Emotion Detection

Image classification using Convolutional Neural Networks

Jonathan H. Krebs, Eric S. R. Bowman, Simon Hilding-Nørkjær, Andreas Stillits & Emilie Jessen

UNIVERSITY OF COPENHAGEN

Motivation

End goal: Detection and isolation of human faces and estimation of the emotional state.

Side quest: Comparing implementation and performance between native and pre-trained models.

Use cases include

- Gauging reactions to situations or products
- Allowing models to consider the human emotional response
- Detect and warn inattentive drivers

Methods

Part A:
Multiclass classification of human emotions using **CNN**

Part B:
Detecting and isolating faces using **MTCNN** in arbitrary photos (Pictures of this class)

Methods

Part A:
Multiclass classification of human emotions using **CNN**

Loss Function:
Categorical Cross Entropy

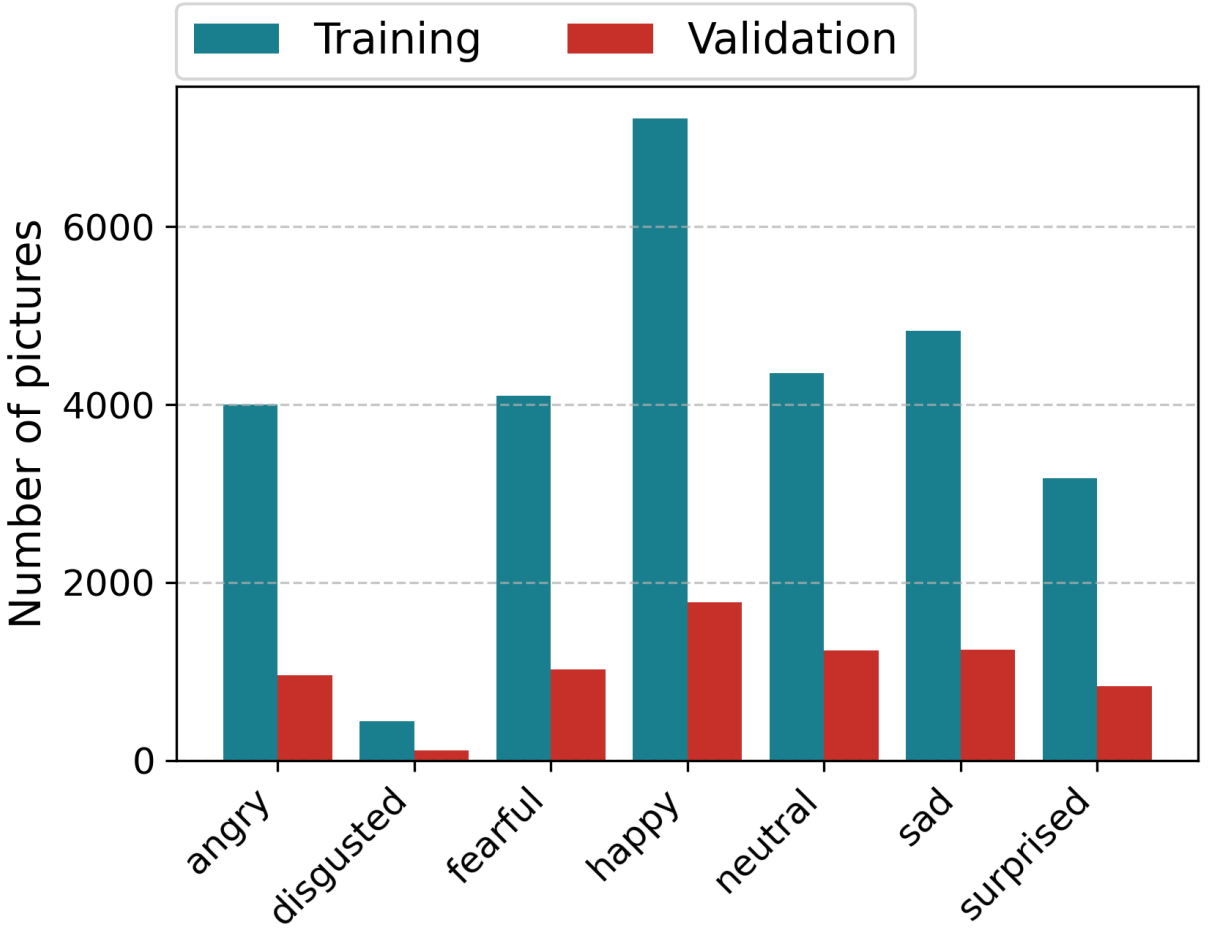
Part B:
Detecting and isolating faces using **MTCNN** in arbitrary photos (Pictures of this class)

Loss Function:
Binary Cross Entropy*
Euclidean distance*

*https://arxiv.org/pdf/1604.02878

Data – an overview

- Labelled images of faces
  - 28.097 for training
  - 7.178 for testing
- Greyscale
- 48x48 pixels
- Seven classes
- Imbalanced data set



Link to data: https://www.kaggle.com/datasets/ananthu017/emotion-detection-fer

Data - a closer look

Investigation

Data - a closer look

Investigation

- Some emotions look similar

Data - a closer look

Investigation

- Some emotions look similar
- Some emotions are unclear, while others are easy detectable

Data - a closer look

Investigation

- Some emotions look similar
- Some emotions are unclear, while others are easy detectable
- Lots of baby pictures (and cartoons)

Data - a closer look

Investigation

- Some emotions look similar
- Some emotions are unclear, while others are easy detectable
- Lots of baby pictures (and cartoons)
- Water marks and hands on face

Data - a closer look

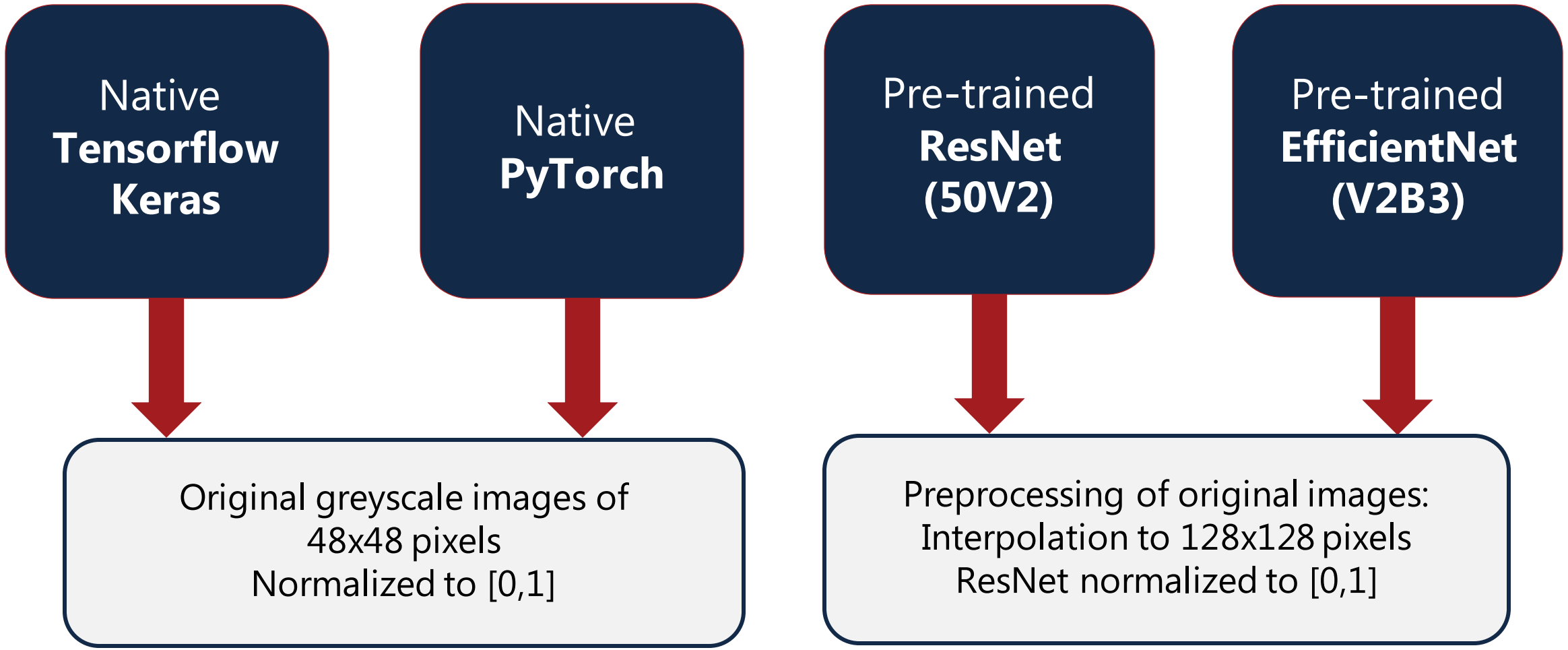
Investigation

- Some emotions look similar
- Some emotions are unclear, while others are easy detectable
- Lots of baby pictures (and cartoons)
- Water marks and hands on face

Managing expectations:
Human accuracy*: **65,5%**
Highest accuracy in litterature**: **73.3%**
Random: **14.3%**

* https://arxiv.org/pdf/1307.0414
**https://arxiv.org/pdf/2105.03588

Models


Native
Tensorflow Keras
Native
PyTorch
Pre-trained
ResNet (50V2)
Pre-trained
EfficientNet (V2B3)
Original greyscale images of 48x48 pixels
Normalized to [0,1]
Preprocessing of original images:
Interpolation to 128x128 pixels
ResNet normalized to [0,1]

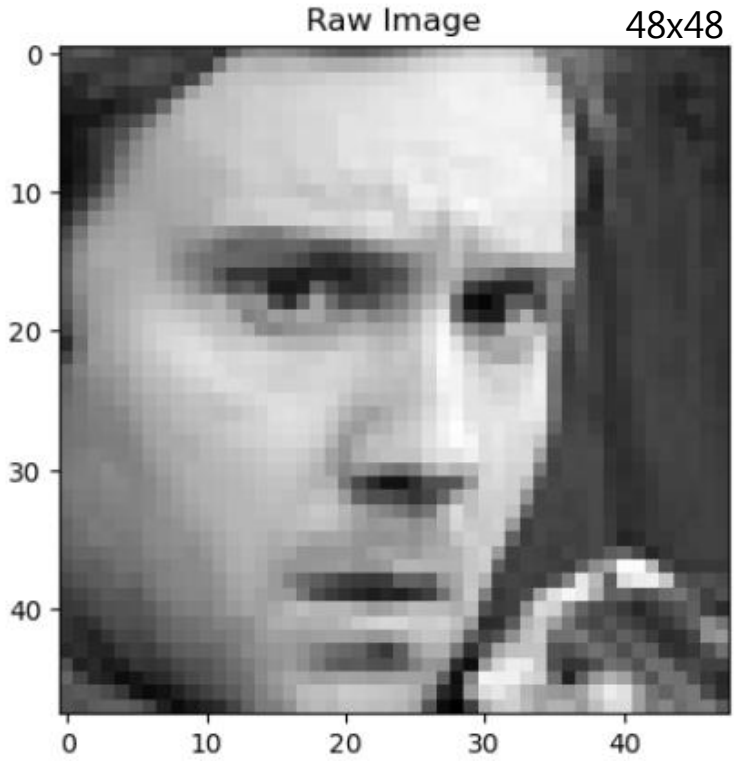
Preprocessing - Bilinear Interpolation

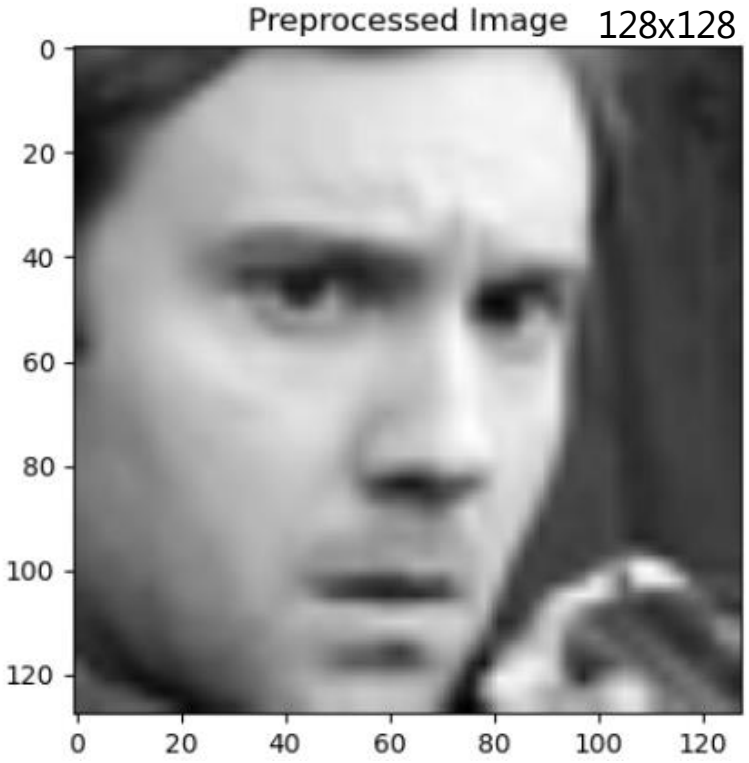
Pros:

- Pre-trained models based on higher resolution
- Easier recognizable for humans

Cons:

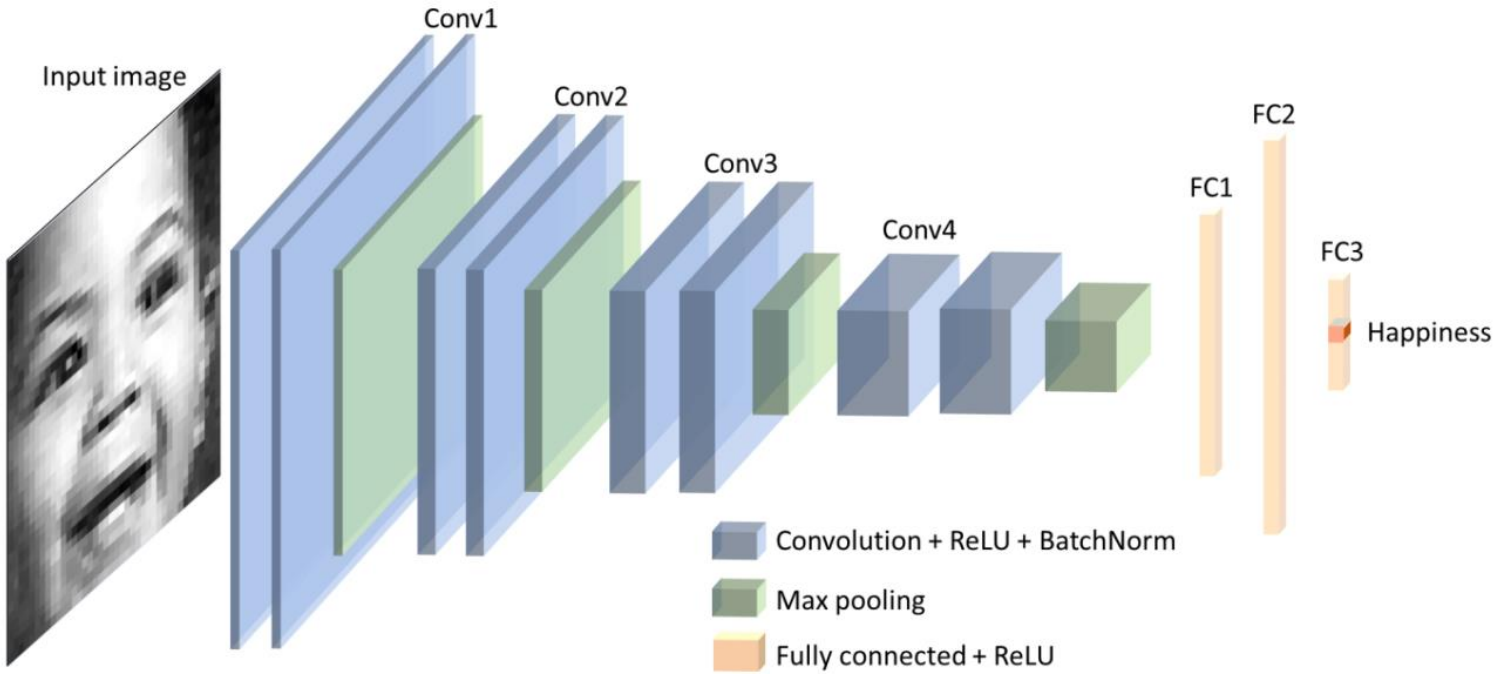
- Fade out edges - more complex kernels
- Adding artificial information?





