Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

Robustness, attention and explainability of neural networks

Dissertation thesis

2024 Mgr. Štefan Pócoš

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

Robustness, attention and explainability of neural networks

Dissertation thesis

Study program:	Informatics
Field of study:	2508 Informatics
Department:	Department of Applied Informatics
Supervisor:	prof. Ing. Igor Farkaš, Dr.

Bratislava, 2024 Mgr. Štefan Pócoš

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Mgr. Štefan Pócoš
Study programme:	Computer Science (Single degree study, Ph.D. III. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Dissertation thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Robustness, attention and explainability of neural networks

Annotation: Explainable AI is becoming a fast developing research area in machine learning using neural networks, standing on the pillars of safety, robustness and transparency, as requirements for successful use of AI. Also, a breakthrough has been achieved by introducing the novel mechanisms of attention. Literature is growing rapidly, providing numerous algorithms focusing on these issues.

Aim:
1. Review the concepts related to robustness, explainability and attention in the context of neural networks.
2. Propose and test novel approaches related to explainability or robustness using deep neural networks on selected tasks (e.g., classification) and analyze the findings.
3. Propose and implement an attention mechanism in a neural network and analyze its properties.

Literature:
Stollenga M.F., Masci J., Gomez F., Schmidhuber J.: Deep Networks with Internal Selective Attention through Feedback Connections, NIPS 2015.
Xu K. et al: Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. ICML 2015.
Waswani A. et al. Attention Is All You Need. NIPS 2017.
Arrieta A.B.: Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. Information Fusion, 2020.

Tutor:	prof. Ing. Igor Farkaš, Dr.
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	doc. RNDr. Tatiana Jajcayová, PhD.
Assigned:	27.01.2020

Approved: 27.01.2020 prof. RNDr. Rastislav Kráľovič, PhD.
Guarantor of Study Programme

.....
Student

.....
Tutor

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Mgr. Štefan Pócoš
Študijný program: informatika (Jednoodborové štúdium, doktorandské III. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: dizertačná
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Robustness, attention and explainability of neural networks
Robustnosť, pozornosť a vysvetliteľnosť neurónových sietí

Anotácia: Vysvetliteľná umelá inteligencia (UI) sa stáva rýchlo sa rozvíjajúcou oblasťou výskumu v oblasti strojového učenia pomocou neurónových sietí. Stojí na pilieroch bezpečnosti, robustnosti a transparentnosti ako požiadaviek na úspešné používanie UI. Prelom sa dosiahol aj zavedením nových mechanizmov pozornosti. Výskum napreduje rýchlo a prináša so sebou veľa algoritmov zameraných na tieto otázky.

Cieľ:
1. Urobte prehľad pojmov súvisiacich s robustnosťou, vysvetliteľnou a pozornosťou v kontexte neurónových sietí.
2. Navrhnite a otestujte nové prístupy súvisiace s vysvetliteľnosťou alebo robustnosťou v hlbokých neurónových sieťach na vybraných úlohách (napr. klasifikácia) a analyzujte zistenia.
3. Navrhnite a implementujte mechanizmus pozornosti v neurónovej sieti a analyzujte jej vlastnosti.

Literatúra:
Stollenga M.F., Masci J., Gomez F., Schmidhuber J.: Deep Networks with Internal Selective Attention through Feedback Connections, NIPS 2015.
Xu K. et al: Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. ICML 2015.
Waswani A. et al. Attention Is All You Need. NIPS 2017.
Arrieta A.B.: Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. Information Fusion, 2020.

Školiteľ: prof. Ing. Igor Farkaš, Dr.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Dátum zadania: 27.01.2020

Dátum schválenia: 27.01.2020 prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

.....
študent

.....
školiteľ

I hereby declare that I wrote this work by myself, only with the help of the referenced literature.

Bratislava, 2024 .....

Mgr. Štefan Pócoš

I want to thank my supervisor Prof. Ing. Igor Farkaš, Dr., for his insightful advice and professional guidance throughout my PhD. A great deal of appreciation goes to I. Bečková for her support while writing this thesis, and also to my parents, who made my studies possible in the first place.

Abstract

Recent advancements in machine learning have led to a state in which achieving more accurate models is often no longer the primary aim. The focus is rapidly shifting towards designing models that can demonstrate diverse but valuable qualities in addition to excellent performance. One of the current challenges in deep learning is the lack of robustness — poor accuracy on out-of-distribution data, which opens the door for deliberate attacks on models, against which they fail to defend. Another concern is the insufficient level of transparency and explainability of deep neural networks. During the interaction with people, the ability to express the model's reasoning is vital, as it leads to greater trust and promotes the deployment of machine learning models. In this work, we aspire to address these two concerns.

One of the key concepts we leverage throughout most of this work is attention. Attention mechanisms in machine learning have brought huge success in the past years. Nevertheless, in our opinion, it is not a fully explored area, and still provides a solid basis for further development. With all this in mind, we focus on three parallel research lines in this work. First, we analyze a group of malicious inputs called adversarial examples. We utilize them in standard deep learning models and propose ways to investigate their distinctions from in-distribution data. Second, in the hope of mitigating the effects of adversarial examples on image classification, we propose and examine an attention-based model, RecViT. Third, we design multiple approaches to build upon the state-of-the-art in the task of addressee classification in the human-robot interaction scenario. By leveraging the attention modules in our model design, we are able to craft human-readable explanations during the addressee estimation. Hopefully, this facilitates smoother and more trustworthy human-robot interaction.

Keywords: attention, explainability, robustness, adversarial examples

Abstrakt

Najnovšie pokroky v strojovom učení nás priviedli do stavu, v ktorom zvyšovanie presnosti modelov často nie je prvoradým cieľom. Úlohou môže byť aj vytvorenie modelu, ktorý okrem excelentnej úspešnosti bude mať aj iné potrebné kvality. Jednou zo súčasných výziev hlbokého učenia je nedostatočná robustnosť, t.j. nízka presnosť na dátach mimo trénovacej distribúcie. Toto otvára dvere pre rôzne útoky, proti ktorým sa modely nedokážu brániť. Ďalším problémom je nedostatočná úroveň transparentnosti a vysvetliteľnosti hlbokých neurónových sietí. Keďže hlboké učenie má v rôznych úlohách ľuďom pomáhať, pri interakcii často vyžadujeme nielen správnu a rýchlu odpoveď, ale aj jej zdôvodnenie. Zvýšená vysvetliteľnosť modelov by preto viedla k širšiemu nasadeniu umelej inteligencie v každodennom živote, a vo všeobecnosti k väčšej dôvere voči týmto modelom. Práve preto sa v tejto práci venujeme robustnosti a vysvetliteľnosti v hlbokom učení.

Jeden z kľúčových konceptov, ktorý v tejto práci využívame, je pozornosť. Mechanizmy pozornosti v ostatných rokoch priniesli obrovské úspechy, no podľa nášho názoru, tento koncept stále nie je dostatočne preskúmaný a ponúka mnohé smery v ktorých by vývoj umelej inteligencie mohol napredovať. Preto v tejto práci skúmame tri paralelné výskumné línie. Po prvé, analyzujeme skupinu škodlivých vstupov, taktiež nazývaných aj adverzariálne vstupy. Tieto dáta používame v štandardných modeloch hlbokého učenia a navrhujeme spôsoby, ako skúmať ich rozdiely od čistých dát. Po druhé, s cieľom zmiernenia účinkov adverzariálnych vstupov na klasifikáciu obrázkov, navrhujeme a testujeme model založený na pozornostnom mechanizme, RecViT. Po tretie, navrhujeme viacero prístupov riešenia odhadu adresáta pri interakcii robota s ľuďmi. Vďaka dizajnu s pozornostným mechanizmom, naše modely ponúkajú spôsoby, ako z nich extrahovať zrozumiteľné vysvetlenia ich rozhodnutí. Dúfame, že práve takáto práca umožní plynulejšiu a dôveryhodnejšiu interakciu človeka s robotom.

Kľúčové slová: pozornosť, vysvetliteľnosť, robustnosť, adverzariálne vstupy

List of Symbols

$\boldsymbol{x}$	general input to a machine learning model
$\boldsymbol{I}$	image input to a machine learning model
l	target class label in a classification task
$\boldsymbol{x}_{adv}$	adversarial input to a machine learning model
$\boldsymbol{\eta}$	perturbation vector
$f_{\boldsymbol{\theta}}$, f	neural network
$\boldsymbol{\theta}$	neural network parameters
f_i	mapping representing i-th layer of a neural network
$\boldsymbol{y}$	output probability vector of a model
$L(\boldsymbol{x}, y; \boldsymbol{\theta})$	loss function of a neural network parametrized by $\boldsymbol{\theta}$
$\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, y; \boldsymbol{\theta})$	gradient of the loss function w.r.t. $\boldsymbol{x}$
$\|\boldsymbol{x}\|_2$	Euclidean norm of a vector $\boldsymbol{x}$
$\|\boldsymbol{x}\|_p$	L_p norm of a vector $\boldsymbol{x}$
$Z(\boldsymbol{x})$	vector of logits (i.e., classifier's output before the softmax)
$\mathbb{E}_{(\boldsymbol{x},y)\sim D}(\cdots)$	expectation when sampling from a distribution D

Acronyms

AE	Adversarial Example
AI	Artificial Intelligence
ALE	Accumulated Local Effects
ALP	Adversarial Logit Pairing
AT	Adversarial Training
BERT	Bidirectional Encoder Representations from Transformers
BIM	Basic Iterative Method
BP	BackPropagation
BPTT	BackPropagation Though Time
CAM	Class Activation Mapping
CNN	Convolutional Neural Network
CW	Carlini & Wagner
DASNet	Deep Attention Selective Network
DL	Deep Learning
DNN	Deep Neural Network
FGSM	Fast Gradient Sign Method
Grad-CAM	Gradient-weighted Class Activation Mapping
GRU	Gated Recurrent Unit
LIME	Local Interpretable Model-agnostic Explanations
LLM	Large Language Model
LR	Learning Rate
LSTM	Long Short-Term Memory

MHA	Multi-Head Attention
ML	Machine Learning
MLP	Multi-Layer Perceptron
MPC	Multi-Party Conversation
NLP	Natural Language Processing
NN	Neural Network
PCA	Principal Component Analysis
PDP	Partial Dependence Plot
PET	Oxford-IIIT Pet
PGD	Projected Gradient Descent
RecViT	Recurrent Vision Transformer
ReLU	Rectified Linear Unit
RMSprop	Root Mean Square Propagation
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
t-SNE	t-distributed Stochastic Neighbor Embedding
UMAP	Uniform Manifold Approximation and Projection
ViT	Vision Transformer
XAI	eXplainable Artificial Intelligence

List of Figures

List of Tables

Contents

Introduction

Machine Learning (ML) is a transformative field that conceptually originated from a model of a biological neuron — the perceptron (McCulloch and Pitts, 1943; Rosenblatt, 1958). Due to the inability of Neural Networks (NNs) to scale to non-linear problems, their popularity at first dropped. However, when the backpropagation was formulated by Rumelhart et al. (1986), NNs again became interesting. Suddenly, it was possible to train a multi-layer perceptron capable of approximating any non-linear function. This opened the door for Deep Learning (DL), i.e., stacking many layers of perceptrons to create large models trained for various tasks. Since then, deep learning has grown into such dimensions that nowadays, we can solve tasks with unprecedented complexities in an end-to-end manner without any explicit programming of the rules generating the decisions.

Deep learning comes with its disadvantages. When employing ML in critical tasks, an erroneous output could lead to serious harm (e.g., medical misdiagnosis, misinterpretation of a traffic sign, heavy financial loss). Due to that, the need to provide explanations alongside the raw outputs has emerged. By design, NNs are not capable of producing them. Therefore, interest in eXplainable Artificial Intelligence (XAI) started to rise, and now it is an intensively researched topic that focuses on bridging the gap between the remarkable performance and the interpretability of the deep learning models (Linardatos et al., 2021).

The lack of interpretability is not the sole flaw of deep learning. It was first shown by Szegedy et al. (2014) that neural networks can be fooled using a class of malicious input called Adversarial Examples (AEs). These inputs often closely resemble realistic, in-distribution data, so the possibility of secretly feeding them into NNs is a highly concerning issue. Numerous defense methods have since been suggested. Sadly, none of them can provide satisfactory robustness (Carlini and Wagner, 2017).

Tightly connected to the previously mentioned topics are methods of attention. Attention mechanisms in the current form, abundantly employed in the state-of-the-art neural networks, originated from works in Natural Language Processing (NLP) (Bahdanau et al., 2015). Later, the attention mechanisms were refined and presented in one of the most influential works of the modern history of Artificial Intelligence (AI), where the transformer was proposed (Vaswani et al., 2017). Nowadays, its adaptation

to the image domain is a state-of-the-art tool for image processing. As attention mechanisms are relatively recent, they provide a solid ground for exciting research, not only to improve the interpretability but also to explore their potential to resist the AEs.

This thesis is structured as follows. In Chapter 1, we briefly introduce *Deep Neural Networks (DNNs)*, their training mechanism, and commonly used architectural designs for image processing (convolutional networks) and sequential processing (recurrent networks). Moving on to Chapter 2, we point out the vulnerable side of deep learning, where we describe the most crucial development of adversarial examples and compare some attacking and defense strategies. Chapter 3 is dedicated to the interpretability and explainability of deep learning, a short taxonomy elaborating on a few, often used techniques. We conclude our theoretical overview in Chapter 4, in which we introduce the attention mechanisms in deep learning, their historical development, and the most prominent publications in the field. In the following chapters, we describe our work, which is divided into three parallel research lines. In Chapter 5, we examine AEs and their distinctions from clean data in deep neural network classifiers. This is followed by the Recurrent Vision Transformer (RecViT) proposal in Chapter 6, using which we aim to narrow the gap between adversarial robustness and generalization. We conclude with Chapter 7, which is dedicated to the proposal of an explainable model for addressee estimation, which can be leveraged in a humanoid robot for a smoother and more trustworthy interaction.

Chapter 1

Artificial neural networks

Neural networks, combined with the training of deep learning models, are now considered one of the key and most influential developments in the relatively new history of machine learning. Thanks to neural networks, humanity achieved the biggest advancements in the history of artificial intelligence. All of this, however, is inspired by the theoretical knowledge about biological principles in the human brain.

1.1 Training of neural networks

Training is crucial to having a working neural network model. But what exactly is training? What should a well-trained network do? Ideally, it should perform the task it was designed to do, flawlessly. Let us make a demonstration on image classification. After training using the training set, we expect the network to correctly classify even the previously unseen images (from the same distribution). Thus, we want it to generalize well and often to output human-like decisions. This is, however, exceptionally hard to formulate precisely. Therefore, during learning, we must be satisfied with optimizing the so-called error function while considering that generalization is our overall goal. An elemental example of an error function is the squared difference between the target and the network output, considering the one-dimensional case.

A neural network consists of numeric parameters (weights) and activation functions. During the training, the weights are modified so that the network output corresponds to the desired one. Only after the *BackPropagation (BP)* was formulated could neural networks be effectively trained (Rumelhart et al., 1986). An optimizer, such as *Stochastic Gradient Descent (SGD)*, is used during the training. SGD performs a small step against the direction of the gradient of the error function for a given input. These steps are repeated many times until the network fulfills our expectations (or, in practice, until it reaches a point where it can no longer learn useful features).

Many beneficial modifications of SGD exist, each with pros and cons. For instance,

keeping the information about previous updates (momentum) can help the network converge faster or overcome local extrema during training (Qian, 1999). Other possibilities are to scale the learning rate per each weight adaptively (i.e., in AdaGrad and RMSprop) (Duchi et al., 2011; Ruder, 2017), or to remove the learning rate as a tunable parameter and automatically determine the best update rate in each step (Zeiler, 2012). One of the most commonly used optimizers is *adaptive moment estimation (Adam)*, which employs exponential moving average of gradients, and similarly constructed normalization of the learning rate (Kingma and Ba, 2014).

Optimizing the performance using training data is often insufficient to reach our goals and teach the network to generalize well. We should also monitor the validation error — error on unseen data. If this error starts increasing, the training should halt, regardless of the error on the training data. Otherwise, the network could get overtrained (i.e., yielding excellent performance on training data while performing poorly on test data). Different approaches, such as *dropout* (Srivastava et al., 2014), prevent overtraining and support regularization. The dropout technique turns off random neurons during training and keeps them active when feeding the test data. Another possibility is to use *weight decay* — an extra modification that keeps the weights small. Thus, it mitigates the accumulation of extreme values.

Data normalization is another crucial step in making models function properly. Normalization does not concern only the input values but also the initialized weights. Those should be initialized such that when computing the forward pass, the final activation neither converges to zero nor diverges to infinity. Various solutions for this have been proposed, depending on what kind of activation functions we use. A quality boost during training can be delivered by using *batch normalization*, proposed in Ioffe and Szegedy (2015). Batch normalization is a technique to standardize the network input and can be applied repeatedly, either to activations before or after a layer output.

The above theory provides only the fundamentals of training deep networks. The general history of individual training methods and their technical description are not the focus of this work, so we will not elaborate on them any further. The following two sections will introduce convolutional and recurrent neural networks. Both concepts serve as inspiration or a base point upon which we build in this work.

1.2 Deep convolutional neural networks

Networks with fully connected layers proved to be an invaluable tool for processing data. But what about images? We can, of course, process an image with fully connected layers as well. To do that, we first need to flatten the image. Even given a smaller image size of 128×128 pixels, the number of inputs is 16384. If we set the number

of neurons in the first hidden layer to achieve a considerable reduction, for example, 100, we need 1 638 400 trainable parameters — more than a million. Training such a network would require a lot of patience and, mainly, a vast amount of training data and computational power. *Convolutional layers* can provide a considerably better solution for image manipulation.

A convolution processes the input image without flattening it, thus it retains the spatial structure. Also, the number of parameters required to calculate the forward pass is only a fraction of the traditional, fully connected layer. All is due to the convolution operator, which is between two functions f and g defined as

$$(f * g)(\boldsymbol{x}) = \int f(\boldsymbol{z})g(\boldsymbol{x} - \boldsymbol{z})d\boldsymbol{z}. \tag{1.1}$$

In the case of discrete objects, the integral gets replaced with a sum

$$(f * g)(i) = \sum_a f(a)g(i - a). \tag{1.2}$$

By adding another dimension to our functions f and g, we get

$$(f * g)(i, j) = \sum_a \sum_b f(a, b)g(i - a, j - b). \tag{1.3}$$

We can view the convolution operator as a measure of overlap between f and g, when one of the functions is "flipped". When a convolutional layer is used in NNs, the convolution kernel is a small matrix $n \times n$ whose values are optimized during training.

We can compute the hidden-layer activations using only a few parameters. For practical reasons, the kernel size is kept relatively small (usually 2×2, 3×3, 5×5, or 7×7). This means that the calculated activation at a certain point with index (i, j) is influenced only by the pixels near (i, j) in the input image. The region of the input image influencing an activation on the hidden layer is called its *receptive field*. Usually, wider kernels or more hidden layers are used to make the receptive fields larger as the receptive fields grow and go deeper into the network.

The convolution is *translation-invariant*, which results from the fact that the kernel is shifted through the input image, and regardless of the position, the same output is produced, given the small region with the same pixel intensities. Translational invariance is often highly desirable. For instance, during image classification, the object of interest can be located anywhere in the image, and we want the network to output the same category, regardless of the object's position.

A convolutional layer deployed in practical applications consists of more than a single kernel, each trained by backpropagation. They tend to learn various features; each filter is sensitive to different structures to capture the most relevant information jointly. In a deep convolutional network, the features learned in the first layers resemble edge detectors, similar to Gabor filters in the human brain. As we go deeper into the

network, the filters tend to be sensitive to increasingly more complex structures, and the last layer often contains filters sensitive to particular objects or parts of them, given that the task is object recognition.

To better manipulate the dimensions of the hidden layer, we use *stride* and *padding*. The padding adds rows and columns of zeros to the activation map to keep the map's original shape even after convolution. On the contrary, stride reduces the number of parameters (shrinks the map) by shifting the kernel by a constant.

It is advisable to use *pooling* after a convolutional layer. Pooling reduces the number of hidden neurons by taking the average or the maximum from small blocks, creating an activation map of a smaller dimension.

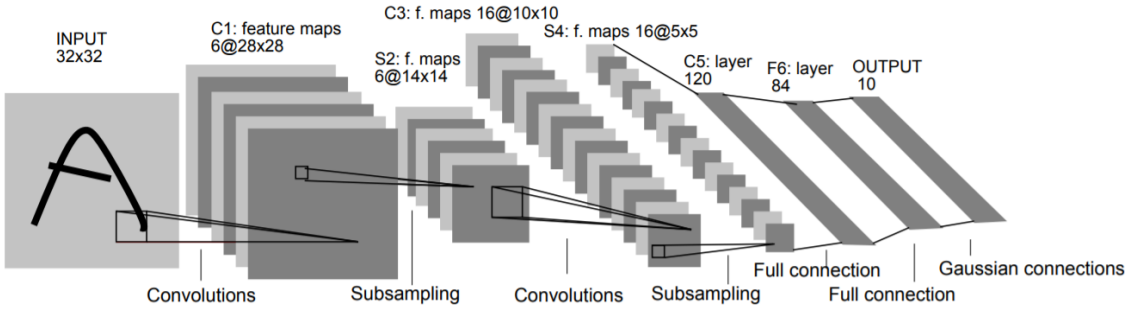


Figure 1.1: The LeNet-5 architecture (LeCun et al., 1998).

The first work, where the power of *Convolutional Neural Networks (CNNs)* was demonstrated, was the creation of LeNet (LeCun et al., 1998), with the architecture depicted in Fig. 1.1. In the following years, convolutional networks were not used very much due to the difficulties of setting the convolutional filters correctly.

A major breakthrough was achieved when AlexNet was introduced (Krizhevsky et al., 2012). AlexNet is a huge convolutional neural network, classifying about 1.2 million images into 1000 categories with the state-of-the-art accuracy, surpassing the previous network by a considerable margin. Since then, CNNs have become the first choice to analyze image data, and many novel architectures incorporating convolutional layers have been proposed.

1.3 Recurrent networks

Previously, we have shown that we can process static data remarkably well by using feed-forward neural networks with convolutional or fully connected layers. On the other hand, let us assume we have a series of inputs $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_t, \in \mathbb{R}^d$, and we want to predict the next one. Considering the previous approach, the only choice is to trim the sequence into equally long parts, such that the input dimension would always be the

same, feed it to the network, and train it by backpropagation to predict the next input. This approach neglects the influence of distant points, thus offering poor performance.

To better manipulate sequential data, *Recurrent Neural Networks (RNNs)* were designed (Elman, 1990). A simple RNN consists of feed-forward architecture and it contains a hidden state $\boldsymbol{h}_t$, which is updated every time step. There are also two trainable matrices $\boldsymbol{W}_x$ and $\boldsymbol{U}_h$ used to compute the hidden state $\boldsymbol{h}_t$, combining both the new input $\boldsymbol{x}_t$ and the previous state of the hidden layer $\boldsymbol{h}_{t-1}$, and a trainable matrix $\boldsymbol{W}_y$ to compute the output $\boldsymbol{y}_t$:

$$\begin{aligned} \boldsymbol{h}_t &= \sigma_h(\boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{U}_h \boldsymbol{h}_{t-1} + \boldsymbol{b}_h), \\ \boldsymbol{y}_t &= \sigma_y(\boldsymbol{W}_y \boldsymbol{h}_t + \boldsymbol{b}_y), \end{aligned} \tag{1.4}$$

where σ denotes the activation functions and $\boldsymbol{b}$ indicates the biases.

Using this approach, the hidden state $\boldsymbol{h}_t$ can theoretically contain information about all previous states and is not limited to a fixed window. Having $\boldsymbol{h}_t$, we can formally condition the output $\boldsymbol{y}_t$ on all previous inputs.

RNNs can be trained by minimizing the cross-entropy loss using BackPropagation Though Time (BPTT) (Werbos, 1990). We can view BPTT as the recurrent network unfolded in time, and thus, the weights can be modified thanks to information from the gradient. Although the training of RNNs seems straightforward, it is often tricky to make them learn more complex tasks.

1.3.1 Long short-term memory

RNNs, especially when multiple layers are stacked on one another (also known as deep RNNs), suffer from the problem of *vanishing* and *exploding gradients*, i.e., gradients progressively becoming extremely small or large, respectively. Another drawback of simple RNNs is their inability to capture long-term dependencies. For example, if $\boldsymbol{x}_1$ significantly influences $\boldsymbol{x}_{100}$, the hidden state at the time step $t = 100$ will barely capture the correct relationship, and the error will be significant. To prevent this kind of issue, the *Long Short-Term Memory (LSTM)* unit was designed (Hochreiter and Schmidhuber, 1997).

An LSTM unit consists of input gate $\boldsymbol{i}_t \in \mathbb{R}^{n \times h}$, forget gate $\boldsymbol{f}_t \in \mathbb{R}^{n \times h}$ and output gate $\boldsymbol{o}_t \in \mathbb{R}^{n \times h}$, which are updated after an input is presented. Using these vector variables along with the cell activation $\tilde{\boldsymbol{c}}_t$, the memory $\boldsymbol{c}_t$ and the hidden state $\boldsymbol{h}_t$ are updated. The exact operations are defined as follows:

$$
\begin{aligned}
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_i \boldsymbol{x}_t + \boldsymbol{U}_i \boldsymbol{h}_{t-1} + \boldsymbol{b}_i), \\
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_f \boldsymbol{x}_t + \boldsymbol{U}_f \boldsymbol{h}_{t-1} + \boldsymbol{b}_f), \\
\boldsymbol{o}_t &= \sigma(\boldsymbol{W}_o \boldsymbol{x}_t + \boldsymbol{U}_o \boldsymbol{h}_{t-1} + \boldsymbol{b}_o), \\
\tilde{\boldsymbol{c}}_t &= \tanh(\boldsymbol{W}_c \boldsymbol{x}_t + \boldsymbol{U}_c \boldsymbol{h}_{t-1} + \boldsymbol{b}_c),
\end{aligned}
\tag{1.5}
$$

$$
\begin{aligned}
\boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t, \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t),
\end{aligned}
\tag{1.6}
$$

where $\odot$ represents the element-wise multiplication, $\boldsymbol{W}$ and $\boldsymbol{U}$ are the trainable matrices and $\boldsymbol{b}$ are the trainable bias vectors.

Afterwards, one can extract the needed information and train the network from the hidden representation $\boldsymbol{h}_t$. Thanks to the specifically designed gates, information can be processed even if it was inserted into the network a long time ago, mitigating the issues with long-term dependencies. The design of LSTM also provides a computationally efficient way to propagate gradients through the memory $\boldsymbol{c}$, reducing the problem of exploding and vanishing gradients.

