AI Frameworks Report:

# GitHub:

Inside GitHub, you will find 2 folders, one is the entire project that I save inside of the repository for if you want to check the code also, so I have everything safely saved. The other folder, which I named “What you need” contains everything you requested on Kaggle to test the project with the best model yourself.

# Project Notebooks (Approaches used)

## Preprocessing\_Trainset:

#### In final code:

* + In the first part of this notebook, I take all files in the train set and change all of their extension to .jpg and take the first part of the mp4s and make those .jpgs for them.
  + In the second part of this notebook, I look for all labels so the names of everyone in the dataset. Then I use the correct labels to remove those from the list and then change all the false labels to the correct ones.
  + In the third part of this notebook, I remove rows where the label is equal to nothing as it would be of no use for training and move their images to a different folder.
  + Afterwards I do some image checks which I made when I first wrote the code as I found out the way the color layers are applied are different in cv2 and mathplotlib. (mathplotlib == rgb, cv2 == bgr)
  + When I get to face detection I first make sure the usage of a GPU is possible through CUDA. (In requirements file the download for Torch and Torchvision has the http for cu118 so GPU usage is possible but when pip installing it, it downloads everything correctly but GPU usage is not possible.)
    - If you want to use GPU here is the pip install:
      * pip install torch torchvision --index-url <https://download.pytorch.org/whl/cu118>
  + Next code part is the function, that uses MTCNN to find faces in the pictures and crop them.
  + The code beneath it calls above function and saves the faces in folders based on the persons name in the Sample\_Submission.csv. Saving is done from left to right, so when incorrect faces are found it could cause wrong pictures in wrong folders, but easy fix that can be done manually.

#### Not in final code:

Can all be found in folder: Entire\_Project 🡪 folder: Not\_End\_Resault 🡪 Code\_Not\_In\_FinalProject 🡪 Preprosessing\_Notes.ipynb

* + Made use of haarcascade’s to find faces but was worse than MTCNN.
  + Made use of dlib face detection model (dlib.cnn\_face\_detection\_model\_v1) but was worse than MTCNN.
  + Tried to use various different ways of changing the function for face detection in the MTCNN code to scale the images differently to hopefully get better results.
  + To make sure all faces I found where actually faces I made a function called redetect to do another detection on cropped images to make sure they were faces, but it caused the faces found to decrease incredibly so not useful.
  + I also tried a different way of processing the images, by using dlib.get\_face\_chip to align the faces to see if it would preform better.
    - Images can be found in folders (Aligned and Aligned\_50)

## Create\_Embedings (Was first called Model\_Training)

#### In final code:

* + By using shape predictor and face rec model from dlib I made embeddings of every face I have and save them.
  + Underneath I use the cosine\_similarity to check the similarity between pictures of the same person and similarity between pictures of other people.

#### Not in final code:

Can all be found in folder: Entire\_Project 🡪 folder: Not\_End\_Resault 🡪 Code\_Not\_In\_FinalProject 🡪 Model\_Training\_Notes.ipynb

* + Different functions that do data augmentation.
  + In the code underneath I used from cv2.face.LBPHFaceRecognizer\_create() to fine tune a model that could predict faces, where I used data augmentation to see if more images would give better results, but was worse than making the embeddings to then predict with cosine\_similarity. I also tried to make use of eigen and fisher facerecognizers, but they were far worse than LBPH.
  + Similar to the embeddings that ended up being the best for these tests in this code I used the embeddings to train a face\_recognision\_model that could be used to do face recognition, but it wasn’t a good model.

## Predict\_With\_CS

#### In final code:

* + In the first part of the code preprocessing is done similar to the preprocessing on the trainset. The only difference in the test set is pictures are not sorted in folder based on who is in the cropped image.
  + For prediction the embeddings of the trainset images were used on embeddings made of the testset images and cosine\_similarity was used to predict who was in the image.
    - This was the best way I found to do face recognition but have recently realized there were a lot more methods I could use to make cosine-similatry work far better.
  + The last piece of code is putting all predictions in the csv file for Kaggle.

#### Not in final code:

Can all be found in folder: Entire\_Project 🡪 folder: Not\_End\_Resault 🡪 Code\_Not\_In\_FinalProject 🡪 Prediction\_Notes.ipynb

* + To improve face recognition in images with MTCNN, I tried to use landmarks to take out images that were not faces, but it didn’t work in the slightest.
  + All other code parts are prediction methods I tried with the various models I made.
  + The last piece of code is a simple check I made to debug.

Next two Notebook project files were made after talking with Bart and testing his transfer learning method to see how well it would work on my preprocessed data.

## Training\_Transferlearning\_Model

* + Inside resize\_with\_padding cropped trainset images are resized to 160 and padded with black bars to make sure all images are 160x160 without having to stretch too much or not at all.
  + Using the facenet model that can detect faces, transfer learning is applied by taking the large model and training it on our trainset data, so it works for our dataset.
  + Then afterwards the weights are saved to be used for prediction.

## Predicting\_With\_Transferlearning\_Model

* + Preprocessing that is used in previous codes is also applied here to take test set images and crop out the faces.
  + Afterwards like before, the cropped images are rescaled to 160x160.
  + In the last code part, the model is remade and then filled in with the weights we saved after training the model and now used to work for this model. Then predictions are made on the cropped testset images and also combined to make the csv file for Kaggle.