

**My code:** <https://github.com/krzysiek581234/DetectioHumanusOptimus>

**And:** [https://github.com/krzysiek581234/France\\_AI\\_YOLO](https://github.com/krzysiek581234/France_AI_YOLO)

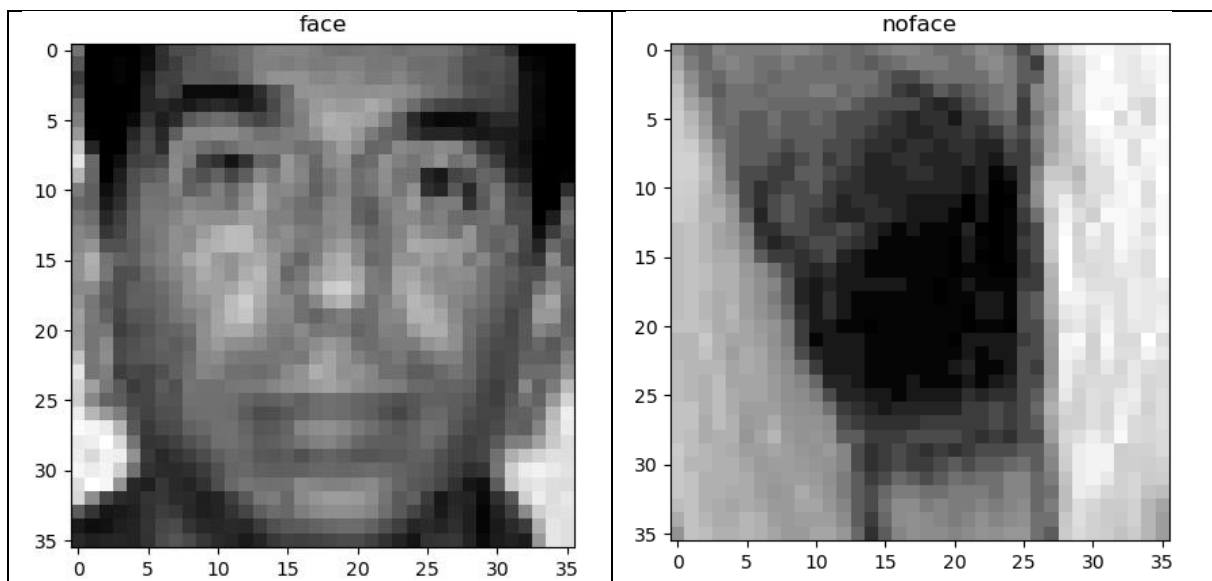
**Entry:**

The project's main objective was to create a neural network capable of accurately identifying human faces in images. To achieve this, the subject of the project focused on the implementation and analysis of a convolutional neural network (CNN) for classifying human faces based on images. This CNN was developed and evaluated with the aim of comparing its effectiveness with alternative approaches, including different neural network architectures and transfer learning solutions.

The project began with supervised learning, utilizing a carefully chosen **XYZ** training dataset that featured close-up images of faces. The primary task for the model was to classify the emotions depicted in these images into two distinct classes: 'face' or 'no face.' This comprehensive analysis and implementation allowed for the evaluation of the CNN's performance and its ability to accurately discern human faces in images.

