



## Assignment of bachelor's thesis

<b>Title:</b>	Real-time Facial Expression Recognition in the Wild
<b>Student:</b>	Martin Vadlejch
<b>Supervisor:</b>	doc. Ing. Patrik Kutílek, MSc., Ph.D.
<b>Study program:</b>	Informatics
<b>Branch / specialization:</b>	Computer Science
<b>Department:</b>	Department of Theoretical Computer Science
<b>Validity:</b>	until the end of summer semester 2021/2022

### Instructions

The goal of this thesis is to come up with methods to identify human emotions from images. Such methods

should be able to detect faces in real-time and quantify their affinity towards different emotions. For implementation, use deep learning methods, trained on commonly used face datasets labelled with emotional affinity. Apart from the standard test dataset, use at least 100 additional photographs of different people, which shall be labelled by at least five respondents via a questionnaire.

Compare the results of the automated method with the questionnaire responses.





**FACULTY  
OF INFORMATION  
TECHNOLOGY  
CTU IN PRAGUE**

Bachelor's thesis

# **Real-time Facial Expression Recognition in the Wild**

*Martin Vadlejch*

Department of Theoretical Computer Science  
Supervisor: doc. Ing. Patrik Kutílek, MSc., Ph.D.

May 12, 2021



---

## **Acknowledgements**

I wish to express my deepest gratitude to my family and friends, for they have always been very supportive of me and helping me achieve my goals. I also wish to express my sincere appreciation to my consultant Ing. Jan Hejda, Ph.D., for his persistent help and insight.



---

# Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the “Work”), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on May 12, 2021

.....

Czech Technical University in Prague

Faculty of Information Technology

© 2021 Martin Vadlejch. All rights reserved.

*This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).*

### **Citation of this thesis**

Vadlejch, Martin. *Real-time Facial Expression Recognition in the Wild*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.



---

# Abstrakt

Strojové rozpoznávání emocí či výrazů tváře je technologie, se kterou se zatím běžně nesetkáváme přesto, že by mohla najít mnohé využití. Jedním z důvodů může být výpočetní náročnost současných konvolučních neuronových sítí, které byly pro tento problém dosavadně použité. V této práci najdeme vhodnou malou a rychlou konvoluční neuronovou síť, která je schopná rozpoznat výrazy z tváře v reálném čase, a to i na mobilních zařízeních. Pro učení sítě použijeme databázi AffectNet, v současnosti největší databázi výrazů tváře v přirozených podmínkách, a pro následnou validaci anotujeme dalších 160 snímků, na kterých vyhodnotíme úspěšnost modelu. Naše síť dosahuje téměř state-of-the-art výsledků, zároveň je zhruba desetkrát menší než ostatní použité neuronové sítě, a na Raspberry Pi 4 dokáže klasifikovat výrazy z tváře s až pěti snímky za vteřinu.

**Klíčová slova** detekce tváře, rozpoznání výrazu tváře, reálný čas, přirozené podmínky, databáze výrazů tváře, konvoluční neuronové sítě, porovnání architektur CNN, mobilní zařízení, Raspberry Pi 4

---

# Abstract

Machine emotion recognition or facial expression recognition is a technology that we do not often see just yet, despite its many possible uses. One reason could be the use of computationally demanding convolutional neural networks in current state-of-the-art models. In this work, we find a suitable lightweight and fast convolutional neural network capable of facial expression recognition in real-time, even on mobile devices. The network is trained on the AffectNet database, currently the largest facial expression database in the wild setting, and to further validate our results, we collect and annotate 160 additional images, on which we evaluate the model performance. Our network achieves near state-of-the-art accuracy while being almost ten times smaller than previous approaches and can recognize and classify facial expressions with up to five frames per second on the Raspberry Pi 4.

**Keywords** facial detection, facial expression recognition, real-time, in the wild, facial expression database, convolutional neural networks, CNN architecture comparison, mobile devices, Raspberry Pi 4

---

# Contents

<b>Introduction</b>	<b>1</b>
<b>1 Definition of Facial Expression</b>	<b>3</b>
1.1 Discrete Emotion Space . . . . .	3
1.1.1 Affective Categories . . . . .	3
1.1.2 Facial Action Coding System . . . . .	4
1.2 Continuous Emotion Space . . . . .	4
1.2.1 Russel’s Circumplex Model . . . . .	5
<b>2 Neural Networks</b>	<b>7</b>
2.1 Artificial Neuron . . . . .	7
2.1.1 Forward Pass . . . . .	8
2.1.2 Backward Pass . . . . .	8
2.1.3 Activation Functions . . . . .	8
2.1.3.1 Binary Classification . . . . .	10
2.1.3.2 Regression . . . . .	10
2.1.4 Cost Functions . . . . .	10
2.2 Multilayer Perceptron . . . . .	11
2.2.1 Multiclass Classification . . . . .	13
2.2.2 Multivariable Regression . . . . .	13
2.2.3 Backpropagation . . . . .	14
2.2.4 Common Problems of ANN . . . . .	15
2.3 Convolutional Neural Networks . . . . .	18
<b>3 Facial Detection</b>	<b>23</b>
3.1 Viola-Jones Object Detection Framework . . . . .	23
3.2 Histogram of Oriented Gradients and Bag of Features . . . . .	24
3.3 CNN-Based Approaches . . . . .	24
3.3.1 Multitask Cascaded CNN . . . . .	25
3.3.2 Single Shot Scale-invariant Face Detector . . . . .	26

3.3.3	CenterFace . . . . .	26
<b>4</b>	<b>Experiments and Results</b>	<b>27</b>
4.1	Facial Expression Databases . . . . .	27
4.2	Our Facial Expression Test Set . . . . .	28
4.3	Proposed Method . . . . .	32
4.3.1	Training Pipeline . . . . .	32
4.3.2	Architecture Experiments . . . . .	33
4.3.3	Data Preprocessing and Augmentation . . . . .	34
4.3.4	Loss Functions . . . . .	34
4.3.5	Optimizers, Learning Rate, and Transfer Learning . . . . .	35
4.3.6	Skew-normalization Techniques . . . . .	37
4.3.7	Simultaneous Prediction of Dimensional and Categorical Labels . . . . .	41
4.3.8	Scaling Factor $\alpha$ of MobileNet Architectures and Speed-Accuracy Tradeoff . . . . .	43
4.4	Evaluation on Custom Test Set . . . . .	45
4.5	Comparison with the State of the Art . . . . .	46
	<b>Discussion</b>	<b>47</b>
	<b>Conclusion</b>	<b>49</b>
	<b>Bibliography</b>	<b>51</b>
	<b>A List of Acronyms</b>	<b>57</b>
	<b>B Contents of the enclosed media</b>	<b>59</b>

---

# List of Figures

1.1	FACS Action Units . . . . .	4
1.2	Russel’s Circumplex Model . . . . .	6
1.3	Transformation of the circumplex model . . . . .	6
2.1	Single Layer Perceptron . . . . .	7
2.2	Too small vs. too large learning rate $\eta$ . . . . .	11
2.3	Multilayer Perceptron . . . . .	12
2.4	Train vs. Validation Loss Comparison . . . . .	16
2.5	Classifier Overfitting . . . . .	17
2.6	Matrix Convolution . . . . .	18
2.7	Matrix Convolution Example . . . . .	19
2.8	Residual Block . . . . .	20
2.9	Inception Module . . . . .	20
3.1	Haar-like Features . . . . .	23
3.2	Histogram of Oriented Gradients Descriptor . . . . .	24
3.3	MTCNN Facial Detector . . . . .	25
3.4	S <sup>3</sup> FD Facial Detector . . . . .	26
4.1	Annotation Application GUI . . . . .	29
4.2	Low Annotator Agreement Examples . . . . .	30
4.3	Test Set Annotation Distribution . . . . .	31
4.4	Model Architecture Visualization . . . . .	33
4.5	Training with Different Loss Functions . . . . .	35
4.6	Training with Exponential Learning Rate Decay . . . . .	36
4.7	Example of Imbalanced Learning Model . . . . .	38
4.8	Example of Weighted Learning Model . . . . .	39
4.9	Example of Imbalanced Dimensional Predictor Error . . . . .	40
4.10	Example of Weighted Dimensional Predictor Error . . . . .	40
4.11	Real-Time FER Application Screenshots . . . . .	44
4.12	More Real-Time FER Application Screenshots . . . . .	45



---

## List of Tables

2.1	Activation Functions . . . . .	9
2.2	Comparison of CNN Architectures . . . . .	21
4.1	Comparison of Facial Expression Databases . . . . .	27
4.2	Test Set Label Distribution . . . . .	28
4.3	Test Set Annotator Agreement . . . . .	30
4.4	Comparison of Architecture Training Times . . . . .	33
4.5	Sample Weights for Discrete Classifiers . . . . .	37
4.6	Metrics on Confusion Matrix . . . . .	38
4.7	Comparison of Weighted Regression Results . . . . .	41
4.8	Simultaneous Prediction Results with Sample Weights . . . . .	42
4.9	Comparison of Simultaneous Predictor with Baseline Predictors . . . . .	42
4.10	MobileNet Depth Scaling Factor $\alpha$ . . . . .	43
4.11	Model Evaluation on Custom Test Set . . . . .	45
4.12	Class Sensitivity Comparison . . . . .	46
4.13	Label Distribution in the AffectNet Database . . . . .	47
4.14	Examples of Misclassified Samples in AffectNet Database . . . . .	48





---

# Introduction

As we seek new ways to better our lives with computers, we strive to make them better understand us, our habits, our physiological and psychological state. Affective computing is a field of study exploring how machines can recognize and interpret human emotions and utilize them for more effective *human-computer interaction* (HCI). Such technologies could find many applications – augmented reality biofeedback devices, social robots, possibly even surveillance cameras [1].

Facial expression is one of many nonverbal communication channels and is an essential part of our interaction with the world, as it conveys real emotions based on facial features and their relative positions, like raised eyebrows, corners of the mouth, dilated pupils, and more [2]. When perceived by an observer, this information is processed by the amygdala, where the facial expression is recognized [3]. As computers do not have amygdalae, we turn to artificial neural networks, for they have been shown to outperform conventional algorithms in *facial expression recognition* (FER) [4].

As great as *convolutional neural networks* (CNN) are for computer vision, they are computationally expensive. Some current state-of-the-art image recognition models have upwards of a hundred million parameters and require tens of GFLOPs<sup>1</sup> for inference, making them unsuitable for both mobile devices and real-time classification.

In this work, our goal is to explore the models of emotion representation, the current state of the art in FER, and determine whether CNN architectures with smaller memory footprints and faster inference times can achieve acceptable classification accuracy to be used in real-world applications.

---

<sup>1</sup>*FLOPs = Floating-point Operations*



---

# Definition of Facial Expression

*Facial expression* is the current state of facial muscles, the movement of eyes, eyebrows, mouth, chin, cheeks, and this altogether conveys the emotion, which is the psychological state itself [2]. For us humans, FER is so instinctive and intuitive that we automatically do it at all times, as if one could not look at a face without immediately trying to determine the expression. It is our inherent nature.

There is a wide variety of human feelings, and those can be projected into facial expressions in different ways. Some are common, others could be culturally or context-specific, thus multiple observers could describe the same expression in various ways [5].

For those reasons, we need a model to classify facial expressions and their underlying emotions. Such a model should be universal, easy to understand and interpret, and easy to represent numerically. There are many models of emotion classification, some more suitable for our task than others.

## 1.1 Discrete Emotion Space

Discrete emotion space is such space in which we have distinct, separate categories. There are two widely used models to describe facial expressions – *affective categories* and the *Facial Action Coding System*.

### 1.1.1 Affective Categories

The simplest model categorizes facial expressions as a finite number of classes based on the emotions they represent. Such a model would use categories like *joy*, *fear*, *sadness*, *disgust*, *anger*, and *surprise* to classify facial expressions [5]. This approach's problem arises when an expression with no directly corresponding class (e.g., *tiredness*) is to be classified. At that moment, the model is not descriptive enough, and another class would have to be added.

### 1.1.2 Facial Action Coding System

The Facial Action Coding System (FACS) [6] is based on contractions and relaxations of individual facial muscles. Those individual muscle contractions are described by their corresponding *action units* (AUs) and intensity scores, which range from A–E with increasing intensity. The actions of muscle groups can be described using *action descriptors* (ADs). Both of those units can further be specified either as unilateral or single side only.

FACS is a complex system to describe facial expressions, but interpreting those and assigning them their corresponding emotions requires a dictionary, such as EMFACS [7].

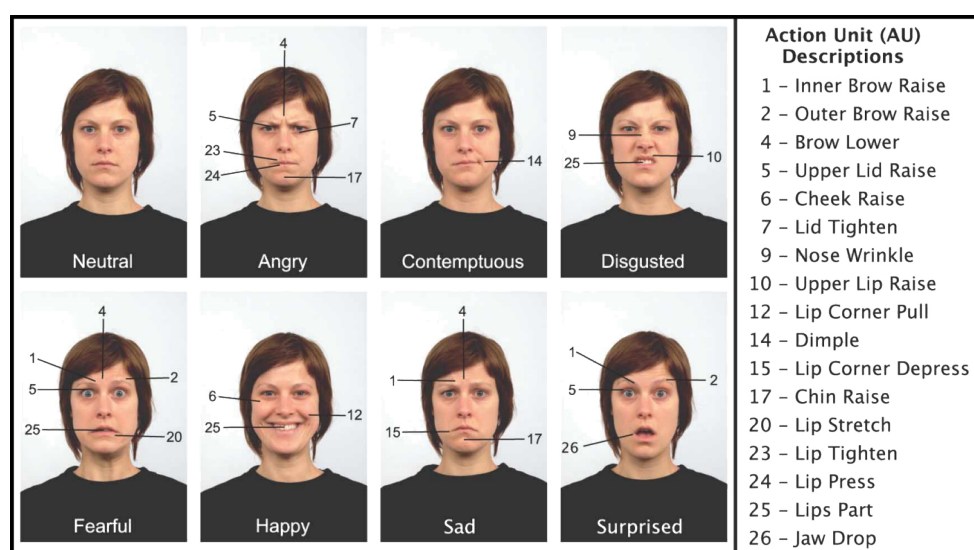


Figure 1.1: FACS Action Units with examples for basic emotions, as described in [8].

## 1.2 Continuous Emotion Space

Continuous emotion space describes the emotions as a dimensional model, in which the classification accuracy is the only limiting factor, as even very similar expressions can be distinguished by slight differences in the corresponding continuous dimensions. Examples include Russel's circumplex model [9], *Positive activation – Negative activation* (PANA) model [10], *Pleasure – Arousal – Dominance* (PAD) model [11], and Plutchik's model [12].

Both PANA and Russel's circumplex model describe the emotion space with two values, Russel's model with *valence and arousal* (VA) axes, whereas PANA's axes are rotated by 45 degrees from VA. In the PAD model, an extra dimension – dominance – is added to VA (pleasure axis is equivalent

to valence in VA). Plutchik’s model is a surface of a 3D cone, often also represented as unfolded cone, called the Plutchik’s wheel. The circular plane surface is divided into eight segments representing basic emotions adjacent based on similarity. Those basic emotions combine into emotional dyads with decreasing intensity with distance from the center.

Russel’s circumplex model seems to be the most common dimensional model used for FER database annotations [4].

### 1.2.1 Russel’s Circumplex Model

Developed by James Russell in 1980, the circumplex model describes all emotions by their valence (horizontal axis) and arousal (vertical axis). Valence represents how positive or negative the emotion is, whereas arousal determines the overall intensity. All discrete emotions can be mapped onto the circumplex model; examples are shown in Figure 1.2.

There seems to be particular disagreement whether to use only the circumplex (bound by unit distance from the origin) or the entire VA plane, with both  $v, a \in [-1; 1]$ . The AffectNet database uses only the circumplex for annotations [4], whereas the Aff-Wild database uses the entire plane [13]. Using the standard circumplex allows for conversion to a polar coordinate system. Each point can be described by angular coordinate  $\phi \in [0; 2\pi]$  rad and radial coordinate  $\rho \in [0; 1]$ , but could be counterintuitive to interpret, as simultaneous maximum valence and arousal lies at  $(\sqrt{\frac{1}{2}}; \sqrt{\frac{1}{2}})$  on the VA plane, as opposed to the intuitive  $(1; 1)$ .

*Handrich et al.* suggest that in the AffectNet database, “there are no samples with both strong valence and arousal (e.g.,  $(1; 1)$ )” [14]. This could be a misinterpretation, as AffectNet is annotated within the bounds of the circumplex, thus  $(\sqrt{\frac{1}{2}}; \sqrt{\frac{1}{2}})$  represents the strongest possible simultaneous valence and arousal.

