# - 2. Development phase/reflection phase -

2.1 Frameworks:

-Keras framework: simple and efficient, lot of tools

-Google Collab: training platform not satisfying, stalls

-PyCharm IDE: for local computing

Libraries:

-Matplotlib: visualisation of data in any form including images

-OS: manipulation and saving the data

-Random: randomised values

Active data paths - assigned to Train_dir and Test_dir

```
# use this for 'AI' set
# TRAIN_DIR = "C:/Users/Szilvi/Meine Ablage/Colab Notebooks/AI/train/"
# TEST_DIR = "C:/Users/Szilvi/Meine Ablage/Colab Notebooks/AI/test/"

# use this for 'AI half' set
TRAIN_DIR = "C:/Users/Szilvi/Meine Ablage/Colab Notebooks/AI half/train/"
TEST_DIR = "C:/Users/Szilvi/Meine Ablage/Colab Notebooks/AI half/test/"

# use this for 'AI2' set
# TRAIN_DIR = "C:/Users/Szilvi/Meine Ablage/Colab Notebooks/AI2/train/"
# TEST_DIR = "C:/Users/Szilvi/Meine Ablage/Colab Notebooks/AI2/test/"
# input_size = 208

# size for images
input_size = 48
```

* Commenting and uncommenting is used to adapt those variables for different sets (three of them)

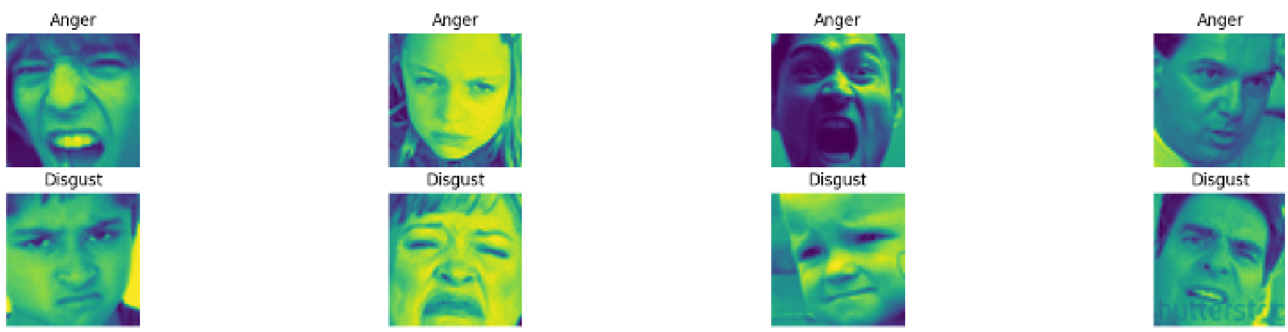
## 2.2 Important program segments

-Pyplot (plt) library presents four random images/category

```
#random choice(emotions)
plt.figure(figsize=(20,15))

for label in range(nr_of_emotions):
      img_folder = TRAIN_DIR + emotions[label]
  for x in range(4):
      # 4 images per emotion
      plt.subplot(len(emotions), 4, label*4+x+1)
      # random image within specific emotion
      random_image = random.choice(os.listdir(img_folder))
      # image drawing with matplotlib
      img = mpimg.imread(img_folder + '/' + random_image)
      plt.imshow(img)
      plt.title(emotions[label])
      plt.axis('off')
plt.show()
```

The result for first two rows :



-TRAIN_DIR and TEST_DIR changed for different sets

-Input_shape and emotions array also altered for different datasets

-Start: normalization and data preparing

-Images zoomed 20%, focus is on face

```
train_datagen = ImageDataGenerator(rescale = 1./255, zoom_range = 0.2)
```

## 2.3 Building a CNN model using Keras

-Two convolutional layers 16 and 32 filters 3x3 ReLu activation

-Batch normalization

-Poor performing, another one conv layer 64 matrices added:

```
# Add convolutional layers with increasing filter number
model.add(Conv2D(16, kernel_size = (3, 3), activation ='relu', input_shape =
input_shape))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(BatchNormalization())

model.add(Conv2D(32, kernel_size = (3, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(BatchNormalization())

model.add(Conv2D(64, kernel_size = (3, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(BatchNormalization())
```