**Data:**

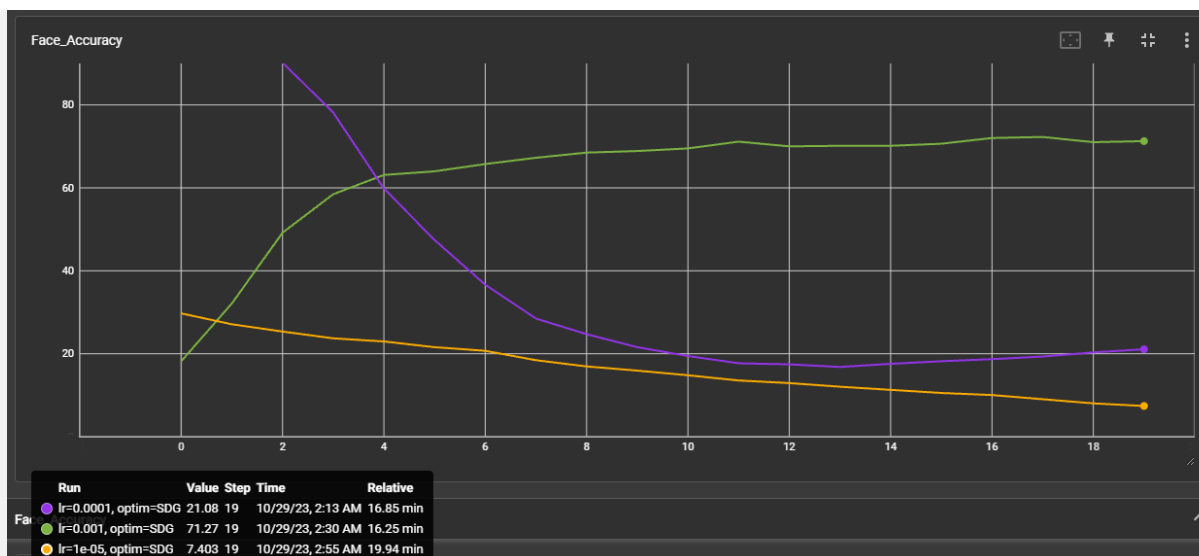
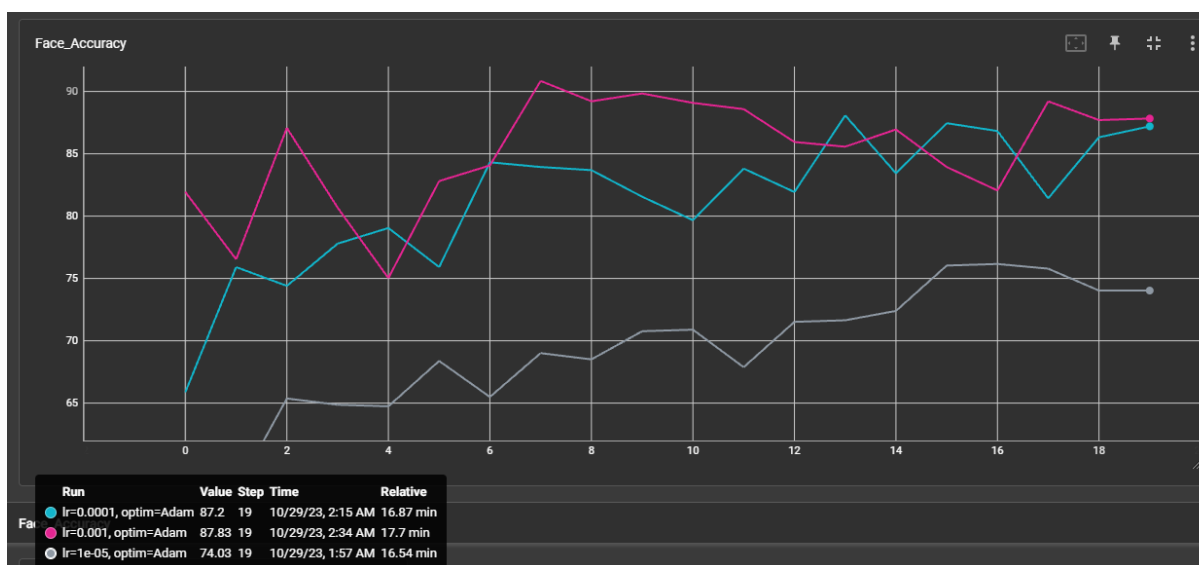
The training process involved a dataset comprising 64,770 facial images, 26,950 non-representational images, all of which were in a format of 36x36 pixels with a single colour channel (grayscale).



## Adam vs SGD + Learning Rate

I conducted experiments in which I analysed the performance of various optimizers, particularly the **SGD** (Stochastic Gradient Descent) and **Adam** algorithms.

**SGD** is a gradient-based optimization algorithm that involves updating weights in the direction opposite to the gradient of the loss function with a small step size. On the other hand, **Adam** (Adaptive Moment Estimation) is a more advanced algorithm that combines elements of momentum and gradient scaling, allowing for dynamic adjustment of the learning rate during training. Unlike SGD, Adam also takes into account first and second-order moments, making it more effective for complex and irregular loss functions.