A closer comparison between those two models could be possible using a transformation such as Shirley-Chiu’s equiareal map [15] to expand the circumplex with low distortion (Figure 1.3). Different transformations such as radial stretching could also be used to preserve the angular coordinate but would result in increased distribution density along diagonal axes.



# Neural Networks

## 2.1 Artificial Neuron

Invented by Frank Rosenblatt in 1958 [17], the mathematical model inspired by biological neurons called the *perceptron* eventually became the basic building block of all modern neural networks.

Compared to its predecessors – *Threshold Logic Units* (TLU) – proposed in 1943 by McCulloch & Pitts [18], Rosenblatt’s perceptron allows for non-uniform weights  $\mathbf{w}$  to account for different influence levels from different inputs. Furthermore, weights are no longer restricted to positive numbers, allowing some inputs to have an inhibitory effect. He also introduces the *bias* input, a constant one with its corresponding weight  $w_0$ ; this affine transformation allows for the shift of activation function  $f(\xi)$  to fit the data better.

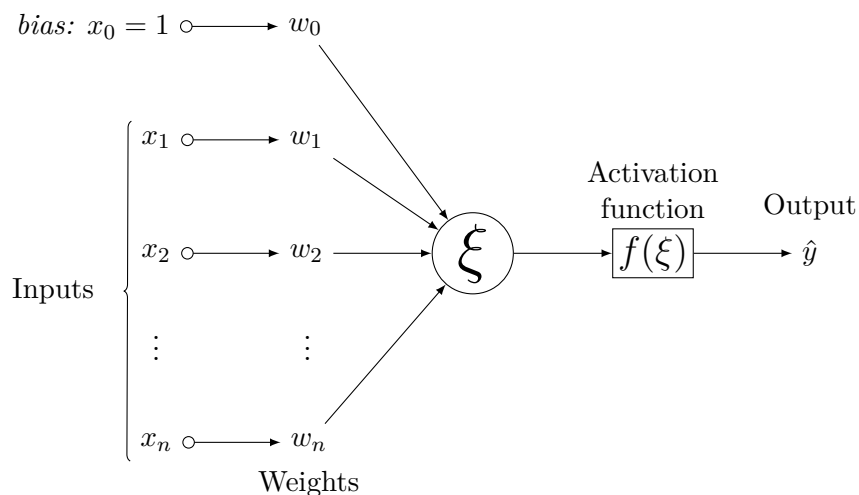


Figure 2.1: Single artificial neuron - *perceptron*.

### 2.1.1 Forward Pass

*Perceptron* is a simple binary classifier that maps its input  $\mathbf{x} = (x_1, \dots, x_n)$  of length  $n$  (to which we prepend an *intercept* or *bias*  $x_0 = 1$ , see Figure 2.1) to an output value  $\hat{y}$  using a dot product  $\sum_{i=0}^n w_i x_i$  to calculate the action potential  $\xi$ . The action potential is then fed to a threshold activation function:

$$\hat{y} = f(\xi) = \begin{cases} 1 & \xi \geq 0 \\ 0 & \xi < 0 \end{cases} = \begin{cases} 1 & \sum_{i=1}^n w_i x_i \geq -w_0 \\ 0 & \sum_{i=1}^n w_i x_i < -w_0 \end{cases}$$

### 2.1.2 Backward Pass

The process of finding the weights is called *perceptron learning*. A single cycle of updating weights for a set of all training examples is called an *epoch*.

---

**Algorithm 1: Perceptron Learning**

---

**Input :** Set of  $m$  training examples  $\{\mathbf{x}_j, y_j\}_{j=1}^m$

**for**  $j \leftarrow 1$  **to**  $m$  **do**

$error \leftarrow y_j - f(w_0 + \sum_{i=1}^n w_i x_{j,i});$

**if**  $error \neq 0$  **then**

$w_0 \leftarrow w_0 + error;$

**for**  $k \leftarrow 1$  **to**  $n$  **do**

$w_k \leftarrow w_k + error \times x_{j,k};$

**end**

**end**

**end**

**Result:** Perceptron with weights updated from a single epoch.

---

With the perceptron being a linear classifier, if the training set is linearly separable, it is guaranteed to converge within a finite number of weight adjustments. If the training set is linearly inseparable (e.g., the XOR function), the perceptron will never learn to classify all examples correctly [19].

### 2.1.3 Activation Functions

Artificial neurons can be very versatile; using different activation functions can be used for different types of predictions. Examples of selected activation functions and their derivatives can be found in Table 2.1.



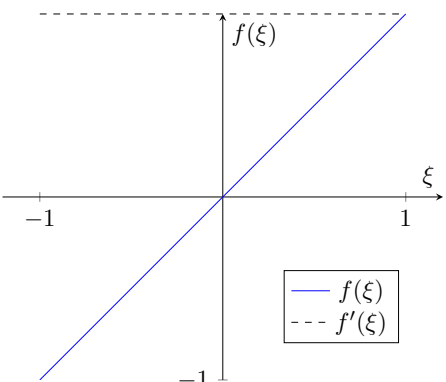
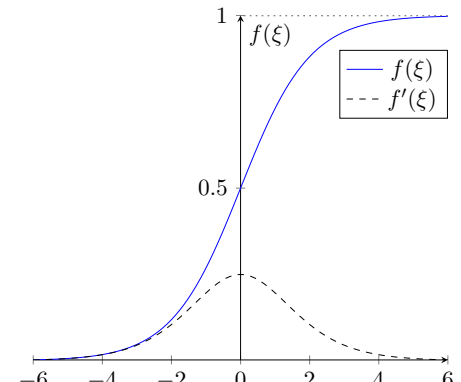
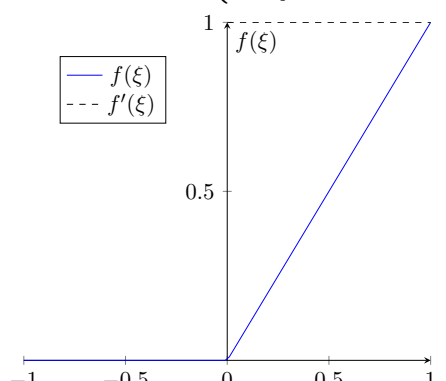
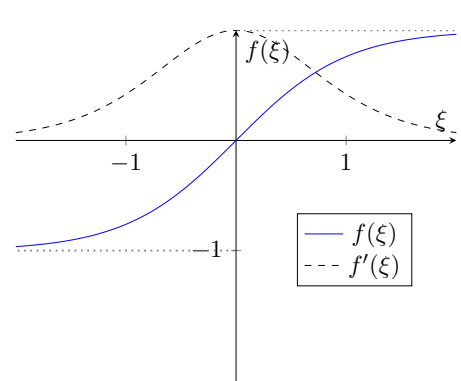
Linear Activation Function	Sigmoid
$f(\xi) = \xi = w_0 + \sum_{i=1}^n x_i w_i$ $f'(\xi) = 1$  <p>Binary classification:</p> $\hat{y} = \begin{cases} 1 & f(\xi) \geq 0 \\ 0 & f(\xi) < 0 \end{cases}$	$f(\xi) = \frac{e^\xi}{1+e^\xi}$ $f'(\xi) = f(\xi)(1 - f(\xi))$  <p>Binary classification:</p> $\hat{y} = \begin{cases} 1 & f(\xi) \geq 0.5 \\ 0 & f(\xi) < 0.5 \end{cases}$ $f(\xi) = \hat{P}(Y = 1   X = x)$
ReLU	Hyperbolic Tangent
$f(\xi) = \begin{cases} \xi & \xi \geq 0 \\ 0 & \xi < 0 \end{cases}$ $f'(\xi) = \begin{cases} 1 & \xi \geq 0 \\ 0 & \xi < 0 \end{cases}$ 	$f(\xi) = \tanh \xi$ $f'(\xi) = \text{sech}^2 \xi$ 

Table 2.1: Activation Functions

### 2.1.3.1 Binary Classification

Binary classification is a prediction of a boolean value, true or false, 1 or 0. The activation function should be both continuous and differentiable<sup>2</sup> so that the *delta rule* and *loss function* can be used in the learning process. In the ADALINE paper [20], linear activations are used, where  $f(\xi) = \xi$ . The output is still interpreted by thresholding the activation value, but the weighted sum of inputs is used to update the weights rather than a binary error.

In more modern artificial neurons, a sigmoid activation function can be found. Its output is a real-valued number between 0 and 1 and, in the context of binary classification, can be interpreted as a probability of belonging to a class (Table 2.1).

### 2.1.3.2 Regression

Prediction of a continuous variable can be achieved using the result of the activation function as output  $\hat{y} = f(\xi)$ . Choosing an appropriate activation function is necessary, as its range determines the possible predictions  $\hat{y}$ . This can be used to limit the prediction range, e.g., linear activation for  $\hat{y} \in \mathbb{R}$ , ReLU activation for  $\hat{y} \in \mathbb{R}^+$ , hyperbolic tangent for  $\hat{y} \in [-1; 1]$  (Table 2.1).

## 2.1.4 Cost Functions

In 1960, the concept of a *cost function* was introduced [20]. A continuous and differentiable activation function  $f(\xi)$  is used instead of the threshold activation function. We can define a quadratic cost function as:

$$\mathcal{C}(\mathbf{w}) = \frac{1}{m} \sum_{j=1}^m (f(w_0 + \sum_{i=1}^n w_i x_{j,i}) - y_j)^2$$

where  $f(\xi) = \xi$  is a linear activation function,  $m$  is the number of training samples, and  $n$  is the total number of weights. The set of weights  $\mathbf{w}$  is obtained by minimizing the sum of squared errors:

$$\min_w \mathcal{C}(\mathbf{w})$$

Using the *delta rule*, we descend opposite to the maximum positive slope (the *gradient*) with the weight updates  $\Delta \mathbf{w}$  moderated by the *learning rate*  $\eta$  to step towards the cost function minimum.

$$\Delta \mathbf{w} = -\frac{\partial \mathcal{C}}{\partial \mathbf{w}} = \left[ -\frac{\partial \mathcal{C}}{\partial w_0} \quad \dots \quad -\frac{\partial \mathcal{C}}{\partial w_n} \right]$$

$$\mathbf{w}_{new} = \mathbf{w} + \eta \Delta \mathbf{w} = \mathbf{w} - \eta \left( \frac{\partial \mathcal{C}}{\partial \mathbf{w}} \right)$$

---

<sup>2</sup>The derivation of activation function has to be defined for all inputs. For activations such as ReLU, the derivative in zero is either considered 0 or 1, based on implementation.

Once the cost function minimum has been found (the gradient is equal to zero), or after the norm of the weight updates decreases under a set tolerance or iteration limit is reached, we say the perceptron has *converged*. The rate at which the model converges depends on both the training set and learning rate. Too low  $\eta$  and reaching minimum takes an excessive amount of iterations, setting  $\eta$  too high and the model might never find the minimum (Figure 2.2). This problem can be alleviated by using a *learning rate decay*, which decreases the  $\eta$  in later iterations [21].

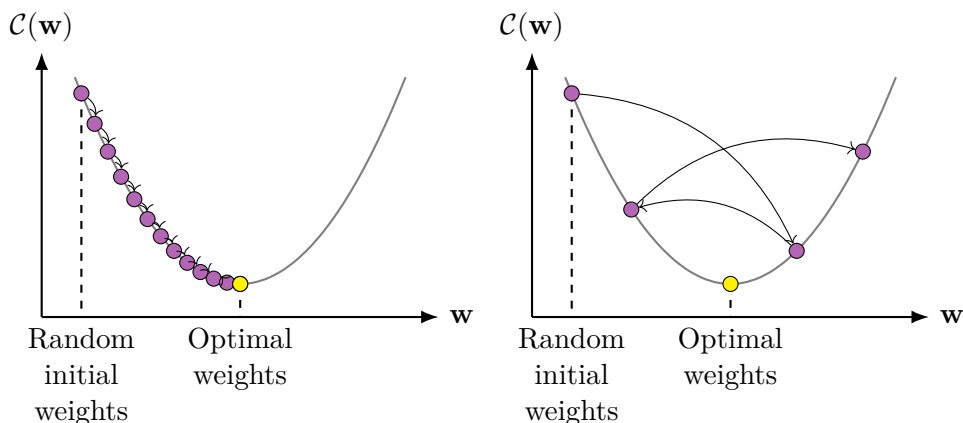


Figure 2.2: Too small vs. too large learning rate  $\eta$ .

For different prediction tasks, different activation and cost functions are used. The previous example where  $f(\xi) = \xi$  and  $\mathcal{C}(\mathbf{w}) = (Y - \hat{Y})^2$  is well suited for regression, but not for binary classification.

Most common cost function for binary classification is the *binary cross-entropy* together with sigmoid activation function (Table 2.1):

$$\mathcal{C}(\mathbf{w}) = \frac{-1}{m} \sum_{j=1}^m y_j \log(f(\xi_j)) + (1 - y_j) \log(1 - f(\xi_j))$$

## 2.2 Multilayer Perceptron

First introduced in 1967 [22], multilayer perceptrons are a class of *feedforward neural networks*, meaning the data only flows in a single direction. In 1974, Paul Werbos developed backpropagation algorithms for the training of *artificial neural networks* (ANN) [23]. We call the individual neurons *nodes*.

The input consists of  $n_0 + 1$  nodes; their activations are set to the input  $\mathbf{x} = \mathbf{x}^{(0)} = (x_1, \dots, x_n)$ , and  $x_0$  is the bias node with constant unit output. After the input layer, there is  $l > 1$  following layers, each with a variable number of nodes  $n_i$ . Except for the bias nodes, every node in the network

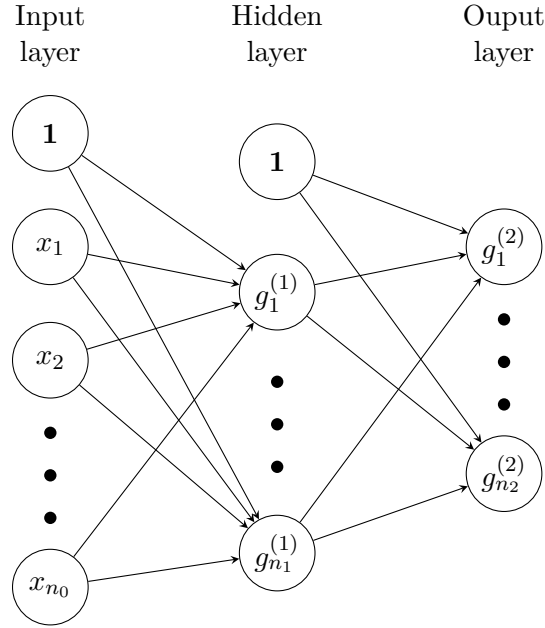


Figure 2.3: Multilayer perceptron with a single hidden layer.

takes outputs from the entire previous layer as its input and adjusts its weights (Figure 2.3). We call this a *densely connected* network.

The activation of individual nodes is defined as:

$$g_j^{(i)} = f(w_{j,0}^{(i)} + \sum_{k=1}^{n_{i-1}} x_k^{(i-1)} w_{j,k}^{(i)})$$

where  $w_{j,k}^{(i)}$  is the  $k$ -th weight of  $g_j^{(i)}$  and  $x_k^{(i-1)}$  is the  $k$ -th element from the vector of outputs  $\mathbf{x}^{(i-1)}$  from the previous layer  $i - 1$ . The output vector  $\mathbf{x}^{(i)}$  from the layer  $i$  can be defined as:

$$\mathbf{x}^{(i)} = f\left(\begin{bmatrix} 1 & x_1^{(i-1)} & \dots & x_{n_{i-1}}^{(i-1)} \end{bmatrix} \begin{bmatrix} w_{1,0}^{(i)} & w_{2,0}^{(i)} & \dots & w_{n_i,0}^{(i)} \\ w_{1,1}^{(i)} & w_{2,1}^{(i)} & \dots & w_{n_i,1}^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1,n_{i-1}}^{(i)} & w_{2,n_{i-1}}^{(i)} & \dots & w_{n_i,n_{i-1}}^{(i)} \end{bmatrix}\right)$$

where  $n_i$  is the number of nodes in layer  $i$ . The output vector from the last layer is also the vector of predicted values:

$$\mathbf{x}^{(l)} = \hat{\mathbf{y}}$$

Neural networks are great for many things, they can have multiple outputs, allowing for multiclass classification, and most importantly, they can solve

linearly inseparable problems (e.g., the XOR function), as the consequent layers transform the data in a way that makes it separable.