1.3.2 Gated recurrent unit

Another approach for processing sequential data provides *Gated Recurrent Unit (GRU)* (Cho et al., 2014a). GRU achieves comparable results to LSTM, as it is derived from it. As an advantage, implementing a GRU unit has a smaller computational demand. The GRU unit consists of the update gate $\boldsymbol{z}_t$, the reset gate $\boldsymbol{r}_t$, and the candidate activation vector $\tilde{\boldsymbol{h}}_t$, and is defined as follows:

$$
\begin{aligned}
\boldsymbol{z}_t &= \sigma_g(\boldsymbol{W}_z \boldsymbol{x}_t + \boldsymbol{U}_z \boldsymbol{h}_{t-1} + \boldsymbol{b}_z), \\
\boldsymbol{r}_t &= \sigma_g(\boldsymbol{W}_r \boldsymbol{x}_t + \boldsymbol{U}_r \boldsymbol{h}_{t-1} + \boldsymbol{b}_r), \\
\tilde{\boldsymbol{h}}_t &= \phi(\boldsymbol{W}_h \boldsymbol{x}_t + \boldsymbol{U}_h(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}) + \boldsymbol{b}_h), \\
\boldsymbol{h}_t &= (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t.
\end{aligned}
\tag{1.7}
$$

Similarly, as in LSTM, $\boldsymbol{W}$ and $\boldsymbol{U}$ denote the weight matrices, $\boldsymbol{b}$ the biases and $\odot$ the operation of element-wise multiplication.

For many years, GRU and LSTM have been the best choices when modeling sequential data with NNs. Different approaches are still being invented, creating yet other modifications of LSTM and GRU. These architectural designs are the predecessors to modern, attentional sequential data processing methods, described in Chapter 4.

Chapter 2

Vulnerability of deep learning

Szegedy et al. (2014) were the first to point out that neural networks exhibit certain counter-intuitive properties. A well-generalizing classifier is expected to produce similar answers to two close inputs. While this usually holds, the so-called *Adversarial Examples (AEs)* violate this property. Thus, AEs are inputs to machine learning models that closely resemble clean data but deliberately cause misclassification. More precisely, let us have a well-trained neural network $f_{\boldsymbol{\theta}}$ parameterized by $\boldsymbol{\theta}$, an input $\boldsymbol{x}$ and its corresponding label l, and assume that $\boldsymbol{x}$ is correctly classified: $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = l$. An adversarial example $\boldsymbol{x}_{adv}$ is a modified input, such that $\|\boldsymbol{x} - \boldsymbol{x}_{adv}\| < \epsilon$, but $f_{\boldsymbol{\theta}}(\boldsymbol{x}_{adv}) \neq\ l$.

The existence of these corrupted inputs indicates potential core flaws in the way neural networks generalize. It also raises concerns about the safety of using machine learning modes in critical applications. Even though in this work, we discuss and address AEs exclusively in the vision domain, the lack of robustness has been shown in other applications, such as speech recognition (Carlini and Wagner, 2018), text and malware classification (Ebrahimi et al., 2018; Hu and Tan, 2017), reinforcement learning (Huang et al., 2017), and many more.

2.1 Adversarial attacks

Besides pointing out the vulnerabilities of machine learning models, Szegedy et al. (2014) also suggested a reliable way to produce adversarial examples. Their computation was, however, relatively slow and required many steps of box-constrained L-BFGS optimization.

Shortly after the discovery of AEs, Goodfellow et al. (2015) investigated their linear nature. They explained the existence of AEs as an emergent property of high-dimensional dot products. Having many dimensions, the "adversarial perturbation" aligns with the model's weights, contributing to large activations of certain neurons, which, propagated further, results in a misclassification. Furthermore, the authors in-

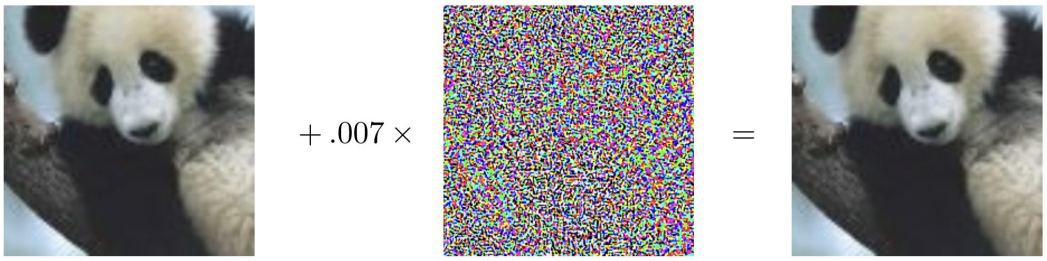


Figure 2.1: Computation of an adversarial example. A small, carefully crafted perturbation (middle) causes a high-confidence misclassification (right) of a previously correctly classified image (left) (Goodfellow et al., 2015).

troduced an efficient way of generating AEs — the *Fast Gradient Sign Method (FGSM)*. Assume we have a neural network classifier $f_{\boldsymbol{\theta}}$ parameterized by $\boldsymbol{\theta}$ and an input-target pair $(\boldsymbol{x}, l)$. Let $L(\boldsymbol{x}, l; \boldsymbol{\theta})$ be the cost function used during the optimization of $f_{\boldsymbol{\theta}}$, then

$$\boldsymbol{\eta} = \epsilon \text{sign}(\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, l; \boldsymbol{\theta})) \tag{2.1}$$

is the analytically computed change of the input $\boldsymbol{x}$ so that the misclassification occurred with high probability. This leads to the following single-step computation of an AE: $\boldsymbol{x}_{adv} = \boldsymbol{x} + \boldsymbol{\eta}$. As demonstrated in Fig. 2.1, the perturbation is often so small that it is imperceptible to a human observer.

Considering the attacker's limitations, adversarial attacks can be split into two main categories: *black-box* and *white-box*. While using black-box attacks, the attacker has very little information about the data or the model. In the white-box setting, the attacker can access any and all vital information, such as the architecture, optimizer, data, and much more. *Gray-box* attacks, which are quite commonly utilized, ease from several limitations of the strict black-box definition; therefore, in this case, little information about the model or the data can be provided[1].

Yet another categorization of the attacks is based on the target specification. The target can be unspecified or fixed to a particular output. When not specifying the target (*non-targeted attack*), the only goal of the attacker is to cause misclassification. The opposite is true when employing a *targeted attack*, where one optimizes the network to produce a specific target value. Nonetheless, since the majority of the known attacks can be written in both forms, further in this chapter, we omit the description of most of the targeted alternatives.

[1]In the literature, gray-box attacks are often labeled as black-box, mainly if the attacker's restrictions are substantial.

2.1.1 White-box attacks

Naturally, further modifications of the existing FGSM attack were proposed. Kurakin et al. (2017) suggested a straightforward extension of the FGSM attack, the *Basic Iterative Method (BIM)*. BIM also stems from the linear approximation of the loss function, but the AEs are found iteratively. In each iteration, a simple FGSM step is applied and the result is clipped to the range of possible pixel values. This results in a slower but stronger attack of generally smaller perceived average perturbation.

The idea of utilizing the linear approximation of the loss function was further refined by Madry et al. (2018), who proposed the *Projected Gradient Descent (PGD)* attack. Similarly to BIM, PGD iteratively computes the gradient w.r.t. the input to progressively converge to an AE. However, in this case, the authors use random restarts from inputs located in the close vicinity of the original image. Due to the repeated computations, the attack has higher potential to overcome local extrema in the multi-dimensional loss landscape.

PGD, BIM, and FGSM are particularly suitable for producing AEs constrained by the L_∞ norm, which translates to having a hard limit for the maximal pixel change. However, as the goal of AEs is to fool the network while resembling the original, the distance measure can vary. Therefore, a more general norm computation, the L_p norm, is often used:

$$\|\boldsymbol{x}\|_p = (|x_1|^p + |x_2|^p + |x_3|^p + ... + |x_{n-1}|^p + |x_n|^p)^{\frac{1}{p}}. \tag{2.2}$$

Keeping this in mind, Carlini and Wagner (2017) introduced three adversarial attacks, each constraining its optimization in different L_p norm (L_0, L_2 and L_∞). All of them proved to generate consistent and unique AEs, but since their L_2 attack (often referred to as the Carlini-Wagner or, shortly, CW attack) stands out the most, we only elaborate on this version. The approach of the proposed L_2 attack is to minimize the joined objective function $\|\boldsymbol{\eta}\|_2 + c \cdot g(\boldsymbol{x} + \boldsymbol{\eta})$ such that $\boldsymbol{x} + \boldsymbol{\eta} \in [0, 1]^n$, where g is a mapping, satisfying $g(\boldsymbol{x} + \boldsymbol{\eta}) \leq 0$ in case of a successful attack. The choice of g was empirically tested and the best effect on the optimization process was yielded by the following function:

$$g(\boldsymbol{x}') = \max(\max\{Z(\boldsymbol{x}')_i : i \neq t\} - Z(\boldsymbol{x}')_t, -\kappa), \tag{2.3}$$

where $Z(\boldsymbol{x}')_i$ is the i-th logit of the input $\boldsymbol{x}'$, t is the target class in case of a targeted attack, and κ is a parameter used for controlling the confidence of the generated AEs. To find an AE with the smallest possible perturbation, a modified binary search is employed to optimize the value of the parameter c.

To be able to use a common optimizer such as Adam (Kingma and Ba, 2014), the authors also introduced a clever reparametrization $\delta_i = \frac{1}{2}(\tanh(w_i) + 1) - x_i$, so that

no clipping was necessary. The resulting AE is always within the correct interval. To obtain an AE, the optimization has the following form:

$$\text{minimize}_{\boldsymbol{w}} \quad \left\| \frac{1}{2}(\tanh(\boldsymbol{w}) + 1) - \boldsymbol{x} \right\|_2^2 + c \cdot f\left(\frac{1}{2}(\tanh(\boldsymbol{w}) + 1 \right). \tag{2.4}$$

Later, Chen et al. (2017) proposed a generalization of the Carlini-Wagner L_2 attack. Here, the objective function is combined with the L_2 and the L_1 perturbation magnitude:

$$\text{minimize}_{\boldsymbol{w}} \quad c \cdot f(\boldsymbol{w}, t) + \beta \|\boldsymbol{w} - \boldsymbol{x}\|_1 + \|\boldsymbol{w} - \boldsymbol{x}\|_2^2, \tag{2.5}$$

while $\boldsymbol{w} \in [0, 1]^p$. The additional utilization of the L_1 norm promotes sparsity in the perturbation. Even though the optimization procedure resembles the CW attack, this version produces qualitatively different AEs.

2.1.2 Black-box attacks

Until now, we have discussed attacks in which the attacker has access to all the required information about the network. However, in reality, the attacker may not have the luxury of unlimited access to the model simply because it is commonly hidden from the users. Luckily for the attackers (and unfortunately for the defenders), it was shown that AEs possess the ability to *transfer*, i.e., an AE fooling one network has high potential to fool another one, trained on a similar task (Szegedy et al., 2014). This was further elaborated on and confirmed in Goodfellow et al. (2015). This opens the door for generating AEs on a substitute model using a white-box attack, then fooling the original one.

Building a surrogate model

Lacking the input corpus, the training of a surrogate model is an immensely difficult task. It often led to methods working under unrealistic settings, such as querying the black-box model too many times (which can be detected by the defender) (Xu et al., 2016). In other cases, sampling of a huge-dimensional input space rendered the training of the substitute model impractical (Srndic and Laskov, 2014). These challenges were eventually resolved by Papernot et al. (2017), who first demonstrated the possibility of employing *practical black-box attacks*. To alleviate the exhaustive sampling of the input space, the authors introduced Jacobian-based dataset augmentation, which approximates the decision boundary of a ML model using as few samples as possible.

In another work, Liu et al. (2017) examined the strength and the predicted class of the AEs transferred from one network to another. Their primary result suggests that having multiple networks and generating AEs to fool all of them results in a particularly strong AEs with high potential to fool another, unprotected network. They also

Figure 2.2: Iterations of the boundary attack. With each call of the black-box model, the image is continuously morphed to be as similar as possible to the original image (Brendel et al., 2017).

noted that the AEs often do not transfer with their target class; nonetheless, when optimized using more networks, the probability of fooling an arbitrary network while keeping the target class increases. This opens space for strong and targeted black-box attacks.

Prompting the network

Assembling a surrogate model is not the only way to produce high-quality black-box attacks. Brendel et al. (2017) proposed the *boundary attack*, in which the attacker starts out by picking an image classified as the intended incorrect label. This is followed by a continuous alteration of the image along the classification boundary to become closer to the given original image while keeping the incorrect prediction label. A downside of this method is the number of calls ($\approx 10^5$), which the attacker needs to make to obtain an AE that closely resembles the original image. The progress of the boundary attack for a specific target is shown in Fig. 2.2.

An attack, which deviates from the strict black-box setting, was introduced in Su et al. (2019), where the vector of output probabilities is used to craft the AEs. The confidence score serves to evaluate the fitness function in a differential evolution algorithm (Storn and Price, 1997). Their main goal, however, was to find only a few pixels to perturb in order for the image to become adversarial. They showed that it is possible to reliably produce AEs with 1 to 3 perturbed pixels, opening a world of attacking possibilities restricted by the L_0 norm.

2.2 Towards robust models

Ever since the discovery of AEs, researchers have been searching for a model that resists the influence of malicious inputs and provides a reliable and proper generalization. The

ultimate goal is to design a model that, in case of receiving out-of-distribution data, behaves as close to human expectations as possible. This section is, therefore, dedicated to a brief overview of several frequently referenced defense techniques.

2.2.1 Defensive distillation

Among one of the earliest and most promising defenses against AEs is *defensive distillation* (Papernot et al., 2016). In the past, in addition to allowing for a better generalization, distillation has been used to reduce the model size. Here, the authors leveraged the generalization properties of distillation to train a more robust network. The training itself consists of two simple steps. First, a *teacher network* is trained using standard backpropagation while setting a high softmax temperature T on the last layer:

$$\text{softmax}(\boldsymbol{x}, T) = \frac{\exp\left(\boldsymbol{x}/T\right)}{\sum_i \exp\left(x_i/T\right)}. \tag{2.6}$$

Second, to promote uncertainty during the inference, a *student network* is trained using the "soft" labels, i.e., the output probabilities from the teacher network. The student network is trained with the same (high) temperature as the teacher network. To harness the benefits of the non-standard training, the softmax temperature is set to $T = 1$ during the inference of the student model. This simple procedure results in a model exhibiting immense robustness gains; according to the authors, in some cases, the success of an attack dropped from 95% to 0.5%.

2.2.2 Noise manipulation

As AEs are usually crafted by adding noise to clear inputs, denoising is a reasonable strategy to alleviate their adverse effects. One of the methods to achieve this is *feature squeezing* (Xu et al., 2017). The authors of the article investigated two distinct image compression techniques: bit depth reduction and spatial median smoothing. To detect an AE, the classification results of the original image vs. the squeezed image are compared. If significant disparities are found, the input is rejected and labeled as adversarial.

The opposite perspective on eliminating the adversarial noise was elaborated on by Liu et al. (2018). Here, instead of denoising, the authors aim to cancel the adversarial perturbation by adding random noise during the inference. This is achieved via "self-ensemble", i.e., an ensemble of multiple outputs from the same model (given the output is non-deterministic). They introduced the *noise layer*, which generates a random noise vector $\boldsymbol{\epsilon}$, where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$, and combines it with the input to the layer $\boldsymbol{x'} \rightarrow \boldsymbol{x'} + \boldsymbol{\epsilon}$. The noise layer is then used before every convolution. The final output is the class with the highest cumulative probability score.

2.2.3 Adversarial training

Back-feeding adversarial examples into the network, also known as *Adversarial Training (AT)*, was first tested in Szegedy et al. (2014). By alternating the training phases (phase 1: clean dataset, phase 2: updated pool of adversarial samples), it was possible to improve the network's robustness on the MNIST dataset (LeCun and Cortes, 2010). Later, Goodfellow et al. (2015) included AEs in a way that disregards the need to actually generate them. They instead incorporated the FGSM attack into the loss function, hence in every computation step, loss for the actual input $\boldsymbol{x}$ and also the hypothetical adversarial input is taken into consideration:

$$\tilde{L}(\boldsymbol{x}, l; \boldsymbol{\theta}) = \alpha L(\boldsymbol{x}, l; \boldsymbol{\theta}) + (1 - \alpha) L(\boldsymbol{x} + \epsilon \text{sign}(\nabla_{\boldsymbol{x}} \text{L}(\boldsymbol{x}, \text{l}; \boldsymbol{\theta})), \text{l}; \boldsymbol{\theta}). \tag{2.7}$$

Using this setup, the adversarially trained network was able to resist the AEs arguably more than before (the attack success dropped from 89.4% to 17.9%). On the other hand, even the rate of transferability (to and from the adversarially trained model) was greatly reduced (19.6 % and 40.9 %).

Advancing in the research line further, the first major and in-depth analysis of adversarial training was done by Madry et al. (2018). Their perspective on the problem of AEs is the following saddle point optimization:

$$\min_{\theta} \rho(\boldsymbol{\theta}), \quad \text{where} \quad \rho(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim D} \left[\max_{\boldsymbol{\eta} \in S} L(\boldsymbol{x} + \boldsymbol{\eta}, l; \boldsymbol{\theta}) \right]. \tag{2.8}$$

The inner maximization corresponds to input perturbation that causes extreme loss values, and the outer minimization should provide a model that resists the adversarial perturbation. The saddle point optimization is claimed to be possible via robust adversarial training using the PGD attack. In the subsequent analysis, the authors successfully used AE to strengthen a network for MNIST classification. However, the optimization of CIFAR10 (Krizhevsky, 2012) did not yield satisfactory robustness within the given perturbation budget.

2.2.4 Adversarial logit pairing

Since adversarial training has gained a lot of popularity, Kannan et al. (2018) proposed a technique for further improvement, the *Adversarial Logit Pairing (ALP)*. In addition to the standard AT, ALP forces the logits of an AE to resemble the logits of their counter-parting clean example. This is achieved by the integration of the following regularization term into the loss function, penalizing the distances of the two logits:

$$\lambda L_Z(Z(\boldsymbol{x}), Z(\boldsymbol{x}_{adv})), \tag{2.9}$$

where λ determines the regularization strength, and L_Z is the loss penalizing the differences in the logits $Z(\cdot)$ of the original $\boldsymbol{x}$ and the adversarial $\boldsymbol{x}_{adv}$ example. While ALP

achieved great improvements when defending against white-box attacks, the robustness against black-box attacks was on par with the state-of-the-art.

2.3 Evaluation of defense strategies

Even though it appears that it is possible, and quite manageable, to train a robust model, the opposite is true. Most of the previously mentioned defense strategies (and considerably more) fail to provide the reported robustness. To support this claim, Athalye et al. (2018) disproved 7 out of 9 defenses published at the International Conference on Learning Representations in 2018. They established three main error types during a defense proposal:

1. **Shattered gradients**: In this case, the defending model has broken gradients. In other words, the gradient becomes non-differentiable, non-existent, or simply provides meaningless direction.

2. **Stochastic gradients**: Due to the non-deterministic nature of inference in some defense mechanisms, the gradient, while available, does not correspond to the optimization necessary to find an AE.

3. **Exploding and vanishing gradients**: When trying to achieve robustness via prolonging the computation time (e.g., more layers or iterative computation of the output), computing the gradient w.r.t. the input can have a similar effect as in the unfolded computational graphs in RNNs — they explode or vanish.

Having pointed out these peculiarities in model design and evaluation, it is now easier to notice how this can lead to erroneous robustness evaluation. When applying white-box attacks to networks based on invalid gradients, they often fail due to incorrect gradient information. However, the inability to find AEs does not prove their nonexistence. In fact, using gradient-free attacks or replacing the corrupted gradients with an approximation of the correct ones (as for defensive distillation) did show that the proposed defenses were indeed invalid.

Defense evaluation is especially tricky when the gradients are broken, but it does not mean that evaluation of other defense types is trivial (Carlini et al., 2019). For example, Tramer (2022) proved that, in theory, it is highly unlikely to achieve the reported detection rate of feature squeezing (Xu et al., 2017), while Lucas et al. (2023) pointed out the flaws in leveraging randomness when defending against white-box attacks.

It was later shown that even a promising defense, such as ALP, is subjected to skewed evaluation. Engstrom et al. (2018) pointed out three main arguments why ALP does not provide proper generalization: 1) by switching to a non-targeted attack, the performance drops significantly; 2) by increasing the number of PGD attack steps,

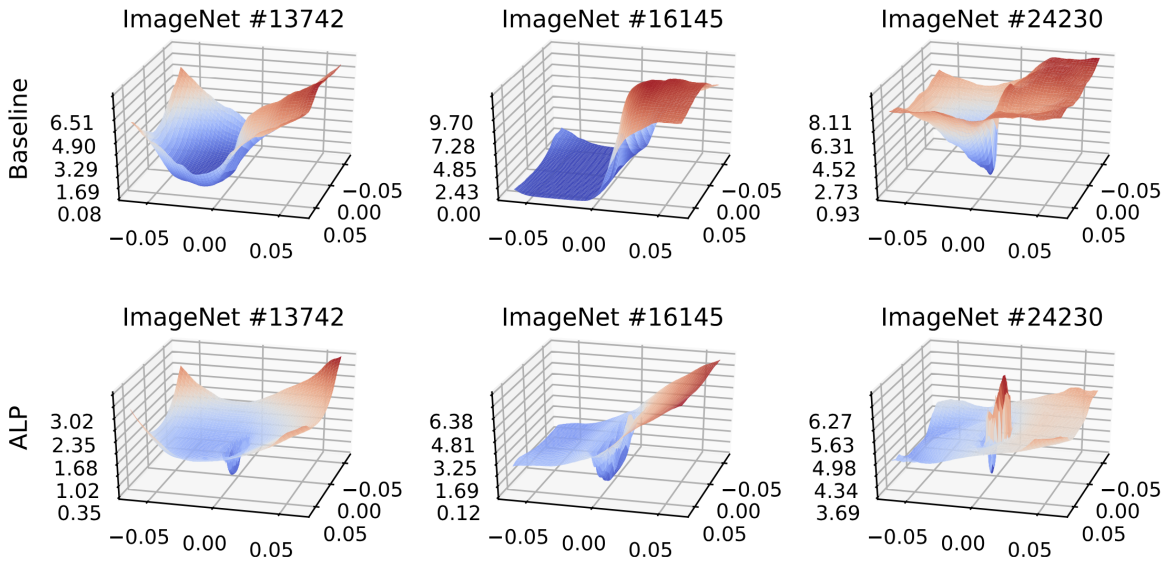


Figure 2.3: Loss landscape visualization for a standard model (top row) vs. a model trained with ALP (bottom row) in three different ImageNet samples (Engstrom et al., 2018).

a higher rate of AEs fooled the network than the originally reported number; and 3) the plot of the loss function after ALP revealed locally distorted gradients (see Fig. 2.3), which means that a strong gradient-free AE can fool the network.

2.4 What next?

While many defenses fail, adversarial training using strong AEs indeed strengthens the deep ML models. The increase of robustness, while often only moderate, comes with a cost of sizeable computational demand during the training, and is usually employed using only specific types of adversarial perturbations, leaving the door open for other attacks (Tramer and Boneh, 2019). Even though there are fair limitations of AT, it is the best method so far for defending against the AEs, and further improvements, such as Kurakin et al. (2018), can make it even better.

One of the advantages the researchers now have is the provided exploration of weaknesses in robustness evaluation (Carlini et al., 2019). Thus, when proposing novel detection or defense methods, researchers have a vast library of tools and knowledge to make the best possible and unbiased evaluations.

Unfortunately, the increase in robustness often comes with lower accuracy on clean data (Tsipras et al., 2019). Several papers promote the idea of an inherent trade-off between robustness and accuracy, i.e., a model cannot resist adversarial examples and, simultaneously, be accurate on clean data (Zhang et al., 2019). On the flip side, some works believe robustness to be achievable by a superb generalization, rendering

robustness and generalization not a contradictory goal (Stutz et al., 2018; Gilmer et al., 2018).

Since robust models are not yet within reach, the attention is shifting towards understanding and explaining the behavior of AEs and out-of-distribution data in the ML systems (Ilyas et al., 2019; Shamir et al., 2022). As artificial intelligence plays an integral part in everyday life, now it is more important than ever to be aware of possible threats. While no permanent solution is found for the robustness problem, new attacks are constantly being developed, exploiting the weaknesses in modern machine learning. An example of this was provided by Cohen et al. (2024) who proposed the "AI worm", an adversarial attack to fool chatGPT-4 and Gemini-pro generative models.

Chapter 3

Explainable and interpretable machine learning

Given the black-box nature of deep learning models, they are commonly required to support their outputs with some reasoning. Due to this, the interest in *eXplainable Artificial Intelligence (XAI)* has steadily grown in the last decade (Linardatos et al., 2021; Barredo Arrieta et al., 2020). While academics discuss XAI on a theoretical level, having models that explain their decisions can also be beneficial from a practical point of view. For developers, it could provide a means for faster and more reliable production. For example, when a deep learning system does not work as expected, it could be possible to discover the root of the problem easier, leading to faster and more efficient model diagnosis. People using DL models would be more confident in the provided output, leading to strengthening trust in machine learning in general. This is especially important in critical applications, where a misjudgment causes catastrophic outcomes. Explainability could provide a way to ensure that the opaque model treats everyone fairly and without biases; therefore, ethical and legal requirements could also be fulfilled if explanations, alongside the predictions, would be provided.

The terms *explainability* and *interpretability* are closely related and often used interchangeably (Zhang et al., 2021). Even though it appears that there is no clear consensus as to what exactly these terms refer to, there have been certain attempts to define them. In a review paper by Angelov et al. (2021), the authors used the definition of interpretability as the "capacity to provide interpretations in terms to be understandable to a human", while explainability "is related to the notion of explanation as an interface between humans and an AI system." These definitions mostly correspond to our usage of these terms in this thesis, yet as with many other definitions regarding this topic, they are not rigorous and lack any mathematical formality. For this reason, we will not indulge in searching for the "best" possible definition and leave this task for future considerations.

3.1 Taxonomy of interpretability

Interpretability techniques are commonly divided into several categories. An especially vital distinction is based on their applicability to specific model families. Interpretability algorithms that are restricted to certain types of models are called *model-specific*. On the other hand, methods that can be applied regardless of the properties of the black-box model are called *model-agnostic*.