\* Two methods have the same settings, the only difference is the optimizer

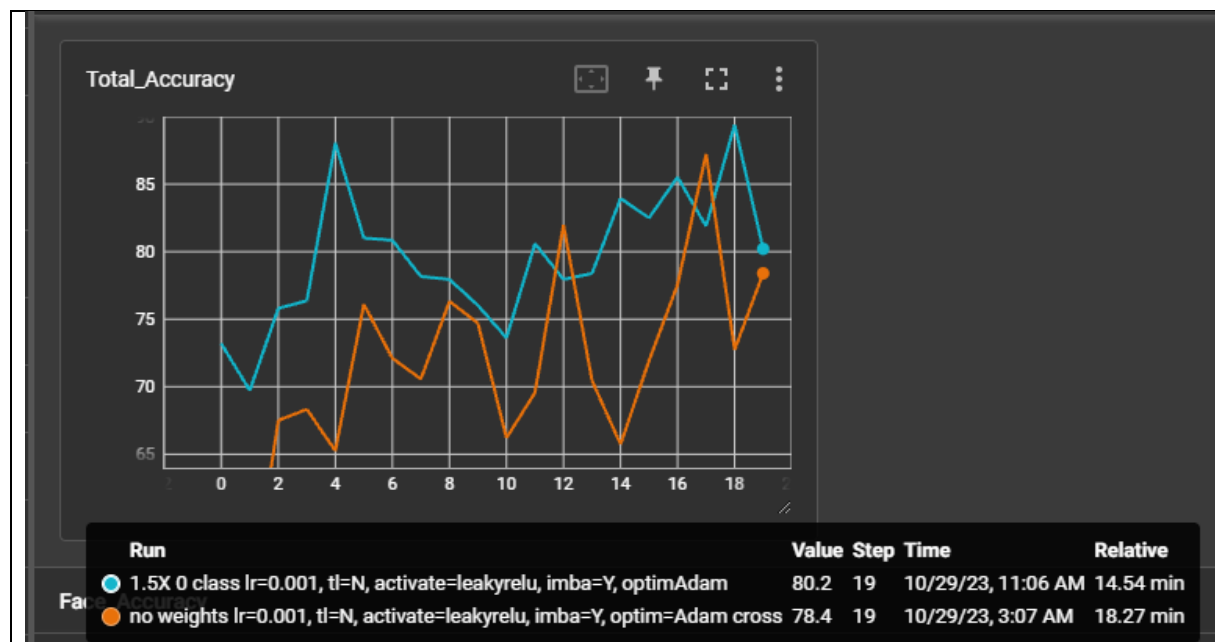
In Our Case Adam performed better that could be, because face image are more complex and because of that Adam have better algorithm which updates the weights of the model.

As we can see SDG works better with higher Learning rate and Adam is not as volatile

### Imbalanced Dataset - Weighted Cross Entropy

Our dataset is imbalanced, with three times as many photos of faces as there are of no faces. This creates an issue because during training, our model would automatically bias towards face prediction.

To address this problem, there are a couple of approaches we can take. We can either acquire more pictures that represent the missing class, or we can adjust the weight during the learning process. I test updating the weights.



As the picture suggests, multiplying the weights of the misrepresentation class could improve the performance of the network on a general scale.

```
nn.CrossEntropyLoss(weight=torch.tensor([1.5, 1.0]).to(device=self.device), reduction='mean')
```

As we can see in the example, weight scaling improves our model's performance by 2 percentage points.

### Architecture NN

Architecture of our network in key part during this project. The default architectural presented in the project is not complex and because it is missing several layers, its ability to detect faces is limited.

I experimented with my own network 4 convolutional layers with leaky relu or relu activation form.

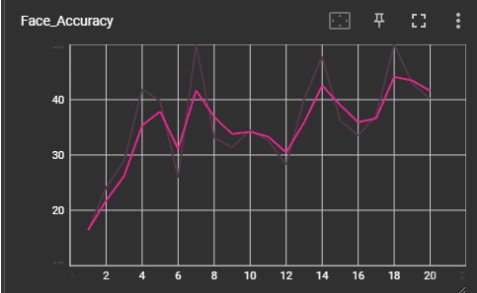
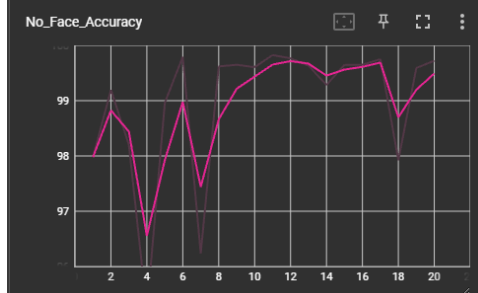
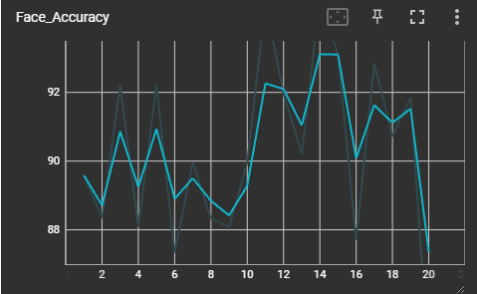
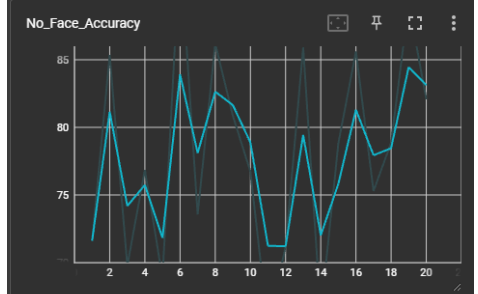
Default network	Main network
<b>Convolutional Layer:</b> The first convolutional layer with 1 input and 6 outputs, with a 5x5 kernel.	<b>Convolutional Layers:</b> The first convolutional layer with 1 input and 8 outputs, with a 5x5 kernel.