Training of neural networks is a complex task, large training sets are used to update many weights throughout the network, and full batch training with cost function calculating the error on the entire training set is very memory-expensive. Full batch training is usually avoided by splitting the test set into fragments called *mini-batches* and calculating the individual prediction errors. This individual error is called *loss*, and the *loss function*  $\mathcal{L}$  is used to update weights after each mini-batch.

The mini-batch size can be as small as a single training example – this is called *online training* – and is usually much slower compared to larger mini-batch sizes and sometimes leads to unstable convergence [24]. Large mini-batches are memory-expensive and, after a certain point, can lead to worse generalization [25]. For those reasons, it is essential to select an adequate mini-batch size hyperparameter.

The cost function  $\mathcal{C}$  is the average  $\mathcal{L}$  across the training set. Training the network on the entire training set single time is called an *epoch*, and multiple epochs are often needed for the model to converge, as the step size needs to be moderated (Figure 2.2).

### 2.2.1 Multiclass Classification

Neural networks improve on the binary classification ability and can classify multiple classes by representing  $n$  classes as  $n$  neurons in the output layer of the network, with the *softmax* activation function:

$$f_k(\xi) = \frac{e^{\xi_k}}{\sum_{i=1}^n e^{\xi_i}}$$

for  $k$ -th output neuron, representing the probability  $\hat{P}(Y = k|X = \mathbf{x})$  of the network input  $\mathbf{x}$  belonging to class  $k$ . The class prediction  $\hat{Y}$  is then determined as the most probable class:

$$\hat{Y} = \arg \max \hat{\mathbf{y}} = \arg \max ([f_1(\xi) \quad \dots \quad f_n(\xi)])$$

The loss used for multiclass classification is most often *categorical cross-entropy*, which is defined as:

$$\mathcal{L}(Y, \hat{Y}) = - \sum_{i=1}^n y_i \log \hat{y}_i$$

### 2.2.2 Multivariable Regression

Similarly, multiple output neurons can be used for multivariable regression. The activation function used determines the range of possible predictions. If,

for example, our task was to predict spatial dimension (e.g., length), using ReLU as the activation function limits the range of possible predictions to  $f(\xi) \in \mathbb{R}^+$ , as negative length is never a valid prediction.

The vector of  $n$  predicted values  $\hat{\mathbf{Y}}$  is defined as:

$$\hat{\mathbf{Y}} = \hat{\mathbf{y}} = [f_1(\xi) \quad \dots \quad f_n(\xi)]$$

When the prediction range cannot be easily limited by the choice of the activation function, a linear transformation can be used to interpret the result, as can be seen, for example in [14]. E.g., if the range of a prediction was to be limited to  $\hat{Y} \in [10; 20]$ , a sigmoid activation function can be used, and the final prediction  $\hat{Y}$  is obtained as:

$$\hat{Y} = 10\hat{y} + 10$$

Different loss functions can be used, commonly used are *mean average error* (MAE), *mean square error* (MSE), *root mean square error* (RMSE), and *root mean square log error* (RMSLE). Each of them has slightly different properties.

$$\mathcal{L}_{MAE}(\mathbf{Y}, \hat{\mathbf{Y}}) = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

$$\mathcal{L}_{MSE}(\mathbf{Y}, \hat{\mathbf{Y}}) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

$$\mathcal{L}_{RMSE}(\mathbf{Y}, \hat{\mathbf{Y}}) = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

$$\mathcal{L}_{RMSLE}(\mathbf{Y}, \hat{\mathbf{Y}}) = \sqrt{\frac{1}{n} \sum_{i=1}^n (\log(y_i + 1) - \log(\hat{y}_i + 1))^2}$$

*MAE* is a metric that represents the average error and does not penalize outliers in the set. *MSE*, on the other hand, penalizes outliers quite heavily. *RMSE* is better suited for scenarios in which the errors are quite large, as it scales their values down but still penalizes outliers in the set. It also represents the *standard deviation*  $\sigma$  of *residuals*<sup>3</sup>. *RMSLE* is a metric similar to RMSE but does not penalize the large outlier errors.

### 2.2.3 Backpropagation

To train a neural network, we minimize the network's cost function for the set of weights  $\theta$ .

$$\min_{\theta} \left( \frac{1}{m} \sum_{j=1}^m \mathcal{L}(Y_j, \hat{Y}_j) \right)$$

---

<sup>3</sup>*Residual* = prediction error

In full batch training,  $m$  is equal to the number of training samples. In mini-batch training,  $m$  is the number of samples for which the weights are updated at a single iteration. This number is commonly referred to as the *mini-batch size* or *batch size*.

The average mini-batch loss (mini-batch cost) is calculated and used to find the gradients and update weights after every mini-batch:

$$\theta = \theta + \eta \nabla \theta = \theta - \eta \left[ \frac{\partial \mathcal{C}}{\partial w_{1,0}^{(1)}} \quad \cdots \quad \frac{\partial \mathcal{C}}{\partial w_{n_l, n_l-1}^{(l)}} \right]$$

Compared to single perceptron learning, reaching the global minimum is very difficult, as the descent can get stuck in local minima or at saddle points. For those reasons, additional techniques such as *momentum* can potentially improve both the learning speed and convergence by accelerating the SGD in the relevant direction and dampening oscillations [26]. It is defined as:

$$\theta = \theta + \eta \nabla \theta^i + \gamma \nabla \theta^{i-1}$$

where  $\gamma$  is the *momentum rate*, and  $\nabla \theta^{i-1}$  is the previous gradient. In practice, using just momentum is not ideal, as the accumulated descent inertia can sometimes make the descent overshoot and does not slow down quickly enough. More complex sets of rules for the descent have been invented, and different optimizations are suitable for different problems. Those sets of optimization rules are commonly referred to as *optimizers*. Commonly used optimizers are *Nesterov accelerated gradients* (NAG), *Adagrad*, *RMSprop*, and *Adam*.

NAG seeks to improve the momentum overshooting of simple momentum term by first calculating the current gradient, then calculating a second gradient based on a momentum-only update, and then correcting the descent direction on the combined gradients. Adagrad gives individual weights their individual learning rates so that infrequently updated weights receive more significant updates, which improves the learning on sparse data. RMSprop improves on Adagrad by instead of accumulating past gradients using an exponential moving average of squared gradients. Adam is basically RMSprop with added momentum [26].

Although Adam has been shown to converge significantly faster at the beginning of the training [27], in the later stages, SGD can take over with better generalization, and hybrid training strategies could benefit both speed and generalization [28].

#### 2.2.4 Common Problems of ANN

One of the common problems of artificial neural networks is *model overfitting*. Many epochs are often needed for the model to converge, and after a certain point, the model starts to remember the training samples instead of generalizing the problem and performs poorly on other data.

To detect when this starts to happen, we split the training data into two separate sets, the training and the *validation set*. The model is never trained on the validation data, but both training and validation losses are evaluated and monitored. Once the validation loss ceases to improve, we consider the model converged. This is called *early stopping*, as it stops the training when the validation accuracy does not improve for the last  $n$  epochs. If the training is not stopped at this point, the training accuracy continues to improve, overfitting the model, and the validation accuracy degrades (Figure 2.4).

The validation set is also used to fine-tune hyperparameters of the network, as improving the validation accuracy most often translates to better *test accuracy*, as both of those sets are considered a representative sample of the real problem. The difference between validation and test set is that, unlike the validation set, the test set is never used to fine-tune the hyperparameters to avoid any possible bias and be a representative evaluation metric of the resulting model's accuracy. It is only used to evaluate the single final model with the best validation accuracy.

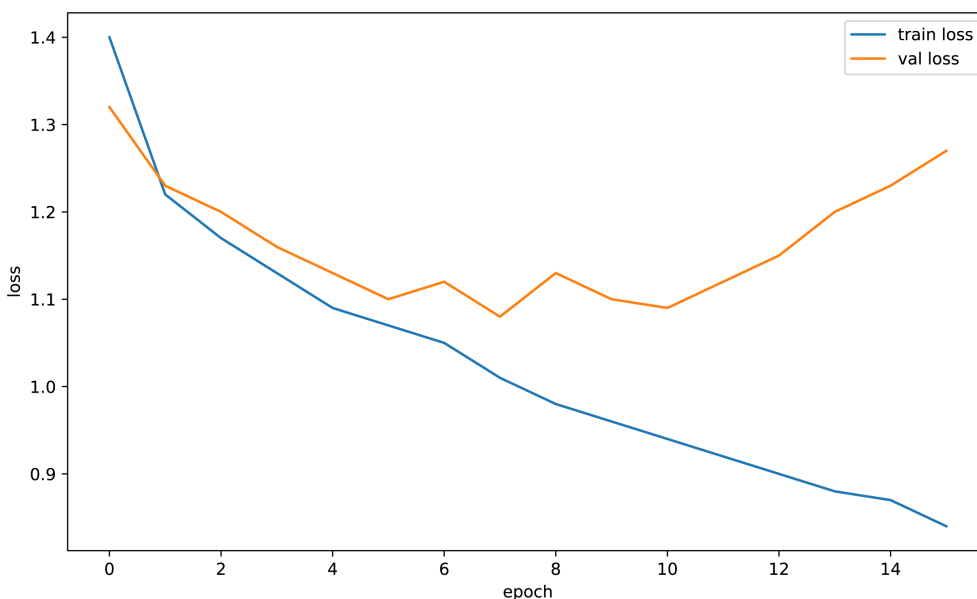


Figure 2.4: Train vs. validation loss comparison.

Another thing to look out for are the *exploding gradients*, a problem with computers' numerical precision. During the gradient descent and backpropagation, the error gradients can accumulate due to large weights in the network and cause an *overflow* in variables, resulting in *not-a-number* (NaN) values and invalid computation. Such problems can be prevented by using *weight regularization* techniques, which penalize the network's increasing weights by adding a regularization term to the loss function.



$L_p$  regularization is defined for the  $p$ -norm as:

$$\|\mathbf{w}\|_p = (|w_1|^p + \dots + |w_n|^p)^{\frac{1}{p}}$$

$$\mathcal{L}(\mathbf{Y}, \hat{\mathbf{Y}}) = Error(\mathbf{Y}, \hat{\mathbf{Y}}) + \lambda \|\mathbf{w}\|_p$$

where  $\lambda$  is the regularization rate and  $Error(\mathbf{Y}, \hat{\mathbf{Y}})$  is a loss function, such as MSE. Commonly used regularizations are L1 and L2, where L1 is commonly used where sparse weights are required, as it promotes zero weights, and L2 more heavily penalizes larger weights [29]. Regularization techniques also improve the model's ability to generalize and reduce overfitting, as overfitting requires more complex weights (Figure 2.5).

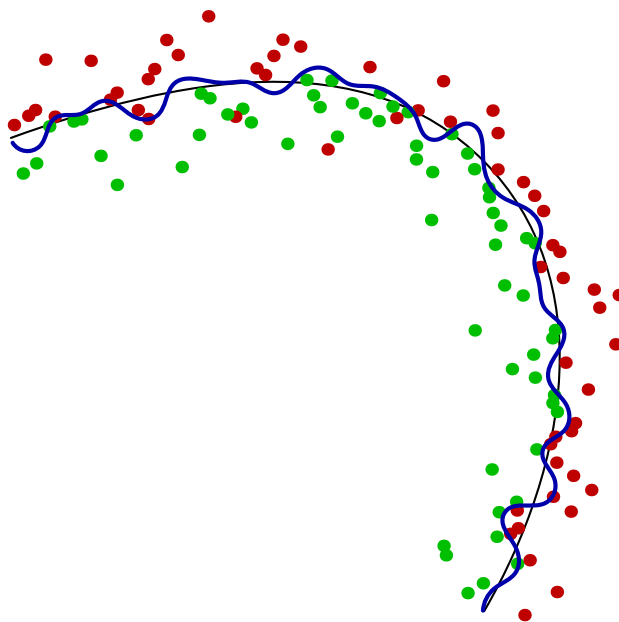


Figure 2.5: The classifier overfitting example is represented by the blue curve.

Other techniques also aim to improve the overfitting problem, commonly used is the *dropout*, which randomly deactivates some nodes during the training process so that no neurons depend excessively on individual inputs. It was even shown to outperform L2 regularization in more complex networks [30].

## 2.3 Convolutional Neural Networks

The motivation behind convolutional neural networks is traditional ANN's unsuitability for large input data, e.g., images. For example, a network accepting  $256 \times 256$  RGB images as input would require 196,608 nodes in the input layer. Each following layer with the same size would require a total of  $196,608^2 = 38,654,705,664$  weights to process the image.

Based on D. H. Hubel and T. N. Wiesel's works, who studied the visual cortices of cats [31], monkeys [32], and their receptive fields, K. Fukushima came up with a network architecture mimicking the hierarchy of simple and complex cells [33]. A *simple cell* works similar to a *Gabor filter* – it responds to an oriented gradient, working like an oriented edge detector, with specific inhibitory and excitatory regions in the *receptive field*. *Complex cells* have larger receptive fields and work as spatially invariant pattern detectors, meaning their receptive fields can not be divided into specific inhibitory or excitatory regions.

This behavior of spatially invariant pattern detectors consisting of inhibitory/excitatory regions can be achieved with *matrix convolution*, which applies a filter called *kernel* to an image, and produces output based on a weighted sum of a window sliding over the image. Convolution is defined as:

$$(\mathbb{I} * \mathbb{K})_{i,j} = \sum_{a=0}^{o-1} \sum_{b=0}^{p-1} \mathbb{I}_{i+a,j+b} \cdot \mathbb{K}_{a,b}$$

where  $\mathbb{I}$  is the image with dimensions  $m, n$ ,  $\mathbb{K}$  is the kernel with dimensions  $o, p$ , and  $(\mathbb{I} * \mathbb{K})_{i,j}$  is element at position  $i, j$  in the resulting matrix  $(\mathbb{I} * \mathbb{K})$  with dimensions  $q, r$ ; where  $q = m - 2\lfloor \frac{o}{2} \rfloor$  and  $r = n - 2\lfloor \frac{p}{2} \rfloor$ .

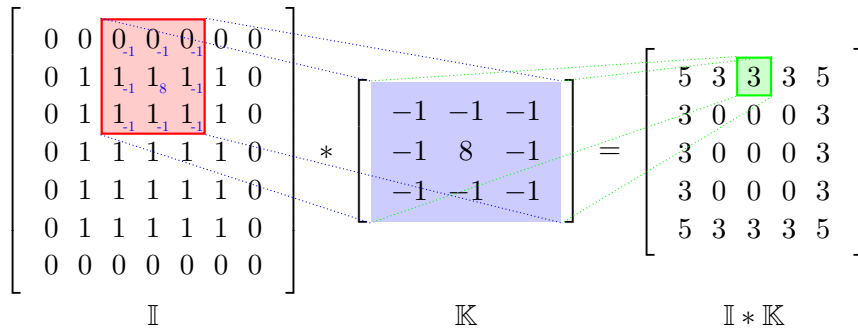


Figure 2.6: Matrix convolution example with edge-detection kernel.

As seen in Figure 2.6, for border values of  $\mathbb{I}$ , the convolution is undefined, as part of the kernel would be out of the image. Different implementations of the matrix convolution might pad the image  $\mathbb{I}$  with zeros or with border values to solve this problem. Another approach is to wrap around the image and take the pixel values from the opposite side.



$$\mathbb{I} \quad \mathbb{K} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad \mathbb{I} * \mathbb{K}$$

Figure 2.7: Matrix convolution example with edge-detection kernel.

Fukushima’s idea was that the consequent layers learn to detect higher-level complex features from the previous lower-level simpler ones. An important step forward came in a 1998 paper, in which LeCun et al. combined Fukushima’s *neocognitron* [33] with *pooling layers* [34] and used the gradient descent to optimize both convolution filters and weights [35]. Their methods eliminated the need for hand-crafting of the convolution filters for pre-determined features, but the training was still very difficult for then-current hardware.

In the 2000s, the development of *graphical processing units* (GPU) capable of massively parallel computation started to revolutionize the artificial intelligence research, as GPU implementations of neural networks were up to 60 times faster than their CPU counterparts by the year 2011 and have reduced the training times for larger networks from months to days [36].

In 2012, Alex Krizhevsky won the *ImageNet* competition, an annual computer vision contest in multiclass object recognition, in which their GPU implementation of CNN broke all previous records with only 16 % error rate (over 10 % better than runner-up) [37], and with it brought lots of attention to neural networks.