Another distinction between interpretability techniques is based on the explanation coverage. *Global* methods focus on explaining the whole model as it is, whereas *local* methods generate explanations for a single input instance, and each time a different input is presented, the explanation needs to be generated anew.

The techniques are often further divided into two other categories. *Post-hoc* methods strive to explain black-box models without changing their architecture or computations, whereas the purpose of *ante-hoc*, also known as *intrinsic* techniques, is to produce models that are interpretable on their own, so that there would be no need to use other methods to explain the model.

It is also important to note that the data type used as an input greatly impacts the choice of interpretability method. Let us make an example. In a mortgage application, one can construct the inputs as vectors encoding crucial aspects, such as age, marital status, income, etc. Using a global method to assess which of the inputs influences the outcome the most is a viable option. On the other hand, applying such a method for image classification (thus searching for a single, most influential pixel) can be computationally very demanding and lead to poor explanations.

3.2 Global methods

In general, global methods are used to determine the relationship between the individual input features and the model output. Even though these methods function properly mainly when applied to simpler models, where the input complexity is also modest (i.e., concatenated features of categorical or numerical data), some of the methods provide exceptionally helpful approximations of the model behavior.

Partial Dependence Plot (PDP) works by altering a chosen feature across all the dataset samples to a fixed value. These newly created input points are used to compute the average output of the model. By progressively selecting all possible values of the selected feature (or, in the case of a continuous feature after a regular sampling), we end up with a relationship between the feature value and the model output. Alternatively, a tuple of features can be fixed simultaneously to produce a 3D plot of their effect on the output. A considerable drawback of this method is the assumption of independent features, which is rarely satisfied in practice. When the data are correlated,

by modification of a single feature, one can create unrealistic data, and feeding them into the model can result in significant errors.

To mitigate the issues with correlated data, Apley and Zhu (2020) proposed the *Accumulated Local Effects (ALE)*. It estimates the partial derivatives by computing the difference in the predictions by slightly altering a single feature in the input. This delicate shift ensures that the data is still realistic and allows the influence of only the feature of interest to be separated. The result is further normalized to differentiate between the effect of the analyzed feature and the result of the input. In cases when the data are too correlated, this method cannot truly capture the whole model.

There are many other methods to interpret the models globally. For example, a model-agnostic method with a simple yet very powerful strategy is to employ a global surrogate. This means replacing a complex black-box model with a simple one that is interpretable by design (e.g., a linear model or a decision tree). In this case, the surrogate model is trained to match the original outputs as closely as possible.

3.3 Local methods

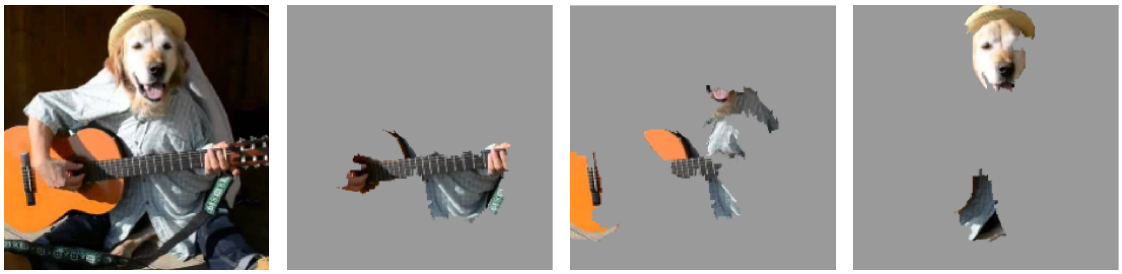
Given the complexity of modern machine learning models, producing global explanations is often an unrealistic wish. To provide explanations, local methods that aim to explain the output for individual input instances are often leveraged.

3.3.1 Model-agnostic methods

One of the easiest ways to create a local interpretation of a model is to use *Local Interpretable Model-agnostic Explanations (LIME)* proposed by Ribeiro et al. (2016). LIME, in its essence, creates a local surrogate simple enough to be interpretable by design. To implement it, one first needs to choose a suitable surrogate model. The second step is to obtain a dataset consisting of randomly perturbed elements of the input around which the model is being approximated. This is followed by the training, where each input contributes with a factor negatively correlating with its proximity to the original point (e.g., a bell curve, having its peak at the input to be explained). The whole process of creating an explanation model can be denoted as the following optimization problem:

$$\text{explanation}(\boldsymbol{x}) = \arg\min_{g \in G} L(f, g, \pi_{\boldsymbol{x}}) + \Omega(g), \tag{3.1}$$

where L is a loss function measuring the discrepancy between the original model f and the local surrogate g given the training data $\pi_{\boldsymbol{x}}$ constructed around a particular input $\boldsymbol{x}$. The surrogate model g is picked from a family of simple and inherently explainable



(a) Original Image (b) Explaining *Electric guitar* (c) Explaining *Acoustic guitar* (d) Explaining *Labrador*

Figure 3.1: Illustration of LIME to explain the highest contributions of image regions (clusters of superpixels) to top three output classes (Ribeiro et al., 2016).

models G. The penalizing term $\Omega(g)$ assesses the complexity of g, yielding high values for complex models.

To use LIME while having numerical features, generating a new dataset around the original point is relatively straightforward. The features can be slightly altered, creating new points close to the original input. The authors also included instructions on dealing with other data types, such as text or images.

- **Text**: In this case, individual words of a sentence can be "turned off," i.e., completely omitted from the sentence. The more words are left out, the farther the new sentence is from the original. These newly created sentences are then used to train the surrogate model.

- **Images**: Random perturbation of pixels would be tedious and not very useful since the pixels in the image are highly correlated. For this reason, the authors proposed to create "superpixels" — interconnected sets of pixels with similar semantic meaning. Those can be found and grouped algorithmically by detecting the neighboring color changes. The superpixels are subsequently used to alter the image. By setting some of them to a neutral activation (for example, gray), a similar strategy to processing the text input can be applied (turning subsets of the superpixels off) to create the dataset. A sample explanation of image data is shown in Fig. 3.1.

Another approach to explaining a model locally is to compute the *Shapley values* (Lundberg and Lee, 2017). In this game-theoretical approach, each feature is a "player" in a game, where the prediction is the "payout." Thus, the goal is to fairly distribute the payout among the players. Computing the Shapley values is quite a common way to explain the model locally.

3.3.2 Model-specific methods

There are also techniques that are specially tailored for a particular model family. Here, we introduce a few of them designed to generate explanations of how neural networks process visual input.

Let us have a grayscale image $\boldsymbol{I}_0 \in \mathbb{R}^{m \times n}$, its corresponding class l, and the class score $y_l = f(\boldsymbol{I}_0)_l$, implemented as a black-box model f. One could ask, how does y_l change in the proximity of $\boldsymbol{I}_0$? Which pixels of the image contribute to class l, or perhaps which pixels contribute to classifying the image to one of the other classes? To have answers for this kind of question can be especially useful when the predictions are not in accordance with the expectations.

Among the first and the most cited works in which the authors calculated image-specific saliency maps in a deep convolutional network is Simonyan et al. (2014). Their motivation for the solution comes from calculating the first-order Taylor expansion, where we are interested in the linear approximation of y_l around the image $\boldsymbol{I}_0$, which can be obtained using partial derivative:

$$\boldsymbol{w} = \left. \frac{\partial f(\boldsymbol{I}_0)_l}{\partial \boldsymbol{I}} \right|_{\boldsymbol{I}_0}. \tag{3.2}$$

Having computed the derivative $\boldsymbol{w}$, they defined the saliency map $\boldsymbol{M} \in \mathbb{R}^{m \times n}$ as $M_{ij} = |w_{h(i,j)}|$, where h is the rearrangement function, mapping the pixel position (i, j) to its corresponding index of the vector $\boldsymbol{w}$ of dimensionality $m \times n$. In the case of an image with multiple color channels, the maximal absolute value through all channels of the derivative is taken, given a fixed position (i, j).

Note that the computation of the derivative consists only of a single backpropagation step. Thus, to create the saliency map, no other operation is required. Even though this method is applicable to multiple types of models, it is included in the category of model-specific methods due to its applicability to models with computable gradients and also the very specific application on image data. The resulting map can be used as an explainability tool to strengthen the trustworthiness of the network and to see whether the correct features of the input were considered when making the decision. On the other hand, this map can also be used as a preprocessing step for other downstream tasks. For example, based on the saliency map defined above, the authors successfully created a weakly supervised automatic object localization (examples are shown in Fig. 3.2).

Guided backpropagation (Springenberg et al., 2015) is yet another method for producing visual explanations in the form of saliency maps, combining *deconvolutional networks* (Zeiler et al., 2010) with the approach of using image-specific gradient information introduced above. These methods, along with *layer-wise relevance propagation* (Montavon et al., 2018), *occlusion maps* (Zeiler and Fergus, 2013), and many more, also

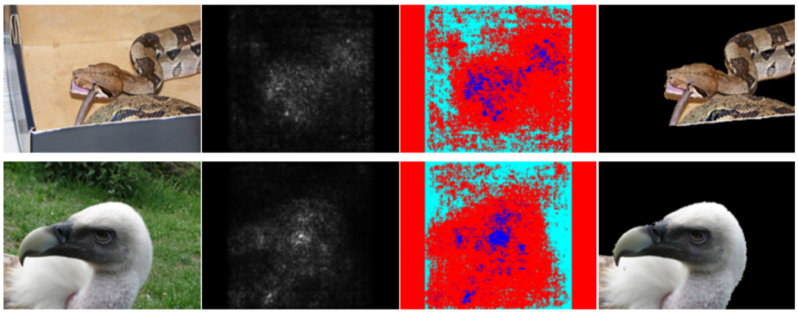


Figure 3.2: Process of semi-supervised object localization. The original image (left) is used to compute the saliency map (middle left). After applying a threshold on the saliency map (middle right), a bounding box is created, which is used to separate the object of interest from the background (right) (Simonyan et al., 2014).

produce insightful information about the salient regions in the images and help make the networks more interpretable. Even though, in recent times, they are somewhat overshadowed by Grad-CAM and its variations (further discussed in the text below), these methods are still in use and provide interesting research opportunities.

A different view on visualization of important image regions is offered by the *Class Activation Mapping (CAM)* (Zhou et al., 2016). CAM creates a map of the same dimensions as the input image, where the value of each pixel is calculated according to its contribution to the output. It uses a specifically designed convolutional neural network trained for image classification. The mapping is obtained as follows:

1. given an image $\boldsymbol{I}_0$, let $f_k(i,j)$ be the activation of the k-th channel at the (i,j) spatial location at the last convolutional layer. Next, F^k is defined as a global average pooling (Lin et al., 2014) of the activations: $F^k = \frac{1}{N}\sum_{i,j} f_k(i,j)$.

2. For a class l, the input to the softmax is $\sum_k w_k^l F_k$, where w_k^l is the corresponding weight of class l to k-th channel. For the CAM algorithm, w_k^l is also the importance of F_k for the class l. Hence, the activation mapping is calculated as:

$$M_c(i,j) = \sum_k w_k^l f_k(i,j) \tag{3.3}$$

As we may notice, the activation map has the same dimensions as the last convolutional layer. To get the importance of each pixel of the input image, the map is simply upsampled to the required dimensions, creating a smoother looking visualisation. The smoothness of the map, however, prevents us from seeing the fine-grained pixel contributions.

Being unable to capture these details is certainly a disadvantage. However, a more severe drawback of CAM is the inability to generalize over different models and its constraint to use convolutional layers, followed by global average pooling. The accuracy of

this kind of model does not get on par with the state-of-the-art architectures, raising the question of whether a more precise model would yield qualitatively different activation mapping. These issues are addressed and solved in the successor of CAM, the *Gradient-weighted Class Activation Mapping (Grad-CAM)* proposed by Selvaraju et al. (2017), which works on a variety of CNN model families, as long as all the network components are fully differentiable.

Grad-CAM does not directly use trained weights to determine the importance of individual feature maps at a given layer as its predecessor CAM, but these weights are replaced by the average partial derivatives w.r.t. the feature map components:

$$\alpha_k^l = \frac{1}{N} \sum_i \sum_j \frac{\partial y_l}{\partial f_k(i,j)}. \tag{3.4}$$

After the importance scores are calculated, the feature maps of a chosen depth (for a given input) are linearly combined, with their corresponding weights α_k^l. In order to keep only the positive influences on the class of interest, ReLU is applied:

$$L^l_{\text{Grad-CAM}} = \text{ReLU}\left(\sum_k \alpha_k^l f_k\right). \tag{3.5}$$

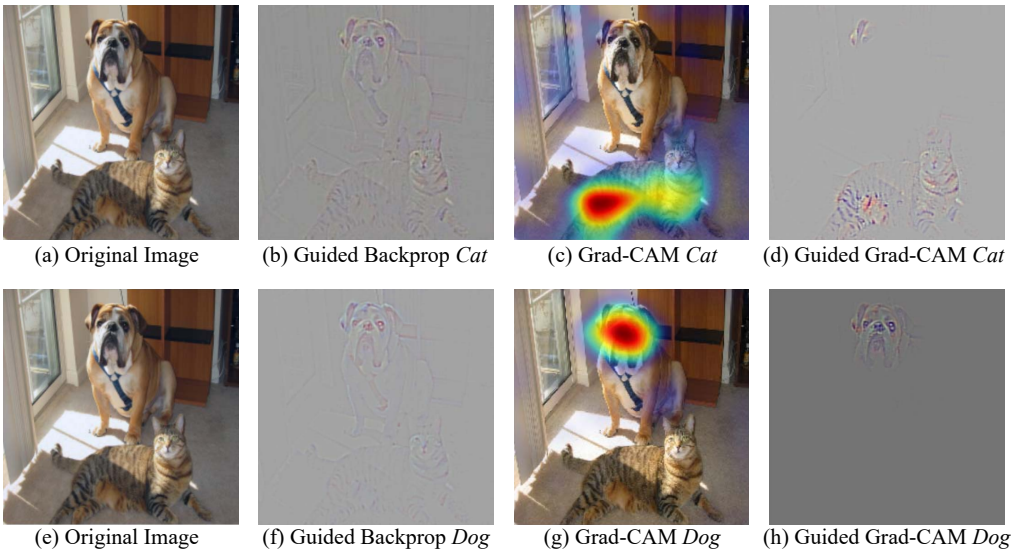
Similarly to CAM, the resulting map is coarse-grained; hence, after the upsampling to the original dimensions, one cannot distinguish the fine details in the input image. For this reason, the authors also applied element-wise multiplication of the $L^l_{\text{Grad-CAM}}$ with the result obtained from guided backpropagation, also known as *Guided Grad-CAM* (see in Fig. 3.3).

Among the major advantages of Grad-CAM is the fact that if the network's components are all differentiable and the initial layers are convolutional, it is capable of producing visualizations regardless of the task. The authors demonstrated this feature in image captioning and visual question-answering tasks.

3.4 Dimensionality reduction

In this section, we introduce techniques that serve for *dimensionality reduction*. Their goal is to transform a set of high-dimensional points into a space of lower dimensions while preserving as much information about the original data structure as possible. In the literature, these methods are rarely labeled as interpretability methods, yet they provide a non-trivial insight into the data. These methods are also often applied to visualize the internal processes in deep black-box models.

The type of dimension reduction matters. For example, one can employ *Principal Component Analysis (PCA)* to preserve the maximum possible variability. Alas, PCA offers only a linear dimensionality reduction and does not consider different data classes.



(a) Original Image (b) Guided Backprop *Cat* (c) Grad-CAM *Cat* (d) Guided Grad-CAM *Cat*

(e) Original Image (f) Guided Backprop *Dog* (g) Grad-CAM *Dog* (h) Guided Grad-CAM *Dog*

Figure 3.3: The process of producing Guided Grad-CAM visualizations of two different target classes that are dominant in the image, cat and dog (Selvaraju et al., 2017).

Therefore, techniques for non-linear dimensionality reduction incorporating information about data point attribution were invented. One such example is *t-distributed Stochastic Neighbor Embedding (t-SNE)* — a technique that is often used to reduce spaces of dimensions reaching up to $10^6 - 10^7$ into a mere 2 or 3 dimensions, allowing 2D or 3D plots of the approximated data (van der Maaten and Hinton, 2008). Given the stochastic nature of t-SNE, after repeating the algorithm, the resulting projection of the same data points can vary, which weakens the reliability and reproducibility of the results. To find the point coordinates after reduction, it employs gradient descent to minimize the Kullback–Leibler divergence between the joint probabilities of the high-dimensional data and the low-dimensional embedding.

Uniform Manifold Approximation and Projection (UMAP), proposed by McInnes et al. (2018), often considered the Riemmanian t-SNE variant, serves the same purpose. It is theoretically well-founded and uses cross-entropy loss instead of the Kullback–Leibler divergence. UMAP is generally considered a more powerful and accurate technique for dimensionality reduction that better preserves the global structure of the data. Another advantage of UMAP is that alongside the projection of a set of points provided at the beginning, it also creates a mapping that can be later used to instantly project new points into a lower dimensional space.

It is worth noting that both t-SNE and UMAP require a proper configuration of their hyperparameters, which can drastically influence the resulting point distribution. Unfortunately, there are no rigorous rules on how to set them, and one needs to use intuition alongside some general recommendations. Due to this and the fact that the methods themselves are highly non-linear, even though they are frequently used to visualize the data, their overall level of explainability remains shallow.

Chapter 4

Attention mechanisms

In this chapter, we introduce some of the key ideas about incorporating attention into machine learning systems. Although in this thesis we mainly focus on applications in computer vision, attention mechanisms in the form that are used in the state-of-the-art models are primarily rooted in the works on *Natural Language Processing (NLP)*.

4.1 The origins of attention

A biological view of attention is provided in Knudsen (2007), where the author summarizes four fundamental processes of human attention: working memory, top-down sensitivity control, competitive selection, and bottom-up filtering for salient stimuli. Each of these processes plays its own role in decision-making executed in the working memory. First, a signal is perceived, and salience filters are applied to extract only the most relevant information. Second, a neural representation is formed. Third, the information is subjected to competitive selection, where only the signals with high intensities are processed. Fourth, the signal arrives in the working memory, ready to render a decision, albeit not the final one. From the working memory, there is also a top-down flow of information, altering neural representations and creating a closed loop, rendering the decision process more informative and reliable.

Just like the human brain model inspired the development of artificial neural networks, the idea of incorporating attention mechanisms into artificial systems also originated from the way humans perceive and process information.

Among the first implementations of attention in neural networks is the work of Schmidhuber and Huber (1991). The authors focused on the task of analyzing and extracting predefined details of 2D objects in a pixel plane, which were subjected to translations and/or rotations. Instead of using a standard feedforward neural network, which requires hundreds or thousands of tediously labeled data, they designed an artificial fovea, which adaptively controlled its movements for target detection. Biological

motivation to model the gaze selection by a neural network was further elaborated on by Denil et al. (2012), where an attention mechanism was used to track and recognize data.

Mnih et al. (2014) proposed an *encoder–decoder* architecture for generating images in multiple steps. In each step, the network focuses on a different part of an image. It was argued that thanks to the incorporation of the attention mechanism, image generation more closely resembled the human drawing process.

Around the same time, an attention mechanism in a form that resembles the current techniques was designed for solving hand-written text synthesis using a recurrent network (Graves, 2014). The authors first demonstrated that RNNs with LSTM units can generate sentences with complex structures by predicting one data point (a word or even a single letter) at a time. They followed by tackling the problem of generating hand-written sentences, providing solid ground in using attention mechanisms for similar applications.

4.2 Natural language processing

The development of attention mechanisms took a rapid turn, predominantly due to the interest in effective processing of natural language. As we shall see in this section, the incorporation of attention in language processing started with the so-called encoder–decoder architecture designed for translation. These ideas have since been repeatedly used and modified to specific needs, contributing to models with extraordinary performance.

4.2.1 Encoder–decoder

During sentence translation, we do not know the length of the translated sentence beforehand. Hence, it is questionable how to approach this task with recurrent networks. This barrier was overcome simultaneously by Sutskever et al. (2014) and Cho et al. (2014b), who proposed a jointly trained encoder–decoder pair, each implemented by a dedicated recurrent network with LSTM or GRU units, respectively.

The encoder is used for creating a fixed-dimensional representation $\boldsymbol{c}$ of the input sentence $(\boldsymbol{x}_1, ..., \boldsymbol{x}_T)$, where $\boldsymbol{x}_i$ are the word embeddings. Simply, words of the input sentence are presented to the encoder, and the hidden activation formed after the last word is considered as the sentence representation $\boldsymbol{c}$. Afterwards, the decoder's hidden state is initialized using $\boldsymbol{c}$, and the output words $(\boldsymbol{y}_1, ..., \boldsymbol{y}_{T'})$ are generated one by one:

$$p(\boldsymbol{y}_t|\boldsymbol{y}_1, ..., \boldsymbol{y}_{t-1}, \boldsymbol{c}) = g(\boldsymbol{y}_{t-1}, \boldsymbol{s}_t, \boldsymbol{c}), \tag{4.1}$$

where g is a non-linear, potentially multi-layered function (outputting the probability

of $\boldsymbol{y}_t$) and $\boldsymbol{s}_t$ is the hidden state of the decoder. The training is fully supervised, maximizing the conditional probability:

$$p(\boldsymbol{y}_1, ..., \boldsymbol{y}_{T'} | \boldsymbol{x}_1, ..., \boldsymbol{x}_T) = \prod_{t=1}^{T'} p(\boldsymbol{y}_t | \boldsymbol{y}_1, ..., \boldsymbol{y}_{t-1}, \boldsymbol{c}). \tag{4.2}$$

As we can see from the equation, in order to find the best translation, exploring all options is intractable. Instead, Sutskever et al. (2014) proposed *beam search*, which keeps in the memory only the top k possible predictions. Experiments also showed that reversing the order of words of the input sequence when feeding the encoder makes a significant improvement of the final translation. Instead of mapping the sentence $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ to $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$, the sentence $\boldsymbol{c}, \boldsymbol{b}, \boldsymbol{a}$ is mapped to $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$ arguing that, in this case, the input word $\boldsymbol{a}$ is much closer to its corresponding word $\boldsymbol{\alpha}$.

The previously mentioned model was further substantially improved upon by Bahdanau et al. (2015), with incorporating an attention mechanism. In the new model, the conditional probability is defined as

$$p(\boldsymbol{y}_i | \boldsymbol{y}_1, ..., \boldsymbol{y}_{i-1}, \boldsymbol{x}) = g(\boldsymbol{y}_{i-1}, \boldsymbol{s}_i, \boldsymbol{c}_i), \tag{4.3}$$

where the main contribution is the usage of distinct context vector $\boldsymbol{c}_i$ to generate every output word. Another deviation from the old approach is the fact that the encoder consists of a bidirectional RNN (Schuster and Paliwal, 1997). The authors defined annotations $\boldsymbol{h}_i$, as concatenations of hidden activations of those stages of the bidirectional RNN, when the i-th word was presented: $\boldsymbol{h}_i = [\boldsymbol{h}_i^{forw}, \boldsymbol{h}_{T-i}^{backw}]$. Each annotation carries the information about the whole sentence, but the i-th annotation $\boldsymbol{h}_i$ has seen the i-th word most recently and thus is primed for its corresponding word.

After the annotations are successfully extracted, the context vector $\boldsymbol{c}_t$ is computed as a weighted combination of the annotations $\boldsymbol{h}_i$:

$$\boldsymbol{c}_i = \sum_{j=1}^{T} \alpha_{ij} \boldsymbol{h}_j, \qquad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}, \tag{4.4}$$

where $e_{ij} = a(\boldsymbol{s}_{i-1}, \boldsymbol{h}_j)$ is the *alignment model*:

$$a(\boldsymbol{s}_{i-1}, \boldsymbol{h}_j) = \boldsymbol{v}_a^T \tanh(\boldsymbol{W}_a \boldsymbol{s}_{i-1} + \boldsymbol{U}_a \boldsymbol{h}_j). \tag{4.5}$$

Here, $\boldsymbol{W}_a \in \mathbb{R}^{n \times n}$, $\boldsymbol{U}_a \in \mathbb{R}^{n \times 2n}$ and $\boldsymbol{v}_a \in \mathbb{R}^n$ are all parameters of the model, jointly trained with the encoder–decoder pair.

The attention mechanism incorporated in the described architecture not only helps the model to perform better, but also allows us to visualize the inner representations in an understandable way, making the model more interpretable. Particularly, by visualizing the weights of the annotations α_{ij}, one can see where the decoder focuses in order to generate the best translation for each predicted word (example in Fig. 4.1). It

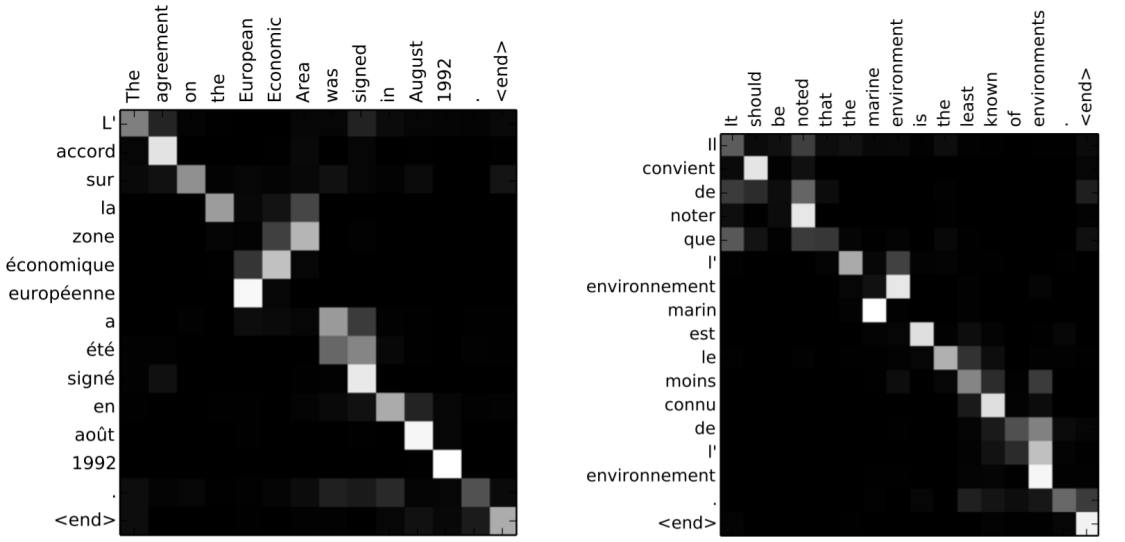


Figure 4.1: An illustration of learned attention weights α_{ij} in encoder–decoder architecture with attention mechanism for selecting specific context (Bahdanau et al., 2015).

was empirically confirmed that the increased attention weights correspond with human intuition.