<p>The second convolutional layer with 6 inputs and 16 outputs, with a 5x5 kernel.</p> <p><b>Pooling Layer (Max Pooling):</b> Max Pooling is applied after both convolutional layers with a window size of 2x2.</p> <p><b>Fully Connected Layers:</b> The first FC layer has <math>16 * 6 * 6</math> inputs and 32 outputs. The second FC layer has 32 inputs and 16 outputs. The third FC layer has 16 inputs and 2 outputs, suggesting two possible output classes.</p>	<p>The second convolutional layer with 8 inputs and 16 outputs, with a 3x3 kernel. The third convolutional layer with 16 inputs and 32 outputs, with a 3x3 kernel. The fourth convolutional layer with 32 inputs and 64 outputs, with a 3x3 kernel.</p> <p><b>Batch Normalization Layers:</b> Batch Normalization layers are used after each convolutional layer to improve stability and accelerate learning.</p> <p><b>Activation Functions:</b> The CNN_NET can use both ReLU and LeakyReLU activation functions depending on the value of the activation variable.</p> <p><b>Max Pooling Layers:</b> Max Pooling is applied after the second, third, and fourth convolutional layers.</p> <p><b>Fully Connected Layers:</b> The CNN_NET has three FC layers: The first FC layer has 576 inputs and 256 outputs. The second FC layer has 256 inputs and 128 outputs. The third FC layer has 128 inputs and 2 outputs</p>
---	---

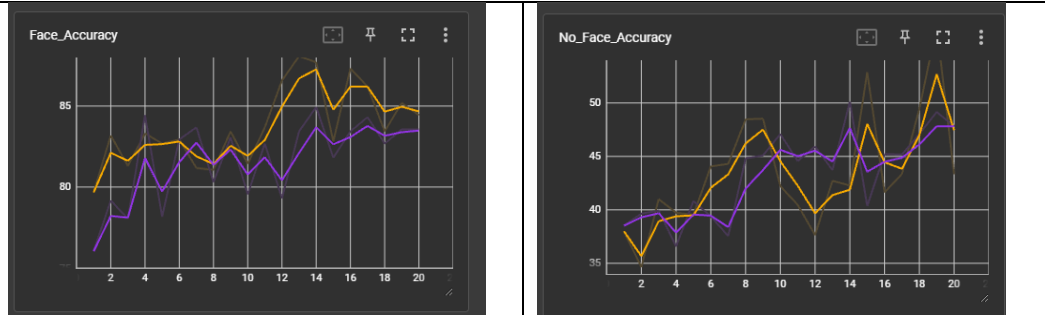
To calculate layers I used:

<https://madebyollin.github.io/convnet-calculator/> - calculator for AI

### Difference in performance

Default network	 
MY CNN	 

Rasnet-18  
Orange  
and violet  
represent  
different  
weights.



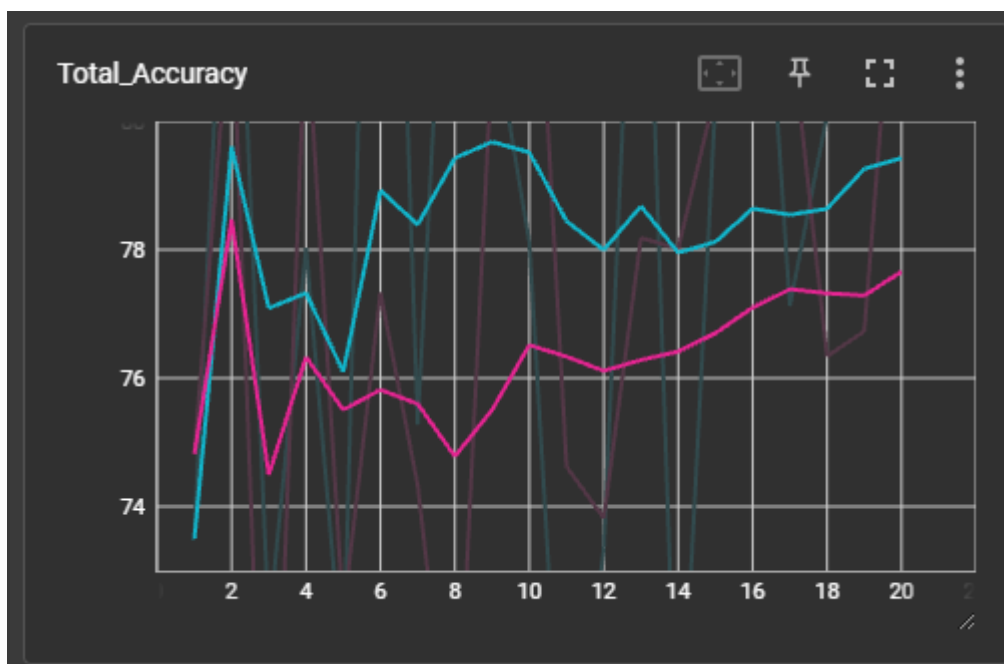
The Resnet-18 model presents an issue in correctly recognizing non-facial images. This poor performance can be attributed to the fact that the training images significantly deviate from the images for which the model is designed. The model is tailored to images of size 224x224 or 299x299 pixels, which is considerably larger than our training images, which are of size 36x36. Furthermore, Resnet-18 is designed for colour images, whereas our dataset consists of grayscale images.