With the increasing speed of GPUs, training larger networks was the primary pursuit for better accuracy<sup>4</sup>. One example of such large networks were the *VGGNets* – smaller *VGG16* and larger *VGG19*, with 16 and 19 weight layers respectively. They experimented with new techniques as different fil-

<sup>4</sup>There is no universal accuracy metric for neural networks, as it is highly dependent on the exact problem. In most modern literature, the network performances are compared using the ImageNet database classification accuracies [38]. We will stick to this convention.

ter sizes [39]. Going deeper than this was becoming an issue, as exploding/vanishing gradients prevented the models from ever properly converging.

This was solved by a new approach introduced in 2015 *ResNet*, where *residual blocks* were used to address the vanishing/exploding gradients problem. Residual blocks work as shortcuts, skipping few layers in the network and applying its output as a second input to the destination layer. Residual blocks helped ResNets converge better and won the 2015 ImageNet competition, with an 8x deeper network than VGG19 [40].

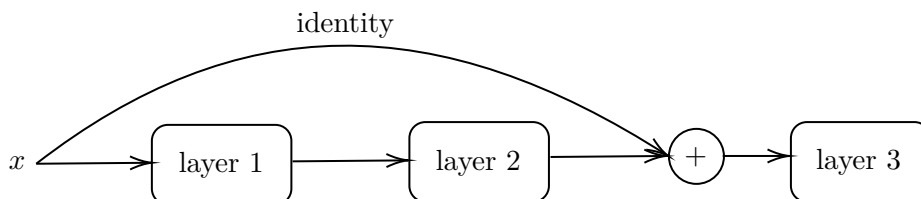


Figure 2.8: ResNet residual block example.

The *Inception* family of CNN architectures took a different approach than ResNets, and instead of going deeper, it went wider. Compared to other architectures, where a single filter size was usually used in a single layer, Inception uses multiple convolution filter sizes and filter concatenation layers. The main idea is that different filter sizes might be suitable for different inputs, which the network learns [41].

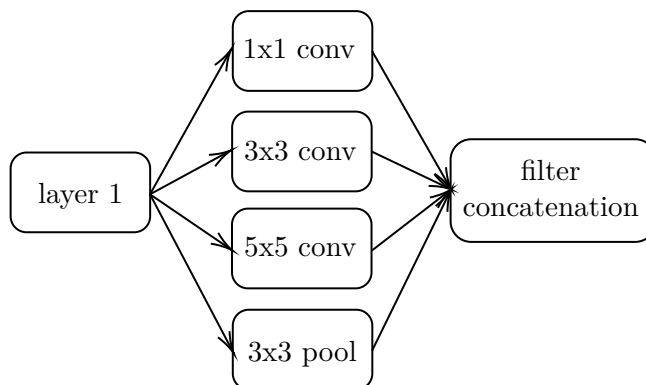


Figure 2.9: Inception module example.

*InceptionV2* and *InceptionV3* were introduced in the same paper and improved on the previous Inception by factorizing larger convolutions like  $5 \times 5$  convolution into two smaller  $3 \times 3$  convolutions [42]. Furthermore, separating a  $n \times n$  convolution into  $1 \times n$  and  $n \times 1$  leads to even greater speedup. Multiple different Inception modules were used as basic building blocks of the entire network. A year later, *InceptionV4* introduced *reduction blocks*, a specialized type of block designed to change the dimensions of its input [43].

In the very same paper, another architecture was also introduced. The *Inception-ResNet* combined two ideas, going both deeper and wider with the convolution layers. The residual connections within the Inception blocks work by adding the block’s input to it’s output before the activation function [43].

There are many different architectures, and each is interesting and unique in different ways. One more notable mention because of its innovative approach is the family of *NASNets*, where the researchers used reinforcement learning instead of architecture engineering to find the best building blocks for a new CNN architecture. Their model outperformed all other state-of-the-art human-invented architectures while reducing the model’s computational demand by 28 % compared to the runner-up [44].

As CNN architectures grew in pursuit of the best possible accuracy, their computational demand grew as well, making them unsuitable for lower-powered mobile or embedded systems. The family of *MobileNets* is specifically designed for those devices – reducing computational demand while preserving as much accuracy possible [45]. Those networks are easily scalable to allow for further latency/accuracy tradeoff tuning. *MobileNetV1* is built with the computationally efficient *depthwise separable convolutions* introduced in Inception networks. *MobileNetV2* uses linear bottlenecks and inverted residual blocks to further improve the accuracy with fewer total network parameters [46].

Model	Params	Size	Accuracy <sup>5</sup>	Depth	GFLOP
AlexNet	60 M	233 MB	0.633 / 0.846	8	0.727
VGG16	138 M	528 MB	0.713 / 0.901	23	16.0
VGG19	143 M	549 MB	0.713 / 0.900	26	20.0
ResNet50	25 M	98 MB	0.749 / 0.921	50	4.0
ResNet152	60 M	232 MB	0.766 / 0.931	152	11.0
InceptionV3	23 M	92 MB	0.779 / 0.937	159	6.0
InceptionResNetV2	56 M	215 MB	0.803 / 0.953	572	13.0
MobileNetV1	4.2 M	16 MB	0.704 / 0.895	88	0.579
MobileNetV2	3.5 M	14 MB	0.713 / 0.901	88	0.3
MobileNetV2 [1.4]	6.9 M	27 MB	0.747 / 0.919	156	0.585
NASNetLarge	89 M	343 MB	0.825 / 0.960	– <sup>6</sup>	23.8
NASNetMobile	5.3 M	23 MB	0.744 / 0.919	– <sup>6</sup>	0.75
YOLOv2	51 M	197 MB	– <sup>7</sup>	10	31.0

Table 2.2: Comparison of selected CNN architectures. [38, 43, 44, 45, 46, 47]

<sup>5</sup>Accuracy values for Top-1 / Top-5 on the ImageNet database.

<sup>6</sup>Given the complexity of NASNets, depth is no longer an applicable metric.

<sup>7</sup>YOLOv2 is an object detection network.



---

## Facial Detection

*Facial detection* algorithms detect a presence of a human face in images. It is closely related to *facial recognition*, which identifies the individual or matches faces in different images. Early development goes as far as the 1970s when T. Kanade demonstrated a fully automated system for *facial feature* localization [48]. In the 1990s, this technology started making its way into law enforcement, using automated identification systems and digital biometric databases [1].

### 3.1 Viola-Jones Object Detection Framework

In 2001, Paul Viola and Michael Jones proposed an object detection framework based on *Haar-like features* [49]. A Haar-like feature consists of several rectangular regions, in which the pixel intensities are summed, and a difference between the regions is then calculated.

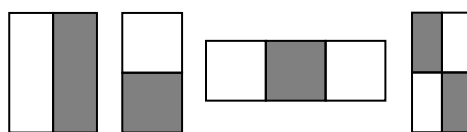


Figure 3.1: Examples of Haar-like features.

They used *integral images* for fast calculation speed. Integral image is a *summed-area table* [50] in which calculation of any Haar-like feature can be achieved in constant time by looking up precomputed sums of different areas of the image and combining them into the desired feature.

For facial detection, the Haar-like features are tailored for common properties of human faces - darker eyes, lighter cheeks, lighter nose bridge, its relative position to the eyes, and more. An *AdaBoost* learning algorithm was used to find 200 such features that achieved a 95 % detection rate, with a false positive rate of  $\frac{1}{14084}$ . To further improve speed and accuracy, the *cas-*

*cade classifier* works as a *degenerate tree* – a positive result from each filter is required to move on to the next one, a negative result immediately rejects the sub-window. The cascade is designed so that most negatives are discarded very early, as the vast majority of sub-windows do not contain a face. The complete face detection cascade consists of 38 stages with more than 6000 features [49].

This technique achieves reasonable accuracy with low computational demands and can be used in real-time applications on embedded devices but struggles with different scales and faces not facing the camera [51].

### 3.2 Histogram of Oriented Gradients and Bag of Features

A different approach to facial detection uses *histograms of oriented gradients* (HOG) and *bags of features*. It aims to be better at detecting partially occluded faces and faces with poor illumination [52].

HOG descriptor works by separating the described region of the image into cells. Oriented gradients are calculated for each cell, and their magnitudes are normalized to compensate for the local illumination. That is done by combining multiple neighboring cells and dividing their histogram values by their L2-norm.

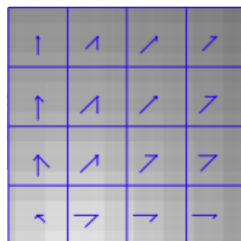


Figure 3.2: Example of histogram of oriented gradients descriptor from [52].

The HOG descriptors are then transformed into  $n$  clusters – those will become  $n$  different codewords. Each image can then be described as a sparse vector of occurrences of those codewords, or a histogram of the features [53].

Those *bags* of features can be used to train a classifier, such as the *Support Vector Machine*. This approach achieved  $> 99\%$  classification accuracy using a codebook size of  $n = 10$  and  $85\%$  with a codebook size of  $n = 100$  [52].

### 3.3 CNN-Based Approaches

Current state-of-the-art facial detectors are based around CNN; examples include the *Multitask Cascaded CNN* (MTCNN) [54], *Single Shot Scale-invariant Face Detector* (S<sup>3</sup>FD) [55], and *CenterFace* [56].



### 3.3.1 Multitask Cascaded CNN

MTCNN sets to improve facial detection and *facial alignment*<sup>8</sup> on poorly illuminated and occluded faces, where Viola-Jones' algorithm accuracy degrades with visual variations of human faces [54]. It achieves this by implementing a cascade of three convolutional networks. The image is resized for each network in advance, building an *image pyramid*. The first one, called the *Proposal Network* (P-Net), finds candidate windows with possible faces. Highly overlapped windows are merged using *non-maximum suppression* (NMS). The second network, called the *Refine Network* (R-Net), further narrows the number of candidate windows so that the last *Output Network* (O-Net) only goes through very high probability windows and detects both the face and *facial landmarks*.

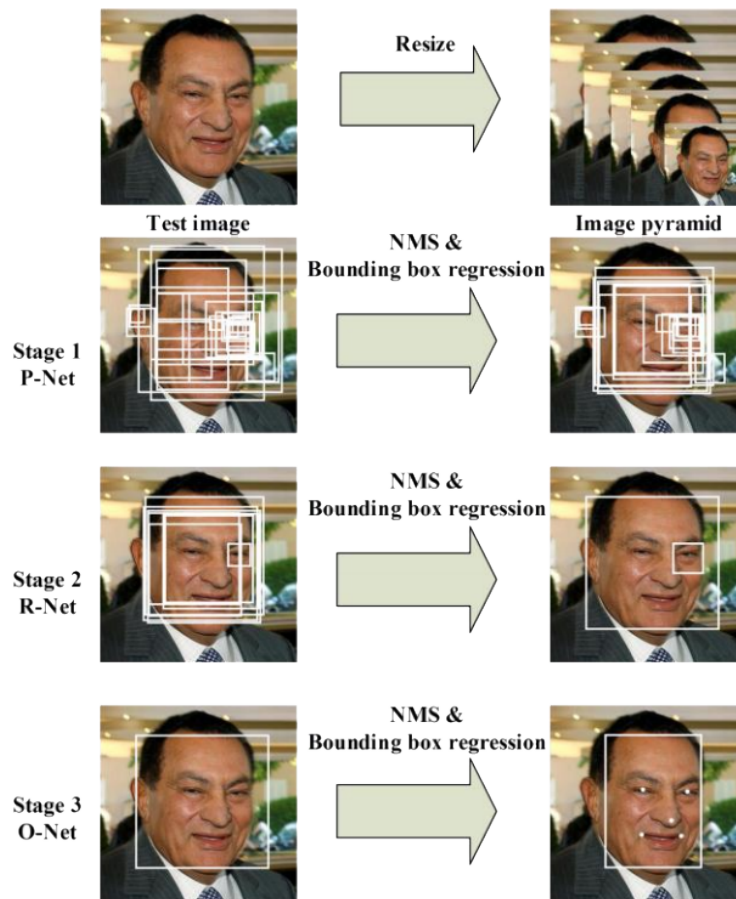


Figure 3.3: An illustration of the MTCNN facial detection pipeline from [54]

<sup>8</sup>Detection of facial landmarks such as mouth corners and eyes.

### 3.3.2 Single Shot Scale-invariant Face Detector

S<sup>3</sup>FD is based on the *Single Shot Detector* (SSD), which separates the image into a set of predetermined default bounding boxes (anchors), and those boxes are adjusted and shifted based on the image shapes inside. It then generates confidence scores for the presence of different object categories (in this case, *face* and *no-face*) and further adjusts the bounding boxes to better match the object’s shape [57].

To develop a robust, scale-invariant facial detector, the S<sup>3</sup>FD develops a CNN architecture based on the VGG16 with extra convolutional layers, gradually scaling the image to detect adequate features at all scales. Additional techniques such as the *max-out background label* allow for better classification of small faces, as less than  $\frac{1}{500}$  of anchors belong to actual faces due to the scale of detection for small faces [55].

Compared to MTCNN, S<sup>3</sup>FD achieves greater detection accuracies but does not perform a facial alignment.

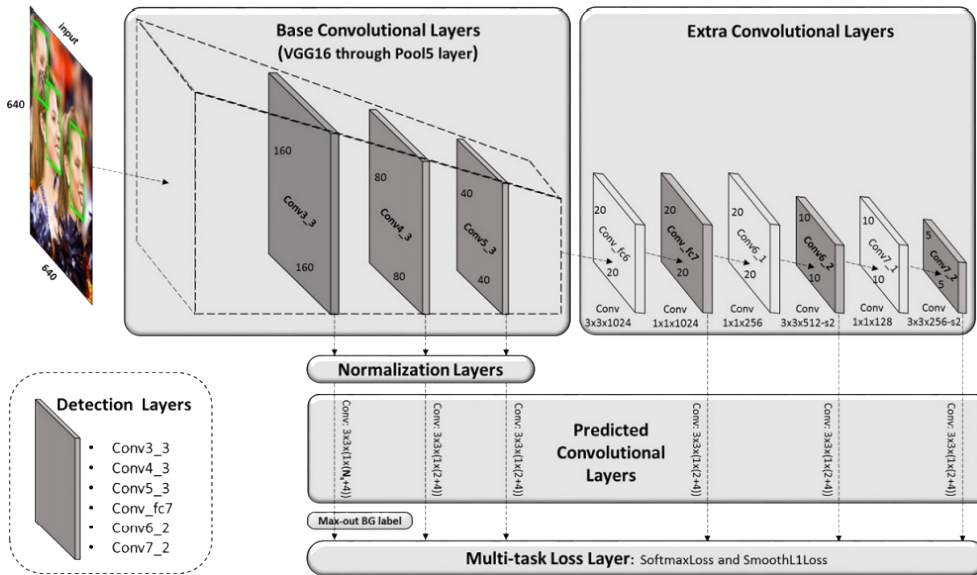


Figure 3.4: An illustration of the S<sup>3</sup>FD architecture from [55].

### 3.3.3 CenterFace

CenterFace uses a *feature pyramid network* [58] for an anchor-free design, and compared to previous approaches, uses a single efficient MobileNetV2 network as a backbone to achieve superior speed and accuracy to that of the resource-heavy VGG16-based (Table 2.2) S<sup>3</sup>FD on the *WIDER FACE* database while being able to run in real-time on single CPU core [56].

The code has been made available on the author’s GitHub repository [59].

## Experiments and Results

### 4.1 Facial Expression Databases

When considering which database to use in this work, the key characteristics taken into consideration were the number of different subjects, the total number of samples, the resolution of the images, and the affective model used for the annotations.

Database	# Subjects	# Samples	Labels	Note
FER-2013 [60]	~35,887	35,887	7 categories	48×48px, wild setting
RAF-DB [61]	~29,672	29,672	7 basic + 12 compound categories	100×100px, wild setting
RaFD [8]	67	8,040	8 categories	1024×681px, posed
Aff-Wild2 [62]	~550	~2,800,000	7 categories + VA + FACS AU	avg. 1030×630, videos
AffectNet [4]	~450,000	~450,000	8 categories + VA	avg. 425×425, wild setting

Table 4.1: Comparison of selected FER databases [4, 8, 60, 61, 62].

The FER-2013 database offers reasonable subject variance but at very low resolution. RAF-DB offers a similar number of samples with better quality and provides labels for *compound emotions* (e.g., “*happily surprised*” . . .) as well as landmark locations, and was annotated by about 40 different annotators [61].