Native Models

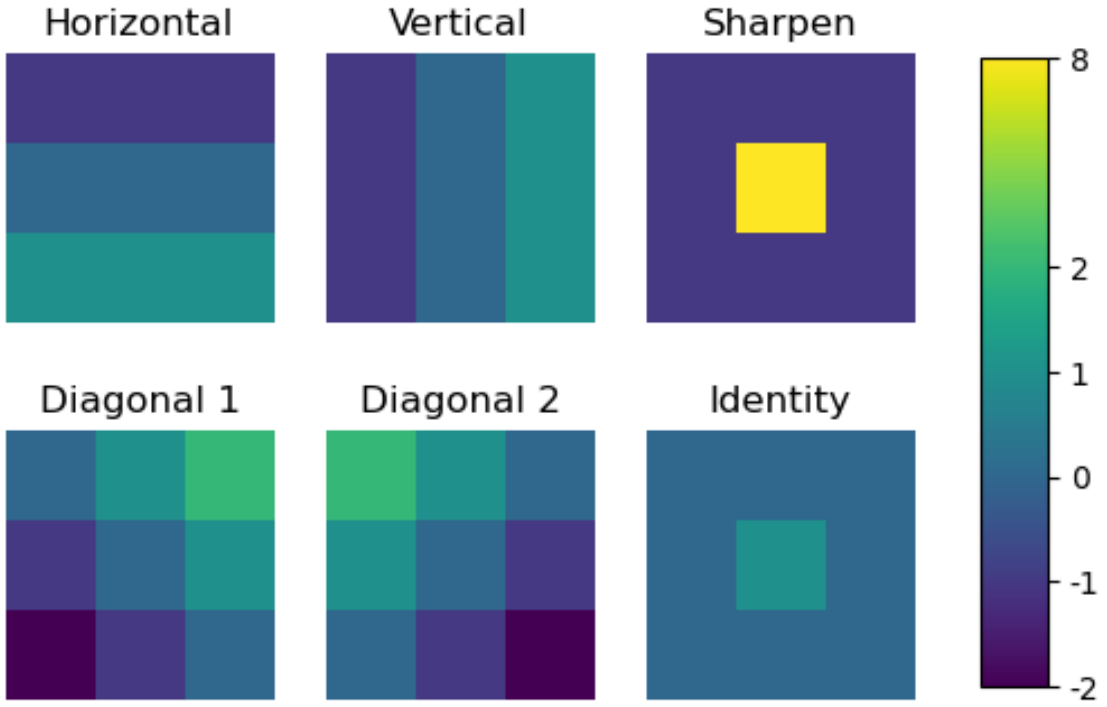
- Trained exclusively on the data
- Smaller network with more room for customization
- More inspectable

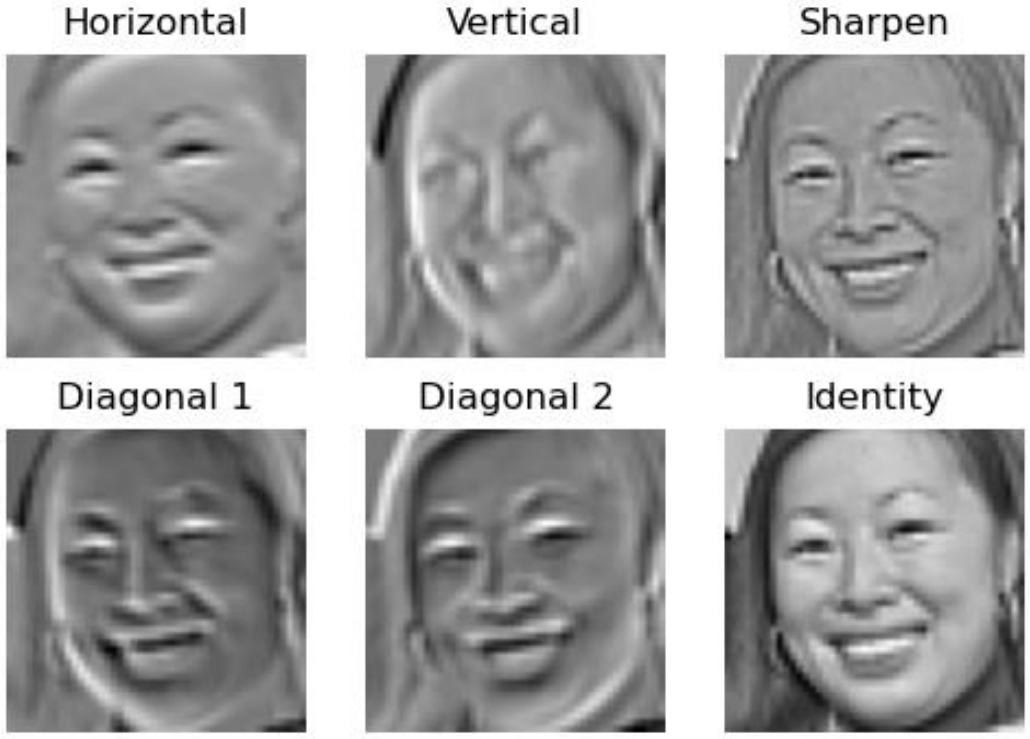


https://arxiv.org/pdf/2105.03588

Native Models

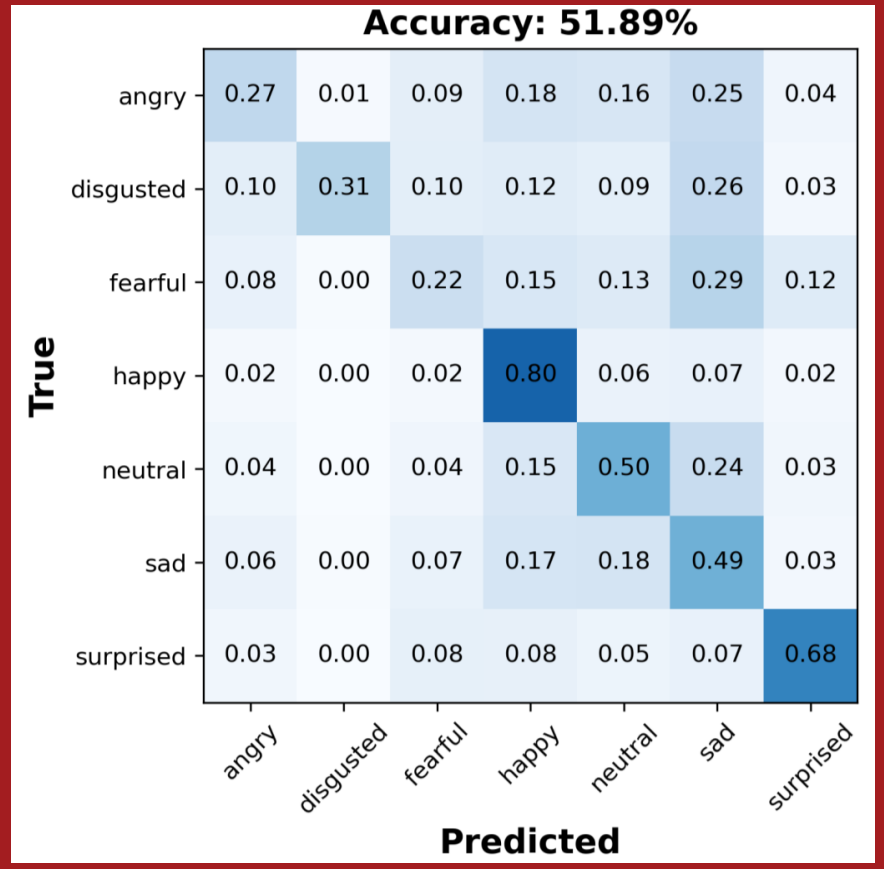
- Experimentation with custom kernels
  - Suited for capturing facial features
  - Horizontal vs. Vertical