In contrast, our CNN is a network that, like Resnet-18, excels in facial image classification and maintains an acceptable level of performance.

Default CNN performs well in the classification of non-facial images but yields subpar results in the case of facial images due to its smaller network size.

### Method to improve performance

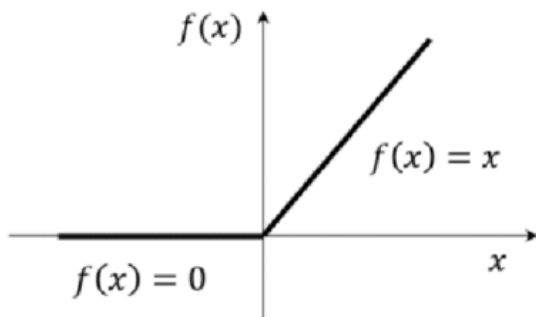
#### Changing the ReLU to Leaky ReLU



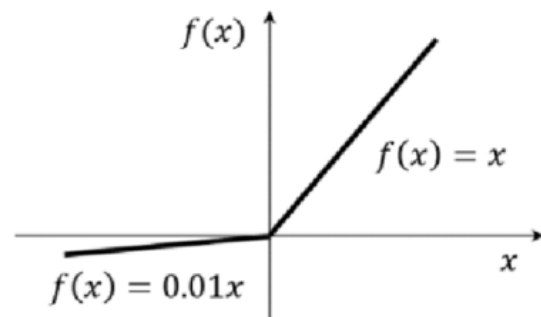
Blue - Leaky ReLU

Pink - ReLU

All other setting are the same



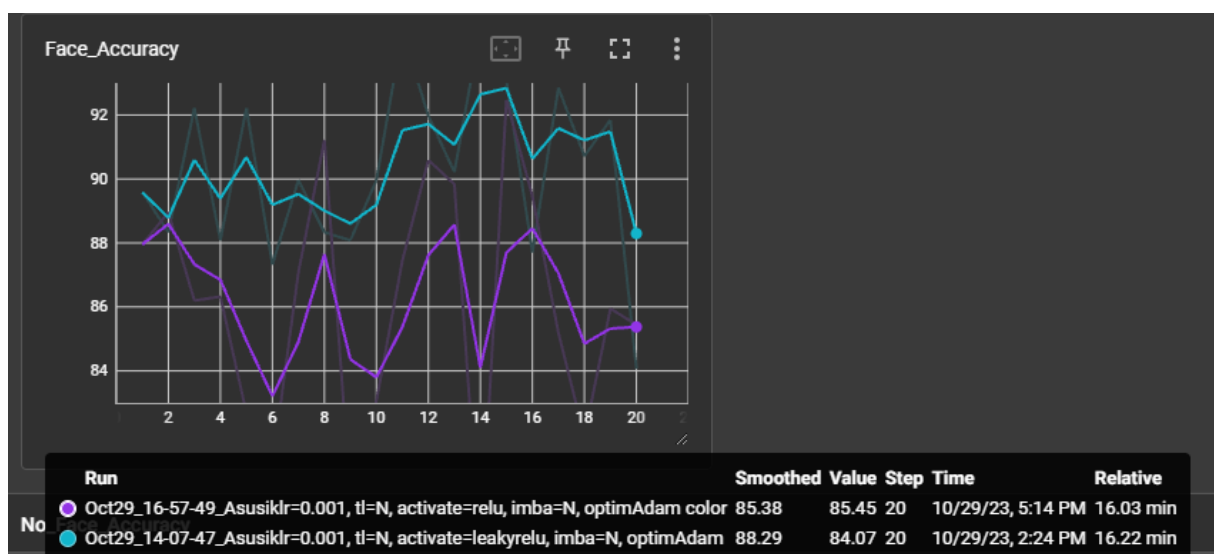
*ReLU activation function*



*LeakyReLU activation function*

