# Emotion Detection in Images

# Overview of the Implementation

## 1. Researching Computer Vision and Transfer Learning

The start of the project included research about how computer vision works and how well-performing models can be built by using the transfer learning approach. The focus for this step lied on the identification of suitable frameworks and candidate base models for transfer learning.

## 2. Preparing a Dataset for Training

Following the initial research, the second step of the project included finding and preparing a suitable dataset. The goal of this step was to ensure that the resulting dataset is of high enough quality to allow for training a reasonably performant model.

## 3. Setting up the Training Environment and Model

With the dataset at hand, the training environment was set up and the model architecture was developed. For this step it was important to look out for pitfalls that result from working with larger image datasets, which includes issues like learning time and hardware constraints.

## 4. Model Evaluation and Image Prediction

To finalize the project a testing environment was created that allows for validating the model's performance with an additional dataset and to test the prediction capabilities firsthand with a small selection of images.

# CNNs and Transfer Learning

Computer vision with convolutional neural networks (CNNs) works by first taking a given image as input and breaking it down into small pieces. These pieces, called ‘feature maps’, are then passed through a series of convolutional layers. The convolutional layers are responsible for extracting and recognizing specific features in the images, such as edges, lines, and shapes. After each layer is applied to the image, a pooling operation is performed to reduce the size of the feature maps while also maintaining important features identified by the convolutional layer.

Once all these operations have been completed on an image, a fully connected neural network is then used to classify it into one or more categories. This type of network takes in all feature maps from each convolutional layer and uses them to determine an output class for the given image. How such an architecture looks like can be seen on the illustration on the right side which represents the schema of the VGG19 network that is used as a base model in this project.

Transfer learning builds on utilizing a pretrained model, usually from a larger and more complex dataset. This pretrained model is then adapted to the new data set, allowing for faster training times and improved accuracy. In computer vision tasks using CNNs, transfer learning works by taking weights from a pre-trained network on ImageNet or other large datasets and applying them to the new task with slight modifications. The lower layers of the network are used as feature extractors while the higher layers are adjusted according to the new task's needs.

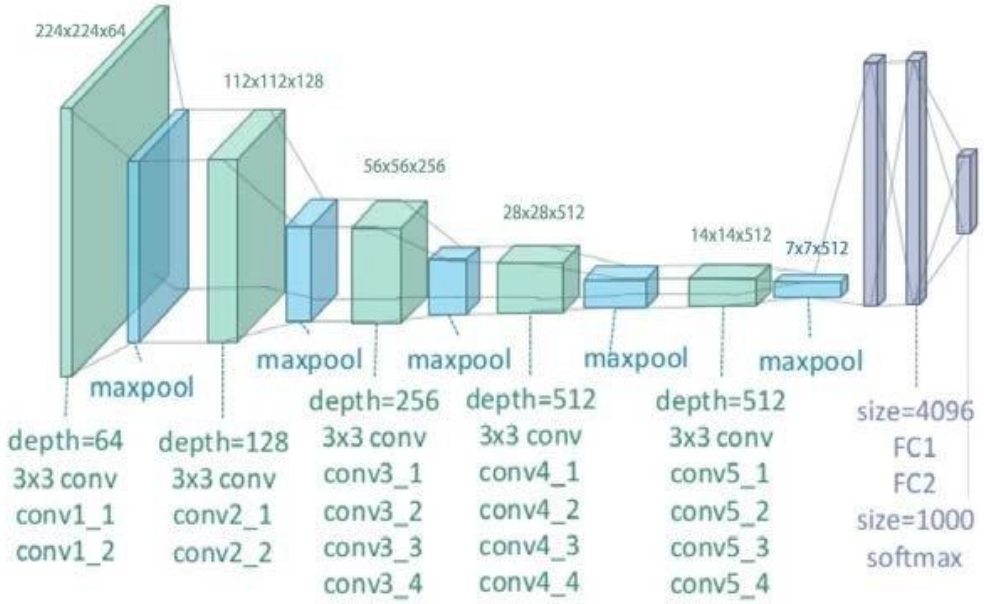
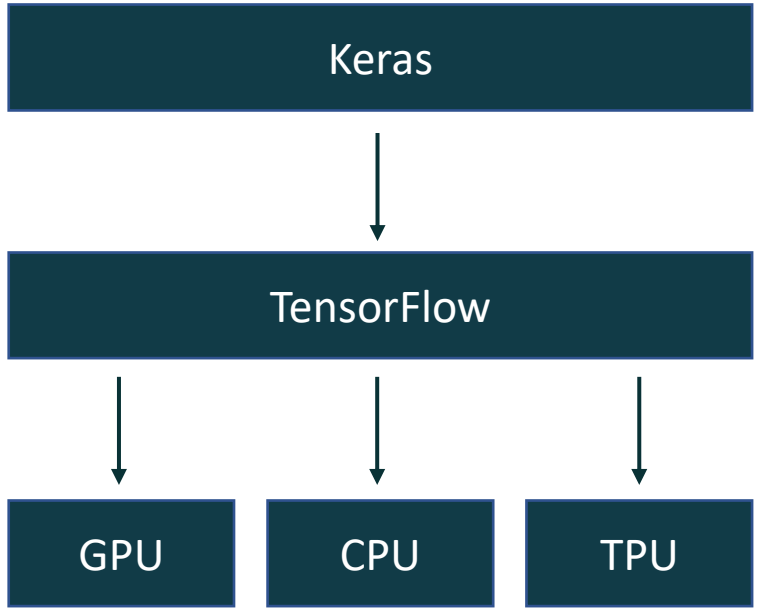