In subsequent work, Luong et al. (2015) proposed and further examined the implementation of *global* and *local attention* into encoder–decoder architecture, where the underlying model was fairly similar to the model used by Sutskever et al. (2014). Both global and local attention produce a hidden state $\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_\mathrm{c}[\boldsymbol{c}_\mathrm{t}; \boldsymbol{s}_\mathrm{t}])$, which is linearly projected and fed through a softmax layer to output the probability distribution for vocabulary words. The difference between these two incorporations of attention is the way how the time-specific context $\boldsymbol{c}_t$ is produced.

Using the global method, the context vector is computed using all the representations of the input words (taking their weighted combination). The only question is how to assign values for the weights $\boldsymbol{\alpha}_t$ corresponding to the input word representations. Three different ways were proposed, where for each hidden representation of the output $\boldsymbol{s}_t$, a score function calculates the values for all possible hidden states $\boldsymbol{h}_i$ of the input sentence, which are then normalized by applying softmax:

$$\mathrm{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \begin{cases} \boldsymbol{s}_t^T \boldsymbol{h}_i & \text{dot,} \\ \boldsymbol{s}_t^T \boldsymbol{W}_a \boldsymbol{h}_i & \text{general,} \\ \boldsymbol{v}_a^T (\boldsymbol{W}_a [\boldsymbol{s}_t; \boldsymbol{h}_i]) & \text{concat.} \end{cases} \tag{4.6}$$

As we can see, the individual methods differ mainly in the way of parametrizing the alignment model. The concatenation method has the highest number of trainable parameters and an analogy was tested in the previous approach by Sutskever et al. (2014). On the other hand, the dot method does not use any trainable weights.

The authors tested an additional way to compute the weights $\boldsymbol{\alpha}_t$, based solely on the decoder's state:

$$\boldsymbol{\alpha}_t = \text{softmax}(\boldsymbol{W}_a \boldsymbol{s}_t). \tag{4.7}$$

On the contrary to the global attention, by incorporating local attention, we do not take into account the representations from the whole input sentence, but only of a particular subset. The model first generates the aligned position p_t and computes the alignment only using a symmetric interval around p_t, $[p_t - D, ..., p_t + D]$, where D is chosen empirically. Two approaches of setting p_t were tested:

1. monotonic alignment: $p_t = t$, assuming that the source and target sentences are roughly monotonically aligned,
2. predictive alignment: $p_t = S \cdot \text{sigmoid}(\boldsymbol{v}_p^T \tanh(\boldsymbol{W}_p \boldsymbol{h}_t))$, where S is the length of the source sentence.

In this case, the alignment weights are calculated as

$$\alpha_t(i) = \text{align}(\boldsymbol{s}_t, \boldsymbol{h}_i) \exp\left(-\frac{(i - p_t)^2}{2\sigma^2}\right), \tag{4.8}$$

where $\sigma = \frac{D}{2}$ is set empirically.

Experimental results showed that the encoder–decoder architecture accompanied by the alignment model computing the attention weights significantly outperforms the model without attention in the task of translating between English and German. The model with the local attention yielded a gain of up to 5.0 BLEU score (Papineni et al., 2002) improvement. The rest of the attentional models also performed very well, their ensemble outperforming the previous state-of-the-art models in translating from English to German and vice versa.

4.2.2 Follow-up work

The encoder–decoder type architecture accompanied with attention proved to be a useful idea for extracting the information from long recurrent networks without relying too much on the representation compressed in a fixed-dimensional vector. However, the concept of extracting information from RNNs using an alignment model is general enough to be applicable to countless tasks. Here, we describe several models where attention in this form (not necessarily in encoder–decoder architecture) was used to improve the state-of-the-art methods.

Hermann et al. (2015) tackled the problem of question answering. Having a query $\boldsymbol{q}$ based on a context $\boldsymbol{c}$ (usually a sentence or a short paragraph), we seek an answer $\boldsymbol{a}$ from our vocabulary; thus, we need to estimate the probability $P(\boldsymbol{a}|\boldsymbol{c}, \boldsymbol{q})$ as accurately as possible. As an input to a recurrent LSTM, the document is presented word by word.

Afterwards, a document-related query $\boldsymbol{q}$ follows, which has the form of a sentence with a missing word in it. The network should correctly fill in the missing word.

The authors suggested an *attentive reader* to process the input. A bidirectional LSTM is employed, where the representations of the input words $\boldsymbol{h}_d(t)$ are extracted as the concatenation of the hidden states in opposite directions, similarly to the previous work. These representations are then used to create a query-specific linear combination according to the attention function:

$$\begin{aligned} m(t) &= \tanh(\boldsymbol{W}_{hm}\boldsymbol{h}_d(t) + \boldsymbol{w}_{um}\boldsymbol{u}), \\ s(t) &= \exp(\boldsymbol{w}_{ms}^T m(t)), \\ \boldsymbol{r} &= \boldsymbol{h}_d \boldsymbol{s}, \end{aligned} \tag{4.9}$$

where $\boldsymbol{u}$ is the concatenation of the first and the last read of the query — representing the question itself. $\boldsymbol{W}$ and $\boldsymbol{w}$ denote the trainable parameters, tuned for extracting the attention weights.

The query-specific representation $\boldsymbol{r}$, along with the representation of the query itself $\boldsymbol{u}$, are then used to compute the output:

$$g^{AR}(\boldsymbol{d}, \boldsymbol{q}) = \tanh(\boldsymbol{W}_{rg}\boldsymbol{r} + \boldsymbol{W}_{ug}\boldsymbol{u}), \tag{4.10}$$

with $\boldsymbol{W}$, as usually, being trainable parameters.

The authors also experimented with an architecture called the *impatient reader*, where the basic mechanisms are similar to the attentive reader but are further enhanced, and the network re-calculates the representation $\boldsymbol{r}$ for each of the words in the query, allowing to re-read the sentence. Both the attentive and the impatient reader showed promising results and surpassed the methods with no attention mechanism.

Another work in the same spirit was *Recognizing Textual Entailment (RTE)* by Rocktaschel et al. (2016). RTE is the task of determining whether two sentences have nothing in common, contradict each other, or the first sentence (premise) entails the second (hypothesis).

The representations from reading the hypothesis $\boldsymbol{Y}$ are linearly combined using the constructed attention weights $\boldsymbol{\alpha}$. The output of the process is a vector $\boldsymbol{r}$, manufactured using the information from the premise and the hidden state when feeding the RNNs with the last word of the hypothesis $\boldsymbol{h}_N$. The resulting representation of the task $\boldsymbol{h}^*$ is then computed as follows:

$$\begin{aligned} \boldsymbol{M} &= \tanh(\boldsymbol{W}^y\boldsymbol{Y} + \boldsymbol{W}^h\boldsymbol{h}_N \otimes \boldsymbol{e}_L), \\ \boldsymbol{\alpha} &= \mathrm{softmax}(\boldsymbol{w}^T\boldsymbol{M}), \\ \boldsymbol{r} &= \boldsymbol{Y}\boldsymbol{\alpha}^T, \\ \boldsymbol{h}^* &= \tanh(\boldsymbol{W}^p\boldsymbol{r} + \boldsymbol{W}^x\boldsymbol{h}_N), \end{aligned} \tag{4.11}$$

where $\boldsymbol{W}$ and $\boldsymbol{w}$ are trainable parameters, $\boldsymbol{e}_L$ is a vector of ones, and $\otimes$ denotes the outer product.

This idea was further enhanced by the method called *word-by-word attention*. In this case, the attentional vector $\boldsymbol{\alpha}_t$ (indicating the premise weights) is calculated for each word of the hypothesis. This allows us to take into consideration each of the words of the hypothesis one by one. The equations are similar to those of the previous system, the main difference being the iterative calculation of $\boldsymbol{M}_t, \boldsymbol{\alpha}_t$ and $\boldsymbol{r}_t$, using previous values of $\boldsymbol{r}_{t-1}$.

Document classification is yet another task, where attention proved to be useful. Yang et al. (2016) used the idea of attention mechanism in recurrent neural networks in the context of text classification. The proposed method for document classification is divided into two parts. First, using an RNN with GRU units, representations of individual words are calculated. This is done in a similar fashion as usual, and attention weights are calculated. Second, the representations of individual words are combined to create the representation of the individual sentences. This process corresponds to attention at a lower level in the hierarchy.

Third, the sentence representations proceed to the higher level in the hierarchy, where they are fed into a different RNN with GRU units. As usual, they are linearly combined with attention weights computed using an alignment model to produce a vector representing the document. The document representation is fed into an additional layer, a fully connected classifier, to output the class of the document.

4.2.3 Transformers

The *transformer* architecture proposed by Vaswani et al. (2017) is one of the most influential works of recent years. As earlier described, RNNs were dominant in processing sequential input data, such as text strings. Here, a feed-forward architecture was proposed to manage sequential data without the need to use any recurrent processing.

The original purpose of the transformer architecture was to handle language translation better than the previous methods, which is achieved by using only the attention mechanism to deal with sequential information. Let us have an input sentence $\boldsymbol{X}$ consisting of n words ($\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]$). The first step is to apply word embedding to the words $\boldsymbol{x}_i$ of the input sentence $\boldsymbol{X}$. To ensure that no information about the position is lost, the so-called *positional encoding* $\boldsymbol{p}(i)$ is generated and added to the corresponding word representations. To compute the encodings, alternating sine and cosine functions are used, changing their frequency according to the index i of the word

and the dimension d, together creating the positional encoding vectors:

$$\begin{aligned} PE(i, 2d) &= \sin(i/10000^{2d/d_{\text{model}}}), \\ PE(i, 2d+1) &= \cos(i/10000^{2d/d_{\text{model}}}). \end{aligned} \tag{4.12}$$

The input vectors are created as $\tilde{\boldsymbol{x}}_{\boldsymbol{i}} = \text{embed}(\boldsymbol{x}_{\boldsymbol{i}}) + \boldsymbol{p}(i)$, compactly represented in the matrix $\tilde{\boldsymbol{X}}$. This input is further linearly projected using three different transformations, forming the *queries* $\boldsymbol{Q}$, *keys* $\boldsymbol{K}$ and *values* $\boldsymbol{V}$.

The queries and the keys are multiplied to create the weights to combine the values according to

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V}, \tag{4.13}$$

where d_k is the dimensionality of keys. This is an integral part of the computation that replaces the recurrence, as in this phase, every word attends to every other word, directly interconnecting the whole sequence in a single operation.

Analogously with multiple convolutional kernels used in CNNs, the attention is also being applied in parallel multiple times in the *Multi-Head Attention (MHA)*:

$$\begin{aligned} &\text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{Concat}(head_1, head_2, ..., head_h)\boldsymbol{W}_o, \\ &head_i = \text{Attention}(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V), \end{aligned} \tag{4.14}$$

where $\boldsymbol{W}_o, \boldsymbol{W}_i^Q, \boldsymbol{W}_i^K$ and $\boldsymbol{W}_i^V$ denote trainable parameters.

The output of the MHA is of the same dimensions as the matrix $\tilde{\boldsymbol{x}}_{\boldsymbol{i}}$, so this step usually does not include dimensionality reduction. For further processing, fully connected feed-forward layers follow. Skip connections are added to the output of both the fully connected and the attention layers, enabling more fluent gradient flow during training. This scheme of MHA preceding fully connected layers is stacked several times to produce better representations of the input.

When the input sentence is processed, a query is no longer obtained from the input representation, but rather from the output (if some words are already translated). The formed representation then goes through fully connected layers. As in the first part (the encoder), the decoder also consists of stacked layers to form a better representation of the sentence. The last step is to compute the output probabilities with a final fully connected layer. A clearer and more detailed explanation of the architecture can be found in Fig. 4.2.

The transformer architecture was a huge step forward, allowing for building the state-of-the-art models of the current times, for example, the *Bidirectional Encoder Representations from Transformers (BERT)*, proposed by Devlin et al. (2019). BERT is a specialized architecture based on the attention mechanism used in the transformer, designed to solve multiple tasks by a large pre-trained language model. Given a specific NLP related task, BERT can be fine-tuned without the need to process too much data.

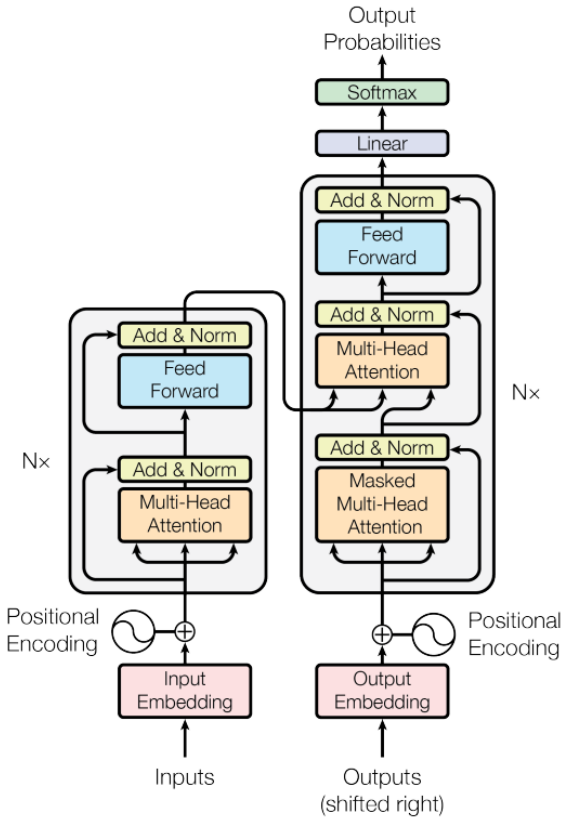


Figure 4.2: Transformer architecture (Vaswani et al., 2017).

Using this strategy, the authors achieved state-of-the-art performance in all 11 tested tasks, such as question answering, sentence pair classification, sentence tagging, and more, demonstrating the supremacy of attention in processing sequences.

4.3 Image processing

Although the rise of attention mechanisms in artificial systems began in the domain of natural language processing, the interest in using attention in vision models has also been slowly growing. In this section, we introduce two important vision models that provided inspiration for our subsequent work, both incorporating attention mechanisms, and then we conclude with the state-of-the-art visual attention model — the vision transformer.

4.3.1 Deep attention selective network

Stollenga et al. (2014) introduced one of the first convolutional architectures modified to use an attention mechanism, the *Deep Attention Selective Network (DASNet)*. The model was based on deep CNNs using maxout units (Goodfellow et al., 2013). In this work, the usual *bottom-up* architecture was enhanced by *top-down* connections, as the

biological processes in the human brain also draw advantage of such connections.

Compared to the previous models, the main modification is the way convolutional filters are manipulated. Each of them is scaled by a factor of a_i ($\boldsymbol{a}$ is set to the vector of ones in the beginning). Thus, the vector $\boldsymbol{a}$ represents an *action*, and the concatenation of specific activation values (channel averages, intermediate activations, and a class probability vector) from the network for a given input represents an *observation*. Hence, to train the network properly, besides the standard training, reinforcement learning is applied to learn the *policy*, which maps the observation to an action determining the filter weights in the next state (when the prediction is repeated with the new state of $\boldsymbol{a}$). This mapping is provided by a trained NN, which is learned using *Separable Natural Evolution Strategies (SNES)* (Schaul et al., 2011).

After a suitable policy is found, the weight vector $\boldsymbol{a}$ is iteratively modified (given a fixed number of iterations T), and the resulting $\boldsymbol{a}_t$ determines the "selected" attention that is applied to the individual filters to produce the prediction.

During the testing of DASNet, the author noted that the biggest increase in accuracy was provided for the images that were in-between two classes. DASNet managed to focus its attention on relevant parts of the input, thus yielding higher accuracy and setting the state-of-art on two benchmark datasets.

4.3.2 Image caption generator

A type of attention-based neural network generating image captions was proposed by Xu et al. (2015). The authors used the theoretical knowledge about attention in recurrent models to build a model that can select specific parts of an image to generate its caption.

First, using the CNN trained in Simonyan et al. (2014) and an input image, the annotation vectors $(\boldsymbol{a}_1, ..., \boldsymbol{a}_L), \boldsymbol{a}_i \in \mathbb{R}^D$ are extracted from a lower convolutional layer as a concatenation of channels on a certain pixel position. Thus, each annotation vector refers to a small region of the input image.

Second, an RNN with LSTM units (encoder) is deployed. The input to the recurrent network consists of the embedded, previously generated word, the previous hidden state of the RNN ($\boldsymbol{h}_{t-1}$), and a vector containing the image data ($\hat{\boldsymbol{z}}_t$). To process the annotation vector, a weight is calculated for each of them. Those are assigned by the attention model f_{att}, conditioned on the previous hidden state $\boldsymbol{h}_{t-1}$ of the RNN and implemented by an MLP. Before the output is further processed, softmax is applied:

$$e_{ti} = f_{\text{att}}(\boldsymbol{a}_i, \boldsymbol{h}_{t-1}), \qquad \alpha_{ti} = \frac{\exp(e_{ti})}{\sum_{k=1}^{L} \exp(e_{tk})}. \tag{4.15}$$

After the weights are calculated, the vector $\hat{\boldsymbol{z}}_t$ is computed as:

$$\hat{\boldsymbol{z}}_t = \phi(\{\boldsymbol{a}_i\}, \{\alpha_i\}) = \begin{cases} \sum_i^L \alpha_i \boldsymbol{a}_i, & \text{soft attention,} \\ \sum_i^L s_{t,i} \boldsymbol{a}_i, & \text{hard attention.} \end{cases} \tag{4.16}$$

We see that the function ϕ is split into two branches, corresponding to two methods of considering the attention: *soft* and *hard*. In the case of soft attention, the resulting representation $\hat{\boldsymbol{z}}_t$ is simply a linear combination of annotations, similarly as done in Bahdanau et al. (2015). On the other hand, the hard attention uses a one-hot vector $\boldsymbol{s}_t$ determined by $p(s_{t,i} = 1 | s_{j<t}, \boldsymbol{a}) = \alpha_{t,i}$.

The final step is to use an MLP to calculate the word probabilities for time t, conditioned on the hidden state $\boldsymbol{h}_t$. The soft attention can be trained by BackPropagation as all the components are fully differentiable. On the other hand, the hard attention needs to be trained by the REINFORCE update rule (Williams, 1992) due to its non-differentiability.

Both methods produce high-quality image captions, resulting in state-of-the-art performance. In addition to good-quality caption generation, using the calculated weights at each time step, it is possible to visualize the regions where the network is focusing when it is generating the caption for a specific word.

4.3.3 Vision Transformers

Motivated by the success of the transformer architecture, many have extended the intended usage of the attention mechanism for alternative tasks. One of them is the *Vision Transformer (ViT)* by Dosovitskiy et al. (2021), who adapted the transformer-like architecture for the task of image classification.

The idea of ViT is the following. An image $\boldsymbol{I} \in \mathbb{R}^{H \times W \times C}$ is reshaped into a sequence of non-overlapping flattened 2D patches $\boldsymbol{x}_i \in \mathbb{R}^{N \times (P^2 \cdot C)}$, the length of the flattened patches is computed according to the desired patch size P and the number of image channels C. Altogether, we have N vectors as an input to the network, each containing information about a small image region. A positional embedding is added to individual image patches to maintain partial information about the location of individual patches within the image.

Similarly as in BERT, an extra learnable embedding, the *class token*, is prepended and is reserved to accumulate the information about the input as a whole. The class token plays a crucial role in classification. Due to its non-specificity to any of the patches, after the inference in the transformer encoder, an MLP head is attached to it to produce the classification. Another key difference from the transformer proposed for NLP is that, in this case, we do not need a decoder and only use the encoder

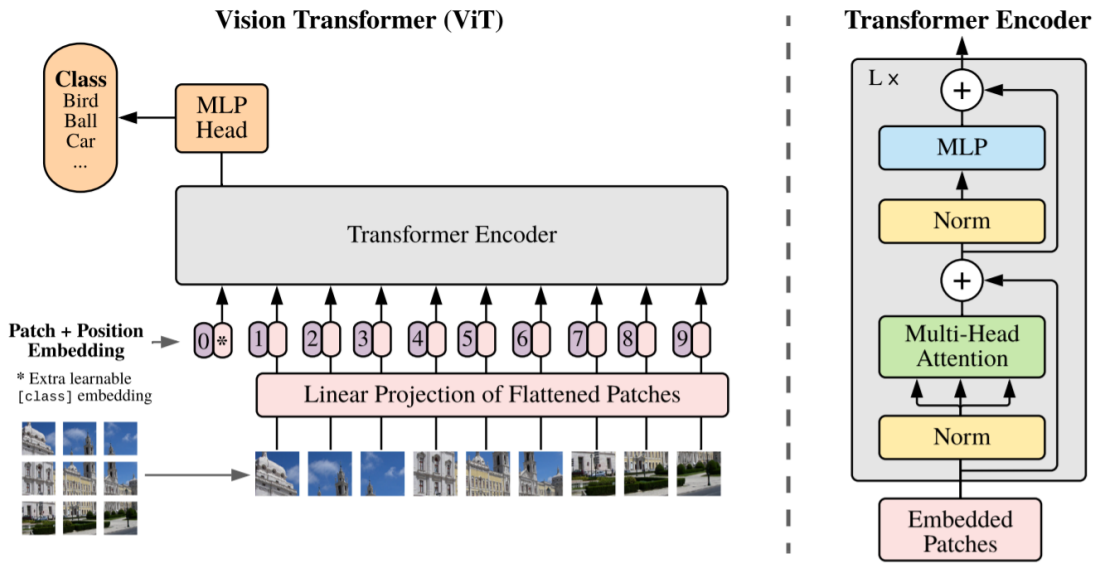


Figure 4.3: Vision transformer architecture (Dosovitskiy et al., 2021).

part. A graphical illustration of the ViT and the exact encoder architecture is shown in Fig. 4.3.

Typically, the ViT is first pre-trained on a large dataset and fine-tuned on a smaller one, much like BERT. The authors provided an extensive study of the proposed architecture and showed that this technique matches, or even outperforms the state-of-the-art image classification networks, with less computational requirements and almost without using inductive biases, as it is in the traditional convolutional networks.

The authors also examined a hybrid architecture in which parts of a pre-processed image served as the input to the ViT. In this version, the input to ViT originates from the feature maps of a convolutional network. This approach yields comparable results to the pure ViT.

Since the first vision transformer was designed, numerous attempts have been made to create similar models with different properties. Among the most successful ViT variants are *Data-efficient image Transformers (DeiT)* by Touvron et al. (2021), which involve distillation through attention to reduce the large data requirements, *Transformer iN Transformer (TNT)*, where the authors further subdivide the image patches to capture the relationships within larger image patches (e.g., 16×16) (Han et al., 2021), and *Swin Transformer* proposed by Liu et al. (2021b) that produces hierarchical feature representations. These, and many more ViT variants, have achieved tremendous success in many computer vision tasks, and the number of benchmark datasets where they achieve state-of-the-art performance is growing (Liu et al., 2021a).

Explainability of Vision Transformers

An advantage of ViTs over CNNs is their innate ability to visualize the important parts of the image, leveraging the attention weights for individual layers and heads (see Fig. 4.4). To this end, the self-attention computation defined in Eq. 4.13 is exploited. There are still many options for attention visualization. Each patch interacts with every other, which produces $n + 1$ attention maps for every head (n is the number of patches and +1 for the class token). In practice, it is often useful to pick a specific map (for example, the one contributing to the class token) and merge the overall influence for every attention head. While this innate attention computation has some flaws, it allows for an accurate examination of the model, without relying on other explanation techniques, such as Grad-CAM for CNNs.

Unluckily, the attention scores in ViTs are more distributed towards the end, and do not always produce reliable attention maps (Kashefi et al., 2023). Therefore, several upgrades regarding the explanation methods have been proposed, that are tailored directly to ViT and its variants. Among them are *attention rollout* and *attention flow*, both proposed by Abnar and Zuidema (2020) or *Grad-SAM*, a gradient-based method to identify the parts of the input image that influence the decision-making (Barkan et al., 2021).

Figure 4.4: Visualization of the attention in ViT model for several input images (Dosovitskiy et al., 2021).

Robustness of Vision Transformers

Although ViTs represent a relatively recent architecture, their robustness against adversarial attacks has already been studied in several works. Among the first were Shao et al. (2022), who concluded that ViTs exhibit greater robustness compared to the more traditional CNNs when evaluated using white-box and transfer attacks.

On the other hand, a similar study of robustness presented a contrasting perspective (Mahmood et al., 2021). The authors found that ViTs, when properly attacked, do not possess higher robustness than CNNs. The empirical evaluation also revealed that AEs generated for CNNs do not effectively transfer to ViTs and vice versa. Leveraging this insight, a black-box defense strategy using ensemble models was proposed. It was able to gain robustness and simultaneously cause only a marginal decrease in clean accuracy.

Bai et al. (2021) supported the claim that ViTs are not superior in robustness by their comprehensive robustness analysis. They criticized previous comparisons for being unfair due to differences in training mechanisms, model sizes, and other factors, highlighting the need for more standardized evaluation criteria.

To summarize the recent advances, after careful analysis, the robustness of ViTs does not seem to be inherently increased. There are still many challenges in evaluating the robustness fairly, mainly when comparing them to other model families. Nevertheless, given their unique approach to data processing and the limited amount of work in this area, many aspects of ViTs are yet to be discovered.

Chapter 5

Manifold proximity analysis

This chapter is dedicated to our analysis of adversarial examples and their behavior in trained deep neural network architectures. Our aim is not only to assess the misclassification rate of the AEs but also to gain deeper insight into the behavior of AEs inside the neural networks. Apart from generating and analyzing six distinct sets of misleading input images, we propose means of visualizing the proximity of their activations to class-specific manifolds. This, along with projections into a low-dimensional space, helps us better understand the behavior of AEs and their distinction from standard, in-distribution, samples.

The content of this chapter is the prime focus of our paper "Examining the Proximity of Adversarial Examples to Class Manifolds" published at the 29th International Conference on Artificial Neural Networks (Pócoš et al., 2022)[1]. It is also available as an extended abstract in Bečková et al. (2022).

5.1 Experiment design

5.1.1 Models and datasets

For our analysis, we use two image benchmark datasets, MNIST (LeCun and Cortes, 2010) and CIFAR-10 (Krizhevsky, 2012). The MNIST consists of 28×28 pixel grayscale images of hand-written digits, whereas the CIFAR-10 contains 32×32 pixel RGB images, depicting ten classes, each of which is a vehicle or an animal. The complexity of CIFAR-10 is generally higher than that of MNIST.