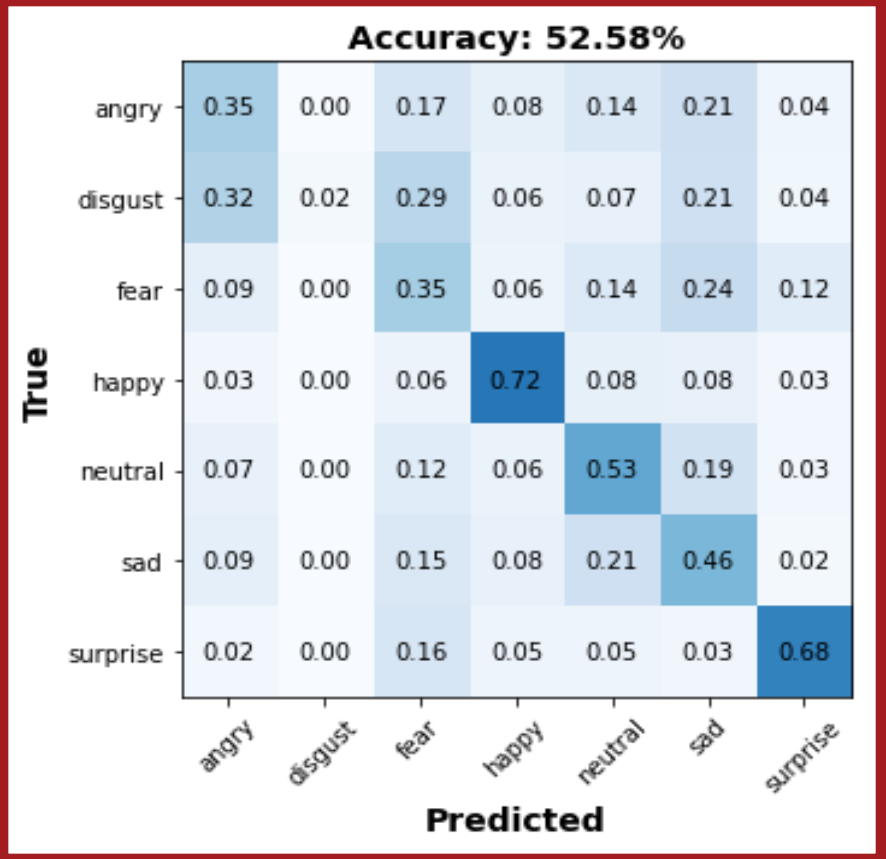


BEST NATIVE MODELS

Tensorflow Keras

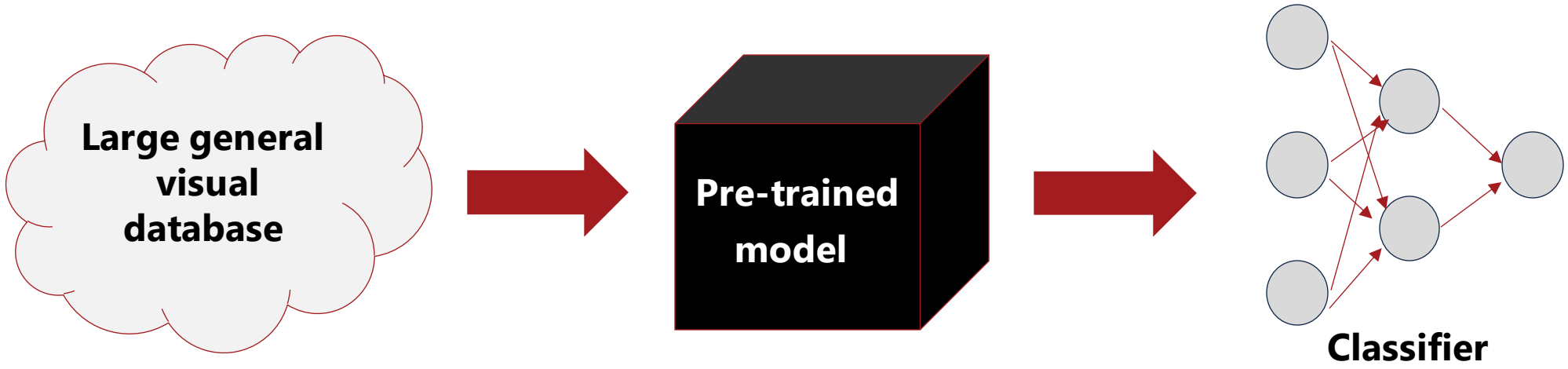


Pytorch

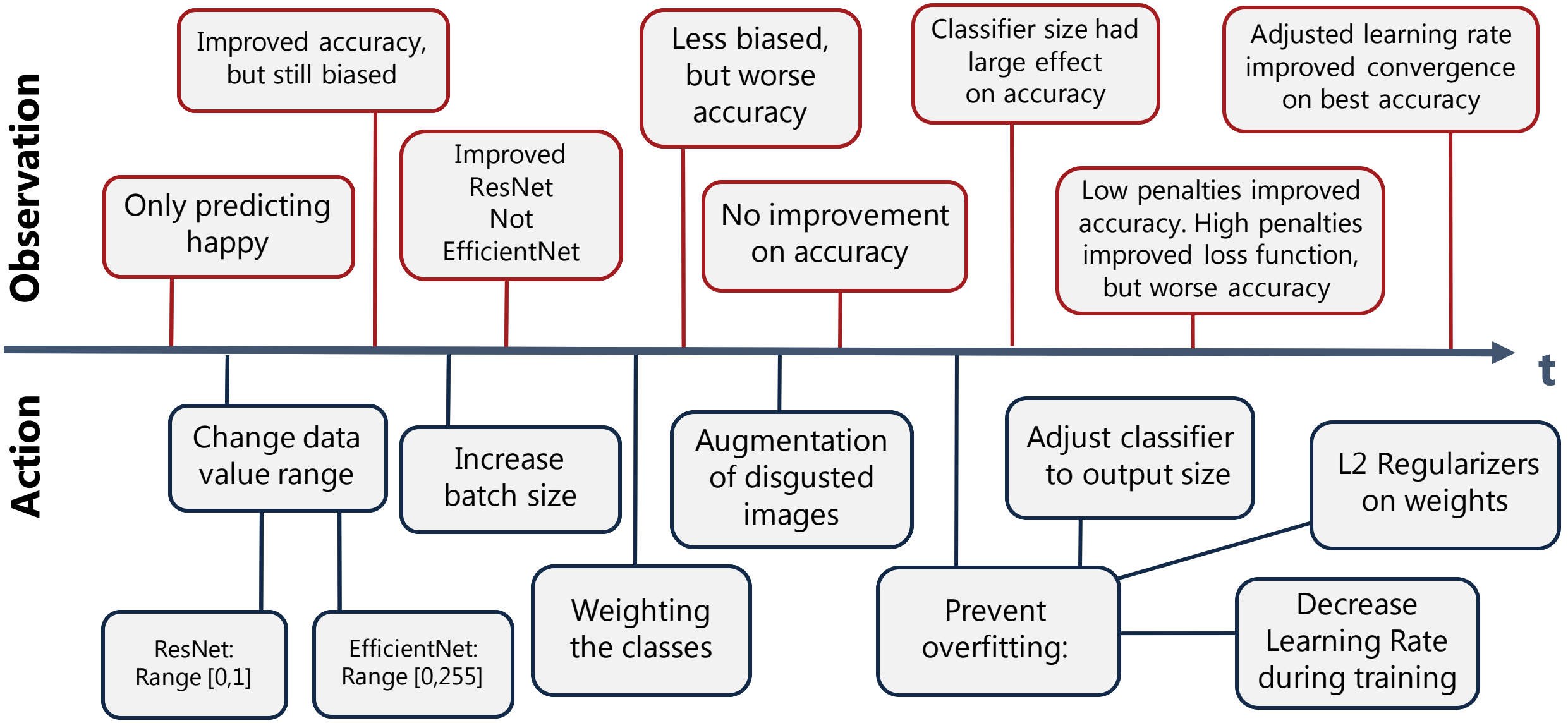


Pre-trained Models

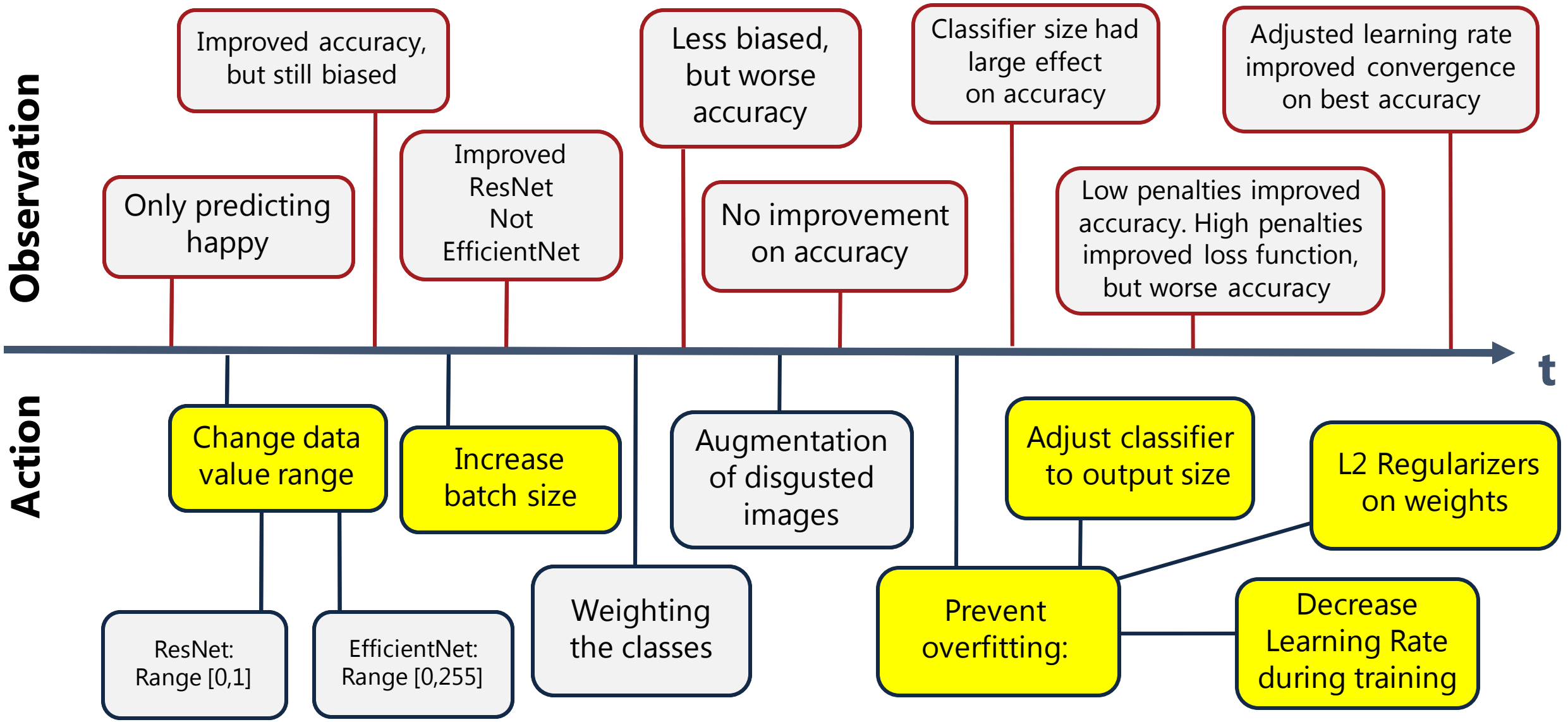
- Pre-trained CNN for image classifications:
  - ResNet & EfficientNet
  - Black box



Pre-trained Models – Learning process



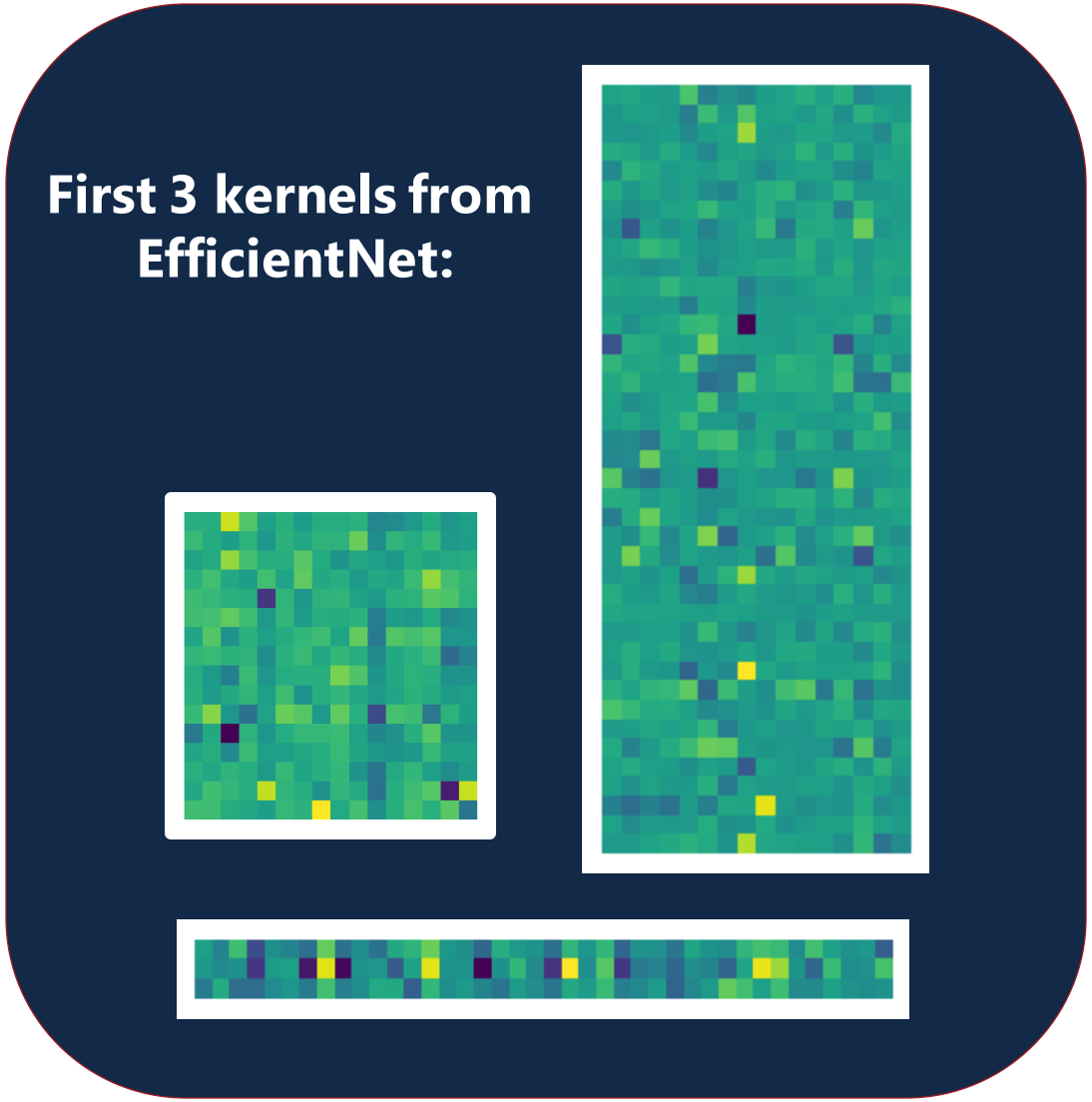
Pre-trained Models – Learning process



Pre-trained Models - What to consider

- Very large networks (EfficientNet: +400 layers)
  - Complex kernels
  - Large output for classifier

- Architecture of the model
  - Frozen vs trainable layers in base model
  - Classifier layers relative to output
  - Prevent overfitting with Regularization, Dropout and Batch normalization



Pre-trained Models

Pre-trained base model
+
Dense output layer

Accuracy:
ResNet: 49,7%
EfficientNet: 50,2%

Pre-trained base mode
+
2 hidden layers
+
Dense output layer

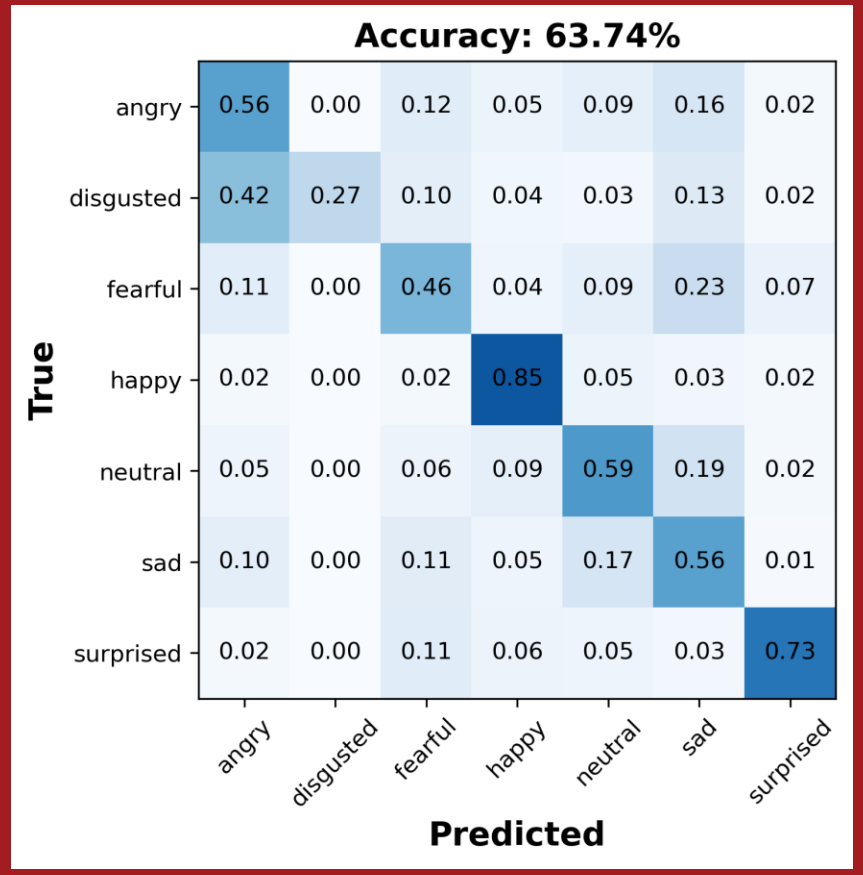
Accuracy:
ResNet: 51.2%
EfficientNet: 56,8%

Pre-trained base model
+
Fine-tuning of top layers
+
2 hidden layers
+
Dense output layer

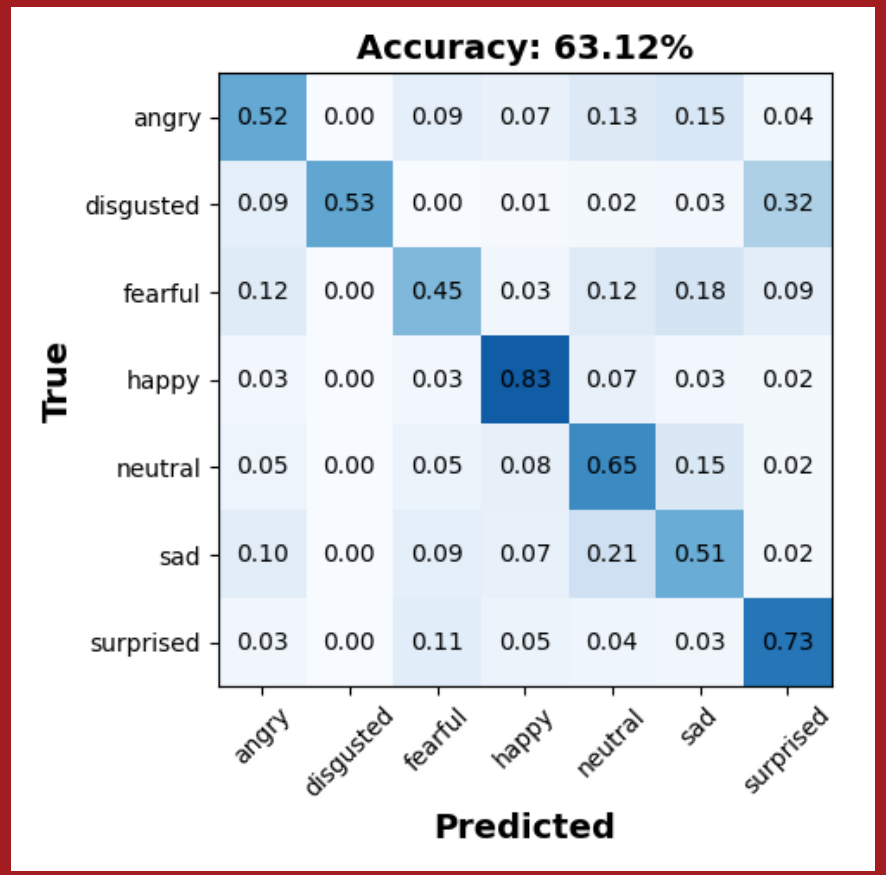
Accuracy:
ResNet: 63,7%
EfficientNet: 63,1%

BEST PRE-TRAINED MODELS

ResNet

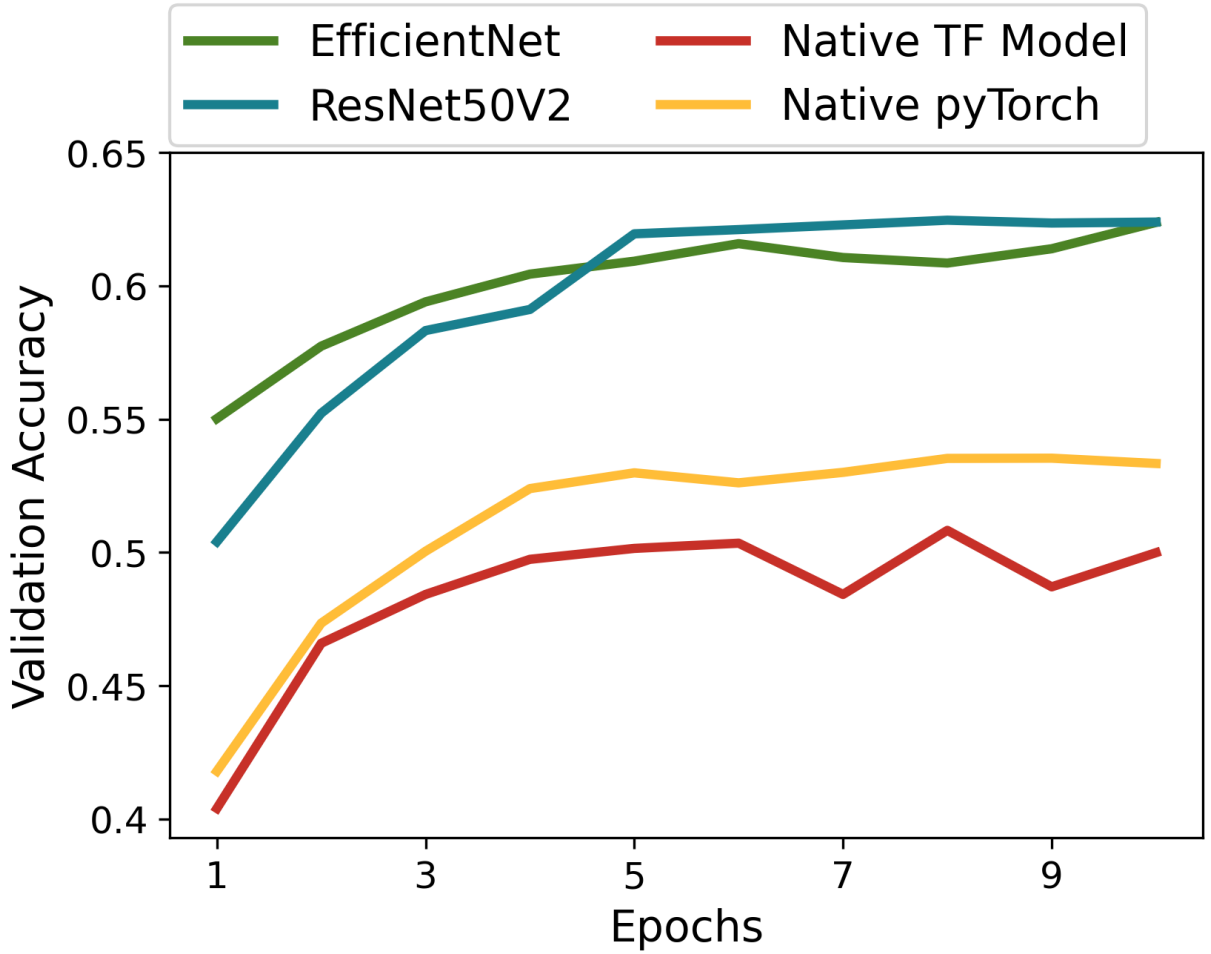


EfficientNet



Comparison of the final models

- Pre-trained gave best accuracy
  - Near human performance
  - ResNet and EfficientNet varied in dominance depending on the testdata
- PyTorch performed better that Tensorflow