*Figure 1. VGG19 model structure. Retrieved from Zheng, Yufeng & Yang, Clifford & Merkulov, Aleksey. (2018). Breast cancer screening using convolutional neural network and follow-up digital mammography.*

# Frameworks and Base Model

For this project I decided to use TensorFlow and Keras as base for training my model. They are two of the most widely used frameworks for creating, training and deploying convolutional neural networks. TensorFlow is a powerful open-source software library that contains a variety of APIs that can be used to construct, train, evaluate and deploy deep learning models. On the other hand, Keras is an open-source library designed to make working with complex neural network architectures simpler. It works on top of TensorFlow and allows to quickly prototype deep learning models without having to dive into the underlying code too much. In addition, both frameworks have built-in support for GPU and TPU acceleration so they can take advantage of hardware accelerators such as NVidia GPUs or Google TPUs when available.




As a base model I decided to use the VGG19 model. The VGG19 model is a deep convolutional neural network developed by the Visual Geometry Group at Oxford University in 2014. It consists of 19 layers, including 13 convolutional layers and 5 fully connected layers. The model was trained using the ImageNet dataset which contains 1.2 million images from 1000 different categories for training. The VGG19 like many other models are included in the Keras library and readily available as Python classes to build upon.

# Dataset Criteria

When selecting datasets for computer vision tasks with convolutional neural networks, there are certain criteria that should be met.

### 1. Correct labels

This means that the class labels must accurately represent what is being seen in each image, especially for a subtle task like recognizing emotions in human faces. Labeling must be accurate and consistent in order for the model to learn meaningful patterns and produce reliable results. To ensure that data is correctly labeled, it should be done manually by experienced annotators who are familiar with emotion recognition tasks. For example, an image labeled “happy” should show a face displaying clear signs of joy or excitement rather than simply a neutral expression or one exhibiting mild satisfaction.

### 2. Balanced Categories

If data is highly unbalanced, the model may learn to recognize only the majority class and completely ignore minority classes. This can lead to a poor performance of the model. To address this issue, it is important to collect more data from minority classes or perform data augmentation techniques such as image flipping or cropping on existing images.

### 3. Right Image Resolution

The size of the images used for training can have a significant impact on the performance of a Convolutional Neural Network (CNN). Generally, larger sized images allow for higher resolution features to be detected and learned by the model. This increased detail can lead to improved accuracy in object detection and classification tasks. On the other hand, larger image sizes also require more computational resources, which can make training longer.

### 4. Training, Validation and Test Data

It is essential to have a training, test and validation set. By having all three sets of data, the model can be thoroughly evaluated and tested to ensure a good performance. The training set is used for the model to learn how to classify images correctly. The validation set is used during development of the model to tune hyperparameters such as the learning rate or the find the right architecture. The test set is then used for evaluating the performance on unseen data after training has been completed.