We employ two strategies for training neural network classifiers for the MNIST dataset: a fully connected (FC) network and a convolutional network. The former

[1]The first and the second author contributed equally to the paper. Both built and maintained the code base for the experiment, including training the networks, generating AEs, and much more. Moreover, each suggested, implemented, and tested one of the methods of computing the proximity to class-specific manifolds.

contains two hidden layers, each having 128 units, whereas the latter uses two convolutional layers, each having 16 filters. We mostly use *Rectified Linear Unit (ReLU)* as an activation function in both cases.

For the classification of CIFAR-10, we use a deep convolutional network comprising three VGG-type blocks, after which an additional fully connected layer of 256 neurons is applied. To ensure proper generalization, we also use dropout and batch normalization.

In this analysis, we deliberately do not work with more advanced models and training regimes. One reason is that having more sophisticated models can unpredictably influence their inner behavior. Furthermore, given the relatively small complexity of the tasks, we reach accuracy that meets our requirements, i.e., 98.0% using the fully connected network, 98.9% for the convolutional network, both trained on MNIST using SGD, and 87.2% for the network trained on CIFAR-10 using Adam optimizer.

5.1.2 Generating corrupted data

As described in Chapter 2, there is no universal approach when defending against all types of AEs. Researchers often work with a relatively narrow subset of all possible AEs, which can result in misleading results. To alleviate similar issues, we opt for as diverse sets of AEs as possible while keeping the evaluation complexity tractable.

One of the critical distinctions of AEs can be the L_p norm, which limits the perturbation magnitude ϵ. We, therefore, employ four types of adversarial attacks, each computing the AEs such that a specific L_p norm of the perturbation is limited. To complete our analysis with out-of-distribution data (that are not AEs), we also generate false-positive images, also called the *rubbish class (RC) examples* — seemingly random noise patterns, forcing the network to be confident about its output category (Goodfellow et al., 2015).

We generate the AEs with the following constraints:

- **L_∞ constraint:** For limiting the maximal change per pixel, we use the PGD attack, introduced by Madry et al. (2018). After setting a fixed perturbation magnitude ϵ, PGD finds a fooling image with perturbation no greater than ϵ. However, in practice, the generated AEs usually lie at the very edge of the ϵ-ball. Thus, to obtain a greater diversity of AEs while limiting the L_∞ norm, we repeat the process of AE generation for $\epsilon \in \{0.01, 0.02, ..., 0.15\}$ for MNIST and $\epsilon \in \{0.01, 0.02, ..., 0.05\}$ for CIFAR-10 images.

- **L_2 constraint:** To produce AEs while limiting L_2 perturbation, we employ the CW attack (Carlini and Wagner, 2017). The aim of the CW attack is, in general, to craft AEs with as small L_2 perturbation as possible, even at the cost of having potentially borderline prediction confidence scores.

- **L_1 constraint:** A subtle yet non-trivial change in perturbation design is to use L_1 norm instead of L_2. Chen et al. (2017) argued that L_1 norm accounts for the total variation of perturbation, thus an attack limiting the L_1 norm can generate AEs with unique properties. For generating these AEs, we use the elastic-net attack, often seen as a generalization of the CW attack.

- **L_0 constraint:** Since L_0 norm restricts the number of changed pixels, AEs constrained by this norm contain only a few, usually significantly perturbed pixels, leaving the rest of the image intact. A common way to generate these examples is to use one-pixel attack (Su et al., 2019). However, as we showed in Bečková et al. (2020), when having access to the gradients, we can leverage them to find the most important pixels to change. To generate AEs, we first compute the gradient w.r.t. the input, and we pick the pixel having the highest absolute gradient. We follow by using grid search to find the best adversarial perturbation for the given pixel. If misclassification occurs, the algorithm is terminated. Otherwise, the computation is repeated until an AE is found or we exceed the maximal number of iterations. Our upper bound for the number of perturbed pixels is 50, corresponding to maximal overall image changes of 6.4% and 4.9% on MNIST and CIFAR-10, respectively.

To supplement our AEs, we also include rubbish class examples. We produce them in two ways, differing only in the initial pixel distribution. In the first method, we randomly generate pixel values from uniform distribution $[0, 255]$, whereas, in the second method, we generate them to match the distribution of pixels in the same position of the datasets. To make the network classify them with high confidence, we iterate the input using PGD attack. The resulting groups are denoted as RC_{rnd} and $\text{RC}_{distrib}$, respectively.

For generating the AEs, we strictly use untargeted attack, as it provides an easier optimization problem[2]. However, when generating the rubbish class examples, we are forced to use the targeted attack while changing the target class uniformly to every viable option. We opt for this simple change because, in the case of untargeted attacks applied to noisy patterns, we end up with a highly uneven distribution of the resulting classes. In the case of MNIST, the majority of crafted inputs converged to class 8, whereas most of the optimized noise patterns crafted using the network trained on CIFAR-10 are classified as frogs. On the other hand, the targeted attack resulted in a roughly even distribution of predicted classes.

Altogether, we generated ≈12,000 corrupted input (the union of AEs and the RC examples) per each of the three networks. A sample of generated data for convolutional networks on MNIST and CIFAR-10 is depicted in Fig. 5.1.

[2]We use Adversarial Robustness Toolbox (Nicolae et al., 2018) for all attacks except L_0.

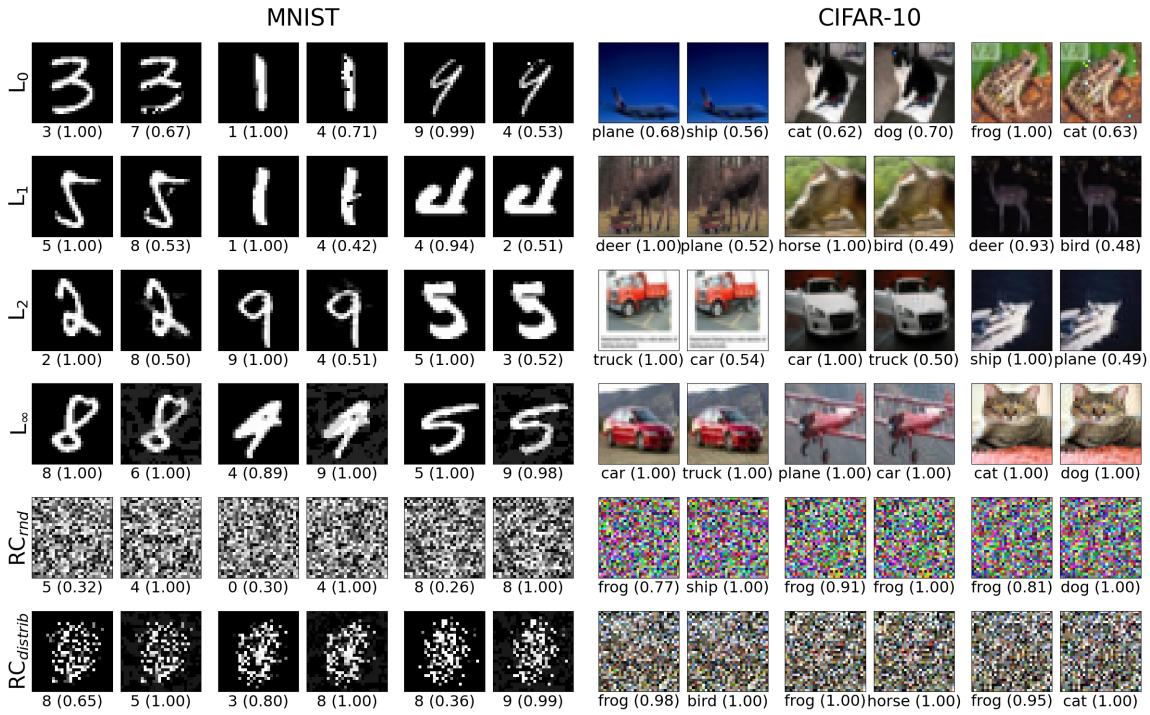


Figure 5.1: The first four rows contain tuples of adversarial examples and their corresponding originals, and the last two rows contain a pair of random noise patterns and the corresponding rubbish class examples. The network confidence and the output class are displayed below the individual images.

5.2 Proximity analysis

Considering the AEs are near the original examples, we presume their latent activations in the networks might be close, especially in the early stages of the network's forward computation. As they are eventually misclassified, we expect the AEs to diverge from the correct class manifolds in the activation space. Therefore, in this section, we search for possible methods of taking a closer look at the behavior of the AEs in trained DNNs.

5.2.1 Distance to classes

To find the region where an AE leans towards the incorrect class during the network's forward pass, we first test a direct technique — computing the distances to the latent activations of test-set images. To be more precise, we create triplets $(\boldsymbol{O}_i, \boldsymbol{A}_i, \boldsymbol{R}_i)$, where $\boldsymbol{O}_i$ denotes the i-th original image, $\boldsymbol{A}_i$ the i-th adversarial image, and $\boldsymbol{R}_i$ stands for the noisy version of the corresponding original image, where the noise magnitude is kept the same as for $\boldsymbol{A}_i$, but is random Gaussian. Having these triplets, we compute the mean of the differences in activations of these three data categories:

1. Images with the correct label (the same as the target label for $\boldsymbol{O}_i$),

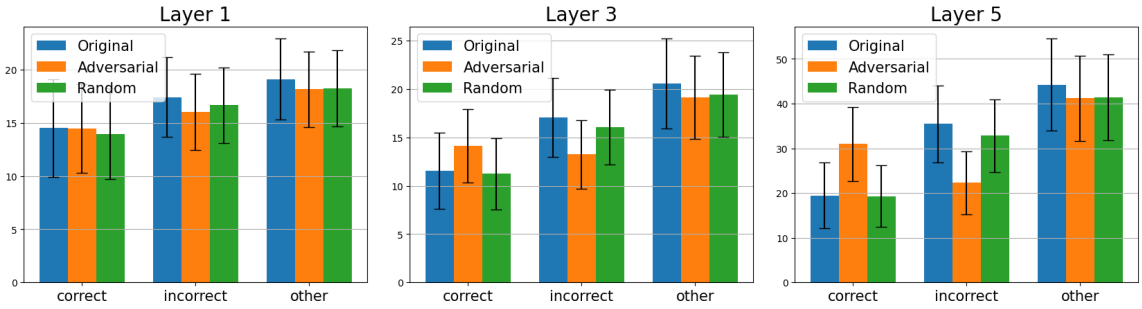


Figure 5.2: Mean distances of original vs. adversarial image to three different groups of data. On the vertical axis, the mean distance of a certain input activation group is depicted. There are three data categories: original, adversarial, and random modification of the original. Their distances are computed towards the correct class, incorrect class and the rest of the classes averaged.

2. images with the incorrect label (the same as predicted for $\boldsymbol{A}_i$),

3. the rest of the test-set images.

A sample showcasing the development of these distances in a fully connected network is shown in Fig. 5.2.

We found that during the early stages of the classification, the distances of AEs have very similar profiles as the distances of original examples. But as we progress through the network further, the AEs are slowly pulled towards the incorrect class. Interestingly, when considering the random inputs, they behave almost identically to the original examples, meaning that random perturbations indeed do not cause out-of-distribution activations.

A downside of this method is that due to the constantly varying number of neurons (i.e., dimensionality), we cannot consistently compare the change of proximity of these three data groups to manifolds throughout the network. Due to this, the scale of the calculated distances is not the same for all the layers. Yet another reason why this method is not the most suitable, is that the distances between high-dimensional data often tend to be similar, thus it does not faithfully capture the underlying relationships. We can see an example of this in Fig. 5.2, where the distances of original examples to the correct class vs. the incorrect class differ only marginally.

5.2.2 Proximity to class-specific manifolds

To overcome the issues with differing numbers of hidden neurons, inspired by Papernot and McDaniel (2018), we use the idea of counting the nearest neighbors in the space of hidden layer representations. This presents a transition in our ideology where we leave out reporting the distances, and rather work with the elements having a certain

activation in the latent space.

Let us fix a network and an attack that we used to generate the AEs. For the sake of nomenclature, we denote the transformations used to compute the output as a series of functions f_i, where $i = 0, ..., l-1$ and l is the number of all transformations (including the linear layers, convolutions, activation function, etc.). For example, in the case of a single-layer perceptron with one activation function, the output vector $\boldsymbol{y}$ corresponding to $\boldsymbol{x}$ is computed as:

$$\boldsymbol{y} = f_1(f_0(\boldsymbol{x})). \tag{5.1}$$

With this in mind, we can represent the i-th hidden activation of any input $\boldsymbol{x}$ as the functional value $F_i(\boldsymbol{x})$, where $F_0(\boldsymbol{x})$ is the input. The following holds:

$$\begin{aligned} &F_0(\boldsymbol{x}) = \boldsymbol{x}, \\ &F_1(\boldsymbol{x}) = f_0(\boldsymbol{x}), \\ &F_2(\boldsymbol{x}) = f_1(f_0(\boldsymbol{x})) = f_1(F_1(\boldsymbol{x})), \\ &\dots \\ &F_l(\boldsymbol{x}) = f_{l-1}(f_{l-2}(F_{l-2}(\boldsymbol{x}))) = f_{l-1}(F_{l-1}(\boldsymbol{x})). \end{aligned} \tag{5.2}$$

Our next step is to split the AEs into $m \times m = m^2$ groups, where m is the number of output classes. Since both datasets we use for testing contain 10 classes, 100 groups per network are created. We denote the subset of all AEs originally belonging to the class o, ending up classified as the class p, where $p \neq o$, as $Adv_{o \to p}$. We also denote the set of input images belonging to the original class as $\boldsymbol{X}^o$, and $\boldsymbol{X}^p$ for the predicted class, given $Adv_{o \to p}$.

Now for a chosen adversarial example $\boldsymbol{x}_{adv} \in Adv_{o \to p}$ and a fixed number k, we compute the class–alignment score on the i-th layer from the nearest neighbors of $\boldsymbol{x}_{adv}$ as

$$\boldsymbol{Z}^i_{o \to p} = \mathrm{KNN}_k(F_i(\boldsymbol{X}^o \cup \boldsymbol{X}^p), F_i(\boldsymbol{x}_{adv})), \tag{5.3}$$

$$\boldsymbol{Z}^i_o = \{\boldsymbol{z} | \boldsymbol{z}_{tgt} = o; \boldsymbol{z} \in \boldsymbol{Z}^i_{o \to p}\}, \tag{5.4}$$

$$\mathrm{score}^{\boldsymbol{x}_{adv}}_i = \frac{|\boldsymbol{Z}^i_o|}{k}. \tag{5.5}$$

$\mathrm{KNN}_k(\boldsymbol{A}, \boldsymbol{x})$ represents the top k nearest neighbors of the point $\boldsymbol{x}$ among all the points from the set $\boldsymbol{A}$. $\boldsymbol{Z}^i_o$ is a subset of $\boldsymbol{Z}^i_{o \to p}$ containing only those points, which belong to the class o, and $|\boldsymbol{Z}^i_o|$ is the magnitude of the particular set.

The results for several AE groups are shown in Fig. 5.3. We observe that even for a fixed network and method of generating the AEs, we get relatively different behaviors. But to properly compare multiple attack methods, we fix an (o, p) pair for all the attacks and plot the scores. Since the AE sets do not have uniform magnitude, we carefully pick those, which have at least 100 elements for every attack method.

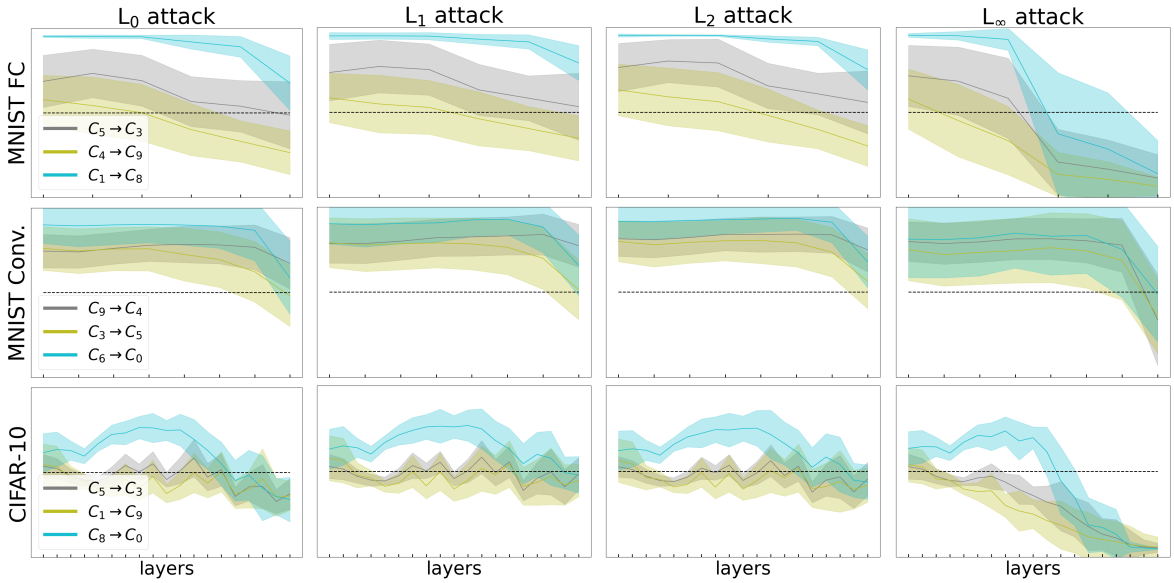


Figure 5.3: Development of the class–alignment scores (y-axis) throughout the network layers (x-axis) for three chosen subsets of AEs, using different attacks. For brevity, we denote the classes of CIFAR-10 with their index numbers. The horizontal line in the middle serves as an indicator of the "breaking point" — a region in which the number of nearest neighbors of the correct class equals the number of the nearest neighbors of the incorrect class.

Looking at the resulting graphs, we can see that the AEs often have common neighbors from the correct class, mainly in the first layers. In the case of L_0, L_1, and L_2 attacks, it usually holds for the successive layers as well. On the other hand, the L_∞ attack reveals the highest tendency to approach the incorrect class during the classification.

The differences are noticeable across the network types as well. When training on MNIST, the CNN prefers to keep the activations of AEs considerably close to the correct class, whereas, in the fully connected network, the AEs slowly shift towards the incorrect class (except for the L_∞, where this happens suddenly about halfway through the network). The network trained on CIFAR-10 demonstrates the highest level of uncertainty when considering the class alignment. In most cases, the alignment fluctuates around 0.5 (the half), however, some particular AE groups evoke closer proximity to the correct class. Additionally, the most drastic shift of the class alignment scores is triggered for L_∞ AEs, which end up surrounded by incorrect class activations.

In Pócoš et al. (2022), we also proposed an alternative method to calculate the proximity scores to class manifolds, enabling us to compare the scores throughout the network. Using the second method, we were able to confirm our findings reported here, as they yield consistent results. For details about the second method, we refer the reader to the full publication.

5.3 Entanglement

Our next step in exploring the neural networks is to look at the entanglement of the AEs with the original examples. It was shown that the manifolds of individual classes tend to separate (disentangle) during the forward pass in the networks, even though they are entangled in the earlier layers (Brahma et al., 2016). Now we are mainly interested in what happens with the activations of AEs and whether they entangle with the test-set data during the classification. We can empirically verify this theory by depicting the manifolds in 2D using UMAP (McInnes et al., 2018).

Once we compute the activations for the input points (adversarial, rubbish class, and clean examples), we use UMAP to nonlinearly project the high-dimensional latent vectors into two dimensions. To ensure the most accurate representation, we experimentally adjust the internal parameters of UMAP. To highlight the corrupted data in the two-dimensional depiction, we colorize them while leaving the test-set examples in grey. Fig. 5.4 provides a demonstration of the MNIST network activations of the clean vs. the corrupted data.

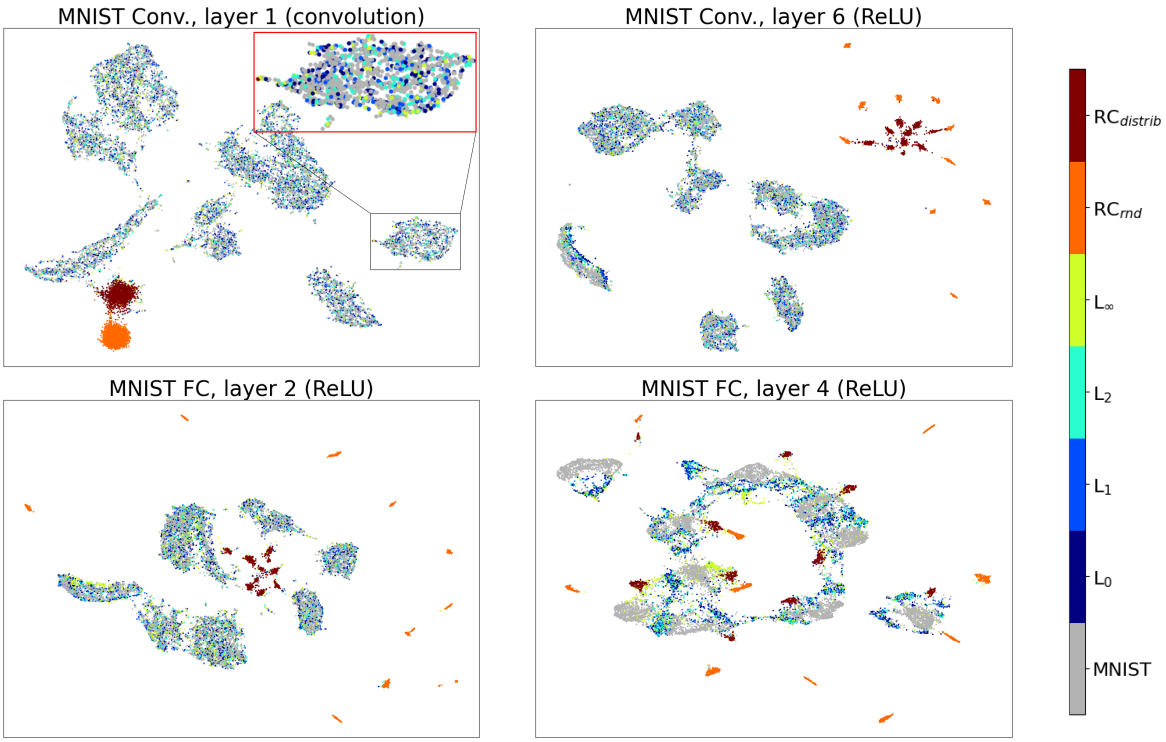


Figure 5.4: UMAP projection of activations of adversarial, rubbish class examples and the testing data. High level of entanglement of AEs with the test set can be observed in the top–left corner, whereas lower entanglement is visible in the bottom–right image.

By analyzing the projections we arrive at the following observations:

1. On the majority of the layers, the AEs are near the test-set activations, as an evidence of entanglement. This holds for all trained networks.

2. The main visual differences between the AEs and the test set are noticeable when approaching the last layers. The PGD attack produces AEs with the highest deviation from the rest of the distribution. This corresponds to the largest shift of class attributes, found in the previous section.

3. Rubbish class activations are well separated from everything else, but on the last layers the group RC_{rnd} is farther from the rest of the data than the group $\mathrm{RC}_{distrib}$. Since we generated the rubbish class examples using the targeted attack where the classes were represented equally, we see the formation of 10 clusters in each rubbish class set, representing the individual classes. Thus, a specificity in the seemingly meaningless noise patterns is observed on the level of internal representations as well.

Although UMAP is stochastic in nature, similarities can be observed with the results obtained via computing the class-alignments. The key observation here is that once more, the AEs prove challenging to distinguish in the latent space, as they frequently evoke activations resembling those from the distribution of clean data. Numerical verification of the entanglement is described in the paper (Pócoš et al., 2022), in which we used soft nearest neighbor loss (Frosst et al., 2019) to quantify disentanglement.

5.4 Summary

The study of AEs and their behavior in deep network architectures exposed various causes of misclassification. We showed that their rate of convergence to the incorrect class manifold can vary a lot, and it does not depend only on the attack method we use but often even on the specific classes we compare. In many cases, an AE does not leave the vicinity of the correct class manifold, yet is misclassified due to an incorrect decision boundary at the very end of the network. This strange behavior can be an effect of complex structures formed by high-dimensional data manifolds.

By using UMAP, we found cases in which the AEs entangle with activations of the test data, rendering the detection methods based on seeking out-of-distribution activations unreliable. The opposite holds for the rubbish class examples, as they exhibit expected behavior. They start far from the input images and tend to get closer to the activations of the clean examples.

Further research in this line of work could be focused on the investigation of adversarially trained networks, as they can already possess a certain level of robustness. Their inner dynamics might help us to take a better approach when detecting out-of-distribution samples. To conclude this topic, we hope this work can be used for the evaluation of novel attacks, or it can provide a way to see how the manifolds of different input types are intertwined.

Chapter 6

Recurrent Vision Transformer

In this chapter, we introduce *Recurrent Vision Transformer (RecViT)*, a novel deep learning architecture designed to address the challenge of adversarial robustness. Through extensive analysis, we demonstrate how RecViT offers valuable insights into this problem. Our findings also highlight the benefits of incorporating top-down information flow into the classification process.

The majority of the analysis described in this chapter constitutes the fundamental focus of our paper, published at the 19th International Conference on Computer Vision Theory and Applications — VISAPP 2024[1] (Pócoš et al., 2024b). The follow-up experiments, designed to find the differences in AE activations when processed by RecViT, can be found as an extended abstract in Pócoš et al. (2024a).

6.1 Related work

Vision transformers are integral concepts of advanced machine learning algorithms. As described in Chapter 4, various versions of ViTs have been proposed, each with unique advantages. Here, we build upon the idea of extending the vision transformer using a biologically motivated top-down connection. The top-down connection, in addition to the predominant bottom-up approach, allows the network to incorporate a different point of view into the classification process.

The greatest source of inspiration for our architectural design is provided by Stollenga et al. (2014), where the authors created *Deep Attention Selective Network (DASNet)*. The core idea of DASNet is to combine the power of CNNs with iterative processing. During the forward pass, a vector comprising hidden activations is constructed, which is fed to a reinforcement-learning-based policy. The output is a weight vector,

[1]The authors' contributions are complementary. The first author (65%) originated with the idea of incorporating a top-down connection into the vision transformer and implemented most of the experiments. The second author (30%) provided essential remarks about the validity of the experiments, facilitated the design of a proper training regime, and generated AEs for the CIFAR-10 dataset.

determining the strength of the convolutional kernels used in the CNN in the subsequent iteration. This results in an iterative classification process, and in each iteration, the network narrows down its focus, being able to make a finer distinction between certain details.