-Max pooling- size of the images to 1/4

-Without padding, size 2 px smaller

-strongest feature remains

-Convolution and pooling make one processing block

-More these blocks better accuracy

-New layers more computation, limits reached

-Dropout increased reliability and precision, low cost

```
model.add(Dropout(0.25))
```

-Forces model to generalize better

-extracts truly universal characteristics

-Compiled with adam optimizer and categorical cross entropy

```
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

-Keras checkpoint callback that saves weights and biases applied

```
# Define the checkpoint callback to save the model every epoch
checkpoint_callback = ModelCheckpoint(checkpoint_filename,
save_best_only=False, save_weights_only=False)
```

-Epochs initially 100, max validation accuracy even before 50

```
# Train the model
history = model.fit(
    train_generator,
    epochs = 50,
    batch_size = 32,
    validation_data = test_generator)
```

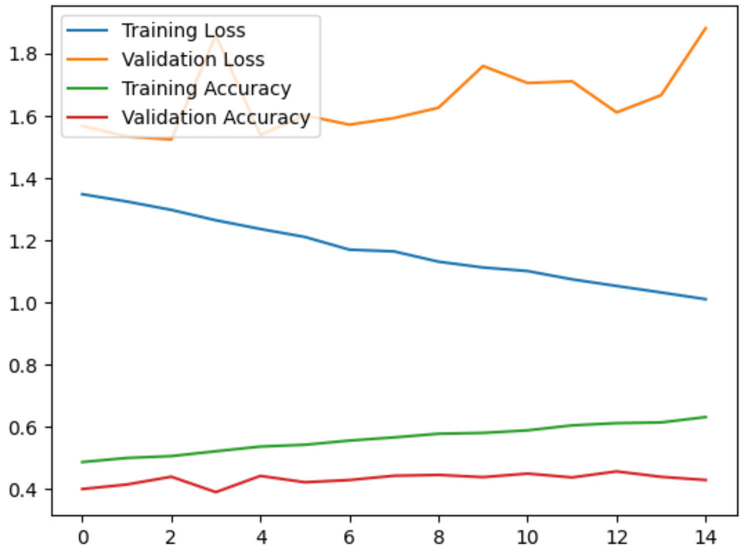
```
Epoch 36  - accuracy: 0.6771 - val_accuracy: 0.5911
Epoch 43  - accuracy: 0.7043 - val_accuracy: 0.6037
Epoch 50  - accuracy: 0.7246 - val_accuracy: 0.6136
Epoch 68  - accuracy: 0.7667 - val_accuracy: 0.6097
Epoch 92  - accuracy: 0.7956 - val_accuracy: 0.6165
Epoch 100 - accuracy: 0.8015 - val_accuracy: 0.6151
```

-Training accuracy grows but validation accuracy similar

-Dense layers with relu and softmax activation function are added

-Model saved as “model.h5”

-Important parameters are plotted:




```
# Plot the loss and accuracy for both the training and validation sets
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.show()
```

-Model and one image from “user images” loaded

```
from keras.models import load_model
import cv2
input_shape = (input_size, input_size, 1)
# Load the saved model
model = load_model(TRAIN_DIR + 'model.h5')

# Load the image
img_path = '/content/drive/MyDrive/Colab
Notebooks/AI/user_images/image1.jpg'
```

-Image is preprocessed, prepared for analyzing

```
img = cv2.resize(img, (input_size, input_size))
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # make grayscale

# Reshape the image to match the input shape of the model
img = img.reshape((1,) + img.shape)

# Normalize the image
img = img / 255.0
```

-Prediction on image with loaded model is called

```
prediction = model.predict(img)
```

-Predicted label and user image are plotted together