Class Photos – Image segmentation

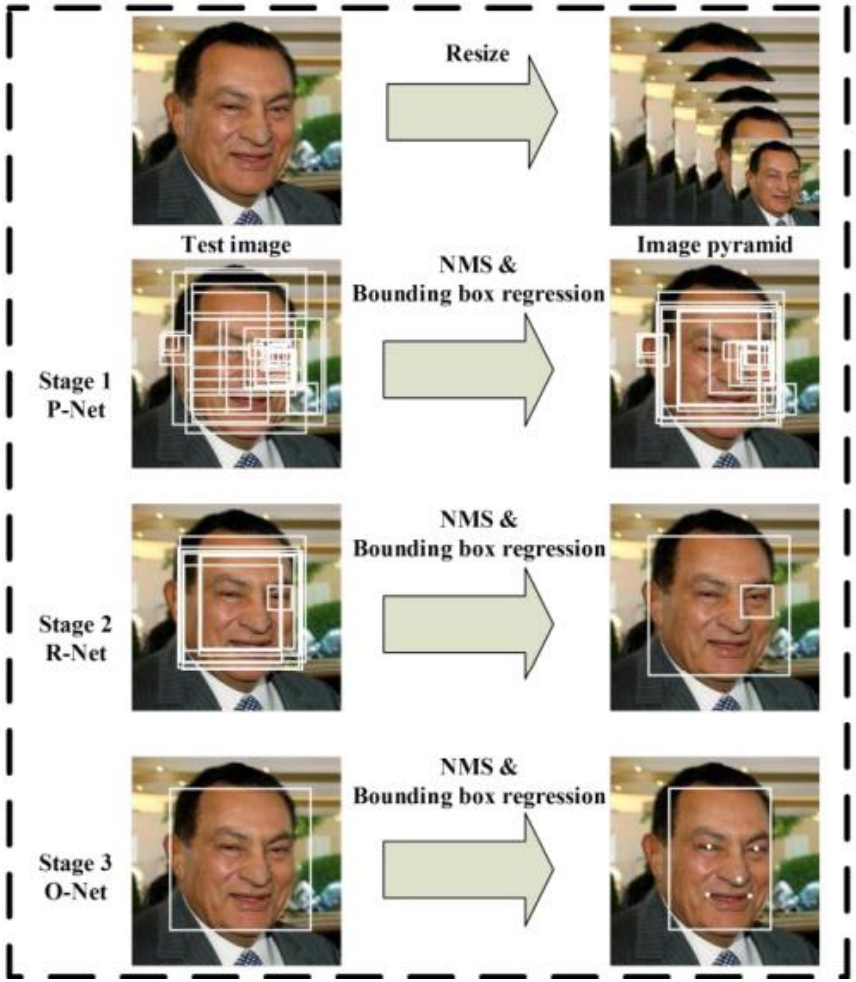
- Large scale applications requires handling versitile inputs
- Automatic face detection
- Pre-processing for model compatability

MTCNN – Multi-Task Cascaded CNN

Pre-trained structure for locating faces

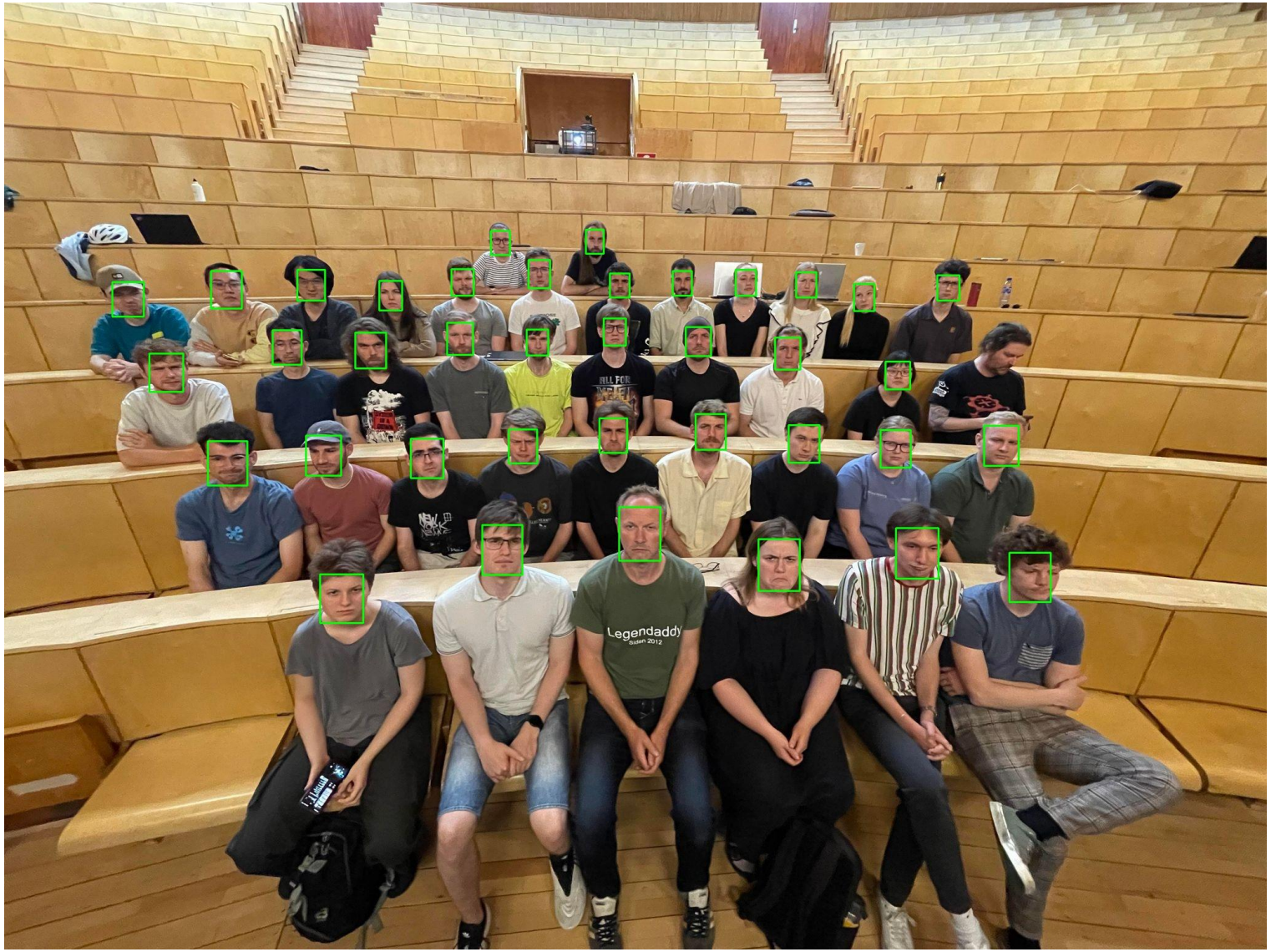
Three Constituents:

- PNet: proposes regions of interest (fast)
- RNet: refines the ROIs (Jaccard Index)
- ONet: outputs final bounds and landmarks



Segmentation Pipeline – Raw Image

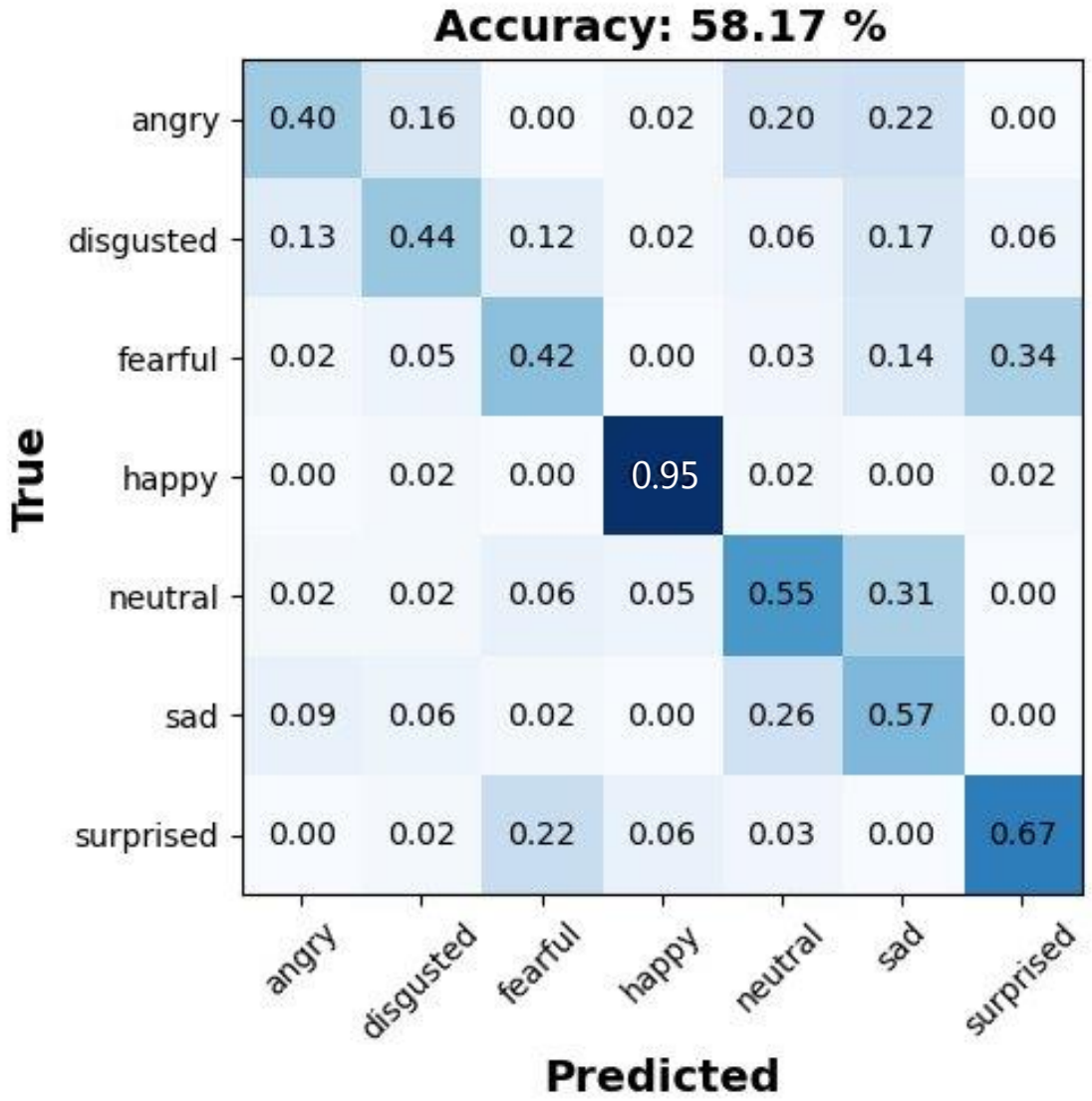
Segmentation Pipeline – Face Recognition



Segmentation Pipeline – Extraction and re-formating

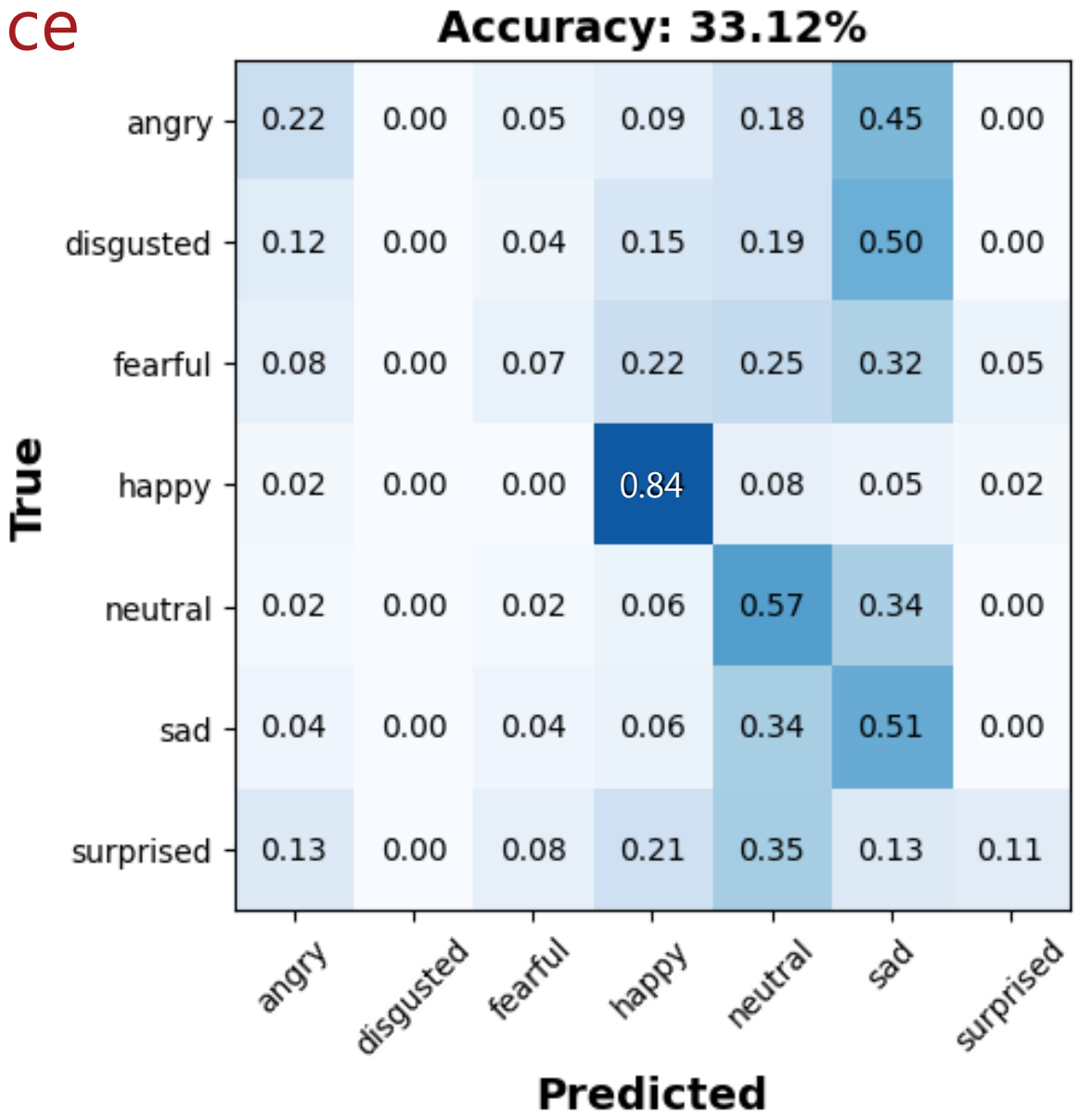
Class Photos - Human Benchmark

- Convincing recognition
- Surprised, fearful are confused
- Guess oftens defaults to sad or neutral
- **Surprisingy difficult!**
  - Low resolution
  - Loss of context