This is, however, not the only similar architecture. There is a number of publications, which in certain ways resemble our work, yet there are fundamental differences between them. The Perceiver (Jaegle et al., 2021) is a transformer-based architecture created to be able to scale to a very large input by progressively transforming the representation by a bottleneck, with potentially shared weights. Gehrig and Scaramuzza (2023) applied recurrence for object detection, but their architecture is composed of multiple parts, such as convolutions, attention modules, LSTM, and more. Presumably, the closest to our work is the modified vision transformer published by Messina et al. (2022). They augment vision transformers with recurrence; however, in their case, the recurrence is also present in the patch tokens. Also, they do not explore the robustness of the proposed model, nor extend the inference with data augmentation.

To the best of our knowledge, RecViT is the first architecture to repeatedly feed the class token into the next iteration phase, omitting the creation of additional weights and keeping the memory complexity stable. We are also unaware of another similar work on a ViT version where the input modification during the inference resembles ours.

6.2 Model design

To use a vision transformer for image classification, we first need to split the image into equally sized patches, which are later embedded to form vector representations. In addition to patch embeddings, to collect the cumulative information about the entire image during the classification, class token can be employed (and we pick this option in our design). It is technically a vector of the same length as the image patch embedding. During the learning phase, its initial values are optimized so that it contains generally neutral information about the given dataset.

After the forward pass, the activation of the class token is inserted into an MLP, producing the network's final decision. In RecViT, however, this is the step where we diverge from the standard output computation. Instead of computing the output class (which, in our case, is optional), we proceed to the second iteration, in which the whole classification process is repeated, but the class token is initialized with its activation from the previous computation. This step results in an initialization of the patch token biased towards the most likely output classes. During the subsequent inference, the self-attention computation can potentially focus on more relevant features, emphasizing

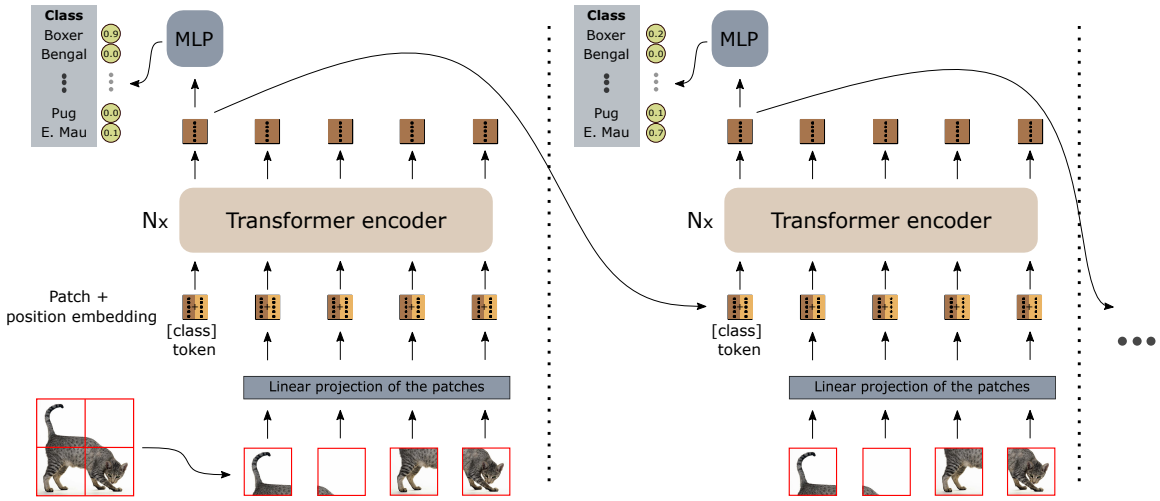


Figure 6.1: Illustration of the first two iterations of the RecViT architecture using an input image from the Oxford-IIIT PET dataset (Parkhi et al., 2012). The large patch size serves solely for visualization purposes, and in practice, it is much smaller.

a subset of particularly relevant classes for the current input.

Overall, we have a fixed number of iterations, using the last one to compute the final output. During this process, all of the used weights are shared, so we do not need to use any additional parameters. Fig. 6.1 shows a detailed scheme of the first two iterations of the architecture.

6.3 Input

6.3.1 Benchmark datasets

To evaluate RecViT's performance, we opt for two benchmark image classification datasets, CIFAR-10 (Krizhevsky, 2012) and Oxford-IIIT Pet (Parkhi et al., 2012), further referred to as the PET dataset. While the former provides a reliable comparison with the state-of-the-art models, by using the latter, we gain insight into the model's operation for more complex and realistic data. The PET dataset comprises 37 classes of dog and cat breeds, where the images have diverse but much higher resolution than CIFAR-10 images. To achieve consistency during training and evaluations, we pad the PET images to a square shape, then we resize them to have the shape of 224×224 pixels.

The major advantage of the PET dataset (to be demonstrated in section 6.6) is the presence of the segmentation masks, i.e., annotations for each input pixel, determining whether it is a part of an object of interest, its boundary, or the background.

6.3.2 Adversarial examples

Since our primary goal is to verify whether the RecViT model possesses more robust behavior against out-of-distribution samples, we need some data to experiment on. For this, we use adversarial examples, as they are often viewed as an approximation of the worst-case noise (Madry et al., 2018). Generating AEs is a trivial task nowadays, yet evaluating the robustness of various networks in an unbiased manner is still a challenge. Therefore, we devise a way of crafting AEs while considering the many logical fallacies researchers are often unable to avoid.

Since the computational graph of RecViT is, in general, deeper than that of a vanilla ViT, we cannot rely on pure white-box attacks, as they would be influenced by the quality and the magnitude of the computed gradients. Therefore, in this case, we opt to build a substitute model on which we will generate the AEs and then transfer them to ViT and RecViT. The choice of the substitute model is, again, non-trivial and should not be arbitrary. It has been shown that networks tend to transfer their AEs to similar architectures with higher success rates (Mahmood et al., 2021). Due to this, the substitute model should not resemble any of the two networks. Hence, a convolutional network could be as impartial to both networks (ViT and RecViT). CNNs, even though they do not transfer to transformer-like architectures so well, meet our criteria.

Training one substitute model and crafting high-confidence AEs is a valid method, but to obtain even stronger and more transferable AEs, we can train multiple dissimilar models (Liu et al., 2017). We thus fine-tune three CNNs pre-trained on ImageNet (Deng et al., 2009), namely AlexNet (Krizhevsky et al., 2012), ResNet (He et al., 2016) and VGG (Simonyan and Zisserman, 2015) for each of the two datasets. After achieving a reasonable classification accuracy (85.03%, 91.52% and 90.75% on CIFAR-10, and 74.27%, 87.22% and 89.15% on PET dataset), we generate adversarial examples using the PGD attack (Madry et al., 2018). To produce AEs with differing perturbation magnitude, we limit the ϵ-ball for these values: $\epsilon \in \{0.01, 0.02, ..., 0.2\}$ for CIFAR-10, and $\epsilon \in \{0.007, 0.022, ..., 0.202\}$ for PET. After this step is completed, we create a set of particularly strong AEs (in theory), which not only fool the network on which they were generated but also the other two trained networks on the same dataset. We denote this category as *Cross-Validated (C-V)* AEs. A small sample of generated AEs with their corresponding original images is shown in Fig. 6.2. We can notice that the AEs are often misclassified with 100% confidence.

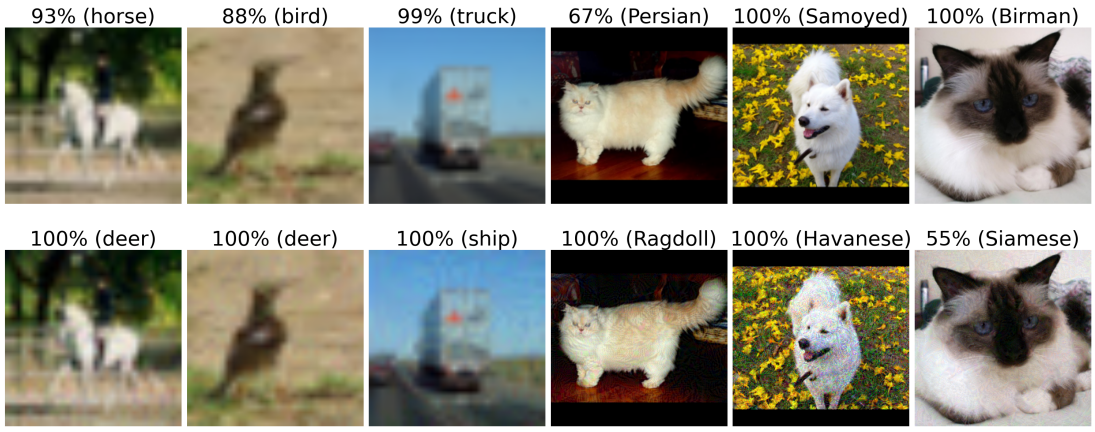


Figure 6.2: Sample of adversarial examples with high potential to transfer to an unknown network trained on CIFAR-10 (left triples) or the PET (right triples) datasets, evaluated on AlexNet.

6.4 Training

To classify the images with great accuracy, we fine-tune a fairly small vision transformer ViT-tiny (Steiner et al., 2022), which was earlier pre-trained on the ImageNet dataset. The training is carried out using standard backpropagation through time, but we distinguish two variants, depending on the time steps used for error minimization.

6.4.1 Error propagation

During the training, after each input, we can get multiple outputs (one per iteration). Therefore, we can distinguish how exactly the error propagates.

In the *method 1 (M1)*, we only optimize the last prediction. This emphasizes the fact that the last prediction is the most important, while the rest can serve for a more general computation of the class token. The loss definition is the following:

$$loss_{M1} = L_{CE}(f_{\boldsymbol{\theta}_2}(\boldsymbol{c}_{k-1}), l; \boldsymbol{\theta}_1), \tag{6.1}$$

where L_{CE} is the cross-entropy loss, $\boldsymbol{\theta}_1$ denotes the model parameters, l is the target class label and $f_{\boldsymbol{\theta}_2}$ is the multi-layer perceptron used to classify the class token $\boldsymbol{c}_{k-1}$ in the final iteration.

In contrast with the previous loss computation, in the *method 2 (M2)*, we optimize the predictions in every time step. This promotes the idea that all of the time steps are equally important. Practically, during the BPTT, we optimize two parallel aspects: the prediction's correctness and the class token usefulness for the following iteration. The optimization procedure is thus defined as follows (using the same nomenclature as in the method 1):

$$loss_{M2} = \sum_{t=0}^{k-1} L_{CE}(f_{\boldsymbol{\theta}_2}(\boldsymbol{c}_t), l; \boldsymbol{\theta}_1). \tag{6.2}$$

6.4.2 Input augmentation

To fully facilitate the power of iterative computation, we do not necessarily need to keep the image unchanged. Instead, we suggest three alternatives for input modification during the inference. An illustration of the non-trivial data augmentation is shown in Fig. 6.3.

- **Vanilla**: No data augmentation. In each iteration, the same image is presented to the network.
- **Random Transform (RandT)**: At the start of each forward pass, random values of translation, rotation, and scale are generated. These values are used to progressively modify the input image in each iteration. Thus, the network has more information and can hopefully adapt to a slightly different view.
- **Blur**: To extract different levels of details during the inference, we progressively blur the image. We start out with a clear image, and after each iteration, we apply Gaussian blurring to the previous input.
- **Invblur**: Here, we use identical inputs as in the "Blur" method, but in reversed order. When using recurrence for data processing, the inputs from different time steps are weighted unevenly. Therefore, this variant should prevent a clean accuracy drop.

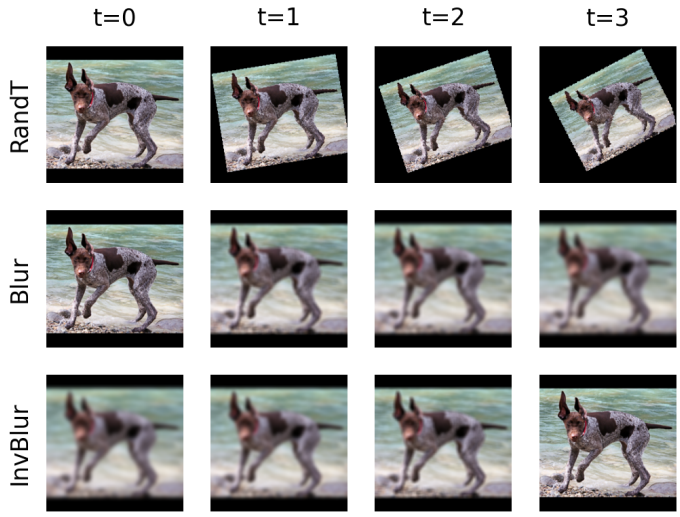


Figure 6.3: Illustration of the three input modification strategies (rows) when feeding them into RecViT consisting of four iterations.

Table 6.1: Accuracy and standard deviation (in %) of RecViT networks using various input augmentation strategies on PET dataset.

	Test set	AlexNet AEs	ResNet AEs	VGG AEs	C-V AEs
RecViT variant	Mean±Std	Mean±Std	Mean±Std	Mean±Std	Mean±Std
Baseline ViT	88.80±0.57	9.26±1.37	14.67±1.90	8.96±1.87	10.66±1.42
Vanilla M1, k=3	89.60±0.54	9.52±1.14	17.19±1.66	10.10±0.97	11.83±1.02
Vanilla M2, k=2	88.67±0.31	9.68±0.90	19.62±1.21	11.49±0.97	13.06±0.81
RandT M1, k=2	89.02±0.24	11.24±0.89	17.58±2.35	12.65±1.56	13.58±1.22
RandT M2, k=2	88.05±0.99	11.17±0.98	18.18±3.94	12.22±0.57	13.28±1.49
Blur M1, k=3	85.23±5.66	12.75±3.34	16.89±3.79	11.36±3.75	13.43±3.56
Blur M2, k=4	87.42±1.56	13.43±2.81	18.83±3.58	12.54±3.14	14.62±3.09
InvBlur M1, k=2	**86.99±0.98**	**15.57±3.60**	**24.91±10.70**	**17.83±6.95**	**18.93±6.63**
InvBlur M2, k=2	85.27±1.28	17.04±1.21	28.65±3.79	17.87±2.03	20.54±2.15

6.5 Results

We train all the possible combinations of loss computation and input modification strategies, with varying numbers of iterations $k \in \{2, 3, 4\}$, resulting in 24 models. To also report their deviations, each configuration is trained five times.

6.5.1 Robustness

The training of our models is followed by their accuracy and robustness evaluation, found in Table 6.1. The table contains the best picks among all the possible values of k. The results show that RecViT's Vanilla variant yields a slight increase in robustness, mainly for ResNet and VGG AEs. After training using M1, there is even an increase in accuracy on clean data. The accuracy and robustness profile of RecViT after feeding with randomly transformed input is comparable with the Vanilla variant. Interestingly, the clean accuracy drops when feeding the network with blurred data while the robustness increases. Nonetheless, in the best configuration (i.e., InvBlur variant using M1, k=2), a 2% drop in clean accuracy causes $\approx 8\%$ robustness increase.

The results of the networks trained on CIFAR-10 are aligned with those described for the PET dataset. More concretely, the Vanilla variant achieves matching robustness and accuracy as the baseline ViT. In this case, using blurring as augmentation, the robustness increases from $\approx 46\%$ to $\approx 68\%$ with a minor drop of $\approx 1\%$ in clean accuracy (see the upper three blocks of Table 6.2).

6.5.2 RecViT vs. ViT

The robustness increase detected after showing blurred versions of the image (along with the clear image) might be a simple effect of the image quality deterioration. When

the image is distorted, we can expect a decline in the classification accuracy, while the robustness can increase due to the partial elimination of adversarial noise. To understand whether the robustness increase is caused exclusively by the input modification or the recurrent architecture plays an important role, we aim to reconstruct the same blurring conditions in an ordinary ViT.

After some experimentation, we quickly found that using a fixed input, we only achieve a little. Even after a single blurring step (the same as in RecViT for $k = 2$), the accuracy quickly drops, rendering it unreasonable to draw any conclusions and to compare the robustness of the networks. Thus, in Alg. 1, we propose a tailored inference of ViT, further denoted as *ViT Blur*, in which the network is presented with input images of the same levels of details as for the corresponding RecViT using InvBlur inputs. For a given k, we start with feeding the clean input into the network. We continue with a progressive blurring of the image altogether $k - 1$ times, whereas after each blurring, we feed the image into the network. After inserting all k images, we use the cumulative output scores to pick the class with the highest value. The blurred inputs are also integrated into the optimization during the training phase.

Algorithm 1 Inference of ViT using variously blurred input

Require: $k \geq 1$ ▷ Number of iterations ($k = 1$ for no blurring)
Require: model, data ▷ ViT and an input image
Require: blur ▷ Blurring function
  $i \leftarrow 1$ ▷ Initializing the counter
  $logits = \vec{\mathbf{0}}$ ▷ Vector of zeros
  while $i \leq k$ **do**
    $i \leftarrow i + 1$ ▷ Increasing the counter
    $logits \leftarrow logits$ + model($data$) ▷ Accumulating the probability scores
    $data \leftarrow$blur($data$) ▷ Blurring the data
  end while
  $output \leftarrow$ argmax($logits$) ▷ Returning the result

From the results provided in Table. 6.2, we see that, in general, the distortion of the images does not increase the robustness as much as in the case of RecViT. More specifically, on CIFAR-10 we achieved the best robustness of 54.61% for ViT (setting $k = 3$), while a similarly accurate RecViT network (M2, $k = 2$) has robustness of 67.67%. On the other hand, the analysis of the PET dataset at first glance does not yield positive results. The robustness for the best RecViT model of 18.93% (M1, $k = 2$) is within the standard deviation of the robustness of a similarly performing ViT Blur model. In further analysis described in the next section, we take a closer look at RecViT's behavior on the PET dataset.

Table 6.2: Comparison of accuracy and robustness (in %) of RecViT InvBlur models with ViT models using the same input modification during training and testing.

	PET		CIFAR-10	
	Test set	C-V AEs	Test set	C-V AEs
RecViT variant	Mean±Std	Mean±Std	Mean±Std	Mean±Std
Baseline ViT	88.80±0.57	10.66±1.42	97.64±0.11	45.78±1.92
InvBlur M1, k=2	**86.99±0.98**	**18.93±6.63**	97.44±0.09	43.89±2.98
InvBlur M1, k=3	86.32±1.25	14.83±5.48	97.43±0.12	44.85±2.12
InvBlur M1, k=4	85.33±2.74	10.58±2.16	**97.44±0.04**	**45.09±2.04**
InvBlur M2, k=2	**85.27±1.28**	**20.54±2.15**	**96.46±0.14**	**67.67±1.67**
InvBlur M2, k=3	78.09±7.14	21.83±4.91	95.12±0.25	69.88±1.95
InvBlur M2, k=4	74.31±5.54	22.36±2.26	94.97±0.15	68.32±1.01
ViT Blur k=2	**86.21±0.83**	**18.20±1.25**	**97.53±0.05**	**50.96±3.13**
ViT Blur k=3	79.19±1.50	15.33±2.03	96.04±1.25	54.61±2.04
ViT Blur k=4	68.45±3.37	12.71±2.21	93.51±1.47	53.11±2.91

Table 6.3: Accuracy and robustness (in %) of the top 3 runs of RecViT models trained on PET dataset using InvBlur as augmentation.

	Test set	C-V AEs
InvBlur M1, k=2	88.13	28.44
InvBlur M1, k=3	87.68	21.11
InvBlur M1, k=4	88.15	12.47
InvBlur M2, k=2	87.14	24.02
InvBlur M2, k=3	85.59	24.28
InvBlur M2, k=4	84.01	26.53

6.5.3 Robustness–accuracy trade-off

We previously pointed out that in the case of CIFAR-10, RecViT offers significant robustness gains at the cost of a marginal drop in clean accuracy. Even though RecViT trained on the PET dataset does not exhibit the same behavior, it demonstrates some curious properties. After a more careful look at the results (see Table. 6.2), we notice an atypically high standard deviation of accuracy and robustness reported for RecViT using InvBlur, mainly when $k = 3$. In this case, the training does not properly converge. Since a broad hyperparameter search was conducted for the Vanilla RecViT, a thorough approach would be to perform a finer hyperparameter search for this particular network. Since we extensively optimized mainly the baseline, comparisons with other variants would be unfair. To be consistent, we would be forced to provide the same optimization for all the tested network versions. Due to time constraints, we omit this step, and we further explore the high variance by training 10 additional runs of the RecViT InvBlur networks, for all the other choices of k and error propagation.

Table 6.3 reports statistics for the top three performing runs of the selected models. The best network (M1, $k = 2$) achieves substantially higher robustness than the adjusted ViT, described in the previous section (28.44% vs. 18.20%), while also performing better on the clean data (88.13% vs. 86.21%).

Further investigation shows that robustness and accuracy are not contradictory goals in our case. The plot of robustness and accuracy of the model version having the biggest deviation (Fig. 6.4) indicates a positive correlation between robustness and accuracy, which is often contradictory in the literature. The calculated average correlation coefficients of 0.57 and 0.81 for RecViT InvBlur networks (trained with M1 and M2, respectively) mean that further enhancing the accuracy via more exhaustive optimization might also heighten the robustness levels of RecViT networks.

6.6 Heatmap visualization

An integral concept when using vision transformers is the inherent self-attention mechanism and the possibility of visualizing the weights for individual image patches. We can peek inside the network directly by extracting its internal components without the need to rely on visualization methods such as Grad-CAM (Selvaraju et al., 2017). This prevents any external influence on the produced attention maps invoked by the hyperparameter choices of the potential visualization method.

6.6.1 Extracting the attention

As described in Chapter 4, the innate self-attention mechanism in vision transformers can be exploited to produce maps of image regions, from which the information is

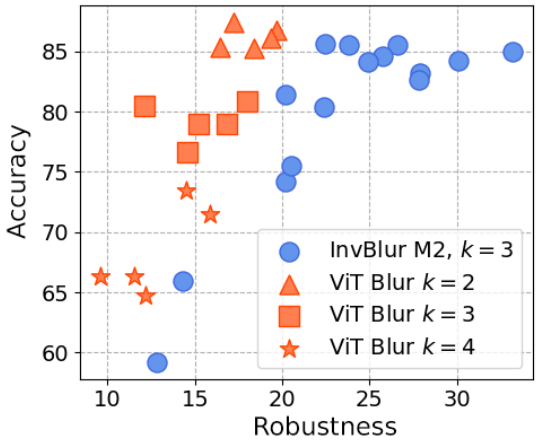


Figure 6.4: The plot of robustness vs. accuracy trade-off for individual RecViT (InvBlur) networks trained with M2 and k=3, compared to ViT Blur models trained on the same data augmentation with various k.

further propagated. Therefore, the intermediary computation of self-attention can provide us with saliency maps.

Our networks consist of 12 layers and 6 attention heads on each of them. After focusing only on the attention scores of the class token, we end up with 78 attention maps per output calculation. To make matters worse, due to RecViT's iterative nature, the total number of attention maps per input image is multiplied by the number of iterations k. Therefore, we merge the attention scores for all the used heads, resulting in $k \times 12$ attention maps in the RecViT. Their quality depends on two main factors: the patch size and the image resolution. Since in this phase, none of those can be tweaked, the network on the PET dataset should yield nicer attention maps.

Upon retrieving the importance scores, we can plot the individual patch positions in the input space with the color taken from a chosen color map. For visually more appealing heatmaps, it is sometimes beneficial to use the Gaussian blur to produce a smoother version of the map in the input space. Fig. 6.5 depicts the original and the smooth attention map for two layers of RecViT after inserting a random image from the PET dataset. We also visualize the overlap with the original image by darkening the unused image parts. According to our expectations, the attention maps for CIFAR-10 (see Fig. 6.6) have seemingly worse quality, yet in many cases, they clearly match the object's position and shape.

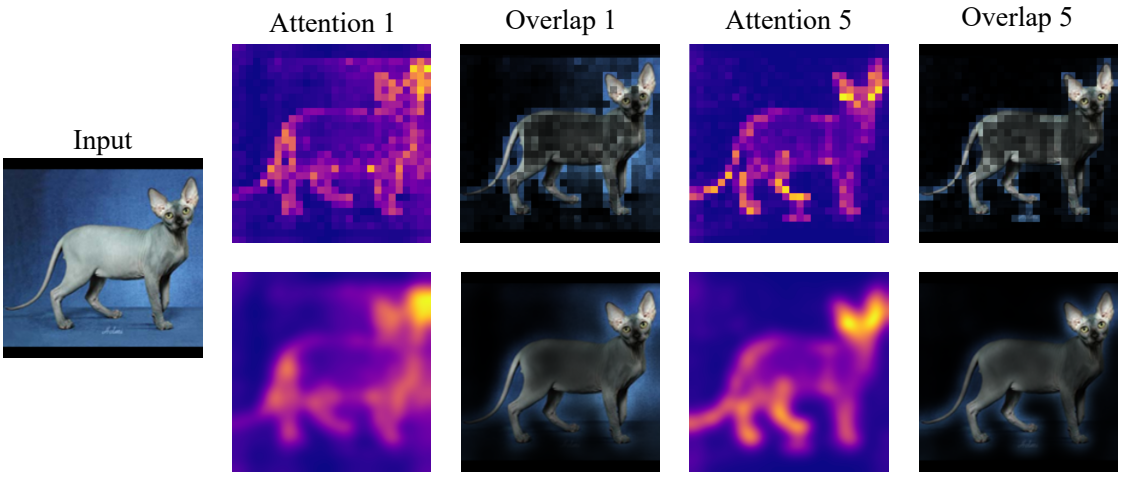


Figure 6.5: Comparison of the original (top) and blurred (bottom) version of RecViT's attention maps, with their overlaps for the images on the first (left) and the fifth (right) layer.