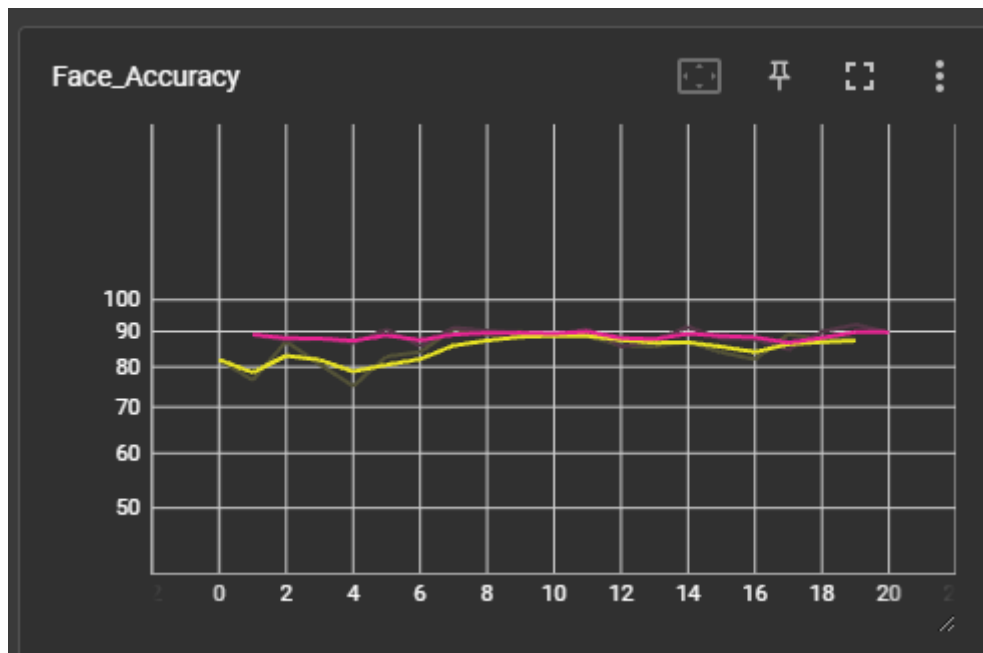
ReLU (Rectified Linear Unit) is an activation function used in neural networks that activates only when the input signal is greater than zero, effectively stopping at zero otherwise. Leaky ReLU is a similar activation function, but it allows a small gradient to flow for  $x < 0$ , helping to mitigate the vanishing gradient problem.

### Adding colour channels



Unfortunately our main data set in scale in Gray scale we cannot see the improvement

### Change the sampler (ImbalancedDatasetSampler)

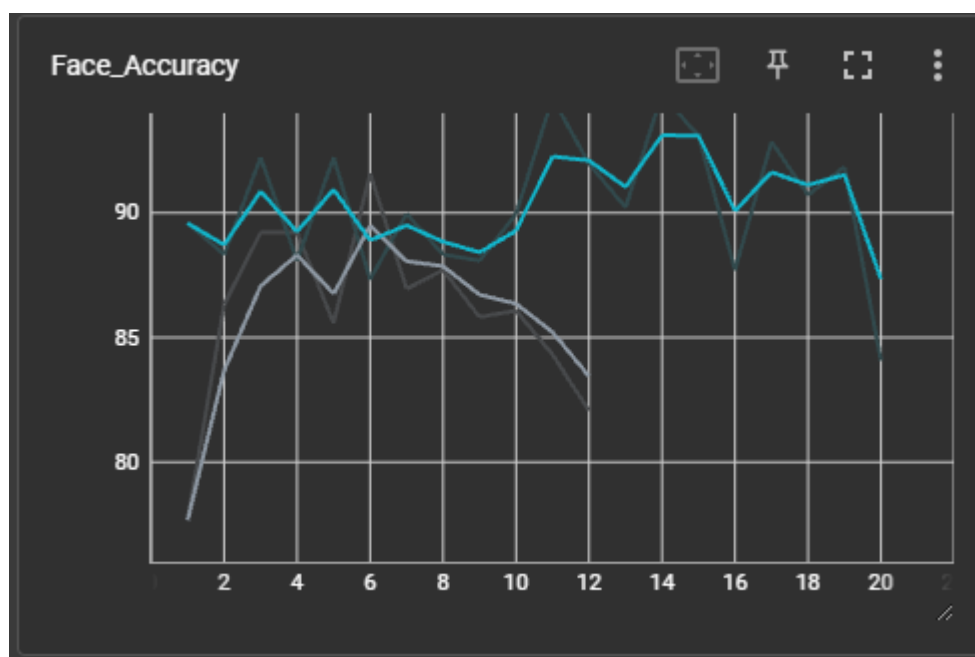


Adding Imbalanced Dataset Sampler is comparable with Subset Random Sampler.