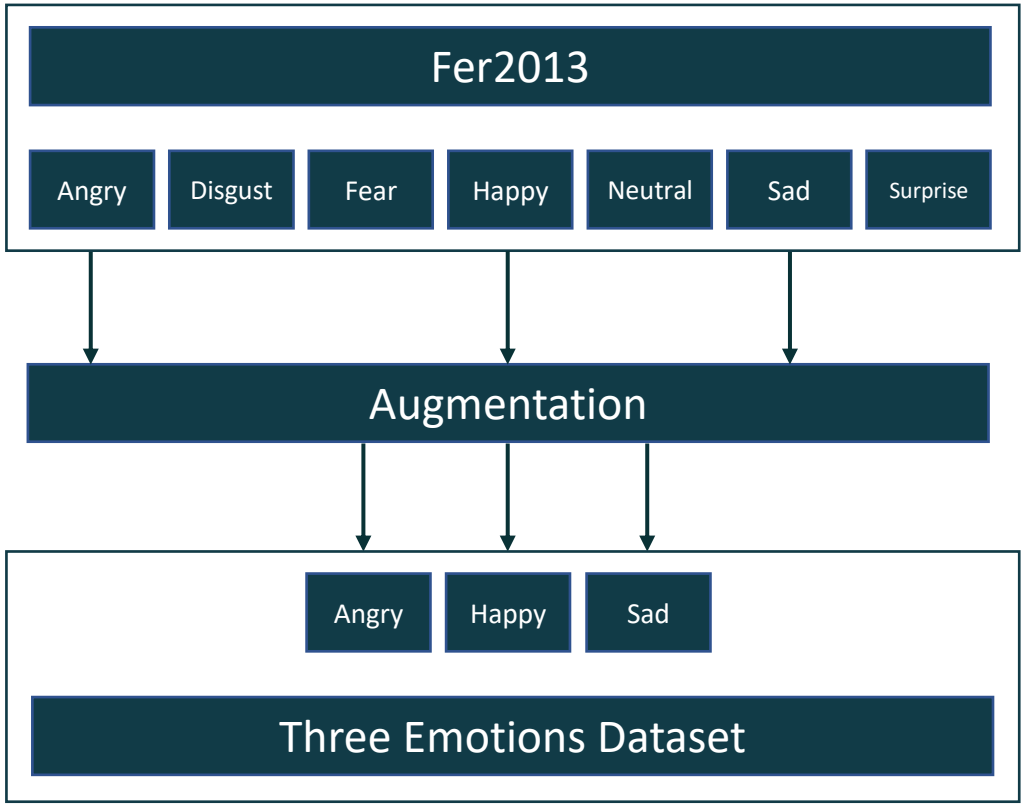
# The Three-Emotions-Dataset

The dataset that was ultimately used for training was tailored to get the best possible performance while staying inside a reasonable frame regarding computation resources and training time. The basis for the custom dataset was the Fer2013 dataset.

The Fer2013 dataset is a publicly available collection of facial images created in 2013, with each image annotated with one of seven emotion labels. Despite its popularity, it is limited by low image quality and unreliable labels, which is not that ideal for facial recognition research.

To alleviate some of these issues, I decided to pick the three most distinct emotions: joy, anger and sadness and use data augmentation to balance the dataset to 5000 images per category.

The final three-emotions-dataset contains 15000 training images sorted by three categories and 3979 test images that were copied from the original Fer2013 dataset. Although in reality there were only 12000 training images, as 20 percent were split and used as validation data.



# Model Training Prerequisites

For successful model training there are certain things that need to be taken into account.

**1. Image preprocessing**

Image preprocessing prior to training a CNN model is essential for accurate results. This involves converting images in array format, normalizing the image data, applying color transforms and performing computations like resizing. Additionally, we have model specific preprocessing steps.

**2. Learning Rate**

The learning rate is also an important factor in training a CNN model. It determines how fast or slow the weights are adjusted during the learning process.

**3. Regularizers and Dropout**

Kernel regularizers are important when training a CNN as they prevent overfitting by adding a penalty term to the loss function. This encourages simpler models that can generalize better on unseen data. Dropout refers to randomly dropping out neurons during each iteration of training, this helps reduce complexity within the network and in turn also prevent overfitting.

**4. Optimizer**

Optimizers are algorithms that are used to adjust parameters in order to minimize our loss function. Common optimizers used for training CNN models include stochastic gradient descent (SGD), Adagrad, Adam and RMSprop. Each optimizer has its own strengths and weaknesses which must be taken into account when selecting one for use in training a specific model.