```
# Display the test image with predicted emotion label
# print (np.shape(img))
plt.imshow(img.reshape(input_size, input_size ,1))
plt.title(predicted_emotion)
plt.axis('off')
plt.show()
```

## 2.4 Why Convolutional Neural Networks (CNN-s)?

-Until recently best for classification

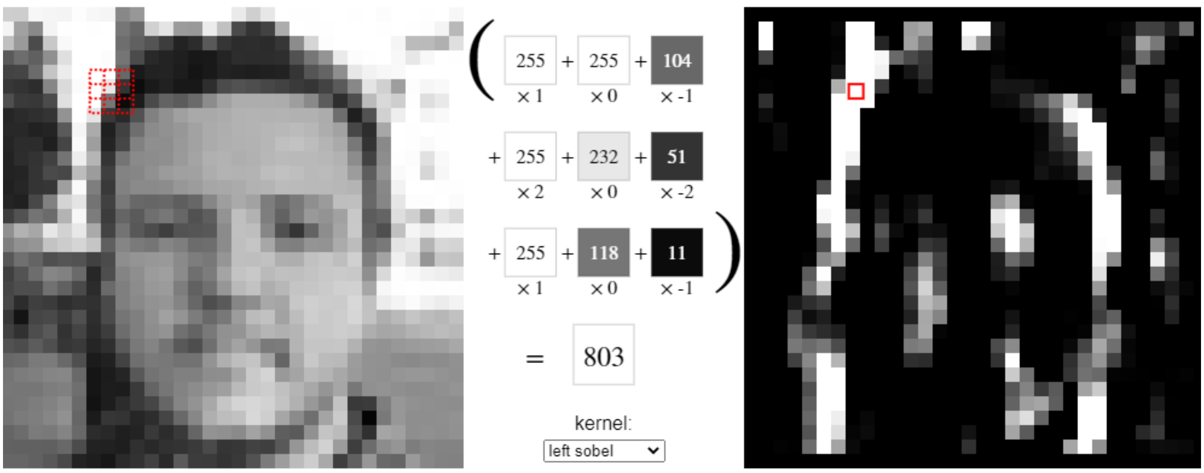
-Exposes image to filter, extract a unique features

-Those features generalizable

-Matrix “Left Sobel” is applied to the input image:

left sobel

$$\begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix}$$




input image output image

-Sobel gives (in this case vertical) features

-Same principle in every filter in CNN layers

-Filters are predetermined for specific features, specific tasks

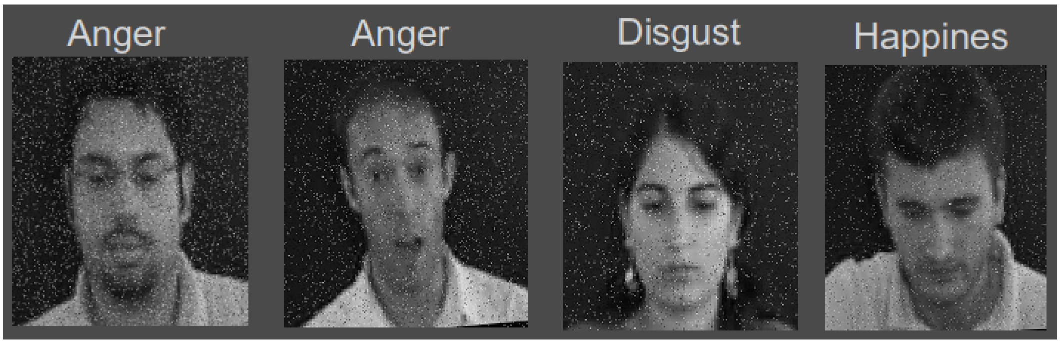
## 2.5 Data sets

-Three Kaggle datasets are taken:

1. Small images with 35900 images, 48 x 48 px (56 MB)
2. Same set with selected 18100 images,  48 x 48 px (34 MB)
3. Bigger images with 14300 images, 416 x 416 px (850 MB)

-Bigger images chosen because of a reasonable size

-had some noise and poorly labelled in some cases, example:




-6 labels: 'Anger', 'Disgust', 'Fear','Happiness', 'Sadness', 'Surprise'

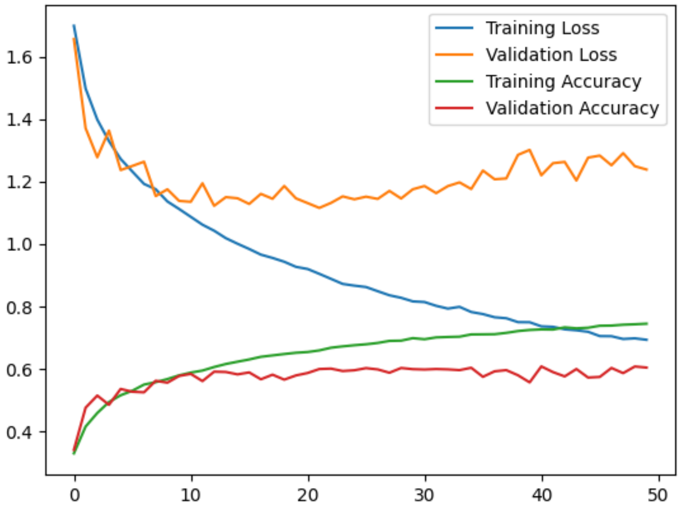
-First and second sets have images size of only 48 x 48 px

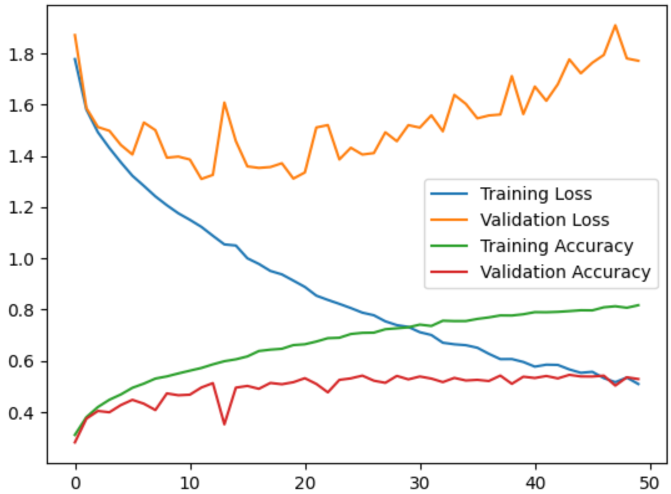
-How much essential information is in small images