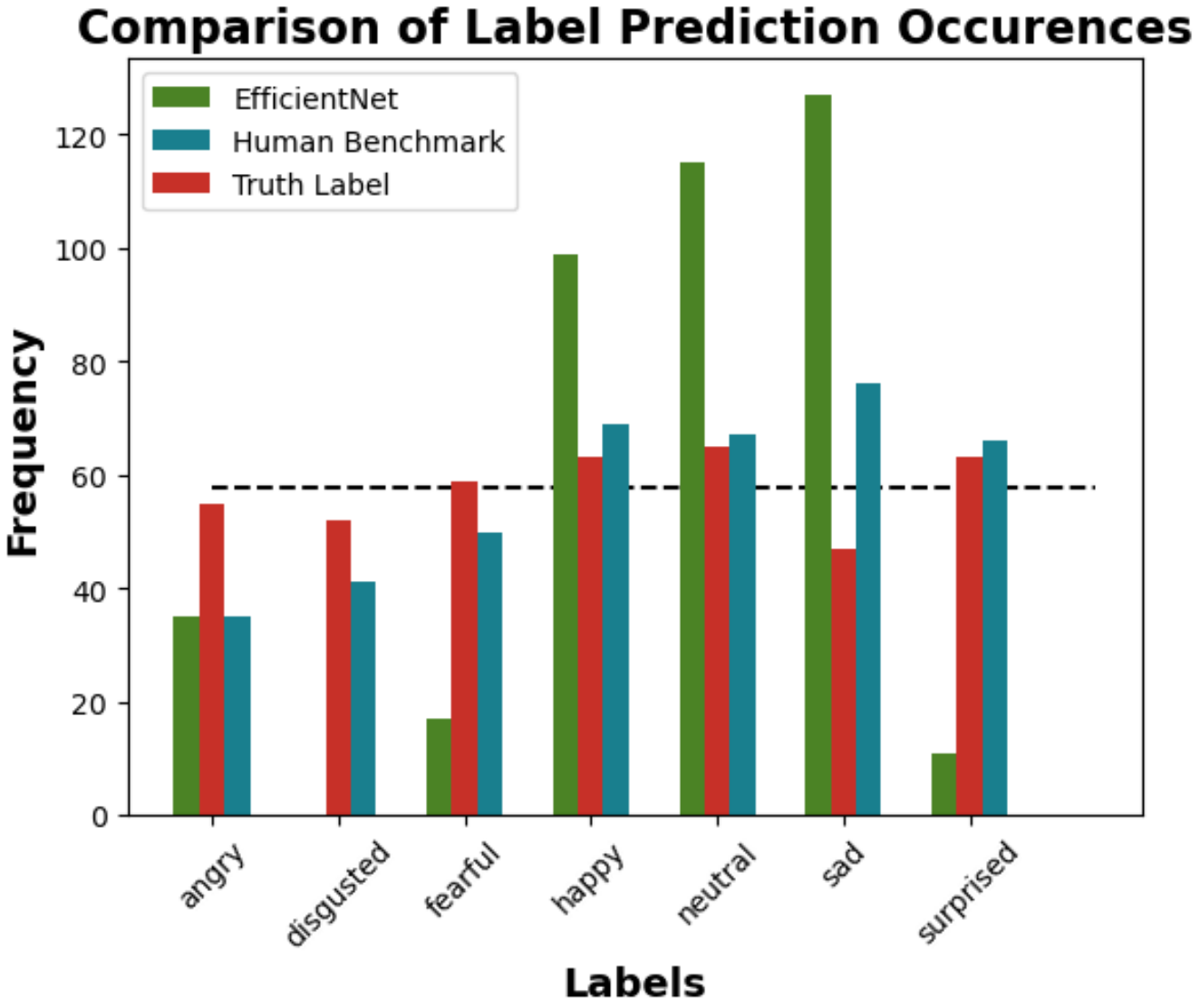
Class Photos – Model Performance

- Comparatively worse than on validation
- Happy is still easily recognisable
- What has the model really learned?



Distribution of predictions

- CNN mainly predicts: happy, neutral, and sad
- Despite balanced predictions on the validation set!
- Evidently the learned triggers do not translate to student photos



La Galerie de Troels

True label: angry
Predicted label: angry

True label: fearful
Predicted label: happy

True label: neutral
Predicted label: neutral

True label: surprised
Predicted label: fearful

True label: disgusted
Predicted label: sad

True label: happy
Predicted label: happy

True label: sad
Predicted label: sad

Accuracy on Troels:
57,1%

CONCLUSION

- Emotion classification is a very complex problem!

- Comparable to human performance
- Within 10 percent points of best in litterature

- Keras vs. Pytorch
- Native vs. Pre-trained

- Future work

Questions & comments?

Link to Github: https://github.com/Emilie-Jessen/AppML-Final-Project-2024/tree/main

Appendix 0

Introduction to topic and data

Contributions

All group members participated evenly.

Topic

Goal

The goal is to train different CNN models – both native and pre-trained algorithms – to recognize facial expressions displaying seven different emotions. We will investigate the differences between native and pre-trained models. The end objective is to use the model on pictures taken of the Applied Machine Learning class of 2024. To reach our end objective, we must utilize a data segmentation algorithm (MTCNN) to identify the faces in the class pictures before applying our models. The project can therefore be considered in two parts: 1) training of CNN's, 2) segmentation by MTCNN and applying CNN's.

Data

Presentation

The dataset contains 35,685 examples of 48x48 pixel gray scale images of faces divided into train and test dataset. Images are categorized based on the emotion shown in the facial expressions (happiness, neutral, sadness, anger, surprise, disgust, fear). The data set can be found here: https://www.kaggle.com/datasets/ananthu017/emotion-detection-fer

Generation

The data set was generated by feeding the Google image search engine a list of 184 emotion related keywords. These were then combined with another list of keywords related to the type of person pictured. The top 1000 results from each query were then processed to isolate the faces and human labelers then doublechecked the labeling of the emotions as well as looked for duplicate images. A more thorough review of the data generation can be found here: https://arxiv.org/pdf/1307.0414

Concerns

48x48 pixels are a small image size and facial features could be lost in the low resolution of the images. Some emotions resemble one another, and in some images, the displayed emotion is vague, whereby the labeling can be questionable. In addition, images of babies and cartoon characters are included, which is also questionable. The data set is unbalanced with the "disgust"-class being under-represented. All these factors could complicate the task, and human accuracy on this data set* has been reported to be 65,5%. The highest accuracy on this data set found in literature** is 73,3%.

* https://arxiv.org/pdf/1307.0414, **https://arxiv.org/pdf/2105.03588

Preprocessing

Normalization

For the two native models and ResNet50V2 the data value range is [0,1], while the EfficientNetV2B3 model takes data input values in the range [0,255], which corresponds to the greyscale-range. An important lesson when dealing with a pre-trained model is confirming the data value range before starting the training.

Rescaling

The two native models work with images of size 48x48 pixels. Both of the pre-trained models are trained on larger images, and we therefore found it a necessity to increase the resolution of the images to 128x128 pixels, as this was the minimum input data size for EfficientNetV2B3*. ResNet50V2 can handle smaller data sizes**, but we decided to keep the resolution equal for the two pre-trained models. The rescaling was done with bilinear interpolation***. We acknowledge, that the interpolation is not providing more information for our model, but our experience points to interpolation increases the accuracy of the two pre-trained models.

Greyscale to RGB

The two native models work with images in greyscale, whereby the shape of the images is: [48, 48, 1]
The two pre-trained models work with images in RGB, and therefore the images must be transformed to RGB. This is done with Tensorflow Keras function grayscale_to_rgb(), which essentially is expanding the single greyscale channel to the three RGB channels. The shape of the images therefore become [128, 128, 3]

*https://arxiv.org/abs/2104.00298, **https://keras.io/api/applications/resnet/, ***https://en.wikipedia.org/wiki/Bilinear_interpolation

Appendix 1

Native models

Native TF Model

Model

The first native model is created using Keras through Tensorflow. The model is a convolutional neural network with no pretrained kernels or weights. It is chosen due to easy implementation as an initial model to work with.

Input

The input is the 48x48 greyscale pictures from the dataset, but rescaled from [0,255] to [0,1]. One of the upsides to using the native model is that it can be tailored to the raw dataset, removing the need for much preprocessing as is the case for the pre-trained network.

Native TF Model

Custom Kernels

For the first layer of the CNN, we defined custom kernels to improve the initialization. Especially the horizontal kernel seems well suited for facial features. Using custom kernels has the added benefit of detering the model from exclusively guessing happy due to the overrepresentation in the data set.

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

-1	-1	-1
-1	8	-1
-1	-1	-1

0	1	2
-1	0	1
-2	-1	0

2	1	0
1	0	-1
0	-1	-2

0	0	0
0	1	0
0	0	0

Native TF Model

Layer (type)	Output Shape	Param #
conv2d_29 (Conv2D)	(None, 46, 46, 6)	60
conv2d_30 (Conv2D)	(None, 44, 44, 128)	7,040
max_pooling2d_18 (MaxPooling2D)	(None, 22, 22, 128)	0
conv2d_31 (Conv2D)	(None, 20, 20, 64)	73,792
max_pooling2d_19 (MaxPooling2D)	(None, 10, 10, 64)	0
conv2d_32 (Conv2D)	(None, 8, 8, 32)	18,464
flatten_9 (Flatten)	(None, 2048)	0
dense_27 (Dense)	(None, 64)	131,136
dense_28 (Dense)	(None, 32)	2,080
dense_29 (Dense)	(None, 7)	231

Total params: 232,803 (909.39 KB)

Trainable params: 232,803 (909.39 KB)

Layers

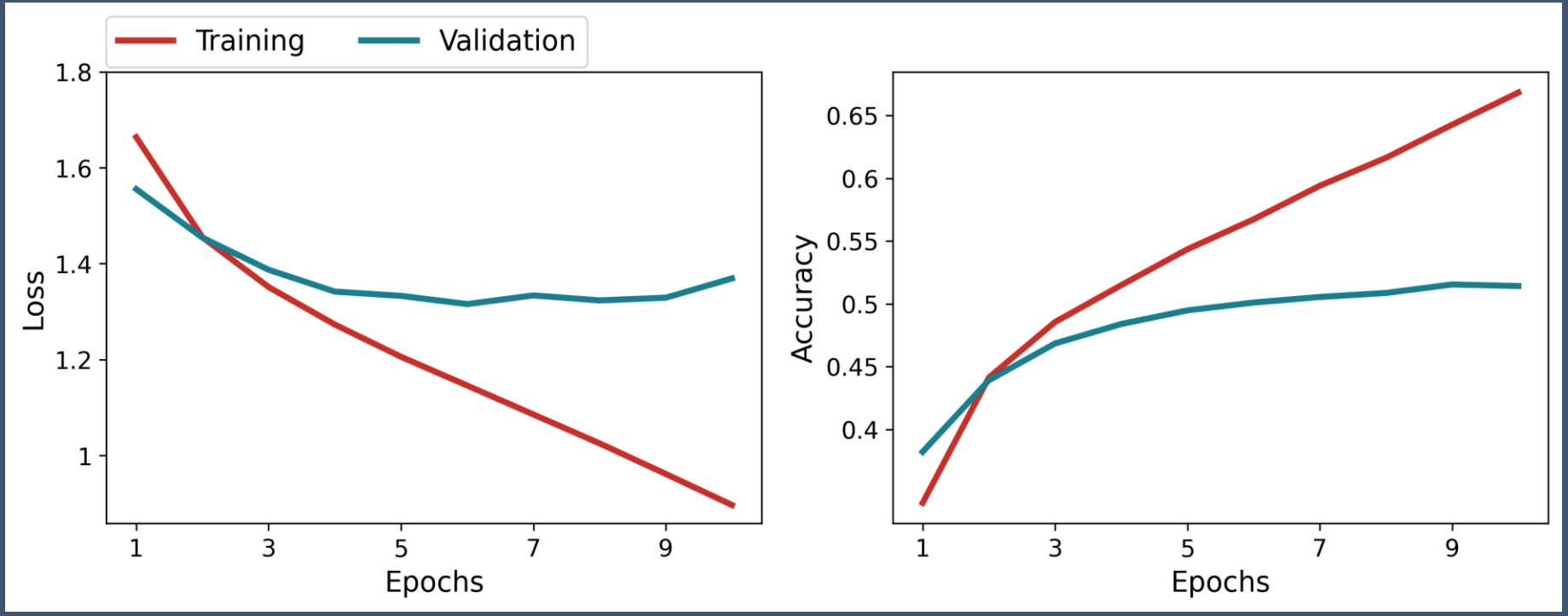
The choice of layers is based on trial an error with a focus on a steady reduction of the dimensionionality. The initial convolution layer consists of the custom designed kernels, with then alternating convolution and maxpooling layers.

The final model contains 232.803 trainable parameters.

Native TF Model

Training

The model is optimized using keras.optimizers.SGD, with a learning rate of 0.002 and momentum 0.9, minimizing categorical cross entropy.



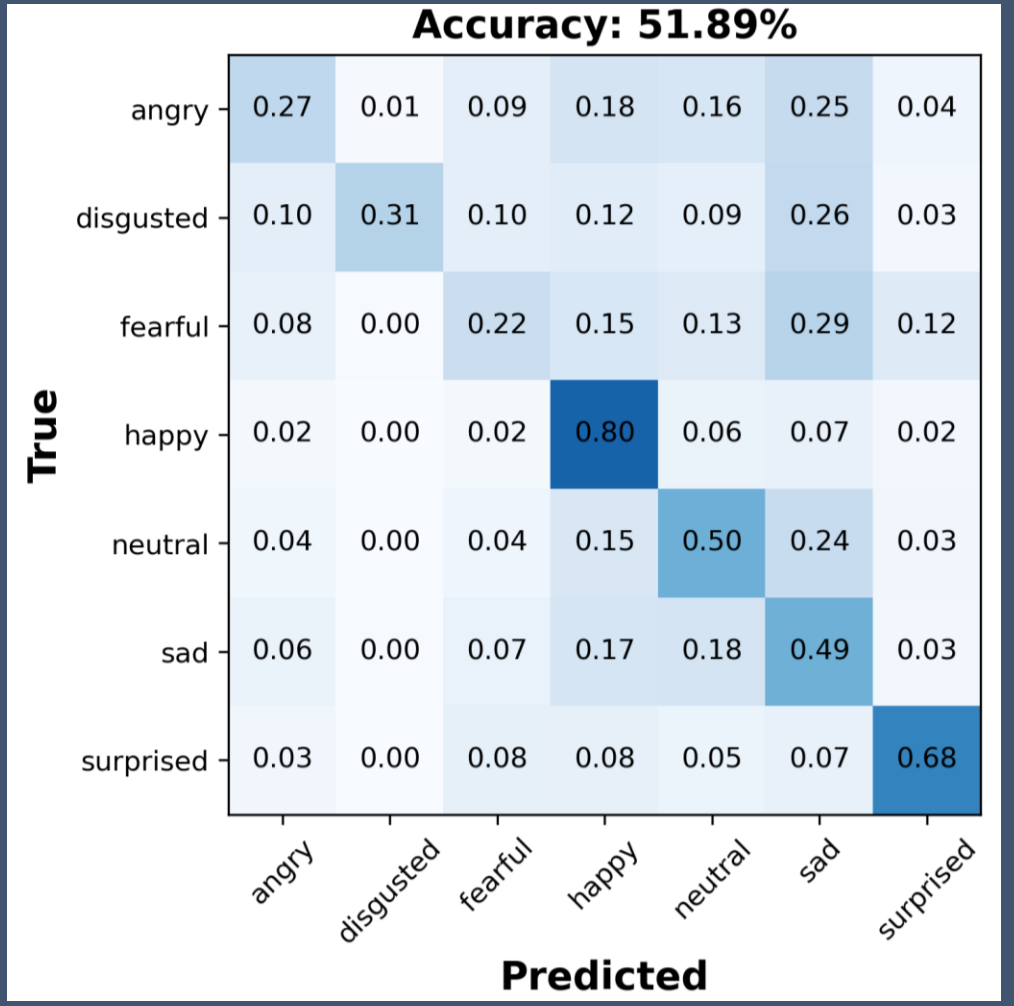
Native TF Model

Results

The model is evaluated on the blind test dataset of 7.178 samples resulting in an accuracy of 51.59% and a loss of 1.336. To judge the final model, a confusion matrix is utilized, where a perfect model is completely diagonal.
We see that the model is capable of recognising happy and surprised, perhaps due to the more easily recognized features such as smiles or open mouths.