### Early Stopping

Early Stopping is a regularization technique in neural network training that involves monitoring the model's performance on validation data and stopping training when its performance on these data starts to deteriorate to prevent overfitting. The best model achieved during training is saved, and after training is completed, it's advisable to test it on test data to assess its generalization ability. This is an effective way to obtain a model with good generalization capability and avoid overfitting to the training data.

In my project we define the patience parameter which represents how many epochs in a row the model will continue before stopping.



**The grayscale network was halted because over the past 5 epochs, it consistently experienced a deterioration in the stability of detection.**

\*The parameters of these two networks are not the same.

### Implementation:

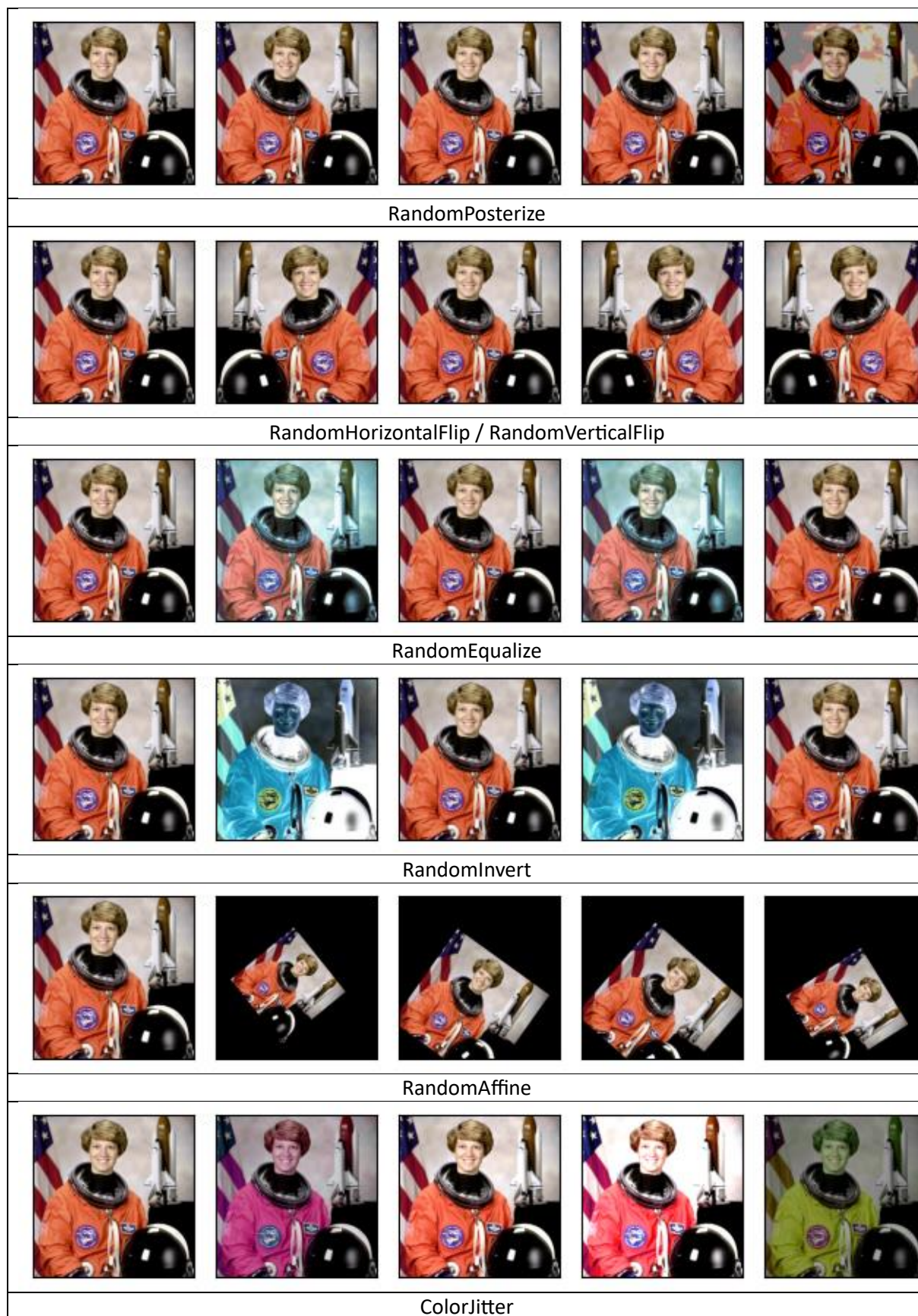
Every epoch I check the performance and we overwrite when the model achieves better performance

```
tempacc = self.validate(model, epoch=epoch)
    if self.best_validation_metric < tempacc:
        self.best_validation_metric = tempacc
        print(f"SAVED {tempacc}")
        file_name = f'CNN_{self.comment}.pth'
        torch.save(model.state_dict(), file_name)
        self.epochs_without_improvement = 0
    else:
        self.epochs_without_improvement += 1
        print(f"Epoch without improvement {self.epochs_without_improvement}")
        if self.epochs_without_improvement > self.patience:
            print(f"Exiting lack of improvement")
            self.writer.close()
        return model
```