6.6.2 Adversarial examples and attention

In a previous study by Kotyan and Vargas (2021), it was pointed out that deep convolutional neural networks lose their focus of attention when an AE is inserted. To test whether similar behavior holds in our RecViT network, we seek differences in the

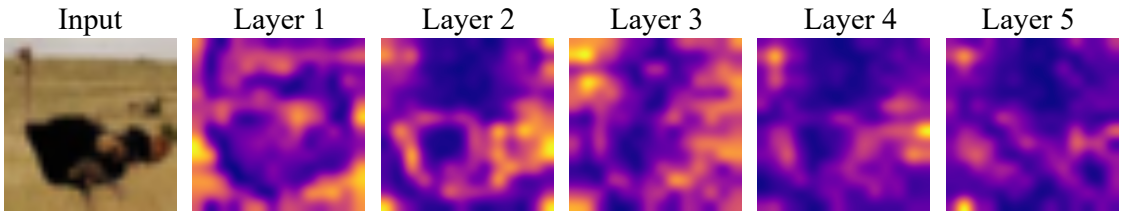


Figure 6.6: Development of the attention maps for a CIFAR-10 input on the first five layers of RecViT.

heatmaps triggered by clean and adversarial inputs. When plotting the activations of AEs and their corresponding clean examples (see Fig. 6.7), we frequently observe that the network does indeed lose focus on AEs. On the other hand, upon processing a clear image, the network generates consistent heatmaps. Our observations, however, are pretty difficult to analyze, and it is hard to draw justifiable conclusions. To alleviate this issue and to produce a numerical assessment of the attention map's "correctness", we use the information encoded in the segmentation map provided for the PET dataset. Therefore, we calculate the overlap of the attention map with the segmentation mask.

There are several possibilities for the overlap computation. Cosine similarity, dot product, or a simple distance measure in L_p norm are usually apt techniques for such purposes. However, these techniques are unsuitable for building statistics across multiple inputs. The varying percentage of pixels tagged as "inside" the object, causes a scaling problem. The same perceived overlap gets mapped to different values, rendering the collection of overlap statistics not comparable.

To mitigate the scaling problem, we propose the following similarity computation. First, we use the segmentation map to construct two binary matrices, $\boldsymbol{S}^+$ containing ones inside of the object and $\boldsymbol{S}^-$ marking with ones the background pixels of the image. Second, to obtain vectors instead of matrices ($\boldsymbol{s}^+$ and $\boldsymbol{s}^-$), we flatten the matrix representations. Similarly, we flatten the merged representation of attention scores to get a vector $\boldsymbol{a}$. Third, we compute the overlap score of the attention map $\boldsymbol{a}$

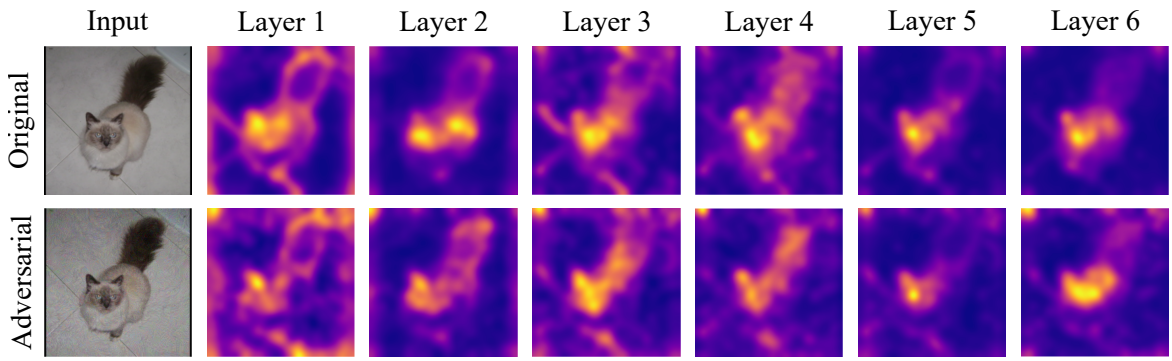


Figure 6.7: Comparison of attention maps in early layers of RecViT, after presenting a clean image of a cat (top row) and its corrupted version (bottom row).

with the positive and the negative part separately, as

$$\text{score}^{+} = \frac{\sum_i a_i s_i^{+}}{\|\boldsymbol{s}^{+}\|_1}, \qquad \text{score}^{-} = \frac{\sum_i a_i s_i^{-}}{\|\boldsymbol{s}^{-}\|_1}. \tag{6.3}$$

The final step is to compute the overall similarity score as the difference between the positive and the negative components: $\text{score} = \text{score}^{+} - \text{score}^{-}$.

Having a clearly defined similarity score, we tested the attention of various RecViT versions. In Fig. 6.8, we compare the least robust RecViT versions (Vanilla) with the most robust versions (InvBlur), where the number of iterations was set to $k = 3$. We have several main observations:

1. The similarity score peaks three times in all networks (except Vanilla M1). In our opinion, this corresponds to the network's highest focus on the relevant image regions prior to the prediction.

2. RecViT Vanilla networks have, in general, smaller similarity scores than their more robust counterpart (InvBlur).

3. The Vanilla M1 network does seem to extract relevant information only during the first iteration. Afterwards, it just retains the produced information (as the attention is spread across the whole image).

4. The Vanilla M2 network evidently loses attention when an AE is presented. The similarity score here is lower than for the original examples throughout the whole network.

5. During the first two iterations of the InvBlur models, the scores for AEs copy the scores of clean examples. We judge that the effect of blurring the image forces the model to focus on the same parts for the AE as for its clean counterpart.

6. The similarity score of the most robust model (InvBlur, M2) uniformly increases in each iteration.

6.7 Summary

Based on the vision transformers, we designed RecViT, a model that incorporates top-down attention flow for image classification. While there is no memory overload in our design, the inference time grows linearly with the number of used iterations. Even though RecViT, at first, demonstrated only marginal robustness gains by using tailored input augmentations, we were able to train the model to become more resistant against the tested adversarial attack. The cost of the heightened robustness is only 1% drop in clean accuracy, verified on two datasets.

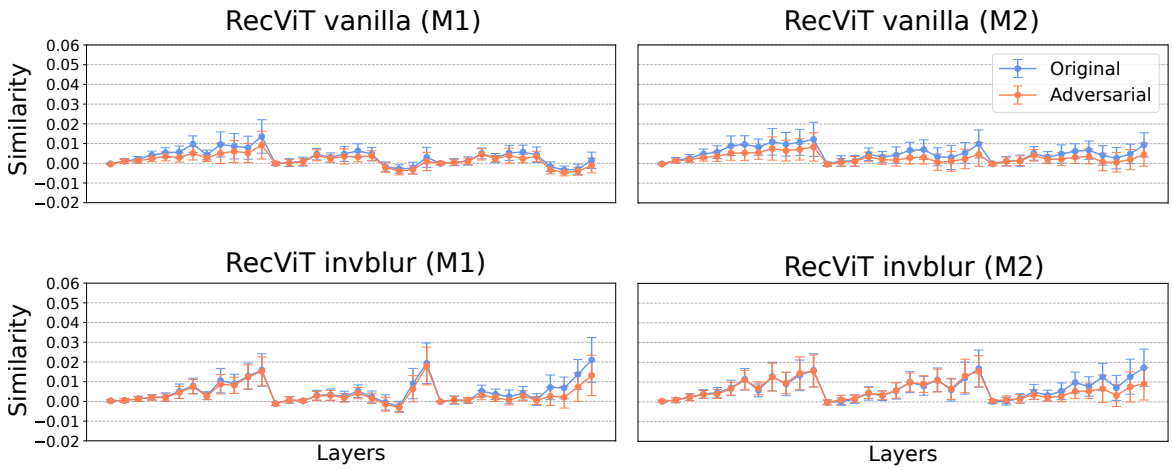


Figure 6.8: Development of heatmap similarity to the segmentation maps of the PET clean examples vs. corresponding corrupted images, computed using 100 samples.

Our analysis shows that the RecViT network possesses the capability to resist AEs better than an unprotected network. We also found a positive correlation between robustness and clean accuracy. In our opinion, improvements can be achieved by a more sophisticated input transformation to incorporate multiple levels of details in the network's decision process. To verify whether a similar level of robustness holds for more tailored AEs, further work should also include testing on adaptive attacks.

Robustness improvement is only one part of our study. We also analyzed the attention of the network. For heatmap visualization, we utilized the inherently available raw attention scores. In our models, we detected and numerically assessed the differences in attention when the network is processing clean vs. adversarial images. Even though the attention maps provide a certain level of feedback, the amount of interpretability remains unclear due to many non-linear transformations preceding the self-attention computation. For future research, it would be beneficial to explore the effects of more sophisticated extraction methods of self-attention explanations.

Chapter 7

Explainable addressee estimation

This chapter introduces a novel approach to addressee estimation in *Multi-Party Conversations (MPCs)*. By building upon the previous work and incorporating various attention mechanisms, we design an architecture that surpasses the former state-of-the-art technique in terms of predictive accuracy, and allows the user to generate real-time explanations of its actions.

The work described in this chapter constitutes a major part of a paper "A Multi-Modal Explainability Approach For Human-Aware Robots in Multi-Party Conversation", which is at the time of writing this thesis under consideration in the Computer Vision and Image Understanding[1] journal. The key concepts, along with some preliminary results, are available as an extended abstract in Bečková and Pócoš (2024).

7.1 Motivation and related work

In a multi-party conversation, the ability to correctly recognize the addressee is vital, as it contributes to smooth conversation. For a machine learning system to fully grasp a particular MPC, the question "Who says what to whom?" should be answered (Gu et al., 2022). In our study, we focus only on the third part of the question (to whom?). People are accustomed to leading complex conversations and, in general, thrive in this task. However, deploying a humanoid robot in real world to lead a smooth conversation is anything but trivial. In order to be successful in an MPC scenario, it needs a well-designed algorithm and proper data to train on.

Earlier approaches for addressee estimation focused on rule-based systems (Traum et al., 2004). In subsequent works, these were substituted for more advanced methods, such as Bayesian networks (Jovanovic, 2007) or support vector machines (Baba et al., 2011). After deep learning gained popularity, various forms of neural networks have been used to analyze multi-party interactions. An example is Minh et al. (2018), who

[1] https://www.sciencedirect.com/journal/computer-vision-and-image-understanding

were the first to train end-to-end deep learning to estimate the addressee, provided the utterance and the gaze information. A similar approach was examined by Tesema et al. (2023), who, along with the proposal of a novel dataset allowing a fine-grained audiovisual analysis (E- MuMMER), used CNNs to estimate the addressee. Building on these advances, Mazzola et al. (2023) proposed and examined an addressee estimation pipeline that combines facial and pose information on the Vernissage dataset (Jayagopi et al., 2013).

7.2 Preparations

7.2.1 Vernissage corpus

Training a model to estimate the addressee requires a solid data corpus. In our work, we make use of the Vernissage dataset (Jayagopi et al., 2013) due to its perfect fit to our needs and the possibility of comparing it with the benchmark accuracy set by Mazzola et al. (2023). The dataset contains video recordings of 13 triadic interactions between two participants and a stationary robot NAO (Shamsuddin et al., 2011), placed in a realistic environment where the participants discuss paintings at an art exhibition.

The dataset is extensively annotated, allowing supervised training algorithms to be employed for a wide range of perception tasks. These include pose estimation, speech recognition, speaker localization, nodding, and many more. For the sake of this study, we only focus on the addressee estimation.

Each video segment is labeled with one of the following tags, according to the addressee's position: ROBOT, RIGHT, LEFT, GROUP, and NO-LABEL. To consistently compare our approach with the baseline model (Mazzola et al., 2023), we limit the possible output values, and we only use the interaction segments in which ROBOT, RIGHT, or LEFT are addressed. We also employ identical data pre-processing, where the data from the robot's cameras are processed into two parallel streams, carrying information about the speaker in two ways: 1) face images and 2) body pose vectors. These serve as inputs to our ML model. The input is further divided into non-overlapping sequences of 10 frames, which we use for training. A sample sequence is shown in Fig. 7.1.

7.2.2 Improving the state-of-the-art

Prior to a detailed description of our explainable model, we seek to improve the state-of-the-art accuracy through subtle changes in the architecture, followed by a thorough hyperparameter search. The previous model (Mazzola et al., 2023) consists of three main steps:

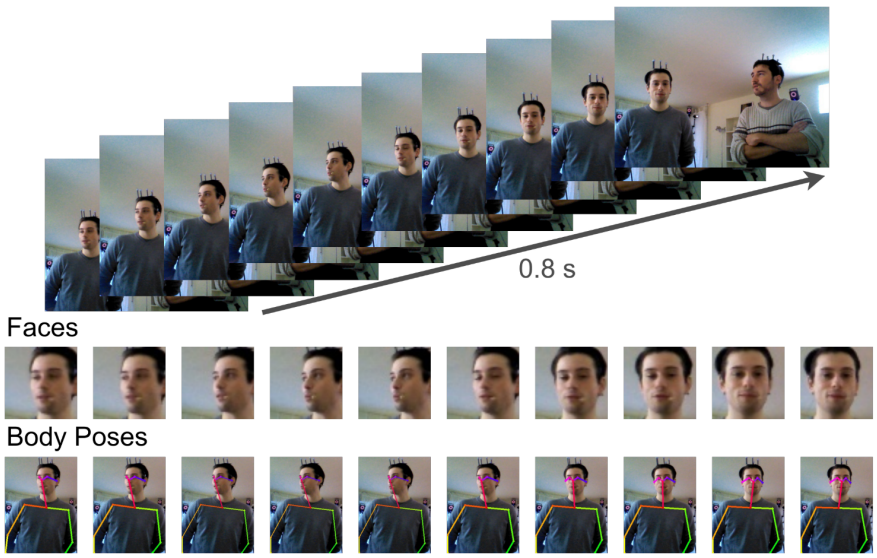


Figure 7.1: A sample sequence from the Vernissage dataset (top) and two data streams created from it (bottom) during the data pre-processing (Mazzola et al., 2023).

1. Parallel processing of the face image and the body pose data for a fixed time step t. In both cases, a convolutional network is employed to produce the face and the pose representation, $\boldsymbol{f}_t$ and $\boldsymbol{p}_t$, respectively.

2. Fusion of face and pose embeddings, producing a single vector representation $\boldsymbol{r}_t$ of the inputs at the given time t. To mitigate the weak influence of the pose vector (compared to the longer face vector), the pose vector is appended 29 times instead of a simple concatenation.

3. Gradual insertion of the fused embeddings $\boldsymbol{r}_t$ into an LSTM network, to integrate the information about all time frames. Finally, a simple MLP is employed to produce the final addressee estimate.

We alter the architecture of Mazzola et al. (2023) by three subtle yet essential tweaks. First, we replace the pose network with a simple MLP. As the body pose data are made of triplets of key points extracted using OpenPose (Cao et al., 2021) (2D coordinates along with their confidence scores), we argue that a shallow MLP does an equally good job, if not better, than a CNN. Second, we revert to a plain concatenation of the face and the pose embeddings. To prevent neglecting the influence of the pose vector, while optimizing the embedding dimensions, we avoid large mismatch in size between the two representations. Third, we replace the LSTM network with a GRU. Based on the complexity of the dataset, the choice of a simpler recurrent network can be beneficial.

To evaluate our model, we train it on all interactions except one, which is kept for testing. This step is repeated, but in each repetition we exclude data from a different

interaction. To also compute the statistical validity of our testing results, we train five runs of every configuration. Table 7.1 contains information about the search range of the number of neurons used and the activation functions. Table 7.2 refers to the parameters relating to the optimization process, and Table 7.3 lists the possible data augmentation strategies, their ranges and chosen values for the best performing model. Bayesian optimization was applied to find the parameters for the best performing model, leveraging the online tool WandB (Biewald, 2020).

Using our approach, we achieve an average absolute improvement of 4.5% F1 score compared to the state-of-the-art (79.51% instead of 75.01%). Moreover, our model requires a substantially lower number ($\approx 0.74\%$) of trainable weights (677,623 instead of 91,706,749). By eliminating the huge memory and computational overload, the deployment of our model in physical robots to provide real-time addressee estimation is much more feasible.

Table 7.1: List of hyperparameters defining the size of the addressee estimation architecture and the activation functions, their ranges and the selected values.

Parameter name	Considered values	Chosen value
normalization	{data stats, ImageNet stats}	data stats
dropout probability	{0, 0.1, 0.2, 0.3}	0.2
hidden neurons face	{128, 129, ..., 400}	256
hidden neurons pose	{16, 17, ..., 40}	32
hidden neurons GRU	{16, 17, ..., 40}	32
output neurons face	{10, 11, ..., 64}	32
output neurons pose	{10, 11, ..., 32}	20
output neurons GRU	{8, 9, ..., 32}	20
activation function face	{ReLU, tanh}	tanh
activation function pose	{ReLU, tanh}	tanh
activation function GRU	{ReLU, tanh}	tanh
CNN	{small, medium, large}	large

7.3 Attentional model of addressee estimation

Having designed and trained a model for estimation of addressees' location in the Vernissage dataset, we now strive to introduce several modifications, so that the model would be inherently capable of producing various kinds of explanations of its decisions. To achieve this, we take advantage of the existence of multiple types of attention mechanisms often used in ML systems (Brauwers and Frasincar, 2023).

To be more specific, we propose three additional changes of our previous model, each bringing its own way of extracting some form of explanations. The overall scheme

Table 7.2: List of optimization-related hyperparameters of the addressee estimation model, their ranges and the selected values.

Parameter name	Considered values	Chosen value
optimizer face	{SGD, Adam, RMSprop}	RMSprop
optimizer pose	{SGD, Adam, RMSprop}	RMSprop
optimizer GRU	{SGD, Adam, RMSprop}	Adam
LR decay pose	[0.5, 1]	0.75
LR decay face	[0.5, 1]	0.9725
LR decay GRU	[0.3, 1]	0.5
learning rate pose	$[e^{-10}, e^{-7}]$	0.00018
learning rate face	$[e^{-10}, e^{-2}]$	0.01
learning rate GRU	$[e^{-10}, e^{-7}]$	0.00009
batch size	{10, 11, . . . , 350}	16
number of epochs	{5, 6, . . . , 50}	15

Table 7.3: List of hyperparameters of the addressee estimation model used during data augmentation, their ranges and the selected values. The parameters describe various transformations of the input image.

Parameter name	Considered values	Chosen value
brightness	[0, 0.5]	0.2
contrast	[0, 0.5]	0.4
saturation	[0, 0.5]	0.45
hue	[0, 0.25]	0.135
angle	[0, 45]	25
crop	[40, 50]	44
kernel size	{1, 3, 5, 7, 9}	7
sigma	[0.1, 3]	0.8

is shown in Fig. 7.2, in which, using the models M1, M2 and M3, elaborately described in this section, we are able to generate the explanations.

7.3.1 Incorporating attention

Face representation We replace the convolutional network processing the face with a vision transformer. The main contribution of ViT is the ability to extract heatmaps, signifying the real-time pixel-wise importance. Its drawback is, in general, poor performance without pre-training on a massive data corpus. However, in this case, we did not observe any performance decrease compared to the CNN approach.

Pose vs. face After calculating the face embedding ($\boldsymbol{f}_t \in \mathbb{R}^{d_{face}}$) via a vision transformer and the pose embedding ($\boldsymbol{p}_t \in \mathbb{R}^{d_{pose}}$) using an MLP, we devise a way to combine

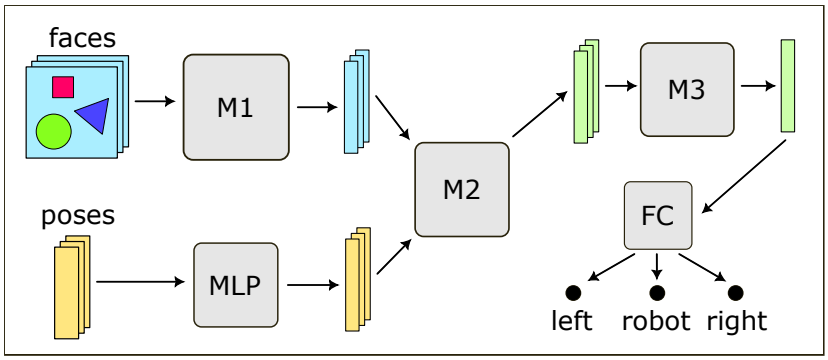


Figure 7.2: Addressee estimation workflow for the explainable model. The network M1 (ViT) computes the face representation, M2 serves to fuse the two data streams, and M3 merges information across the entire sequence.

these representations in a manner that also produces a numerical assessment of their relative contributions. To do this, we leverage the query-less variant of the additive scoring function (Brauwers and Frasincar, 2023), which outputs a single score for a given input vector $\boldsymbol{v}$:

$$\text{score}(\boldsymbol{v}) = \boldsymbol{w}_D^T \, \text{ReLU}(\boldsymbol{W}\boldsymbol{v} + \boldsymbol{b}), \tag{7.1}$$

where $\boldsymbol{w}_D \in \mathbb{R}^{d_{in}}$, $\boldsymbol{W} \in \mathbb{R}^{d_{in} \times d_v}$, $\boldsymbol{b} \in \mathbb{R}^{d_{in}}$ are trainable weights, d_{in} is a tunable hyperparameter, and d_v is the length of the input vector $\boldsymbol{v}$. To be capable of using the same weights for scoring both the face and the pose embeddings, we ensure that $d_{face} = d_{pose} = d_v$. Next, we apply softmax to the scores to compute the relative contributions $(s_{\boldsymbol{f}_t}, s_{\boldsymbol{p}_t})$ of each of the two vectors as:

$$s_{\boldsymbol{f}_t}, s_{\boldsymbol{p}_t} = \text{softmax}(\text{score}(\boldsymbol{f}_t), \text{score}(\boldsymbol{p}_t)). \tag{7.2}$$

The final step is the element-wise addition of the two embeddings, given their corresponding weights, to produce a fused embedding $\boldsymbol{r}_t$ for a given time step t:

$$\boldsymbol{r}_t = s_{\boldsymbol{f}_t} \boldsymbol{f}_t + s_{\boldsymbol{p}_t} \boldsymbol{p}_t. \tag{7.3}$$

Gated Recurrent Unit with attention To fully combine the information about the entire sequence $(\boldsymbol{r}_1, ..., \boldsymbol{r}_n)$ in the baseline architecture, we employ a GRU network. However, to develop an "explainable" alternative, we go beyond the ordinary GRU and include a variation of attention mechanism, in which we use the hidden state of the RNN as a query (Brauwers and Frasincar, 2023). Thanks to the added attention, along with the inference computation, we can extract time-frame-wise importance scores.

Let us denote the stacked representation of the processed inputs through all time frames as $\boldsymbol{R} = [\boldsymbol{r}_1, \boldsymbol{r}_2, ..., \boldsymbol{r}_n]$. These are linearly projected to form keys, queries, and values:

$$\boldsymbol{Q}_r = \boldsymbol{W}^Q \boldsymbol{R}, \quad \boldsymbol{K}_r = \boldsymbol{W}^K \boldsymbol{R}, \quad \boldsymbol{V}_r = \boldsymbol{W}^V \boldsymbol{R}, \tag{7.4}$$

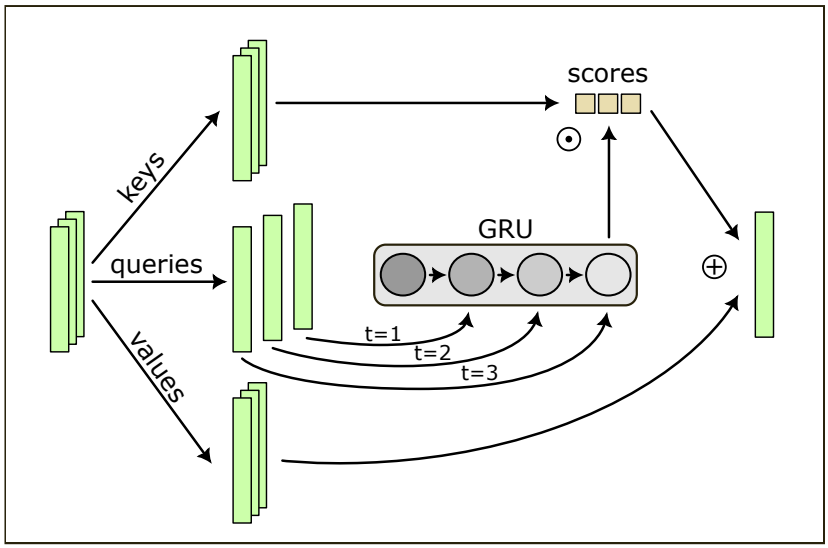


Figure 7.3: Processing of sequential data using the GRU network combined with an attention mechanism. The input sequence (left) is projected into keys, queries, and values. The queries are fed into the GRU, and the final hidden representation is used to extract the importance scores applied to the values.

where the $\boldsymbol{W}^Q, \boldsymbol{W}^K$ and $\boldsymbol{W}^V$ are trainable weights. The query vectors are then gradually fed into the GRU to construct the embedding $\boldsymbol{q}$, integrating information about all time frames. Using $\boldsymbol{q}$, we compute its similarities with the keys encoded in the matrix $\boldsymbol{K}$, providing the significance of each time frame contribution ($\boldsymbol{c} = \boldsymbol{K}\boldsymbol{q}$). Finally, the sequence representation $\boldsymbol{u}$ is formed after an element-wise addition of the values $\boldsymbol{V}$, using the weights contained in $\boldsymbol{c}$:

$$\boldsymbol{u} = \sum_{i=1}^{n} c_i \boldsymbol{V}_i. \tag{7.5}$$

A fully connected layer taking $\boldsymbol{u}$ as an input produces the final addressee estimation. Fig. 7.3 provides a graphic illustration of our recurrent block.

7.3.2 Performance

Even though we focused our model design mainly on improving the inherent explainability, after training the network, we achieved comparable accuracy with the previously created, non-explainable model. In the confusion matrices (see Fig. 7.4) of the explainable and the non-explainable models, we see very similar misclassification rates. Both models have slightly higher accuracy when the addressee is not the robot. The average F1 score of the model after implementing all the explanations decreased only marginally, and is roughly on par with the basic version (79.40% and 79.51%, respectively).

non-explainable model

real class	predicted class: robot	predicted class: left	predicted class: right
robot	75.54%	13.91%	10.55%
left	14.48%	81.30%	4.22%
right	14.54%	6.08%	79.38%

explainable model

real class	predicted class: robot	predicted class: left	predicted class: right
robot	74.74%	11.34%	13.91%
left	12.71%	81.05%	6.23%
right	12.71%	6.08%	81.21%

Figure 7.4: Confusion matrices for the non-explainable (left) and the explainable (right) model.

7.3.3 Extracting explanations

Vision Here, we exploit the nature of the self-attention computation and simply visualize the raw attention scores. Due to the modest image resolution of 50×50 pixels, the quality of the heatmaps is limited; nonetheless, they provide real-time feedback about the important regions of the input image. A sample visualization of the heatmaps using two frames of the Vernissage dataset can be found in Fig. 7.5.