In the underepresented catagory of disgusted, the model is capable of making guesses, but does so with a comparable frequency to the dataset and only when certain.

When in doubt the model also seems to default to the sad label.



Native PyTorch Model

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 128, 48, 48]	1,280
ReLU-2	[-1, 128, 48, 48]	0
Conv2d-3	[-1, 128, 48, 48]	147,584
ReLU-4	[-1, 128, 48, 48]	0
BatchNorm2d-5	[-1, 128, 48, 48]	256
MaxPool2d-6	[-1, 128, 24, 24]	0
Conv2d-7	[-1, 64, 24, 24]	73,792
ReLU-8	[-1, 64, 24, 24]	0
Conv2d-9	[-1, 64, 24, 24]	36,928
ReLU-10	[-1, 64, 24, 24]	0
BatchNorm2d-11	[-1, 64, 24, 24]	128
MaxPool2d-12	[-1, 64, 12, 12]	0
Conv2d-13	[-1, 32, 12, 12]	18,464
ReLU-14	[-1, 32, 12, 12]	0
Conv2d-15	[-1, 32, 12, 12]	9,248
ReLU-16	[-1, 32, 12, 12]	0
BatchNorm2d-17	[-1, 32, 12, 12]	64
MaxPool2d-18	[-1, 32, 6, 6]	0
Flatten-19	[-1, 1152]	0
Linear-20	[-1, 256]	295,168
ReLU-21	[-1, 256]	0
Linear-22	[-1, 64]	16,448
ReLU-23	[-1, 64]	0
Linear-24	[-1, 7]	455

Total params: 599,815
Trainable params: 599,815

Layers

The CNN was built trying to replicate the VGG architecture found in the best performing model in the litterature. The linear layers were adjusted to improve the efficiency and reduce overfitting.

The network has 4 blocks of convolution layers separated by maxpooling for efficiency. These blocks extract features which the fully connected layers then classify into the 7 emotions.

Native PyTorch Model

Training

The model is optimized using SGD optimizer, with a learning rate of 0.0015, batch size of 64 and momentum 0.9, minimizing categorical cross entropy. L2 regularization was implemented by setting the weight decay to 0.001 and batch size to 64.

Several sizes of the fully connected layers were tested with a few sizes with no big difference observed on performance.

Native PyTorch Model

Results

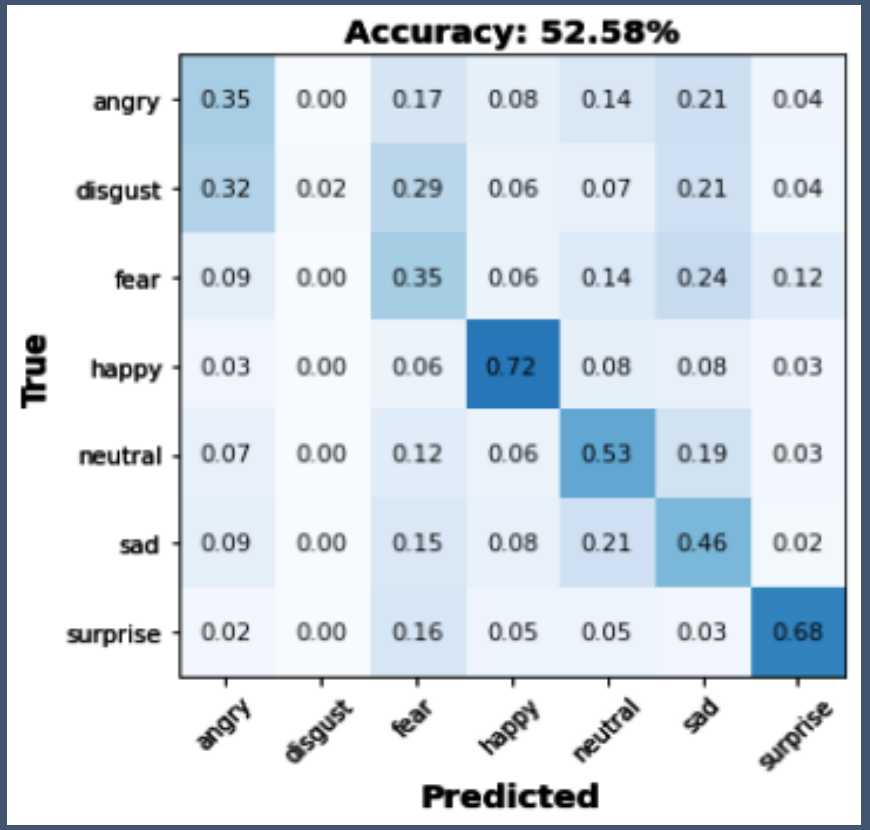
The model is evaluated on the blind test and achieves an accuracy of 52.58% and a validation loss of 1.333.

The model performs well on the data-rich emotions, happy and surprise, as well as performing decently on neutral and sad though with an overlap between the classification of the two.

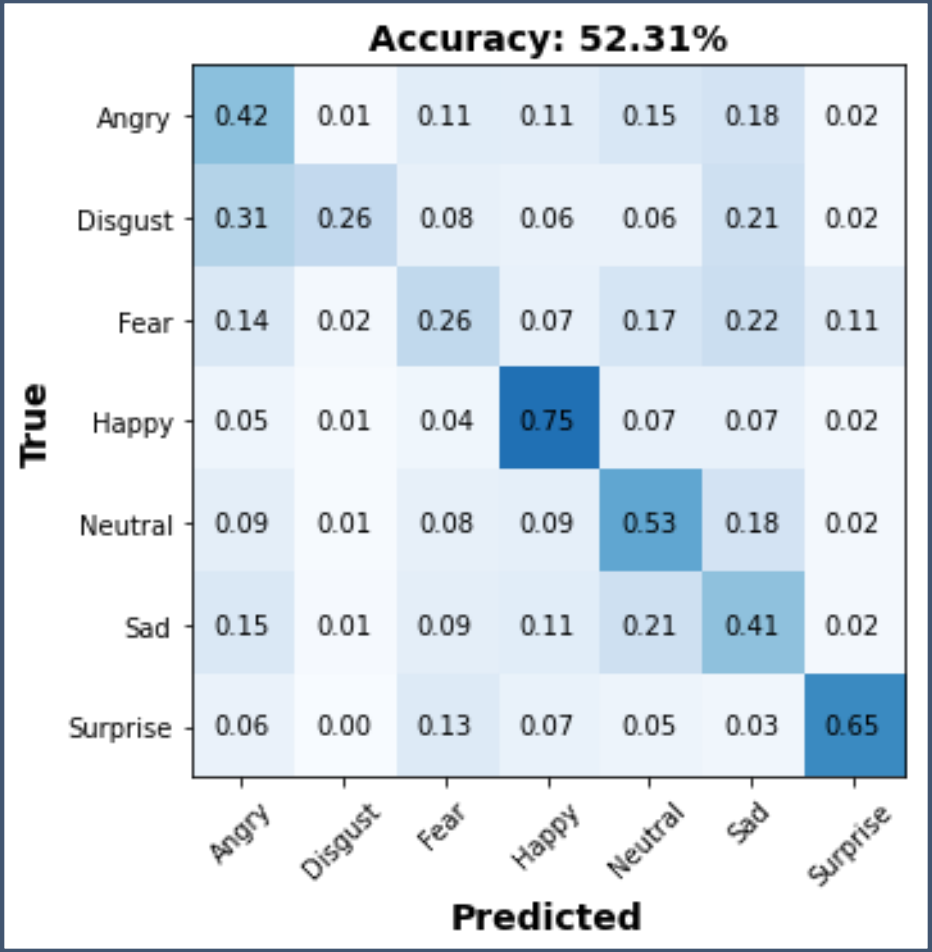
The model struggles with fear and angry and barely predicts disgust at all.

When in doubt the model also seems to default to the sad label.

This model is not too well optimized and a more thorough search of hyperparameters is necessary. Other improvements include running more epochs with a learning rate scheduler as well as tuning the momentum.



Data Augmentation PyTorch



Additional disgusted data

A random combination of the transformations: Rotation, translation, shear and flip was applied to the disgusted data to even out the data imbalance.

This produced a validation accuracy of 58.5%, though the accuracy of the model on the test data was much lower at 52.3%. This behaviour was also observed in the EfficientNet.

It does improve the ability to produce some disgust predictions though these are still not common enough.

Appendix 2

Pre-trained models

Pre-trained models

General idea of pre-trained models

Two different pre-trained image classification models were chosen for this project: ResNet50V2 and EfficientNetV2B3. Both models are trained on the large visual database ImageNet*. The idea behind using a pre-trained model, which is trained on a large and general data set, is that it will serve as a generic model of the visual world. The generic model will take advantage of pre-learned feature maps to be specialized for the task at hand. Specializing the model for the current task is done by adding a new classifyer on top of the pre-trained layers. The models come with pre-established weights, which can either be frozen or trainable for fine-tuning. When the weights of the base model are frozen, only the weights in the top classification layer will be trained for the specific purpose. For fine-tuning of the model, the layers of the base model can be unfrozen, whereby the weights of the base model will be trained as well**. We chose the unfreeze some of the top layers and with a small learning rate (order 10e-3) fine-tune the weights. The intuition was that the model would be in the neighbourhood of a minimum and by choosing a small learning rate, we could nudge the model slightly closer to the minimum to gain the last few percentages of performance.

*https://www.image-net.org/, **https://www.tensorflow.org/tutorials/images/transfer_learning

ResNet50V2

ResNet50v2 is a large pre-trained convolutional neural network that excels in high-accuracy image classification.
To adapt the pre-trained ResNet50v2 for facial emotion recognition, a Sequential classifier was appended to the top of the model. The base model's output reduced in size and dimensions using Global average pooling to interface with fully connected Dense layers, with Dropout layers and Batch normalization interspersed to mitigate overfitting. Care was taken to maintain reasonable dimensionality transitions between layers to prevent loss of information.

Training the model

The model was trained using the NAdam-optimizer with a learning rate of 0.0005 to minimize the Categorial-Cross-Entropy-loss with the monitored metric being accuracy. The training was done with a batch size of 64 over 10 epochs.
A Keras callback function was used to reduce the Learning rate on plateau with a patience of only 1 epoch a free choice of reduction factor.

Fine-tuning of the base model

The architecture of ResNet50v2 contains 191 layers. For fine-tuning the last 50 layers were unfrozen and finetuned using a learning rate of 0.0005. This resulted in a large improvement from 50.1% to 63,7% in accuracy.

The architecture of ResNet50V2: https://www.researchgate.net/publication/359153551_Deep_Learning-Based_Digital_Image_Forgery_Detection_System

ResNet50V2

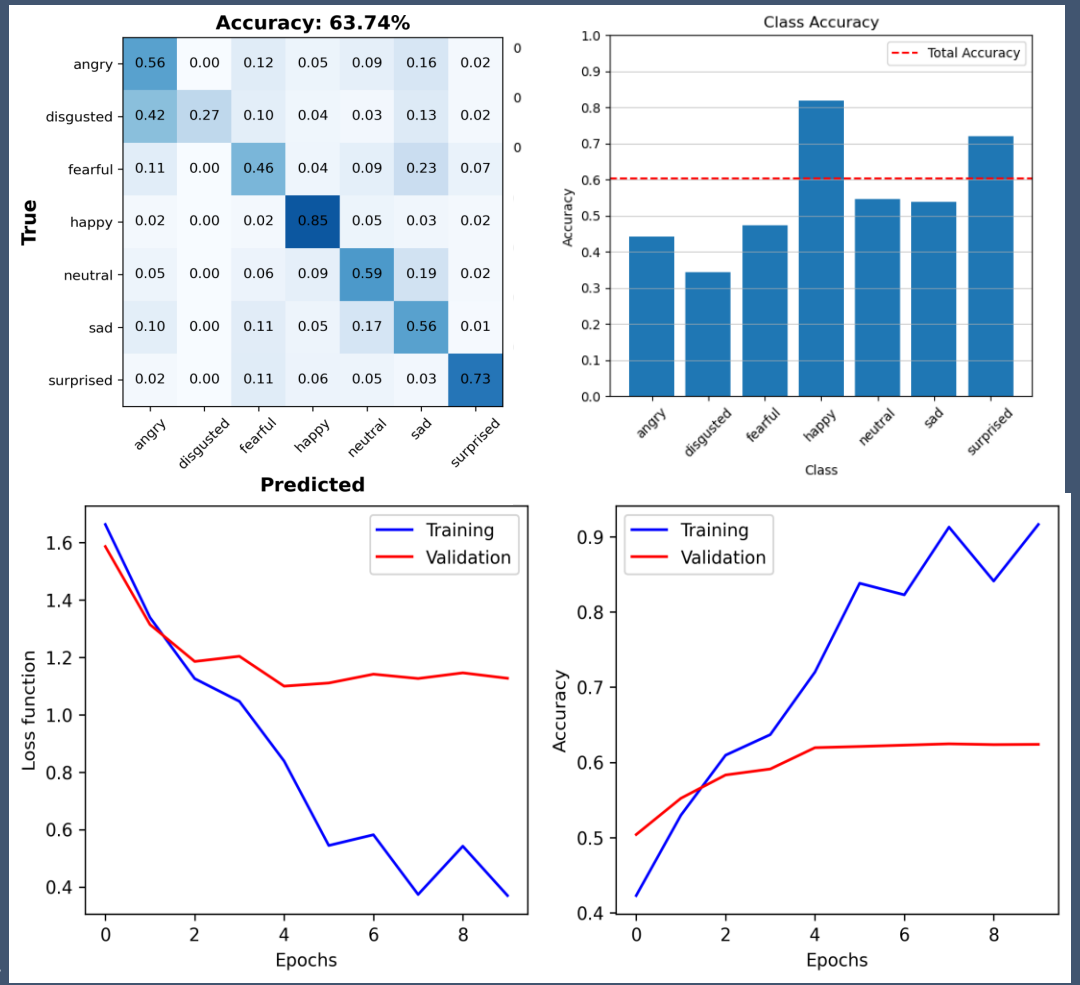
Model Fine-tuning the top 50 layers:

Layer (type)	Output Shape	Param #
resnet50v2 (Functional)	(None, 2048)	23,564,800
dense_34 (Dense)	(None, 256)	524,544
batch_normalization_20 (BatchNormalization)	(None, 256)	1,024
dropout_22 (Dropout)	(None, 256)	0
dense_35 (Dense)	(None, 32)	8,224
batch_normalization_21 (BatchNormalization)	(None, 32)	128
dropout_23 (Dropout)	(None, 32)	0
dense_36 (Dense)	(None, 7)	231

Total params: 24,098,951 (91.93 MB)

Trainable params: 16,622,663 (63.41 MB)

Non-trainable params: 7,476,288 (28.52 MB)



The architecture of ResNet50V2: https://www.researchgate.net/publication/359153551_Deep_Learning-Based_Digital_Image_Forgery_Detection_System