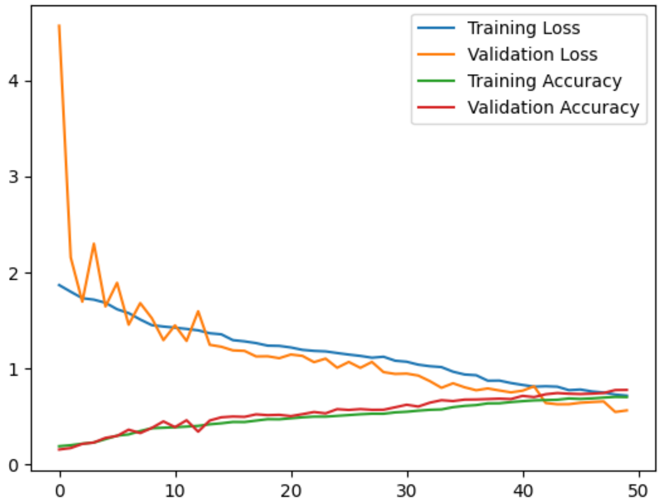
```
1. set - Epoch 50/50 accuracy: 0.7451 - val_accuracy: 0.6047
2. set - Epoch 50/50 accuracy: 0.8172 - val_accuracy: 0.5292
3. set - Epoch 50/50 accuracy: 0.7036 - val_accuracy: 0.7782
```

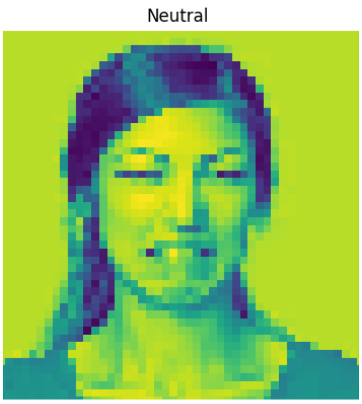
**Left up**: plotted  training data set 1., **right up**: plotted  training data set 2,

**left down**: plotted  training data set 3., **right down**: example prediction




-Quality, quantity of the data, computing power influence quality

-2nd set ccuracy of prediction not going over 0.6

-Much bigger than the statistical chance 0,143

-Far from a usable prediction

## 2.6 Conclusion

-Even simple CNN-s extract important features

-Tensorflow has great filters and all needed tools

-Hyperparameters influence precision 5 - 10%.

-Number of conv layers improves accuracy 35 - 40%

-Small images gave 18% worse results (60% and 78%)

-Set 2: biggest training accuracy, smallest validation accuracy-overfitted

-Set 3: most precise and harmonized

## 2.7 Possible development

-Find face position and angle with semantic segmentation

-Rotate and crop face to ideal position

-Use those images to train the network

-Similar when predicting:

    -Position and angle detection

    -Adjustment and classification

    -Another network to expel unusable images

-CNN classifies only cleaned, centred and rotated data