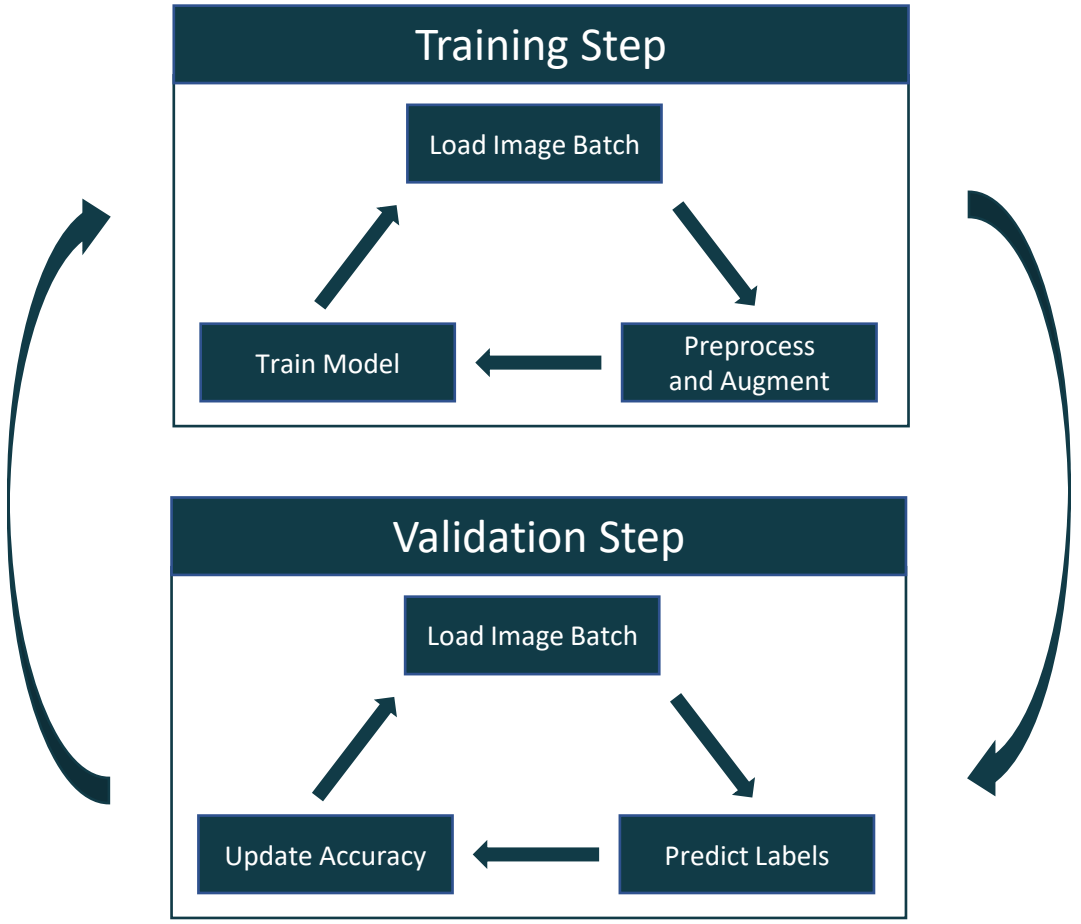
# The Training Environment

The training environment was set up inside a Jupyter notebook and trained on Kaggle, because of the free GPU acceleration.

The image data processing is set up with the ImageDataGenerator class of the Keras library. This class allows for an easy setup of a Python generator that loads, augments and preprocesses the image data during training in predefined batches. This has the advantage of avoiding RAM limitations. The generator is set up to augment the data by rotating, shifting and zooming into it and to rescale the image arrays to be between 0 and 1. Additionally the VGG19‘s preprocessing function is passed which converts the images to BGR and flips them on the head. We also scale up the 48 x 48 images of the dataset to 224 x 224 which is the ideal input size of the VGG19 net.

The actual model consists of the VGG19 with the two 4096 neuron dense layers swapped with two 128 neuron dense layers accompanied by dropout layers, regularizers and a final 3 neuron dense layer representing the classes. The VGG19‘s last convolutional block is also set to be trainable to extract the features of the facial emotions.

We have a relatively low learning rate of 1-e5 given the fine nuances to be learned and optimize for categorical crossentropy with the Adam optimizer. To check for overfitting during training we split 20 percent of the training data and use it as validation set.



# Model Testing

After training the model for around 75 epochs, the training stopped because the validation loss did not decrease anymore. On the final training iteration, the model achieved a accuracy of around 80 percent on the training data and 75 percent on the validation data which is according to my research reasonably high for this task and the used dataset. The final model configuration resulted from trial and error after a lot of attempts with other parameters and model structurs which yielded unsatisfying results.

Additionally a notebook was set up for examining the model with the test dataset and to try some manual predictions. Evaluating the model with the test dataset resulted in around 78 percent accuracy. Given the inherit ambiguity of facial emotion recognition and the limitations of the dataset this result can be seen as acceptable.

Manual testing with random images from the internet proved to work reasonably well for images that are also clear to identify for a human. The process of manual prediction is described on the right.