ResNet50V2

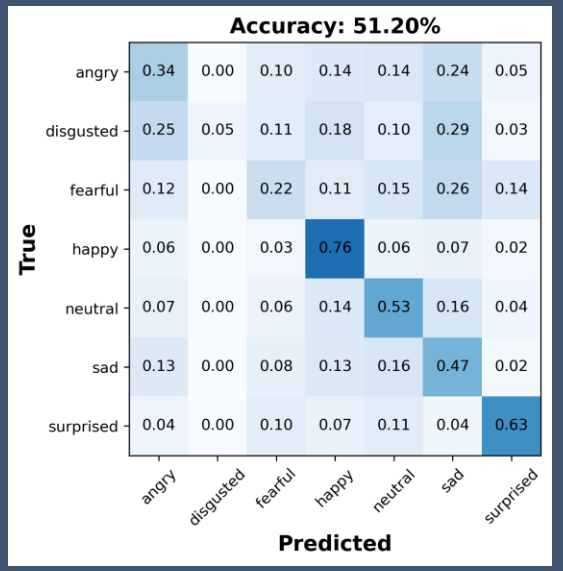
2 hidden layers, NO fine tuning:

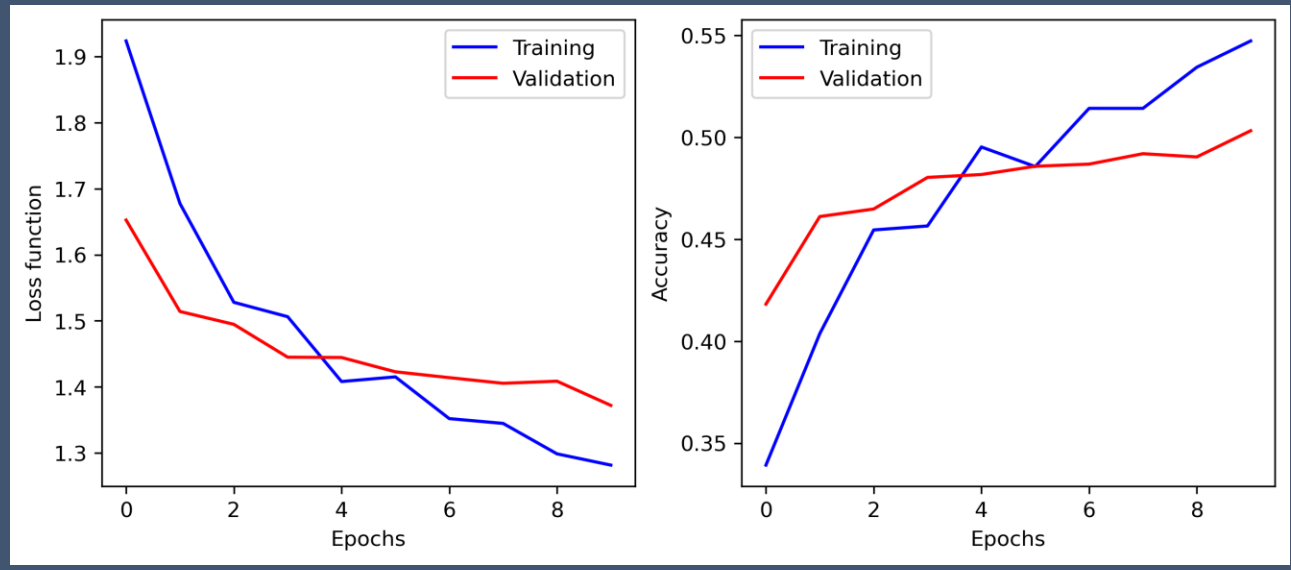
Layer (type)	Output Shape	Param #
resnet50v2 (Functional)	(None, 2048)	23,564,800
dense_31 (Dense)	(None, 256)	524,544
batch_normalization_18 (BatchNormalization)	(None, 256)	1,024
dropout_20 (Dropout)	(None, 256)	0
dense_32 (Dense)	(None, 32)	8,224
batch_normalization_19 (BatchNormalization)	(None, 32)	128
dropout_21 (Dropout)	(None, 32)	0
dense_33 (Dense)	(None, 7)	231

Total params: 24,098,951 (91.93 MB)

Trainable params: 533,575 (2.04 MB)

Non-trainable params: 23,565,376 (89.89 MB)





The architecture of ResNet50V2: https://www.researchgate.net/publication/359153551_Deep_Learning-Based_Digital_Image_Forgery_Detection_System

ResNet50V2

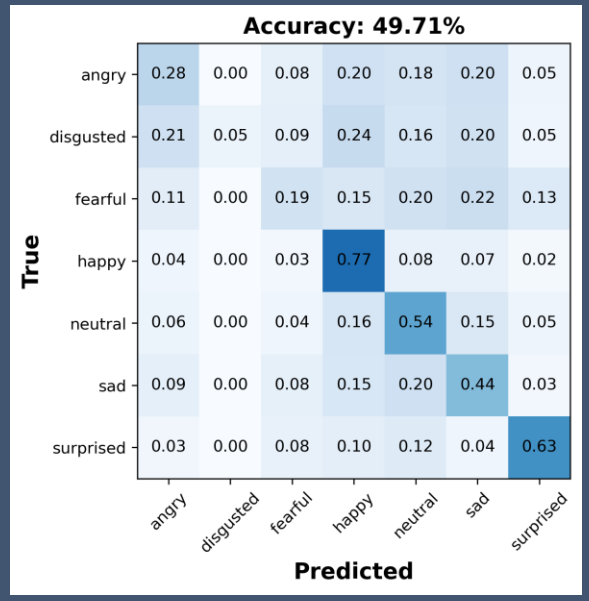
Only 1 Dense layer output:

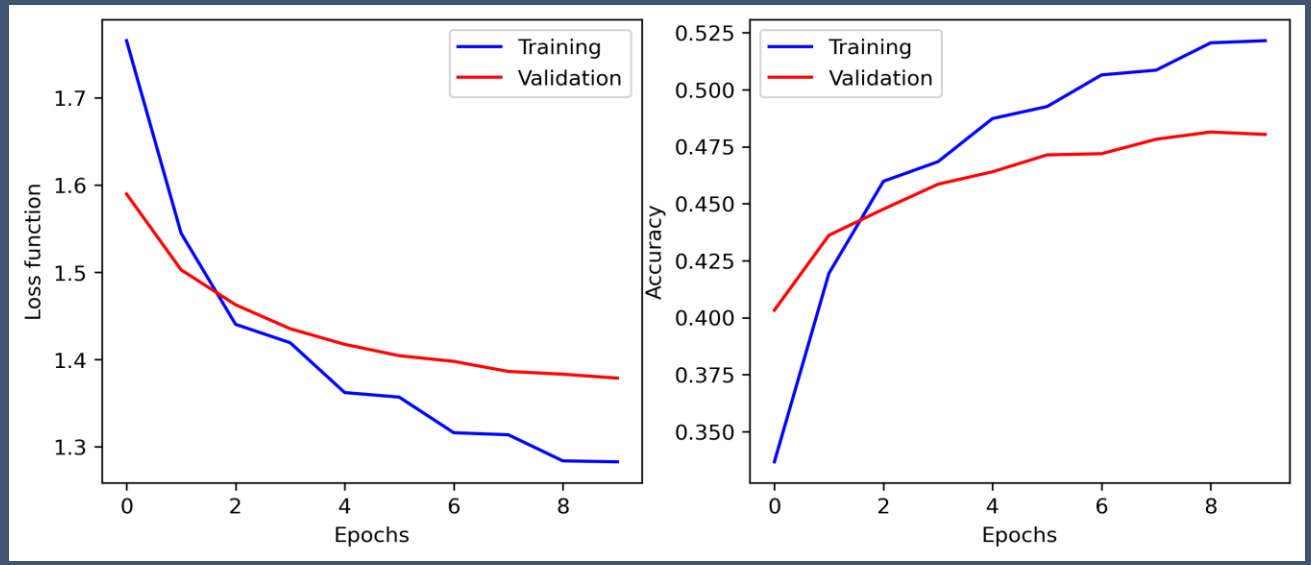
Layer (type)	Output Shape	Param #
resnet50v2 (Functional)	(None, 2048)	23,564,800
dense_30 (Dense)	(None, 7)	14,343

Total params: 23,579,143 (89.95 MB)

Trainable params: 14,343 (56.03 KB)

Non-trainable params: 23,564,800 (89.89 MB)



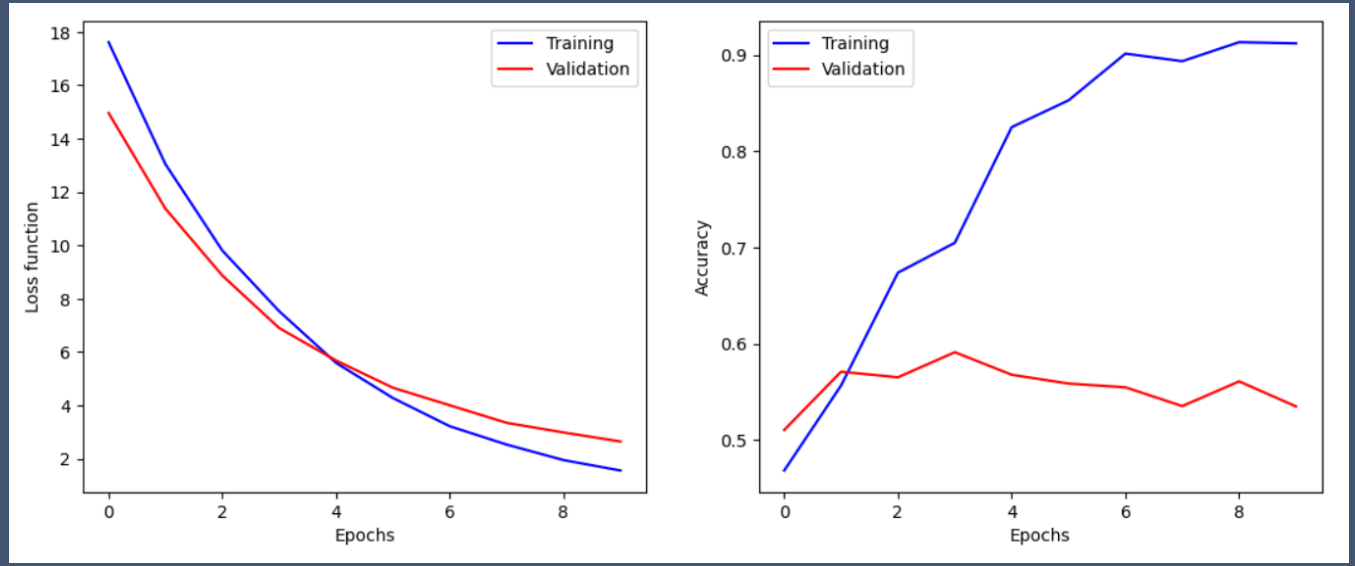


The architecture of ResNet50V2: https://www.researchgate.net/publication/359153551_Deep_Learning-Based_Digital_Image_Forgery_Detection_System

ResNet50V2

Notes on L2 Regularization:

- Adding the square of the weights to the loss function (i.e. Penalizing overcomplexity)
- Made for preventing overfitting, but hard to get right
- Plot showing example of too high penalties resulting in accuracy dropping while loss is being optimized.



EfficientNetV2B3

Architecture of classification layers on top of the pre-trained model

Layer (type)	Output Shape	Param #
efficientnetv2-b3 (Functional)	(None, 4, 4, 1536)	12930622
flatten (Flatten)	(None, 24576)	0
dense (Dense)	(None, 1024)	25166848
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 256)	262400
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 7)	1799

Total params: 38361669 (146.34 MB)
Trainable params: 31582061 (120.48 MB)
Non-trainable params: 6779608 (25.86 MB)

Architecture is the same for excl./incl. finetuning, but number of trainable parameters increases. Here shown for incl finetuning.

EfficientNet is a convolutional neural network architecture, which has proved highly effective in providing high accuracy while being smaller and faster than similiar CNN model families*.

To tailor the pre-trained base model to the specific task of recognizing emotions in facial expressions, a Sequantial classifier structure was put on top. The output from the base model was flattened with a Flatten-layer to make the convolutional layers of the base model compatible with the fully connected Dense-layers. To prevent overfitting Dropout-layers are added in between the Dense-layers. When building the classifier, we took into consideration that the difference in dimensionality between layers should not be too large.

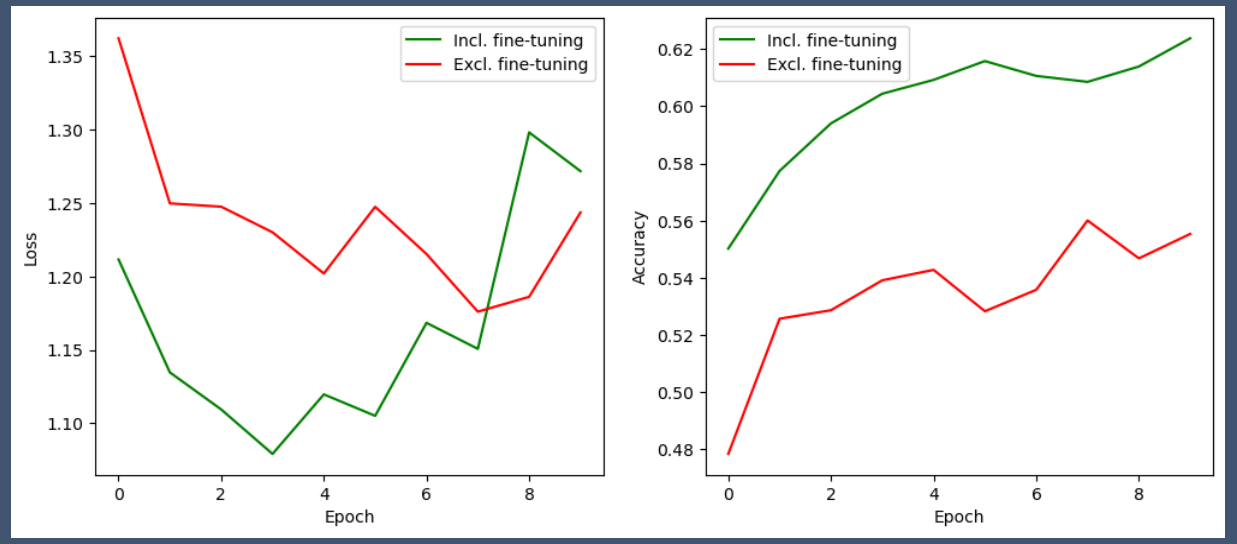
To further develop the model, other architectures of the classifier could be investigated. Trying average/maximum pooling instead of a Flatten-layer could maybe improve the performance.

*https://arxiv.org/abs/1905.11946

EfficientNetV2B3

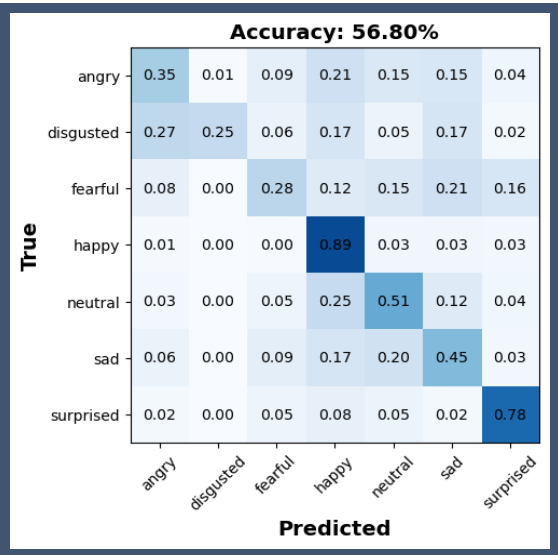
Training the model

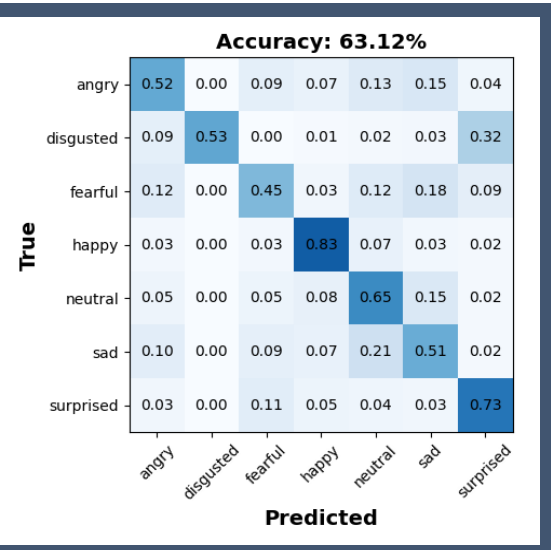
The model was trained using the Adam-optimizer to minimize the Categorial-Cross-Entropy-loss with the monitored metric being accuracy. The training was done with a batch size of 16 over 10 epochs. Early stopping was implemented with a patience of 3, which means that the training will stop if the monitored metric – accuracy – stopped improving on the validation data set over three epochs. For visularization early stopping was turned off.