The RaFD database offers high-quality pictures with multiple samples of each emotion taken from different angles. However, the models were primarily

Caucasian, and the pictures were taken in a very controlled environment, and posed samples with low subject variance might not generalize well for *in-the-wild* recognition with various environmental conditions and illumination.

Aff-Wild2 database is very different from other works, as it contains videos with labels for each frame, as well as audio, and is labeled with both categorical and dimensional models, as well as FACS action units. *Handrich et al.* show that simultaneous training on both VA and categorical labels improves the model’s accuracy [14]. It is possible that adding action units might increase the accuracy further. Using videos also allows for the use of *recurrent neural networks* (RNN), where the data input is a temporal sequence of multiple consequent images, to improve the prediction based on visual and audio context. The authors also experimented with such architectures [62].

As our aim is a lightweight classifier based on a single image, using Aff-Wild2 would require extensive preprocessing (facial detection and alignment), and given the nature of the database, subject variance is very low and might lead to overfitting when using individual frames.

AffectNet is our database of choice for this work. It was created by querying different search engines in multiple languages and crawling the web for images, offers very high subject variance as well as high-resolution and both categorical and dimensional labels [4].

## 4.2 Our Facial Expression Test Set

To further validate the results achieved on the AffectNet database, we collect our own modest test set. This test set had to be collected similarly to the aforementioned AffectNet – by querying search engines due to pandemic restrictions. The queries were in English using keywords like “*happy woman, surprised man...*”. The queries were also limited to results published in the last year to lower the chance of finding pictures that are already contained in the AffectNet database.

The images were manually sorted, contain no watermarks or occluded faces, and are manually cropped to facial regions. The resulting set contains 160 images, 82 males, 78 females; 131 of those subjects were labeled as an adult, 130 were of caucasoid origin, 15 of mongoloid origin, and 15 of negroid origin [63].

Neutral	Sad	Happy	Angry	Surprise	Fear	Contempt	Disgust
36	28	23	22	19	14	11	7

Table 4.2: Distribution of expression labels on our test set.

Five different annotators were instructed on how to annotate both dimensional and categorical labels. A GUI application was developed to display individual pictures for annotation one at a time (Figure 4.1). Figure 1.2 was

## 4.2. Our Facial Expression Test Set

displayed with two axial sliders and a marker representing the current annotation on the circumplex model for the dimensional labels. When in doubt, annotators were to use the marked reference points and consider the valence and activation. The choice of one label did not limit the second label, as was the case in the AffectNet database, e.g., choosing the “*Happy*” category did not limit valence range to positive values, but annotators were asked to stay within the circumplex unit bound to stay consistent with the AffectNet database.

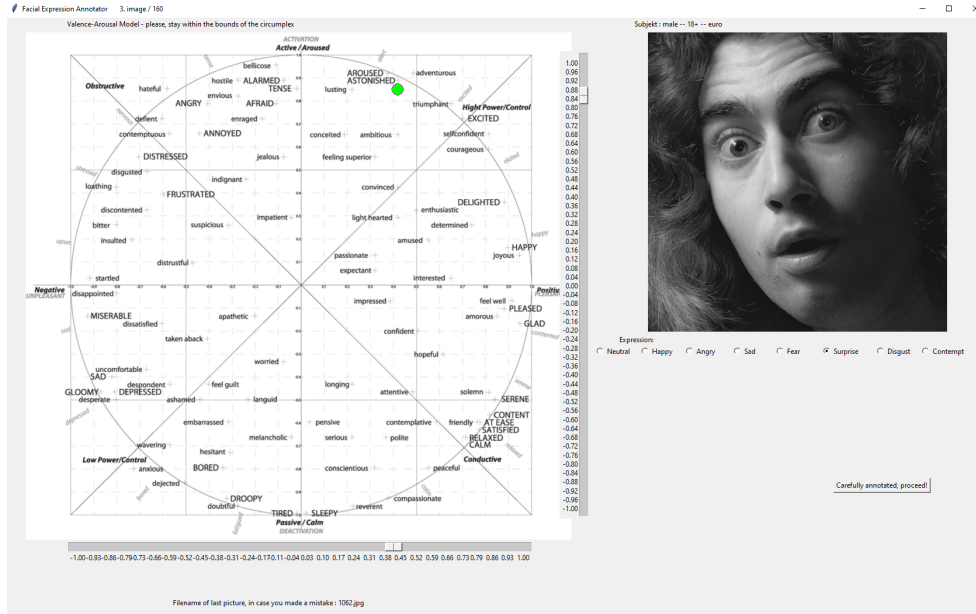


Figure 4.1: Illustration of the annotation application GUI.

Four different metrics were used to evaluate the annotator agreement – accuracy, RMSE, defined in Section 2.2.2, *Concordance Correlation Coefficient* (CCC), defined in [64], and *Sign Agreement Metric* (SAGR), defined in [4].

$$SAGR = \frac{1}{n} \sum_{i=1}^n \delta(\text{sign}(\hat{\theta}_i), \text{sign}(\theta_i))$$

$$\delta(a, b) = \begin{cases} 1 & a = b \\ 0 & a \neq b \end{cases}$$

The SAGR metric is relevant to assess whether the general directions of the annotations agree (e.g., both predict positive valence, despite the actual values).  $\delta(a, b)$  is the *Kronecker delta function*, and  $\theta_i, \hat{\theta}_i$  are the individual compared labels. The CCC metric is defined as:

$$\rho_c = \frac{2\rho\sigma_x\sigma_y}{\sigma_x^2 + \sigma_y^2 + (\mu_x - \mu_y)^2}$$

#### 4. EXPERIMENTS AND RESULTS

---

where  $\mu_x, \mu_y$  are means of  $x, y$ ;  $\sigma_x^2, \sigma_y^2$  are their variances and  $\rho$  is their correlation coefficient.

Label	Metric	Ours	AffectNet
Valence	RMSE	0.455	0.340
	SAGR	0.768	0.815
	CCC	0.630	0.821
Arousal	RMSE	0.528	0.362
	SAGR	0.680	0.667
	CCC	0.449	0.551
Expression	Accuracy	60.3 %	60.7 %

Table 4.3: Evaluation of annotator agreement.

The AffectNet database evaluates annotator agreement on 36,000 samples labeled by two annotators. We report the average of those metrics between five annotators. Our annotators also reported they often felt like the expression could not be described accurately with the categorical model or that the expression could not be annotated accurately without knowing the context. Interestingly, some faces with very high agreement on the discrete label still had a considerable disagreement about the dimensional labels.



Figure 4.2: Selected images with low annotator agreement rate.

Median values were used to evaluate the discrete labels due to outlier values that would otherwise skew the average. For the discrete labels, the

majority classes ( $> 3$  votes) were used. In few cases with no clear majority (e.g., 3:2, 2:2:1...), three additional annotators decided the final label.

Compared to AffectNet, we did not limit dimensional labels to regions based on the selected discrete expression. That allows us to see that compared to Figure 1.2, some categorical labels can be found outside of their expected regions. The scatter plot of final labels (Figure 4.3) shows, that for example, the “*surprise*” label, which would be expected in high arousal and mildly positive valence region, is still the closest match to a VA region representing the “*startled*” expression, according to the Figure 1.2. “*Neutral*” was often used in low arousal and low positive valence region, where it was the closest match to expressions like “*longing*” or “*pensive*.” In AffectNet, those expressions where only the dimensional labels can be accurately annotated but are outside of expected categorical regions are commonly labeled as “*None*.” We did not use this approach, as our network will not predict a wildcard category.

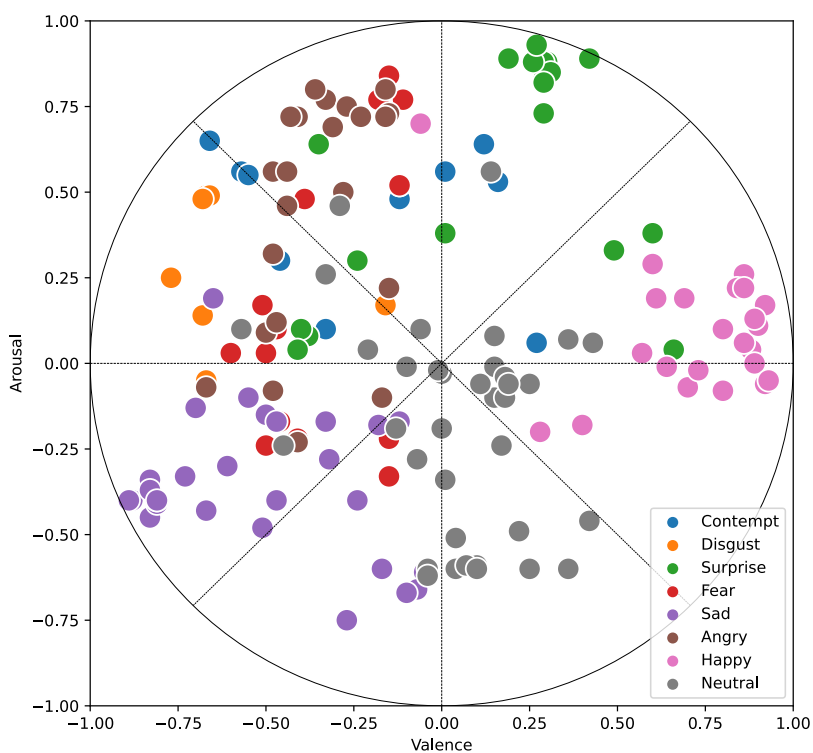


Figure 4.3: Distribution of annotation values on the test set.

### 4.3 Proposed Method

Multiple lightweight CNN architectures were tested to find an expression classifier with reasonable accuracy and inference time. All models were implemented in *Keras* [65], an API for the *TensorFlow* machine learning framework [66]. After the baseline comparison of different architectures, two were selected for further hyper-parameter tuning. Some of the experiments were:

- Optimizer testing – Adam, RMSprop
- Influence of learning rate cap for the aforementioned adaptive LR optimizers
- Improvement in generalization performance with different *data augmentation* techniques
- Comparison of inference speedup and accuracy hit when using RGB or grayscale input images
- Use of different loss functions
- Comparison of separate categorical and dimensional models to a simultaneous one with a combined loss function
- *Transfer learning* using pre-trained weights from the ImageNet database
- The scaling factor  $\alpha$  for the MobileNet family of networks
- *Skew-normalization* techniques for both categorical and dimensional labels

A log of those experiments was kept, and the training sessions were recorded using the *TensorBoard* toolkit. The resulting model with the highest validation accuracy was tested on a reserved 20% image test set and our custom test set. Inference speed was measured, and an application for real-time expression recognition was developed to evaluate the real-world usability of the model.

#### 4.3.1 Training Pipeline

Due to the size of the database being over 50GB, an efficient pipeline had to be implemented. The images were stored on an NVME SSD, and during the training epoch, each mini-batch was dynamically loaded to the main memory, the CPU processed data augmentation, and the prepared batch was then processed on GPU. The main bottleneck when training was the GPU for larger and CPU for smaller models. The total processing time for the experiments in this work was over 300 hours on a system with an Intel i7-5775c, GTX 1070TI GPU, and 32GB of RAM.



### 4.3.2 Architecture Experiments

The different architectures tested include MobileNetV1 [45], MobileNetV2 [46], NASNetMobile [44] and EfficientNet-B0 [67]. Those networks are implemented in Keras [47], and a Keras Functional API was used to wrap those architectures into a final model. The original classification top layers were stripped and replaced with new classification layers – eight nodes for discrete label classification, two nodes for dimensional label regression, or both for a simultaneous prediction (Figure 4.4).

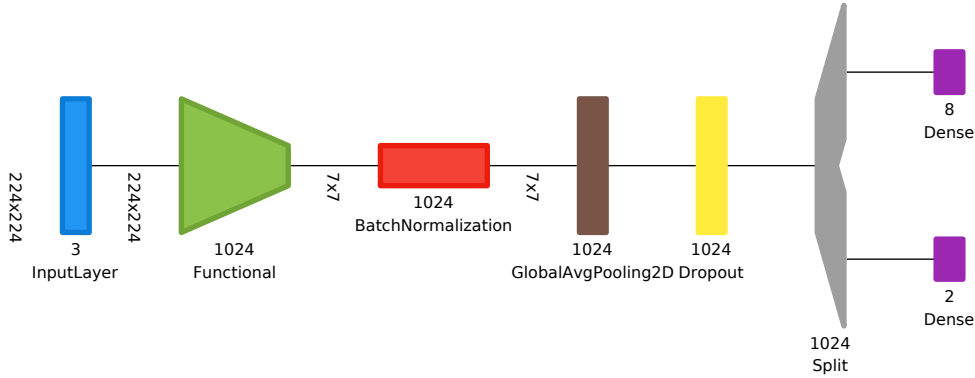


Figure 4.4: Visualization of the model architecture generated using Net2Vis [68].

The functional block is followed by a *Batch Normalization* layer to reduce the internal covariate shift and accelerate the training [69]. *Global Average Pooling* is a substitute for *Dense* layers in CNN and works by taking averages of feature maps and converting them into a feature vector [70]. A *Dropout* layer with  $p = 0.5$  is used for further regularization [30].

The mini-batch size used for training varied based on the model – a larger mini-batch size results in faster training time but might cause the video memory to overflow. Such overflow into main memory would increase the training time by over 20 times.

Architecture	Expected Input	Mini-batch Size	Time / Epoch
MobileNetV1	$224 \times 224$	128	1800 sec
MobileNetV2	$224 \times 224$	96	2050 sec
EfficientNet-B0	$224 \times 224$	48	3000 sec
NASNetMobile	$224 \times 224$	32	3400 sec
Xception	$299 \times 299$	16	10800 sec

Table 4.4: Comparison of training times of selected lightweight architectures to a larger network such as Xception [71].

After the initial evaluation, EfficientNet-B0 and NASNetMobile brought marginal improvement to validation accuracy at best. Thus we used MobileNets for further experiments, as they are much faster to train.

### 4.3.3 Data Preprocessing and Augmentation

The AffectNet database contains undesirable images – those either contain watermarks, are distorted, animated, or not a human face. Those are labeled as the “*Non-face*” category. Another category that had to be removed is the wildcard “*None*” category, which is assigned to images with dimensional labels only. The third category that had to be removed was the “*Uncertain*” category, which also does not contain dimensional labels. At last, six images in TIFF format had to be removed, as they would crash the input pipeline due to a bug in the latest Pillow library. There were 287,645 images left in the training set with those removed, 20% of which were randomly reserved as a test set.

Each mini-batch was dynamically loaded to memory during the training, resized to  $224 \times 224$  pixels, and pixel values were converted from  $[0; 255]$  range to  $[-1; 1]$ . No facial cropping was performed.

We used random horizontal flip,  $\pm 10\%$  random brightness change,  $\pm 10$ -degree random rotation, and up to 20% random zoom for *data augmentation*<sup>9</sup>, yielding minor improvements on validation loss. This data augmentation was used in all later models. We also tested the sample-wise normalization, which led to slightly worse results.

Using grayscale<sup>10</sup> instead of RGB sped up the training by around 10% (mostly by reducing the CPU load for data augmentation) and led to only marginally worse validation loss ( $< 2\%$  difference). However, it did not speed up the inference time.

### 4.3.4 Loss Functions

For classification, categorical cross-entropy loss was used together with the softmax activation function (Section 2.2.1).

For regression, a hyperbolic tangent activation function was used (Table 2.1). MAE and MSE losses (Section 2.2.2) were tested, with almost identical results. *Mollahosseini et al.* used a Euclidean ( $L_2$ ) loss [4]:

$$\mathcal{L} = \frac{1}{2N} \sum_{n=1}^N \|\hat{y}_n - y_n\|_2^2$$

---

<sup>9</sup>*Data augmentation* = Techniques to increase the diversity of training samples and reduce overfitting by modifying the data without altering their ground truth.

<sup>10</sup>In Keras, conversion to grayscale is handled by the Pillow library, using the ITU-R 601-2 luma transformation ( $0.299R + 0.587G + 0.114B$ ) and Floyd-Steinberg dithering [72].

whereas *Handrich et al.* used cross-entropy loss [14], defined as:

$$\mathcal{L}_{v,a} = -y_t \log(y_p) - (1 - y_t) \log(1 - y_p)$$

$$y_t = \left[ \frac{1}{2}(v + 1); \frac{1}{2}(a + 1) \right]$$

$$y_p = [\sigma(v'); \sigma(a')]$$

The cross-entropy loss requires the activation function of the dimensional predictor also to be sigmoid and thus cannot be directly interpreted as predicted values but instead must be transformed as  $[\hat{v}; \hat{a}] = 2([\hat{v}'; \hat{a}'] - 0.5)$ .