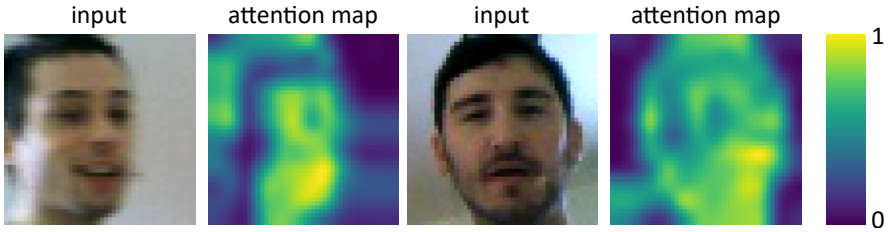


Figure 7.5: Visualization of the attention maps extracted from the penultimate layer of the vision transformer.

Merging modalities To compare the relative importance of face and pose information, we simply extract the weights used to combine the two embeddings. The knowledge of the significance of each of the factors can provide the speaker cues about what the classification is based on. In case the model makes an incorrect estimate, the speaker can potentially re-adjust their position to improve the accuracy of the subsequent or even the current estimate.

To determine whether our model is capable of producing informative explanations, we analyzed the score distribution for the best trained model. We discovered that the face and the pose in the Vernissage dataset are equally important. The average score for the pose is 0.5 with a standard deviation 0.008. The face vector gets an equal contribution since the softmax function is employed to determine the relative scores. This practically means that no valuable feedback is provided.

To explore our possibilities, we tested the effect of having various sizes of pose and face embedding dimensions. We hypothesized that if the embedding dimension is low enough, in order for the network to make the best possible classification, it needs to treat the two modalities unequally. Especially in cases when some information is corrupted (e.g., partially occluded face or erroneous extraction of body pose keypoints). After training the network with various lengths of pose and face vectors, we noticed a strong negative correlation of -0.87 ($p = 0.001$) between the dimensionality used for the vector embeddings and the logarithm of the deviations of the score. This translates to a higher expressiveness of the network when the dimensions are lower. It also means that when optimizing the model, keeping the dimensions of the pose and face vectors low is beneficial for explainability.

We also found that the expressiveness of the network heavily relies on other components integrated into the architecture. For example, after replacing the ViT with a CNN and using GRU without the attention, the explanation capability increases. The pose information is in this case slightly prevalent, having a score of ≈ 0.62 with a standard deviation of 0.15. Whereas for a comparable embedding dimensionality in the whole combined architecture, the pose has an average score of 0.5 and a deviation of 0.04.

Verbal explanations The attention weights provided in the recurrent network offer an ideal way of retrieving information about those time frames, that have the highest impact on the classification.

Using these activations, we design a method to automatically generate a verbal cue about the time of the most important part of the input sequence. To capture this, we use the average of the attention weights after computing the similarities of the query from the GRU with the keys. The average of a sliding window is compared to a threshold value θ, and if it exceeds the number $1/k + \theta$, where k is the length of the sequence, an output sentence is generated based on the sliding window's location. As a proof of concept, in this experiment we distinguish between three possible regions of importance: the beginning, the middle, and the end of the interaction.

Given sequences of length 10, two sample verbal cues are shown in Fig. 7.6. We can see the weights of the individual time frames distinguished by the color and size of the red dots. Even though these explanations are particularly easy to interpret, we must be aware that mixed information about the face and pose is used. This means that the visualization of the faces alongside the attention scores is merely a feature to exploit and cannot fully serve to assess the credibility of the explanations.

To further explore the properties of our explainable model, we analyze the scores, given 10-frame-long sequences of the Vernissage dataset. When considering only the stack of the score values for all the input sequences independently, they precisely follow

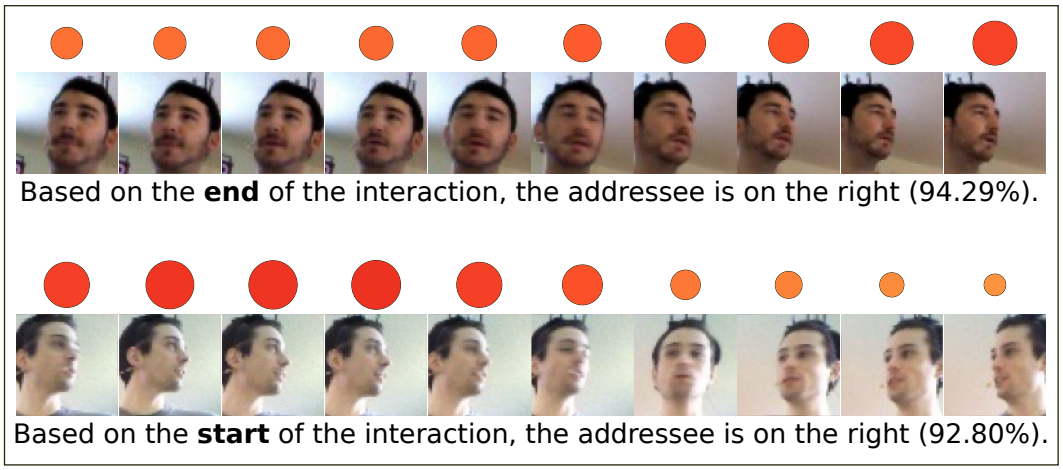


Figure 7.6: Two sequences with their corresponding attention scores (dots) generated by the GRU network. The color and size of the dots correspond to the magnitude of the attention score for each time frame. In both cases, the network generated an explanation for its correct estimation "right".

a Gaussian distribution with a mean of ≈ 0.1 and a standard deviation of ≈ 0.0055. This means that the network has properly learned to consider all the time frames equally. The deviation, although subtle, allows us to detect the irregularities, verbalize them, and provide to the user.

However, a question arises. What threshold value should we use? It clearly influences the rate at which the verbal cue is generated. In Fig. 7.7 we see the probabilities of triggering a verbal response for differing threshold values collected on the sequences of the Vernissage dataset. We empirically verified that threshold values around 0.02 yield verbal cues aligning with our expectations. They are not too overwhelming, and mostly trigger when the speaker is rapidly moving (in cases of higher ambiguity). By choosing a lower value, the noise patterns can overrule the valuable information, yielding responses that are difficult to interpret or verify.

7.4 Implementation in iCub robot

To showcase our model's functionality in a real-world application, we implemented it into the humanoid robot iCub (Metta et al., 2010). Since iCub's perception differs from the NAO's (which was used to collect the Vernissage dataset), we fine-tuned our model on the data provided in Saade (2023). These data are analogous to those from the Vernissage dataset, at least when it comes to the addressee estimation. They contain five interactions altogether, where the addressee is always labeled. Using this dataset, we can prepare the model for real-life interaction, after which a *Large Language Model (LLM)* is employed to enable smooth conversation with the participants while also

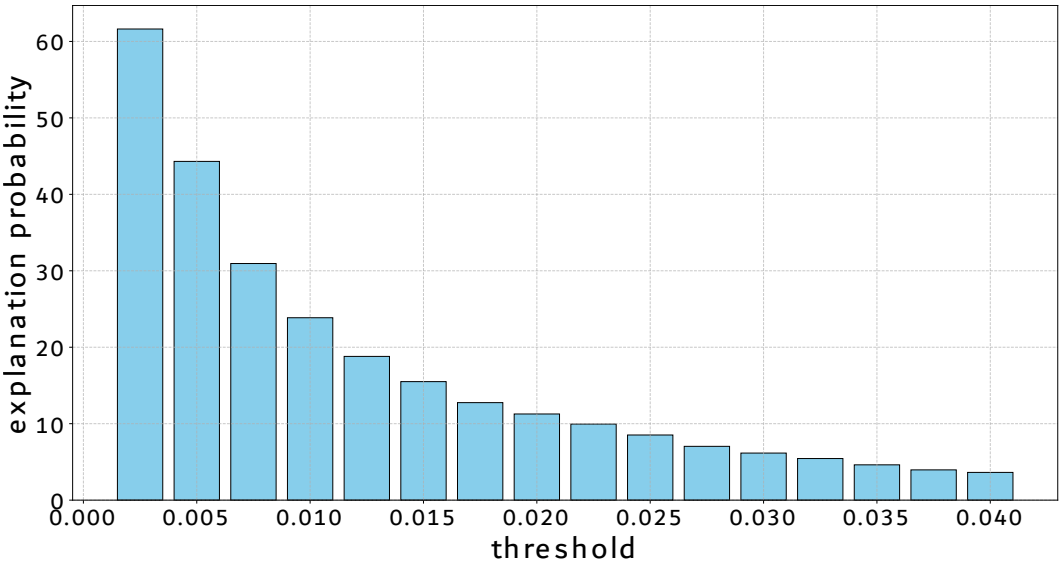


Figure 7.7: The influence of threshold value (x-axis) on the probability (y-axis) of a verbal explanation triggered at the end of a sequence.

providing explanations.

All three forms of explanations are integrated into the robot. The face saliency and the relative importance of the two data streams are visualized on an external screen during the conversation. Meanwhile, the output sentence is being played using the iCub's built-in microphone. To test the likeability and the overall usefulness of the explanations, a pilot user study was conducted. However, for a more complex understanding of the robot's usefulness and the influence of individual explanation methods, more thorough evaluations are necessary.

7.5 Summary

The explainable addressee estimation model we designed in this work offers verbal and visual explanations alongside the addressee estimation. Even though explainability seems to come at no cost of accuracy, it is important to address its limitations. Given the surprisingly high sensitivity of the frequency and quality of the explanations on the hyperparameter setup used during training, a classical error optimization might not guarantee the model's production of meaningful explanations. Thus, it would be useful to devise a training method that aligns with achieving remarkable accuracy while generating valuable and discriminating explanations.

Apart from the human-grounded evaluations provided by various user studies, evaluating the level of explainability presents perhaps one of the greatest challenges, due to the unavailability of standard evaluation metrics for explanations in this task. Also, during the training of a model for addressee estimation, the performance heavily relies

on the quality and representativeness of the dataset, which may not always align with real-world scenarios. Thus, to continue in this line of research, a data-centric approach might yield further substantial improvements both in the quality of the explanations and in the model accuracy. For example, it can involve capturing more interactions in varying light conditions to make the model more applicable to real-life scenarios.

Bibliography

Abnar, S. and Zuidema, W. (2020). Quantifying attention flow in transformers. In Jurafsky, D., Chai, J., Schluter, N., and Tetreault, J., editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4190–4197, Online. Association for Computational Linguistics.

Angelov, P., Soares, E., Jiang, R., Arnold, N. I., and Atkinson, P. M. (2021). Explainable artificial intelligence: an analytical review. *WIREs Data Mining and Knowledge Discovery*, 11.

Apley, D. W. and Zhu, J. (2020). Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 82(4):1059–1086.

Athalye, A., Carlini, N., and Wagner, D. (2018). Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*.

Baba, N., Huang, H.-H., and Nakano, Y. I. (2011). Identifying utterances addressed to an agent in multiparty human–agent conversations. In Vilhjálmsson, H. H., Kopp, S., Marsella, S., and Thórisson, K. R., editors, *Intelligent Virtual Agents*, pages 255–261, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations*.

Bai, Y., Mei, J., Yuille, A. L., and Xie, C. (2021). Are transformers more robust than cnns? In *Advances in Neural Information Processing Systems*, volume 34, pages 26831–26843. Curran Associates, Inc.

Barkan, O., Hauon, E., Caciularu, A., Katz, O., Malkiel, I., Armstrong, O., and Koenigstein, N. (2021). Grad-sam: Explaining transformers via gradient self-attention maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21. ACM.

Barredo Arrieta, A., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., and Herrera, F. (2020). Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115.

Bečková, I. and Pócoš, Š. (2024). Explainable addressee estimation. In *Cognition and Artificial Life 2024*, pages 34–35. Flow.

Bečková, I., Pócoš, Š., and Farkaš, I. (2020). Computational analysis of robustness in neural network classifiers. In Farkaš, I., Masulli, P., and Wermter, S., editors, *Artificial Neural Networks and Machine Learning – ICANN 2020*, pages 65–76, Cham. Springer International Publishing.

Bečková, I., Pócoš, Š., and Farkaš, I. (2022). Skúmanie vzdialeností adverzariálnych vstupov k jednotlivým triedam v hlbokých neurónových sieťach. In *Kognice a umělý život 20*, pages 160–161. České vysoké učení technické v Praze.

Biewald, L. (2020). Experiment tracking with weights and biases. Software available from wandb.com.

Brahma, P. P., Wu, D., and She, Y. (2016). Why deep learning works: A manifold disentanglement perspective. *IEEE Transactions on Neural Networks and Learning Systems*, 27(10):1997–2008.

Brauwers, G. and Frasincar, F. (2023). A general survey on attention mechanisms in deep learning. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):3279–3298.

Brendel, W., Rauber, J., and Bethge, M. (2017). Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *International Conference on Learning Representations*.

Cao, Z., Hidalgo, G., Simon, T., Wei, S., and Sheikh, Y. (2021). OpenPose: Realtime multi-person 2D pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 43(01):172–186.

Carlini, N., Athalye, A., Papernot, N., Brendel, W., Rauber, J., Tsipras, D., Goodfellow, I., Madry, A., and Kurakin, A. (2019). On evaluating adversarial robustness. *arXiv preprint arXiv:1902.06705*.

Carlini, N. and Wagner, D. (2017). Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy*, pages 39–57.

Carlini, N. and Wagner, D. (2018). Audio adversarial examples: Targeted attacks on speech-to-text. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 1–7.

Chen, P.-Y., Zhang, H., Yi, J., and Hsieh, C.-J. (2017). EAD: Elastic-net attacks to deep neural networks via adversarial examples. *arXiv preprint arXiv:1709.04114*.

Cho, K., van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014a). On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111. Association for Computational Linguistics.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014b). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734. Association for Computational Linguistics.

Cohen, S., Bitton, R., and Nassi, B. (2024). Here comes the AI worm: Unleashing zero-click worms that target genai-powered applications. *arXiv preprint arXiv:2403.02817*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE.

Denil, M., Bazzani, L., Larochelle, H., and de Freitas, N. (2012). Learning where to attend with deep Architectures for image tracking. *Neural Computation*, 24(8):2151–2184.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159.

Ebrahimi, J., Rao, A., Lowd, D., and Dou, D. (2018). HotFlip: White-box adversarial examples for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 31–36, Melbourne, Australia. Association for Computational Linguistics.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179–211.

Engstrom, L., Ilyas, A., and Athalye, A. (2018). Evaluating and understanding the robustness of adversarial logit pairing. *arXiv preprint arXiv:1807.10272*.

Frosst, N., Papernot, N., and Hinton, G. (2019). Analyzing and improving representations with the soft nearest neighbor loss. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 2012–2020.

Gehrig, M. and Scaramuzza, D. (2023). Recurrent vision transformers for object detection with event cameras. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13884–13893.

Gilmer, J., Metz, L., Faghri, F., Schoenholz, S., Raghu, M., Wattenberg, M., and Goodfellow, I. (2018). Adversarial spheres. In *International Conference on Learning Representations*.

Goodfellow, I., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*.

Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1319–1327, Atlanta, Georgia, USA. PMLR.

Graves, A. (2014). Generating sequences with recurrent neural networks. arXiv:1308.0850 [cs.NE].

Gu, J.-C., Tao, C., and Ling, Z.-H. (2022). Who says what to whom: A survey of multi-party conversations. In *Proceedings of the 31st International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 5486–5493. International Joint Conferences on Artificial Intelligence Organization. Survey Track.

Han, K., Xiao, A., Wu, E., Guo, J., XU, C., and Wang, Y. (2021). Transformer in transformer. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 15908–15919. Curran Associates, Inc.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Hu, W. and Tan, Y. (2017). Generating adversarial malware examples for black-box attacks based on gan. *arXiv preprint arXiv:1702.05983*.

Huang, S., Papernot, N., Goodfellow, I., Duan, Y., and Abbeel, P. (2017). Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*.

Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., and Madry, A. (2019). Adversarial examples are not bugs, they are features. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 448–456, Lille, France. PMLR.

Jaegle, A., Gimeno, F., Brock, A., Vinyals, O., Zisserman, A., and Carreira, J. (2021). Perceiver: General perception with iterative attention. In *International Conference on Machine Learning*, pages 4651–4664. PMLR.

Jayagopi, D. B., Sheiki, S., Klotz, D., Wienke, J., Odobez, J.-M., Wrede, S., Khalidov, V., Nyugen, L., Wrede, B., and Gatica-Perez, D. (2013). The vernissage corpus: A conversational human-robot-interaction dataset. In *8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 149–150.

Jovanovic, N. (2007). *To Whom It May Concern - Addressee Identification in Face-to-Face Meetings*. Doctoral thesis, University of Twente, Netherlands.

Kannan, H., Kurakin, A., and Goodfellow, I. (2018). Adversarial logit pairing. *arXiv preprint arXiv:1803.06373*.

Kashefi, R., Barekatain, L., Sabokrou, M., and Aghaeipoor, F. (2023). Explainability of vision transformers: A comprehensive review and new perspectives. *arXiv preprint arXiv:2311.06786*.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Knudsen, E. I. (2007). Fundamental components of attention. *Annual Review of Neuroscience*, 30(1):57–78.

Kotyan, S. and Vargas, D. V. (2021). Deep neural network loses attention to adversarial images. *arXiv preprint arXiv:2106.05657*.

Krizhevsky, A. (2012). Learning multiple layers of features from tiny images. *University of Toronto*. Doctoral Thesis.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

Kurakin, A., Boneh, D., Tramèr, F., Goodfellow, I., Papernot, N., and McDaniel, P. (2018). Ensemble adversarial training: Attacks and defenses. In *International Conference on Learning Representations*.

Kurakin, A., Goodfellow, I., and Bengio, S. (2017). Adversarial examples in the physical world. In *International Conference on Learning Representations*.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324.

LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database. `http://yann.lecun.com/exdb/mnist/`.

Lin, M., Chen, Q., and Yan, S. (2014). Network in network. *arXiv preprint arXiv:1312.4400*.

Linardatos, P., Papastefanopoulos, V., and Kotsiantis, S. (2021). Explainable AI: A review of machine learning interpretability methods. *Entropy*, 23(1).

Liu, X., Cheng, M., Zhang, H., and Hsieh, C.-J. (2018). Towards robust neural networks via random self-ensemble. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y., editors, *Computer Vision – ECCV 2018*, pages 381–397, Cham. Springer International Publishing.

Liu, Y., Chen, X., Liu, C., and Song, D. (2017). Delving into transferable adversarial examples and black-box attacks. In *International Conference on Learning Representations*.

Liu, Y., Zhang, Y., Wang, Y., Hou, F., Yuan, J., Tian, J., Zhang, Y., Shi, Z., Fan, J., and He, Z. (2021a). A survey of visual transformers. arXiv:2111.06091 [cs.CV].

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. (2021b). Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10012–10022.

Lucas, K., Jagielski, M., Tramèr, F., Bauer, L., and Carlini, N. (2023). Randomness in ML defenses helps persistent attackers and hinders evaluators. *arXiv preprint arXiv:2302.13464*.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Luong, T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2018). Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*.

Mahmood, K., Mahmood, R., and van Dijk, M. (2021). On the robustness of vision transformers to adversarial examples. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7818–7827.

Mazzola, C., Romeo, M., Rea, F., Sciutti, A., and Cangelosi, A. (2023). To whom are you talking? A deep learning model to endow social robots with addressee estimation skills. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–10.

McCulloch, W. and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147.

McInnes, L., Healy, J., Saul, N., and Großberger, L. (2018). UMAP: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861.

Messina, N., Amato, G., Carrara, F., Gennaro, C., and Falchi, F. (2022). Recurrent vision transformer for solving visual reasoning problems. In *International Conference on Image Analysis and Processing*, pages 50–61. Springer.

Metta, G., Natale, L., Nori, F., Sandini, G., Vernon, D., Fadiga, L., von Hofsten, C., Rosander, K., Lopes, M., Santos-Victor, J., Bernardino, A., and Montesano, L. (2010). The iCub humanoid robot: An open-systems platform for research in cognitive development. *Neural Networks*, 23(8):1125–1134.

Minh, T. L., Shimizu, N., Miyazaki, T., and Shinoda, K. (2018). Deep learning based multi-modal addressee recognition in visual scenes with utterances. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 1546–1553.

Mnih, V., Heess, N., Graves, A., and Kavukcuoglu, K. (2014). Recurrent models of visual attention. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Montavon, G., Samek, W., and Müller, K.-R. (2018). Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15.

Nicolae, M.-I., Sinn, M., Tran, M. N., Buesser, B., Rawat, A., Wistuba, M., Zantedeschi, V., Baracaldo, N., Chen, B., Ludwig, H., Molloy, I., and Edwards, B. (2018). Adversarial robustness toolbox v1.2.0. *arXiv preprint arXiv:1807.01069*.

Papernot, N. and McDaniel, P. (2018). Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv preprint arXiv:1803.04765*.

Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., and Swami, A. (2017). Practical black-box attacks against machine learning. In *Proceedings of the ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, page 506–519, New York, NY, USA.

Papernot, N., McDaniel, P., Wu, X., Jha, S., and Swami, A. (2016). Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy (SP)*, pages 582–597.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA. Association for Computational Linguistics.

Parkhi, O. M., Vedaldi, A., Zisserman, A., and Jawahar, C. V. (2012). Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Pócoš, Š., Bečková, I., and Farkaš, I. (2022). Examining the proximity of adversarial examples to class manifolds in deep networks. In *Artificial Neural Networks and Machine Learning – ICANN 2022*, pages 645–656, Cham. Springer Nature Switzerland.

Pócoš, Š., Bečková, I., and Farkaš, I. (2024a). Explainability of vision transformer with top-down connection. In *Cognition and Artificial Life 2024*, pages 28–29. Flow.

Pócoš, Š., Bečková, I., and Farkaš, I. (2024b). RecViT: Enhancing vision transformer with top-down information flow. In *Proceedings of the 19th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 3: VISAPP*, pages 749–756. INSTICC, SciTePress.

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "Why should i trust you?": Explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Rocktaschel, T., Grefenstette, E., Hermann, K. M., Kocisky, T., and Blunsom, P. (2016). Reasoning about entailment with neural attention. In *International Conference on Learning Representations*.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

Ruder, S. (2017). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

Saade, P. (2023). Optimizing the portability of an addressee estimation model in the icub social robot. Master's thesis, University of Genoa.

Schaul, T., Glasmachers, T., and Schmidhuber, J. (2011). High dimensions and heavy tails for natural evolution strategies. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, page 845–852. Association for Computing Machinery.

Schmidhuber, J. and Huber, R. (1991). Learning to generate artificial fovea trajectories for target detection. *International Journal of Neural Systems*, 02(01n02):125–134.

Schuster, M. and Paliwal, K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. (2017). Grad-CAM: Visual explanations from deep networks via gradient-based localization. In *IEEE International Conference on Computer Vision*, pages 618–626.

Shamir, A., Melamed, O., and BenShmuel, O. (2022). The dimpled manifold model of adversarial examples in machine learning. *arXiv preprint arXiv:2106.10151*.

Shamsuddin, S., Ismail, L. I., Yussof, H., Ismarrubie Zahari, N., Bahari, S., Hashim, H., and Jaffar, A. (2011). Humanoid robot NAO: Review of control and motion exploration. In *IEEE International Conference on Control System, Computing and Engineering*, pages 511–516.

Shao, R., Shi, Z., Yi, J., Chen, P.-Y., and Hsieh, C.-J. (2022). On the adversarial robustness of vision transformers. *arXiv preprint arXiv:2103.15670*.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Workshop at International Conference on Learning Representations*.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations (ICLR)*, pages 1–14.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2015). Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Srndic, N. and Laskov, P. (2014). Practical evasion of a learning-based classifier: A case study. In *IEEE Symposium on Security and Privacy*, pages 197–211.

Steiner, A. P., Kolesnikov, A., Zhai, X., Wightman, R., Uszkoreit, J., and Beyer, L. (2022). How to train your ViT? Data, augmentation, and regularization in vision transformers. *Transactions on Machine Learning Research*.

Stollenga, M. F., Masci, J., Gomez, F., and Schmidhuber, J. (2014). Deep networks with internal selective attention through feedback connections. *Advances in Neural Information Processing Systems*, 27.

Storn, R. and Price, K. (1997). Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, pages 341–359.

Stutz, D., Hein, M., and Schiele, B. (2018). Disentangling adversarial robustness and generalization. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6969–6980.

Su, J., Vargas, D. V., and Sakurai, K. (2019). One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2014). Intriguing properties of neural networks. In *International Conference on Learning Representations*.

Tesema, F. B., Gu, J., Song, W., Wu, H., Zhu, S., Lin, Z., Huang, M., Wang, W., and Kumar, R. (2023). Addressee detection using facial and audio features in mixed human–human and human–robot settings: A deep learning framework. *IEEE Systems, Man, and Cybernetics Magazine*, 9(2):25–38.

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., and Jegou, H. (2021). Training data-efficient image transformers & distillation through attention. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10347–10357. PMLR.

Tramer, F. (2022). Detecting adversarial examples is (Nearly) as hard as classifying them. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S., editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pages 21692–21702. PMLR.

Tramer, F. and Boneh, D. (2019). Adversarial training and robustness for multiple perturbations. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Traum, D. R., Robinson, S., and Stephan, J. (2004). Evaluation of multi-party virtual reality dialogue interaction. In Lino, M. T., Xavier, M. F., Ferreira, F., Costa, R., and Silva, R., editors, *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04)*, Lisbon, Portugal. European Language Resources Association.

Tsipras, D., Santurkar, S., Engstrom, L., Turner, A., and Madry, A. (2019). Robustness may be at odds with accuracy. In *International Conference on Learning Representations*.

van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Werbos, P. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2048–2057, Lille, France. PMLR.

Xu, W., Evans, D., and Qi, Y. (2017). Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*.

Xu, W., Qi, Y., and Evans, D. (2016). Automatically evading classifiers: A case study on pdf malware classifiers. In *Proceedings of the 2016 Network and Distributed Systems Symposium*.

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., and Hovy, E. (2016). Hierarchical attention networks for document classification. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489. Association for Computational Linguistics.

Zeiler, M. D. (2012). Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zeiler, M. D. and Fergus, R. (2013). Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901.

Zeiler, M. D., Krishnan, D., Taylor, G. W., and Fergus, R. (2010). Deconvolutional networks. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2528–2535.

Zhang, H., Yu, Y., Jiao, J., Xing, E. P., Ghaoui, L. E., and Jordan, M. I. (2019). Theoretically principled trade-off between robustness and accuracy. In *International Conference on Machine Learning*.

Zhang, Y., Tiňo, P., Leonardis, A., and Tang, K. (2021). A survey on neural network interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(5):726–742.

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., and Torralba, A. (2016). Learning deep features for discriminative localization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2921–2929.