Finetuning of the base model

The architecture of EfficientNetV2B3 contains 409 layers. For finetuning the last 109 layers were unfrozen and finetuned using a learning rate of 10e-3. This resulted in an improvement from 56,8% (left confusion matrix) to 63,1% in accuracy (right confusion matrix).





Appendix 3

Dealing with imbalanced data set

Imbalanced data set

The data set is imbalanced in the sense, that the "disgusted"-category is under-represented by a factor of approx. 10, while the "happy"-category is larger than the rest by approx. a factor of 2. This results in very few predictions from the models on facial expressions displaying "disgusted" and many predictions of facial expressions displaying "happy". With the EfficientNetV2B3-model we tried two different ways of dealing with this bias.

Class Weighting

The classes were given a weight depending on the number of images in the class:

Class_weight_i = N_total_images / N_images_i

In this way the classes with fewer entries would be given a larger weight. Weighting the classes in this fashion resulted in a lower accuracy than without weighting (51% compared to 56%). However, this method resulted in increased number of images being predicted as disgusted. The class weighting method thereby removed a bit of the bias, but decreased the accuracy. See confusion matrix on the next slide.

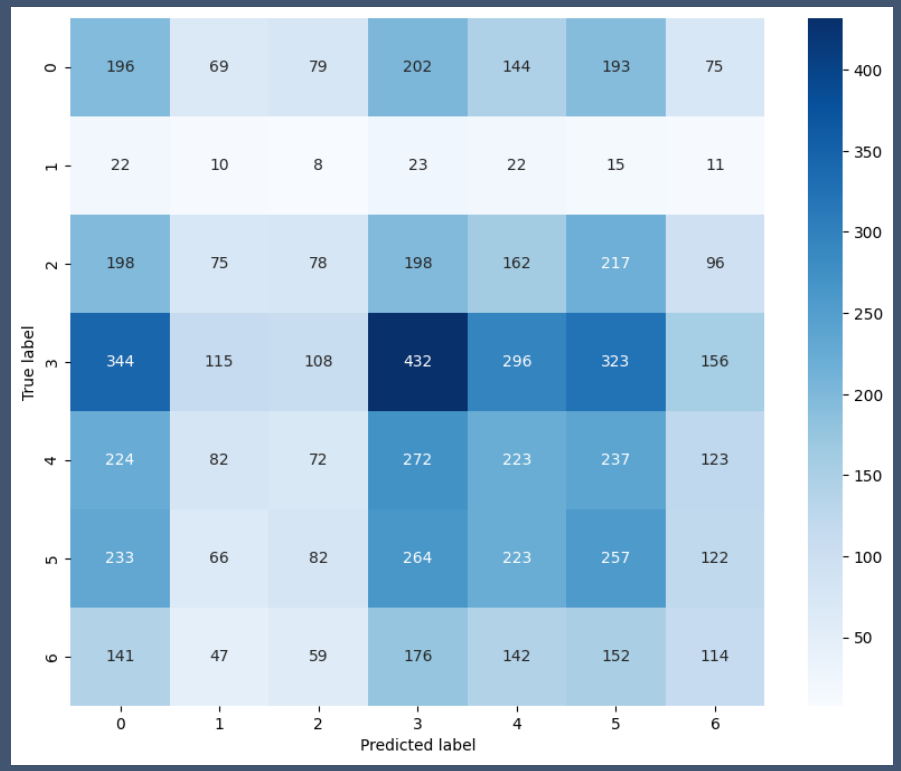
Data augmentation

The number of images in the "disgusted" category was increased by augmenting the images. The images were flipped left/right, up/down and rotated to extend the data set.

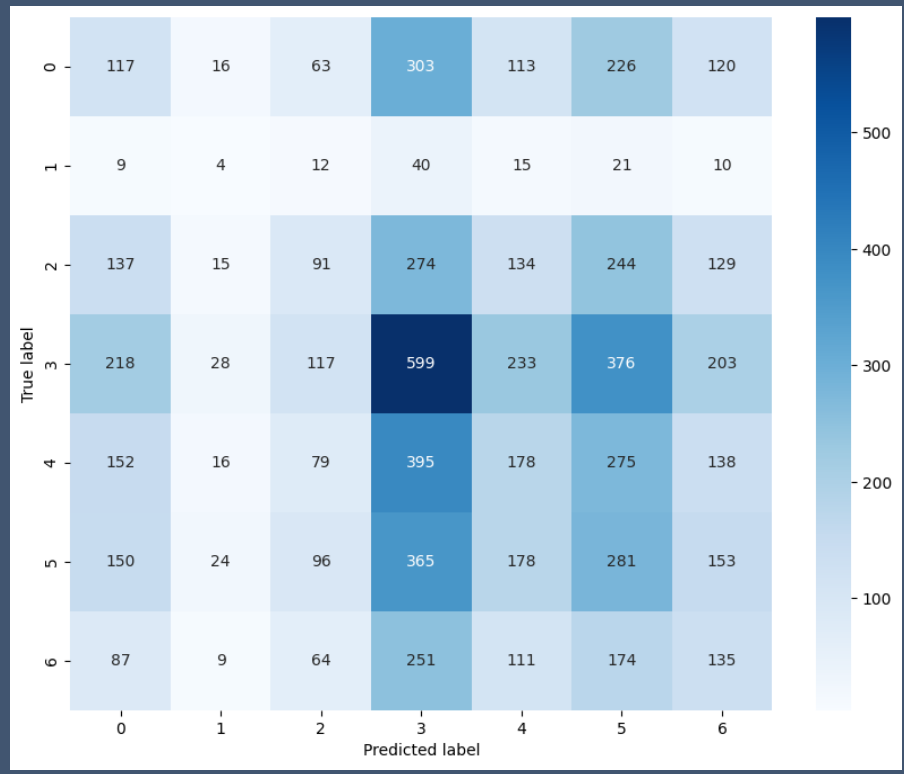
The result of the data augmentation was that it performed quite well on the training set, but when used on the test data the overall accuracy decreased. In addition the number of correct labelled "disgusted"-images in the test set decreased compared to the raw data set and including weighting. This could point to the model learning features of the augmented "disgusted"-images, which are not in the raw images. We have afterwards been thinking, that we shouldn't have flipped the images up/down and rotations, as this would require the kernels to acknowledge that the features aren't pointing the right way.

Imbalanced data set

Class Weighting
Loss: 1,283
Accuracy: 0,514

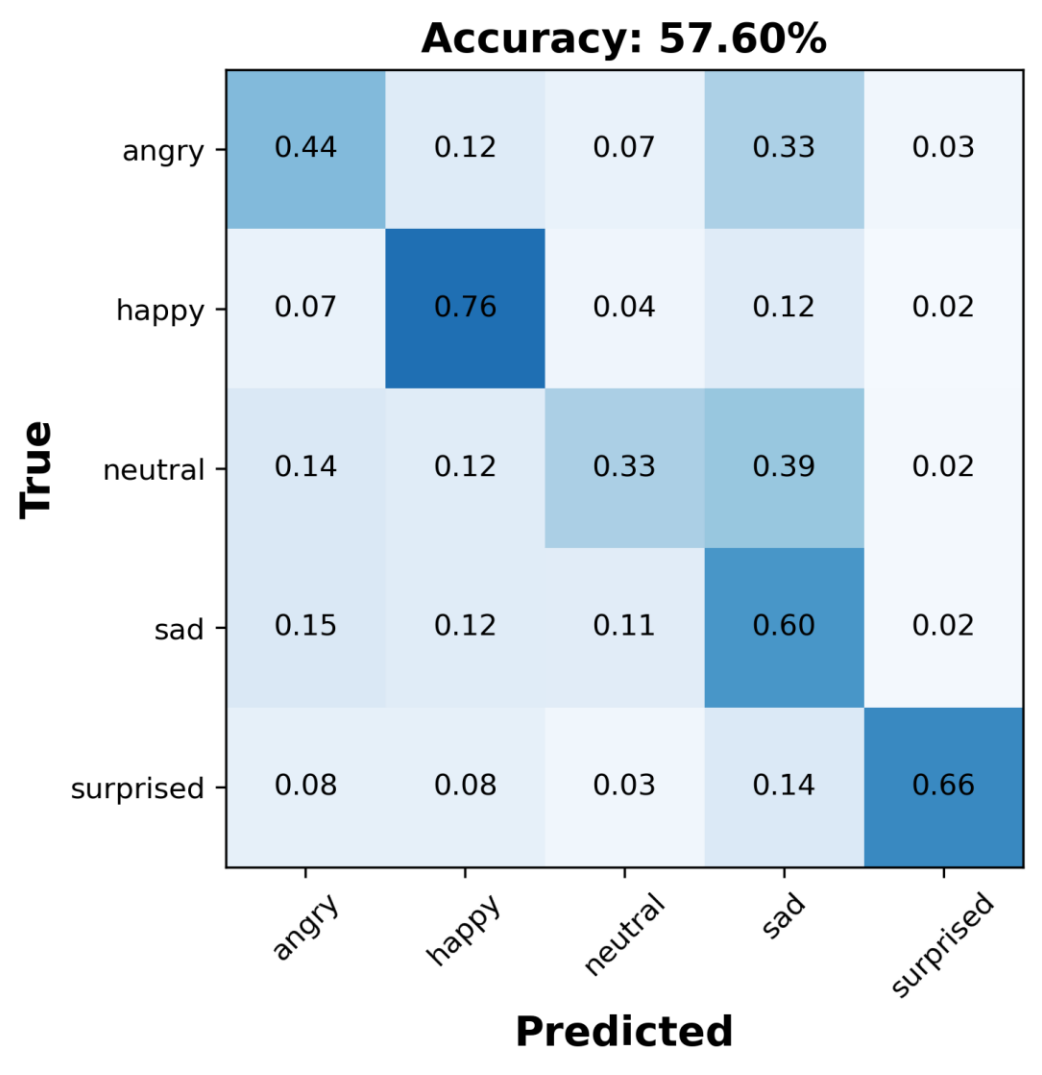


Data augmentation
Loss: 1,196
Accuracy: 0,569



0: angry, 1: disgusted, 2: fearful, 3: happy, 4: neutral, 5: sad, 6: fearful

Native TF model - reduced



Reduced Problem

In an attempt to remove some of the issues with the dataset, the problem is reduced to only 5 categories; dropping disgusted and fearful.

Layers

A similar CNN is constructed, now with only 5 outputs due to the reduced problem.

Results

In the reduced problem we see an increase in the performance of the native Tensorflow Keras model with an accuracy of 57.6%.
While the

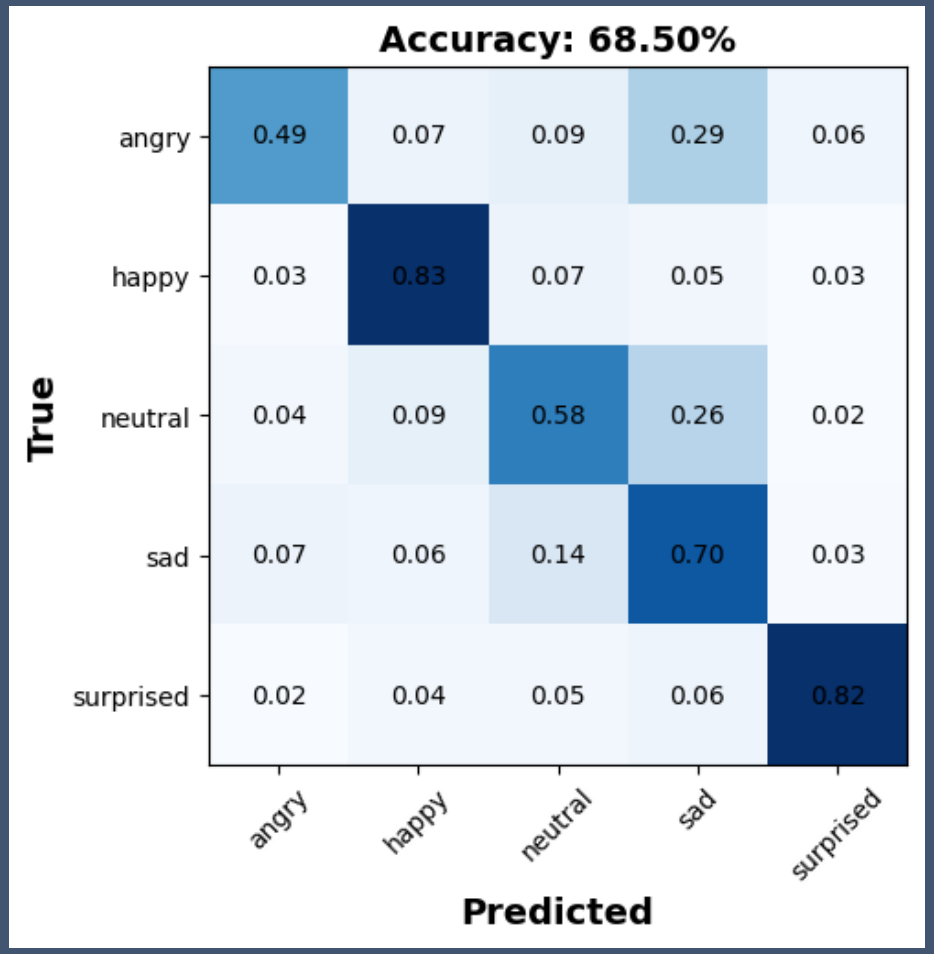
EfficientNet - reduced

Reduced Problem

In an attempt to remove some of the issues with the dataset, the problem is reduced to only 5 categories; dropping disgusted and fearful. The same architecture are utilised as with the complete data set except the output layer having 5 dimensions instead of 7.

Result

As expected removing the difficult categories results in improved accuracy: 68,5%. The resulting confusion matrix for the EfficientNetV2B3 is shown to the right. A clear diagonal structure is showing.



Appendix 4

Data segmentation

Multi-Task Cascade CNN

We segment the AppML2024 class images using a pre-trained model known as MTCNN. It functions via a cascade of three specialized networks with interlayered regularizations.

Firstly, a fast shallow CNN (PNet) proposes a bunch of candidate bounding boxes. These are then tidied up using a non-maximum suppression algorithm to prevent duplicates.
Basically, the candidates are ranked based on PNet confidence whereafter the top one absorbs any other candidate with a sufficiently high Jaccard Index / IoU – defined as pixel intersection / pixel unification.

Secondly, a deeper CNN (RNet) refines the choice of boundary boxes and further discards false candidates.

Finally, the deepest output CNN (ONet) settle for a boundary box and further locates the positions of eyes, nose, mouth, etc.

https://arxiv.org/pdf/1604.02878

Anomalies and misdetections

Since MTCNN does not have perfect performance, we naturally have some instances of faces failling to be detected or non-face objects passing for faces.

Because we acheived no more than order 400 segmented images, it was feasible to manually correct such anomalies, but for large scale data mining one would have to comply with a certain amount of corrupt data using a fully automatic segmentation method.