During the simultaneous training of a network with two predictors, the combined loss of both predictors is minimized. For this reason, the choice of appropriate losses is essential, as using poorly chosen losses might cause minor improvement in one predictor to outweigh more significant improvement in the second predictor.

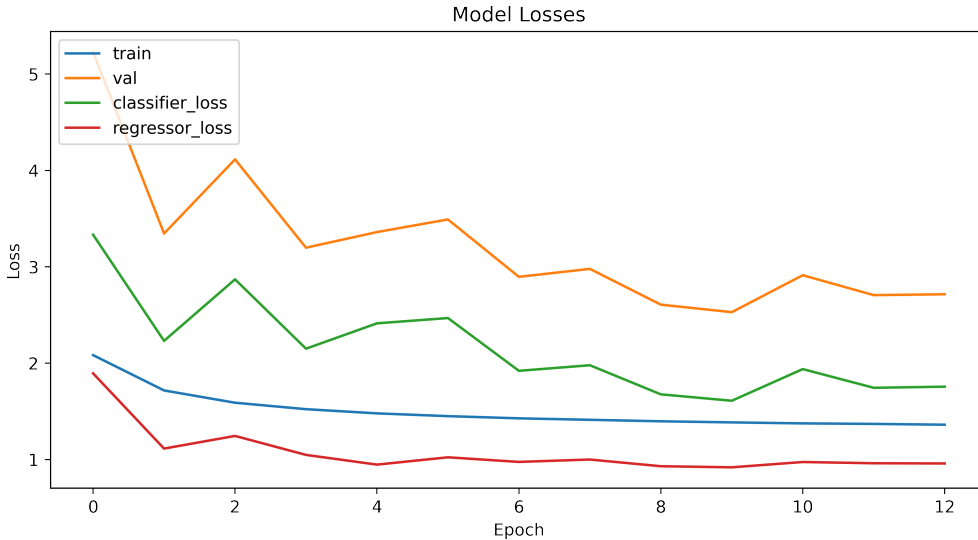


Figure 4.5: Example of model training with different predictor loss functions.

We tested several different losses, and the difference between those was marginal at best. Other metrics such as validation RMSE were also monitored during all experiments to compare the performance of those models.

### 4.3.5 Optimizers, Learning Rate, and Transfer Learning

The tested optimizers were RMSprop and Adam [27], with learning rates  $\eta_{initial} = [10^{-2}; 10^{-3}; 10^{-4}]$  and learning rate decay  $\rho = [0.8; 0.85; 0.9]$  after each epoch.

In our experiments, the learning rate decay did little to help with further model convergence – possibly due to the adaptive learning nature of both

## 4. EXPERIMENTS AND RESULTS

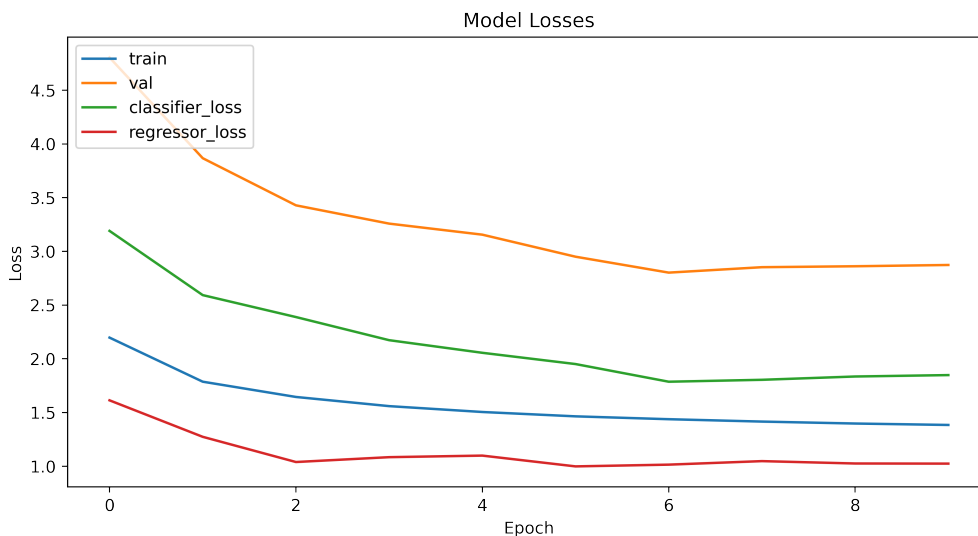


Figure 4.6: Training with  $\eta = 0.001$  and exponential weight decay  $\rho = 0.85$ .

RMSprop and Adam, where learning rate only works as a value cap for individual learning rates – but the descent was much smoother (Figure 4.6 vs. Figure 4.5).

Both optimizers seem to reach a similar plateau around epochs seven to twelve, after which the validation loss improves very slowly. *Early-stopping* was used when validation loss does not improve for three epochs, the last best weights are restored, and the training is stopped. However, due to the low total number of epochs and some variance between different models, we ended up training for a set number of epochs for consistency. Adam optimizer with learning rate  $\eta_{initial} = 0.001$  and  $\rho = 0.85$  after each epoch yielded the best results.

*Transfer learning* is a method of transferring knowledge from a different but related domain to a new learner [73]. The general idea is that we can take the trained weights from a network trained for another task, lock the convolution layers with already trained filters, and fine-tune the network by training the output layers on our data. Keras offers weights of networks trained on the ImageNet database for object classification [47].

We experimented with two different methods of transfer learning. First, we initialized the network with the pre-trained weights, locked the parameters except for the last classification layers, and trained the network until convergence. After reaching a plateau, the weights in the network were unlocked, and the entire network was fine-tuned with very small learning rates  $\eta = [0.0001; 0.00001]$ . Training the network with the base model locked reduced the training time by 35 %, and a plateau was reached on average between epochs six to eight. Fine-tuning reached a plateau in the same number of epochs. This approach resulted in no improvement in validation loss.

Using the second method, we initialized the network with the pre-trained weights instead of random weights, and the network was trained until a plateau was reached. Neither this approach improved the validation loss.

### 4.3.6 Skew-normalization Techniques

Due to the highly skewed nature of the AffectNet database, skew-normalization techniques might be required for good generalization, as some categories are outnumbered twentyfold by others. *Mollahosseini et al.* show that *weighted loss* can improve expression classification [4].

We define weighted loss as assigning the training samples weights inversely proportional to their class’ frequencies:

$$\text{weight}_c = \frac{\text{total number of samples}}{\text{number of classes} \times \text{class } c \text{ samples}}$$

When evaluating the model performance on skewed data, the accuracy metric might be misleading. Other metrics as the  $F_1$  score have been proposed as more appropriate [14]. For binary classification, the  $F_1$  score is defined as:

$$F_1 = \frac{\text{true positive}}{\text{true positive} + \frac{1}{2}(\text{false positive} + \text{false negative})}$$

F1 score can be extended to multi-class classification by using the mean of class-wise  $F_1$  scores:

$$F_1 = \frac{1}{N} \sum_{i=1}^N \frac{\text{tp}_i}{\text{tp}_i + \frac{1}{2}(\text{fp}_i + \text{fn}_i)}$$

where  $N$  is the number of classes and  $\text{tp}_i$ ,  $\text{fp}_i$ ,  $\text{fn}_i$  are the respective counts for the class  $i$ .

To further evaluate the model performance on a skewed test set, *Mollahosseini et al.* use a 200-fold unskewed down-sampling evaluation [4]. We use down-sampling with 500 random samples from each class per trial. Mean metrics from 200 trials are reported under the “*Norm*” column.

	AffectNet				Ours			
	Imbalanced		Weighted		Imbalanced		Weighted	
	Orig	Norm	Orig	Norm	Orig	Norm	Orig	Norm
Accuracy	0.72	0.54	0.64	0.63	0.77	0.50	0.62	0.60
F1 Score	0.57	0.52	0.55	0.62	0.52	0.46	0.48	0.60

Table 4.5: Comparison of weighted loss results using MobileNetV1 trained for 15 epochs vs. AlexNet [4].

#### 4. EXPERIMENTS AND RESULTS

$$\begin{aligned} \text{Sensitivity} & & \text{Miss Rate} \\ \text{TPR} &= \frac{\sum \text{true positive}}{\sum \text{condition positive}} & \text{FNR} &= \frac{\sum \text{false negative}}{\sum \text{condition positive}} \\ \text{Precision} & & \text{False Discovery Rate} \\ \text{PPV} &= \frac{\sum \text{true positive}}{\sum \text{predicted condition positive}} & \text{FDR} &= \frac{\sum \text{false positive}}{\sum \text{predicted condition positive}} \end{aligned}$$

Table 4.6: List of metrics on confusion matrix.

Predicted Category	Confusion matrix								TPR/FNR
	Neutral	Happy	Sad	Surprise	Fear	Disgust	Anger	Contempt	
Neutral	11571 20.10%	1484 2.58%	1438 2.50%	845 1.47%	188 0.33%	113 0.20%	1269 2.20%	211 0.37%	17119 67.59% 32.41%
Happy	1587 2.76%	24708 42.91%	145 0.25%	446 0.77%	54 0.09%	78 0.14%	160 0.28%	405 0.70%	27583 89.58% 10.42%
Sad	833 1.45%	133 0.23%	3012 5.23%	89 0.15%	142 0.25%	67 0.12%	298 0.52%	14 0.02%	4588 65.65% 34.35%
Surprise	287 0.50%	197 0.34%	37 0.06%	1197 2.08%	221 0.38%	19 0.03%	72 0.13%	4 0.01%	2034 58.85% 41.15%
Fear	75 0.13%	22 0.04%	91 0.16%	199 0.35%	570 0.99%	37 0.06%	80 0.14%	0 0.0%	1074 53.07% 46.93%
Disgust	27 0.05%	36 0.06%	40 0.07%	6 0.01%	19 0.03%	227 0.39%	75 0.13%	5 0.01%	435 52.18% 47.82%
Anger	697 1.21%	112 0.19%	306 0.53%	69 0.12%	95 0.16%	221 0.38%	2987 5.19%	28 0.05%	4515 66.16% 33.84%
Contempt	45 0.08%	104 0.18%	4 0.01%	2 0.00%	0 0.0%	3 0.01%	15 0.03%	59 0.10%	232 25.43% 74.57%
TPR/FNR	15122 76.52% 23.48%	26796 92.21% 7.79%	5073 59.37% 40.63%	2853 41.96% 58.04%	1289 44.22% 55.78%	765 29.67% 70.33%	4956 60.27% 39.73%	726 8.13% 91.87%	57580 76.99% 23.01%
	Neutral	Happy	Sad	Surprise	Fear	Disgust	Anger	Contempt	TPR/FNR

Figure 4.7: Example of imbalanced learning model, with no skew-normalization techniques used, compared to Figure 4.8.

In our experiments, using sample weights improved the validation loss significantly. The results can be further evaluated on the *confusion matrices* when classifying the 20 % test set. The confusion matrix compares true and predicted labels for individual classes. Furthermore, the matrix shows the *sensitivity* (TPR), *miss rate* (FNR), *precision* (PPV), and *false discovery rate* (FDR) for each class (Table 4.6).

Although the model in Figure 4.7 achieves higher test accuracy, its sensitivity for the “*contempt*” category is very low with only 8.13 %, whereas when using sample weights, the sensitivity is 50.28 %, but at the cost of a higher false discovery rate.

Predicted Category	Neutral	7692 13.36%	1388 2.41%	694 1.21%	184 0.32%	38 0.07%	31 0.05%	553 0.96%	134 0.23%	10714 71.79% 28.21%
	Happy	336 0.58%	18833 32.71%	50 0.09%	117 0.20%	10 0.02%	21 0.04%	52 0.09%	105 0.18%	19524 96.46% 3.54%
	Sad	1974 3.43%	413 0.72%	3054 5.30%	116 0.20%	96 0.17%	70 0.12%	373 0.65%	29 0.05%	6125 49.86% 50.14%
	Surprise	1767 3.07%	1550 2.69%	252 0.44%	1863 3.24%	270 0.47%	43 0.07%	232 0.40%	36 0.06%	6013 30.98% 69.02%
	Fear	501 0.87%	262 0.46%	375 0.65%	403 0.70%	769 1.34%	60 0.10%	292 0.51%	6 0.01%	2668 28.82% 71.18%
	Disgust	301 0.52%	461 0.80%	189 0.33%	27 0.05%	37 0.06%	386 0.67%	491 0.85%	19 0.03%	1911 20.20% 79.80%
	Anger	1256 2.18%	266 0.46%	372 0.65%	94 0.16%	65 0.11%	135 0.23%	2814 4.89%	32 0.06%	5034 55.90% 44.10%
	Contempt	1295 2.25%	3623 6.29%	87 0.15%	49 0.09%	4 0.01%	19 0.03%	149 0.26%	365 0.63%	5591 6.53% 93.47%
	TPR/FNR	15122 50.87% 49.13%	26796 70.28% 29.72%	5073 60.20% 39.80%	2853 65.30% 34.70%	1289 59.66% 40.34%	765 50.46% 49.54%	4956 56.78% 43.22%	726 50.28% 49.72%	57580 62.13% 37.87%
	Neutral	Happy	Sad	Surprise	Fear	Disgust	Anger	Contempt	PPV/FDR	
	True Category									

Figure 4.8: Example of a model trained with sample weights.

Using sample weights for the classification problems is relatively common [74]. We test whether the dimensional predictor could also benefit from a weighted approach since the distribution of labels is very uneven (Figure 1.3,

#### 4. EXPERIMENTS AND RESULTS

mind the logarithmic scale). Similar to how total classifier accuracy can be misleading, the RMSE values of dimensional predictors also do not apply to all predictions equally (Figure 4.9). To further evaluate the performance, we split the VA space into  $71 \times 71$  regions by their ground truth and measure RMSE per region (Figures 4.9 and 4.10). The mean RMSE of non-empty regions is reported as *normalized RMSE* in Table 4.5.

Compared to categorical classification, the choice of sample weights in the dimensional space is not as simple. We experimented with class weights, as class prevalence determines the sample density in the corresponding VA region. The second approach is a region-based one – the VA space is divided into regions, and samples inside of each region are assigned weights as:

$$\text{weight}_{\text{region}} = \frac{\# \text{ of samples in region}}{\# \text{ of non-empty regions} \times \# \text{ of samples in region}}$$

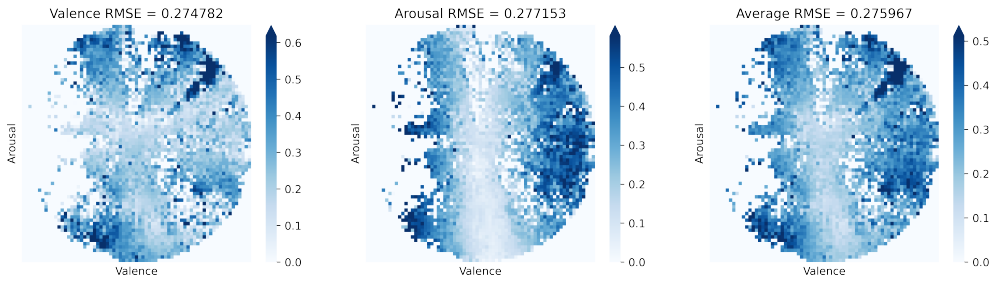


Figure 4.9: Heatmap of RMSE prediction error for test samples from different regions of the VA space.

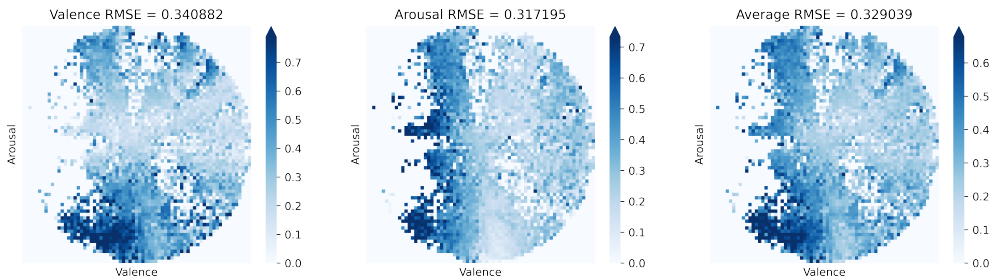


Figure 4.10: Heatmap of RMSE prediction error for test samples from different regions of the VA space when using class weights.

This approach with region-based weights has two significant problems. Since the annotations are bound by the circumplex, the border regions group samples from a smaller area. This problem could be solved by defining the regions in radial coordinates or using a completely different method to calculate the sample weight based on the local sample distribution density. The second



	AffectNet		Ours					
	Imbalanced		Imbalanced		Class Weights		7x7 Regions	
	Val	Aro	Val	Aro	Val	Aro	Val	Aro
RMSE	0.394	0.402	0.286	0.233	0.397	0.297	0.317	0.284
SAGR	0.728	0.670	0.823	0.696	0.755	0.660	0.808	0.668
CCC	0.541	0.450	0.809	0.579	0.660	0.438	0.765	0.525
N. RMSE	-	-	0.278	0.271	0.340	0.317	0.307	0.300

Table 4.7: Comparison of regression results when using different sample weights.

problem is that some regions with very few samples suddenly have a very high weight due to the uneven distribution, possibly hurting the generalization and overfitting the classifier.

Although the weighted approach to dimensional prediction did not improve the accuracy of the predictions, the predictions are somewhat more consistent (Figures 4.9 vs. 4.10). The region-based approach introduced different accuracy levels at the boundaries of different regions, and it would not perform well in simultaneous predictors.

### 4.3.7 Simultaneous Prediction of Dimensional and Categorical Labels

A classifier that simultaneously predicts multiple different expression models – such as dimensional and categorical models – can be desirable for multiple reasons. One would be the added versatility of such classifiers, allowing for multiple applications at virtually no additional computational cost. The second reason is that other works show improved classifier performance in categorical and dimensional predictions when the model learns simultaneously on different labels [14].

Our guess as to why this is is the high annotator disagreement rate, and describing each sample in terms of multiple expression models could lower the total annotation error, especially in borderline samples.

Table 4.5 shows that sample weights are beneficial for categorical predictors. We test whether the same applies to simultaneous predictors.

For our experiments, we used the MobileNetV1 architecture and compared results with YOLOv2 architecture [14]. We further evaluate the performance using skew-normalized metrics defined in Section 4.3.6.

The skew-normalized metrics show that using sample weights is still beneficial to the classifier. Despite the slightly worse dimensional prediction accuracy, the classifier results are very similar to the results in Figures 4.7 and 4.8 – using sample weights improved the sensitivity for underrepresented classes

#### 4. EXPERIMENTS AND RESULTS

	Handrich et al.		Ours			
	Imbalanced		Imbalanced		Class Weights	
Accuracy	0.79		0.77		0.62	
F1 Score	0.61		0.55		0.47	
VA RMSE	0.269	0.228	0.285	0.236	0.399	0.298
VA CCC	0.845	0.606	0.822	0.598	0.644	0.411
N. Accuracy	-		0.5275		0.5802	
N. F1 Score	-		0.4988		0.5809	
N. VA RMSE	-	-	0.281	0.284	0.342	0.319

Table 4.8: Comparison of imbalanced and weighted learning approach using MobileNetV1 trained for 15 epochs to a simultaneous classifier built on the YOLOv2 architecture.

IMBALANCED	Categorical	Dimensional		Simultaneous	
Accuracy	0.77	-		0.77	
F1 Score	0.53	-		0.55	
VA RMSE	-	0.286	0.233	0.285	0.236
VA CCC	-	0.809	0.579	0.822	0.598
N. Accuracy	0.51	-		0.53	
N. F1 Score	0.49	-		0.50	
N. RMSE	-	0.278	0.271	0.281	0.284
CLASS WEIGHTS	Categorical	Dimensional		Simultaneous	
Accuracy	0.62	-		0.62	
F1 Score	0.48	-		0.47	
VA RMSE	-	0.397	0.297	0.399	0.298
VA CCC	-	0.660	0.438	0.644	0.411
N. Accuracy	0.60	-		0.58	
N. F1 Score	0.60	-		0.58	
N. RMSE	-	0.340	0.317	0.342	0.319

Table 4.9: Comparison of the simultaneous predictor and baseline predictors, both with and without weighted learning using MobileNetV1 architecture.

significantly. We consider this approach more balanced than using unweighted learning, as shown by the skew-normalized metrics in Table 4.8.

As to whether the simultaneous training improves the total classifier performance, we compare it to both baseline predictors. Table 4.9 shows that our results did not show any significant improvement or degradation in performance compared to other works [14]. The weighted approach still results in a better categorical classifier with only minor accuracy hit to the dimensional predictions. We believe that using a different type of sample weighting might further narrow the difference between those predictors.

### 4.3.8 Scaling Factor $\alpha$ of MobileNet Architectures and Speed-Accuracy Tradeoff

The MobileNet family of networks is designed for easy width scaling with a single parameter – this makes it a very versatile network architecture for use on different devices and can be tailored to the available resources [45, 46]. To determine what accuracy can be achieved at different computational costs, we experimented with the network width scaling parameter  $\alpha$ .

We used the MobileNetV1 architecture for those experiments, with an input shape of  $224 \times 224$ . Values of the width multiplier  $\alpha$  tested were [0.25, 0.35, 0.5, 0.75, 1, 1.4]. A learning rate  $\eta_{initial} = 0.001$  and a weight decay  $\rho = 0.85$  after each epoch were used for 15 epochs. Mini-batch size of 128 was used to train  $\alpha < 1$  networks and 96 for  $\alpha \geq 1$  due to GPU memory constraints.

MOBILENETV1 $\alpha$ -SCALE	0.25	0.35	0.5	0.75	1	1.4
Million Parameters	0.222	0.419	0.836	1.843	3.243	6.288
Model Size (MB)	0.83	1.58	3.16	6.99	12.31	23.91
RPi4 Inference Time (ms)	108	137	177	256	335	515
N. Accuracy	0.533	0.550	0.567	0.584	0.580	0.589
N. F1 Score	0.534	0.553	0.569	0.585	0.581	0.590
N. Valence	0.366	0.361	0.352	0.341	0.342	0.335
N. Arousal	0.342	0.330	0.327	0.323	0.319	0.318

Table 4.10: Comparison of performance and inference times of different MobileNetV1 sizes, scaled with  $\alpha$ .

The models were further benchmarked on Raspberry Pi 4 (4GB), representing the mobile devices we target in this work. The models were benchmarked in TensorFlow version 2.3.0. The inference time could potentially be improved even further with tools like TensorFlow Lite [75].

Although the model inference time is fast enough for real-time classification, we first need to detect the faces. Using the OpenCV [76] implementation of the Viola-Jones algorithm requires 59ms on the Raspberry Pi 4 to process a  $1920 \times 1080$  frame, whereas the CNN-based CenterFace requires 790ms. We prefer the Viola-Jones algorithm on the Raspberry Pi 4 for two reasons – firstly, much lower latency, and secondly, it only detects large and unoccluded faces that can be classified accurately. CenterFace can detect even small and partially occluded faces that might be too low quality for expression recognition. However, it can still be used with smaller MobileNets and achieve over a frame per second throughput. This changes when more processing resources are available, as CenterFace is a superior facial detector with a lower false detection rate and would be the preferred detector.

## 4. EXPERIMENTS AND RESULTS

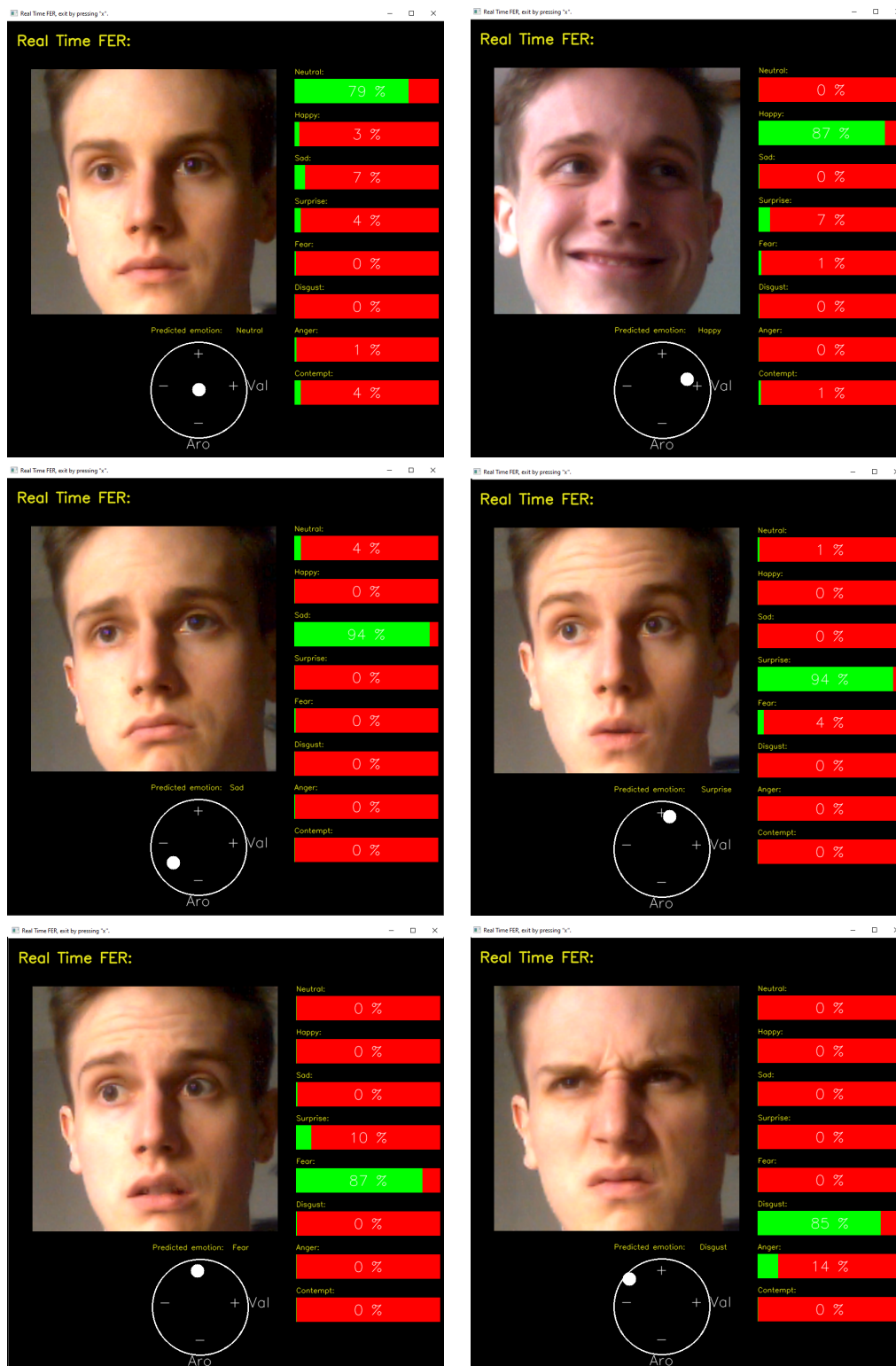


Figure 4.11: Screenshots from a real-time expression recognition application.

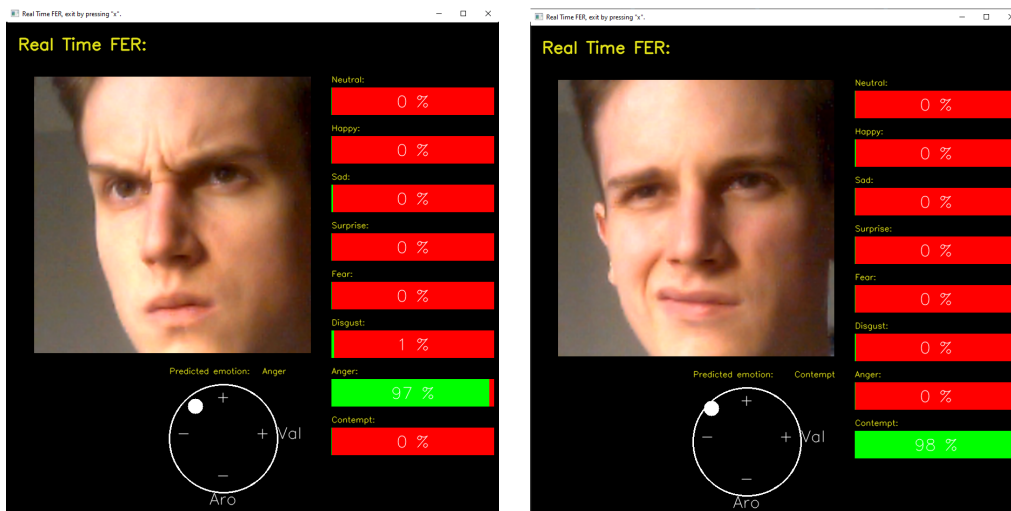


Figure 4.12: More screenshots from a real-time expression recognition application.

When running on the i7 5775c CPU, the Viola-Jones algorithm requires 13 ms on average, CenterFace 90 ms, and  $\alpha = 1$  MobileNet 52 ms. On GTX 1070TI, the inference time of the same network is 454 ms per batch of 256 images or 4 ms per single image. Those GPU benchmarks were severely bottlenecked by the CPU image preprocessing – CUDA usage never exceeded 60 %.

#### 4.4 Evaluation on Custom Test Set

To further validate our results on the AffectNet database, we evaluate the model performance on our test set described in Section 4.2.

	Imbalanced		Class Weights	
Accuracy	0.606		0.606	
F1 Score	0.527		0.550	
VA RMSE	0.339	0.345	0.339	0.399
VA SAGR	0.756	0.738	0.744	0.650
VA CCC	0.733	0.618	0.665	0.477
N. Accuracy	0.561		0.554	
N. F1 Score	0.503		0.537	
N. VA RMSE	0.251	0.258	0.273	0.321

Table 4.11: Comparison of imbalanced and weighted models on our custom test set. Normalized accuracy and F1 metrics on the smaller test set were sampled with five images per class per trial.

CLASS SENSITIVITY (%)	Imbalanced	Class Weights
Neutral	63.89	69.44
Happy	95.65	78.26
Sad	57.14	57.14
Surprise	68.42	63.16
Fear	0.00	35.71
Disgust	71.43	42.86
Anger	77.27	68.18
Contempt	9.09	27.27

Table 4.12: Comparison of class sensitivities of the two models.

Despite the modest size of our test set, the results are very close to those measured on the AffectNet database. Although the accuracy difference between those two models is relatively small, the class sensitivities show that class weights result in a more balanced classifier.

## 4.5 Comparison with the State of the Art

*Mollahosseini et al.* proposed multiple baseline solutions to AffectNet classification – both conventional approach (hand-crafted HOG features + SVM) and CNN models are tested for categorical and dimensional predictions on an unpublished test set, counting the 36 thousand pictures annotated by two different annotators. An unskewed validation set of 500 random images from each expression category is used for hyper-parameter tuning. They also show that weighted loss is beneficial to the categorical classifier and propose skew-normalized metrics to evaluate the model performance.

*Handrich et al.* used a YOLOv2 architecture that simultaneously predicts both categorical and dimensional values and performs facial detection in real-time. They also perform a cross-database evaluation on multiple databases to further validate their results. Their model also outperforms the originally proposed baseline solutions.

Our solution combines the two previous works and applies weighted loss to a simultaneous classifier built on the lightweight MobileNet architecture. Our solution significantly improves the categorical classification accuracy, which we measure by skew-normalized metrics proposed in [4]. We propose a similar skew-normalized metric to evaluate the dimensional predictor better. Although the accuracy of dimensional predictor is slightly worse when using the weighted approach, our models built on the MobileNet architecture are about ten times smaller and significantly faster compared to YOLOv2, allowing for real-time FER on low-power devices like the Raspberry Pi 4, with a near state-of-the-art accuracy and easy scalability for devices with more resources. On a modern GPU, the throughput can achieve over 500 frames per second.

---

## Discussion

We want to discuss the importance of skew-normalized metrics used in this work and the labels in the AffectNet database.

Firstly, the misleading *accuracy* metric – due to the skewed distribution of labels, even a model that always predicts happy expression would have 46 % accuracy, which is way higher than the intuitively expected 12.5 %. The skew-normalized accuracy metric would, in this case, result in the expected 12.5 %, which is precisely how we would like to measure such a model, under the assumption that all facial expressions are equally probable in the real world.

Neutral	80276	25 %
Happy	146198	46 %
Sad	29487	9 %
Surprise	16288	5 %
Fear	8191	3 %
Disgust	5264	2 %
Anger	28130	9 %
Contempt	5135	2 %

Table 4.13: Label distribution in the AffectNet database.

Secondly, the relatively large annotator disagreement rates – determining the ground-truth value is often challenging, even for trained human annotators. For this reason, as pointed out in [14], there is a large number of potentially mislabeled samples (Table 4.14). Those might hinder the training of classifiers and result in lower accuracy metrics, despite being predicted correctly. The normalized accuracy metrics are very close to the measured annotator agreement rates in the AffectNet database, leading us to believe that further advancements in facial expression recognition will soon require the databases to improve on the label accuracy, possibly by having multiple highly trained annotators evaluate the data.



Table 4.14: Examples of misclassified images in the AffectNet database taken from [14]. True positives are the examples of correctly annotated samples, and false positives are samples that all display the “*fear*” expression but are annotated wrongly. False negatives are samples with wrong “*fear*” annotations that display other expressions.



---

## Conclusion

In this work, we set out to find whether a convolutional neural network with a small memory footprint and fast inference time can achieve reasonable accuracy in facial expression recognition to be used in real-time applications and on mobile devices. To train and evaluate our models, we used the AffectNet database – currently the largest database of facial expression images.

We tested different architectures and training hyperparameters and found that MobileNetV1 achieved the fastest inference times while not compromising accuracy. It can also be easily scaled with a simple hyperparameter to suit the desired application better. To further allow for better scalability, our proposed approach separates facial detection and expression recognition problems so that both can be adjusted for the target processing power. The two facial detectors tested were the Viola-Jones algorithm and a MobileNetV2-based CenterFace. While the first detector is faster, the latter achieves state-of-the-art results if the available resources allow for its use.

We tested both categorical and dimensional expression predictors and combined them into a single versatile model for simultaneous prediction of eight basic emotions and valence/arousal values. We further show that using weighted loss for such a model improves the categorical classification accuracy, which we measure using skew-normalized metrics.

To further validate our results, we collected a modest test set of 160 images, which were annotated via a GUI annotation application by five different people. All were instructed on how to annotate both categorical and dimensional labels properly. The annotator disagreement rate for categorical labels was very similar to that in AffectNet, while the disagreement for dimensional labels was considerably higher. We attribute this to the restriction of the VA region during the annotation process based on the selected expression category in AffectNet.

The results on our test set are very similar to those on the AffectNet database, with the weighted loss approach resulting in similar accuracy but higher sensitivity for underrepresented classes like *fear* and *contempt*.

## CONCLUSION

---

To demonstrate the classifier's performance in a real-time scenario, we developed a simple application. It tracks faces from a webcam input using the Viola-Jones algorithm and uses one of six pre-computed models with different parameter counts, from mere 220 thousand up to over six million. The fastest model achieves over five frames per second (fps) on the Raspberry Pi 4, the largest model over two fps, with  $1920 \times 1080$  video input resolution. The same models can achieve over 500 fps on modern GPU, allowing for high-speed processing of videos or multiple camera inputs.

---

## Bibliography

- [1] Gates, K. *Our Biometric Future: Facial Recognition Technology and the Culture of Surveillance*. New York University Press, 2011, ISBN 978-0-8147-3209-0.
- [2] Russell, J.; Fernández-Dols, J.; et al. *The psychology of facial expression*. 1997.
- [3] Williams, M.; Morris, A. P.; et al. Amygdala Responses to Fearful and Happy Facial Expressions under Conditions of Binocular Suppression. *The Journal of Neuroscience*, volume 24, 2004: pp. 2898 – 2904.
- [4] Mollahosseini, A.; Hasani, B.; et al. Affectnet: A database for facial expression, valence, and arousal computing in the wild. *IEEE Transactions on Affective Computing*, 2017.
- [5] Ekman, P.; Friesen, W. V. Constants across cultures in the face and emotion. *Journal of personality and social psychology*, volume 17, no. 2, 02 1971: pp. 124–129.
- [6] Ekman, P.; Friesen, W. V. *Facial action coding system*. 1978.
- [7] Ekman, P.; Friesen, W. V.; et al. EMFACS-7: Emotional facial action coding system. *Unpublished manuscript, University of California at San Francisco*, volume 2, no. 36, 1983: p. 1.
- [8] Langner, O.; Dotsch, R.; et al. Presentation and validation of the Radboud Faces Database. *Cognition and Emotion*, volume 24, 2010: pp. 1377 – 1388.
- [9] Russell, J. A. A circumplex model of affect. *Journal of personality and social psychology*, volume 39, no. 6, 1980: p. 1161.
- [10] Watson, D.; Tellegen, A. Toward a consensual structure of mood. *Psychological bulletin*, volume 98, no. 2, 1985: p. 219.

- [11] Mehrabian, A. *Basic dimensions for a general psychological theory: Implications for personality, social, environmental, and developmental studies*, volume 2. Oelgeschlager, Gunn & Hain Cambridge, MA, 1980.
- [12] Plutchik, R. E.; Conte, H. R. *Circumplex models of personality and emotions*. American Psychological Association, 1997.
- [13] Kollias, D.; Tzirakis, P.; et al. Deep Affect Prediction in-the-Wild: Aff-Wild Database and Challenge, Deep Architectures, and Beyond. *International Journal of Computer Vision*, volume 127, no. 6-7, Feb 2019: p. 907–929, ISSN 1573-1405.
- [14] Handrich, S.; Dinges, L.; et al. Simultaneous Prediction of Valence/Arousal and Emotions on AffectNet, Aff-Wild and AFEW-VA. *Procedia Computer Science*, volume 170, 2020: pp. 634–641, ISSN 1877-0509.
- [15] Shirley, P.; Chiu, K. A Low Distortion Map Between Disk and Square. *Journal of Graphics Tools*, volume 2, no. 3, 1997: pp. 45–52.
- [16] Paltoglou, G.; Thelwall, M. Seeing Stars of Valence and Arousal in Blog Posts. *IEEE Transactions on Affective Computing*, volume 4, no. 1, 2013: pp. 116–123.
- [17] Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, volume 65 6, 1958: pp. 386–408.
- [18] McCulloch, W. S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, volume 5, no. 4, 1943: pp. 115–133.
- [19] Minsky, M.; Papert, S. *Perceptrons - an introduction to computational geometry*. 1969.
- [20] Widrow, B.; Hoff, T. *An adaptive "ADALINE" neuron using chemical "memistors"*. 1960.
- [21] You, K.; Long, M.; et al. How Does Learning Rate Decay Help Modern Neural Networks? *arXiv 1908.01878*, 2019.
- [22] Ivakhnenko, A. G.; Lapa, V. G. Cybernetics and Forecasting. *Nature*, volume 219, 1968: pp. 202–203.
- [23] Werbos, P. *Beyond Regression : "New Tools for Prediction and Analysis in the Behavioral Sciences"*. 1974.
- [24] Smith, S. L.; Le, Q. V. *A Bayesian Perspective on Generalization and Stochastic Gradient Descent*. 2018.

- 
- [25] Mishkin, D.; Sergievskiy, N.; et al. Systematic evaluation of convolution neural network advances on the Imagenet. *Computer Vision and Image Understanding*, volume 161, Aug 2017: p. 11–19, ISSN 1077-3142.
- [26] Ruder, S. An overview of gradient descent optimization algorithms. *arXiv 1609.04747*, 2017.
- [27] Kingma, D. P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv 1412.6980*, 2017.
- [28] Keskar, N. S.; Socher, R. Improving Generalization Performance by Switching from Adam to SGD. *arXiv 1712.07628*, 2017.
- [29] Ng, A. Y. Feature Selection, L1 vs. L2 Regularization, and Rotational Invariance. In *Proceedings of the Twenty-First International Conference on Machine Learning, ICML '04*, New York, NY, USA: Association for Computing Machinery, 2004, ISBN 1581138385, p. 78.
- [30] Phaisangittisagul, E. An Analysis of the Regularization Between L2 and Dropout in Single Hidden Layer Neural Network. In *2016 7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS)*, 2016, pp. 174–179.
- [31] Hubel, D.; Wiesel, T. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, volume 148, 1959.
- [32] Hubel, D.; Wiesel, T. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, volume 195, 1968.
- [33] Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, volume 36, April 1980: pp. 193–202.
- [34] Weng, J. J.; Ahuja, N.; et al. Learning recognition and segmentation of 3-D objects from 2-D images. In *1993 (4th) International Conference on Computer Vision*, 1993, pp. 121–128.
- [35] LeCun, Y.; Bottou, L.; et al. *Gradient-based learning applied to document recognition*. 1998.
- [36] Ciresan, D. C.; Meier, U.; et al. *Flexible, high performance convolutional neural networks for image classification*. 2011.
- [37] Krizhevsky, A.; Sutskever, I.; et al. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, volume 60, 2012: pp. 84 – 90.

- [38] Bianco, S.; Cadène, R.; et al. Benchmark Analysis of Representative Deep Neural Network Architectures. *IEEE Access*, volume 6, 2018: pp. 64270–64277.
- [39] Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv 1409.1556*, 2015.
- [40] He, K.; Zhang, X.; et al. Deep Residual Learning for Image Recognition. *arXiv 1512.03385*, 2015.
- [41] Szegedy, C.; Liu, W.; et al. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015: pp. 1–9.
- [42] Szegedy, C.; Vanhoucke, V.; et al. Rethinking the Inception Architecture for Computer Vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016: pp. 2818–2826.
- [43] Szegedy, C.; Ioffe, S.; et al. *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*. 2017.
- [44] Zoph, B.; Vasudevan, V.; et al. Learning Transferable Architectures for Scalable Image Recognition. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018: pp. 8697–8710.
- [45] Howard, A. G.; Zhu, M.; et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv 1704.04861*, 2017.
- [46] Sandler, M.; Howard, A.; et al. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *arXiv 1801.04381*, 2019.
- [47] Chollet, F.; et al. Keras documentation: Keras Applications. [online], [visited 2021-04-15]. Available from: <https://keras.io/api/applications/>
- [48] Kanade, T. Identification of Human Faces. In *Computer recognition of human faces*, Birkhäuser Basel, 1977, ISBN 978-3-0348-5737-6, pp. 47–63.
- [49] Viola, P.; Jones, M. Rapid object detection using a boosted cascade of simple features. 2001.
- [50] Crow, F. Summed-area tables for texture mapping. *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, 1984.

- 
- [51] Made Kris Raya, I. G. N.; Jati, A. N.; et al. Analysis realization of Viola-Jones method for face detection on CCTV camera based on embedded system. In *2017 International Conference on Robotics, Biomimetics, and Intelligent Computational Systems (Robionetics)*, 2017, pp. 1–5.
- [52] Cerna, L. R.; Cámara-Chávez, G.; et al. *Face Detection: Histogram of Oriented Gradients and Bag of Feature Method*. 10 2013.
- [53] Zhang, Y.; Jin, R.; et al. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, volume 1, no. 1-4, 2010: pp. 43–52.
- [54] Zhang, K.; Zhang, Z.; et al. Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks. *IEEE Signal Processing Letters*, volume 23, no. 10, Oct 2016: p. 1499–1503, ISSN 1558-2361.
- [55] Zhang, S.; Zhu, X.; et al. S<sup>3</sup>FD: Single Shot Scale-invariant Face Detector. *arXiv 1708.05237*, 2017.
- [56] Xu, Y.; Yan, W.; et al. CenterFace: Joint Face Detection and Alignment Using Face as Point. *arXiv 1911.03599*, 2019.
- [57] Liu, W.; Anguelov, D.; et al. SSD: Single Shot MultiBox Detector. *Lecture Notes in Computer Science*, 2016: p. 21–37.
- [58] Lin, T.-Y.; Dollár, P.; et al. Feature Pyramid Networks for Object Detection. *arXiv 1612.03144*, 2017.
- [59] Xu, Y.; Yan, W.; et al. CenterFace. [online] GitHub repository <https://github.com/Star-Clouds/CenterFace>, 2019, [visited 2021-04-15].
- [60] Goodfellow, I.; Erhan, D.; et al. Challenges in representation learning: A report on three machine learning contests. *Neural networks : the official journal of the International Neural Network Society*, volume 64, 2013: pp. 59–63.
- [61] Li, S.; Deng, W. Reliable Crowdsourcing and Deep Locality-Preserving Learning for Unconstrained Facial Expression Recognition. *IEEE Transactions on Image Processing*, volume 28, no. 1, 2019: pp. 356–370.
- [62] Kollias, D.; Zafeiriou, S. Expression, Affect, Action Unit Recognition: Aff-Wild2, Multi-Task Learning and ArcFace. *arXiv 1910.04855*, 2019.
- [63] UNESCO. The Race Question. [online] <https://unesdoc.unesco.org/ark:/48223/pf0000128291>, [visited 2021-04-20].
- [64] Lin, L. A concordance correlation coefficient to evaluate reproducibility. *Biometrics*, volume 45 1, 1989: pp. 255–68.

- [65] Chollet, F.; et al. Keras. [online], [visited 2020-12-15]. Available from: <https://keras.io/>
- [66] Abadi, M.; Agarwal, A.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015, [visited 2020-12-16]. Available from: <https://www.tensorflow.org/>
- [67] Tan, M.; Le, Q. V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *arXiv 1905.11946*, 2020.
- [68] Bäuerle, A.; Ropinski, T. Net2Vis: Transforming Deep Convolutional Networks into Publication-Ready Visualizations. *arXiv 1902.04394*, 2019.
- [69] Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv 1502.03167*, 2015.
- [70] Lin, M.; Chen, Q.; et al. Network In Network. *arXiv 1312.4400*, 2014.
- [71] Chollet, F. Xception: Deep Learning with Depthwise Separable Convolutions. *arXiv 1610.02357*, 2017.
- [72] Clark, A. Pillow (PIL Fork) Documentation. 2015. Available from: <https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf>
- [73] Zhuang, F.; Qi, Z.; et al. A Comprehensive Survey on Transfer Learning. *arXiv 1911.02685*, 2020.
- [74] Cui, Y.; Jia, M.; et al. Class-Balanced Loss Based on Effective Number of Samples. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019: pp. 9260–9269.
- [75] Abadi, M.; Agarwal, A.; et al. TensorFlow Lite: Performance Best Practises. 2015, [visited 2021-05-06]. Available from: [https://www.tensorflow.org/lite/performance/best\\_practices](https://www.tensorflow.org/lite/performance/best_practices)
- [76] Bradski, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. Available from: <https://opencv.org/>



---

## List of Acronyms

<b>AD</b>	Action Descriptor
<b>ANN</b>	Artificial Neural Network
<b>API</b>	Application Programming Interface
<b>AU</b>	Action Unit
<b>CCC</b>	Concordance Correlation Coefficient
<b>CNN</b>	Convolutional Neural Network
<b>EMFACS</b>	Emotion FACS
<b>FACS</b>	Facial Action Coding System
<b>FDR</b>	False Discovery Rate
<b>FER</b>	Facial Expression Recognition
<b>FLOPs</b>	Floating-point Operations
<b>FNR</b>	False Negative Rate
<b>FN</b>	False Negative
<b>FP</b>	False Positive
<b>FPS</b>	Frames Per Second
<b>GPU</b>	Graphical Processing Unit
<b>GUI</b>	Graphical User Interface
<b>HCI</b>	Human-Computer Interaction
<b>HOG</b>	Histogram of Oriented Gradients

## A. LIST OF ACRONYMS

---

**MAE** Mean Average Error

**MSE** Mean Square Error

**MTCNN** Multitask Cascaded CNN

**NMS** Non-Maximum Suppression

**O-Net** Output Network

**P-Net** Proposal Network

**PAD** Pleasure – Arousal – Dominance

**PANA** Positive Activation – Negative Activation

**PPV** Positive Predictive Value

**R-Net** Refine Network

**RAF-DB** Real-world Affective Faces Database

**RMSE** Root Mean Square Error

**RMSLE** Root Mean Square Log Error

**RNN** Recurrent Neural Networks

**RaFD** Radboud Faces Database

**ReLU** Rectified Linear Unit

**S<sup>3</sup>FD** Single Shot Scale-invariant Face Detector

**SAGR** Sign Agreement Metric

**SSD** Single-shot Detector

**SVM** Support Vector Machine

**TLU** Threshold Logic Unit

**TN** True Negative

**TPR** True Positive Rate

**TP** True Positive

**VA** Valence – Arousal

---

## Contents of the enclosed media

Some large files could not be submitted due to size limits. For this reason, all models can also be found on the FIT CTU GitLab repository at:  
<https://gitlab.fit.cvut.cz/vadlemar/real-time-facial-expression-recognition-in-the-wild>.

```
| src ..... all source files and pre-trained models  
├─ BP_Vadlejch_Martin_2021.pdf ..... this thesis in the PDF format  
├─ BP_Vadlejch_Martin_2021.zip ..... LATEX source files for this thesis  
└─ readme.md ..... further description of all the submitted files
```