### Data Augmentation

Data Augmentation is a technique that is widely used in machine learning. It involves generating new training samples by introducing various modifications to input data. In my case, I employ techniques such as randomly reducing the number of bits representing pixels (RandomPosterize), flipping images horizontally and vertically with certain probabilities (RandomHorizontalFlip and RandomVerticalFlip), random histogram equalization (RandomEqualize), and color inversion (RandomInvert) with specified probabilities. Additionally, I apply techniques like random affine transformations with options for rotation, translation, and shearing (RandomAffine), Gaussian blur (GaussianBlur), and adjustments to brightness, contrast, saturation, and hue (ColorJitter) with designated probabilities. All of these methods help me create diverse training data variations and enhance the performance of my machine learning model.





\*examples from

[https://pytorch.org/vision/main/auto\\_examples/transforms/plot\\_transforms\\_illustrations.html#sphx-glr-auto-examples-transforms-plot-transforms-illustrations-py](https://pytorch.org/vision/main/auto_examples/transforms/plot_transforms_illustrations.html#sphx-glr-auto-examples-transforms-plot-transforms-illustrations-py)

```

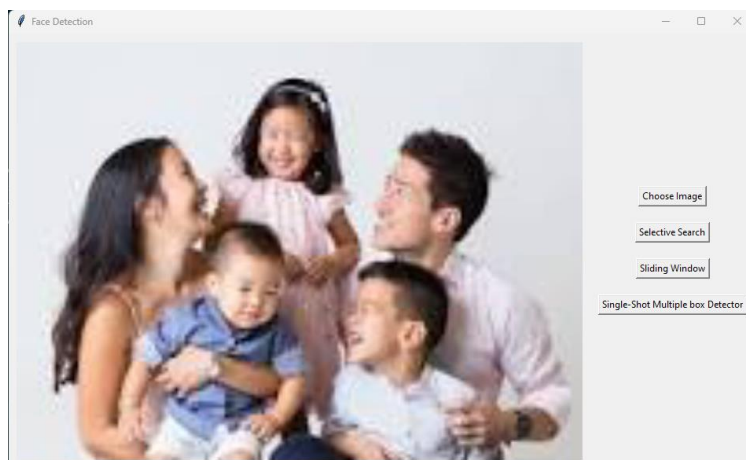
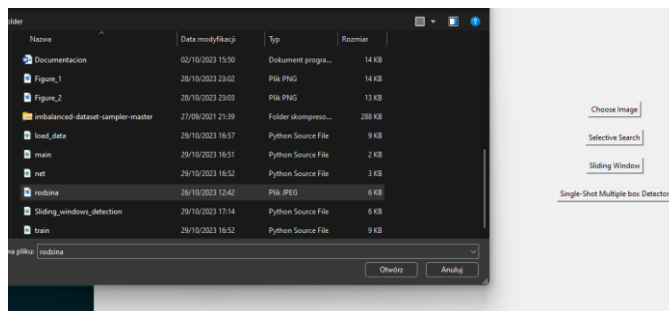
transform = torchvision.transforms.Compose(
    [
        torchvision.transforms.RandomPosterize(4, 0.1), #
        torchvision.transforms.RandomHorizontalFlip(p=0.2), # 20% - chance to do that
        torchvision.transforms.RandomVerticalFlip(p=0.2), #
        torchvision.transforms.RandomEqualize(p=0.1), #
        torchvision.transforms.RandomInvert(p=0.1),

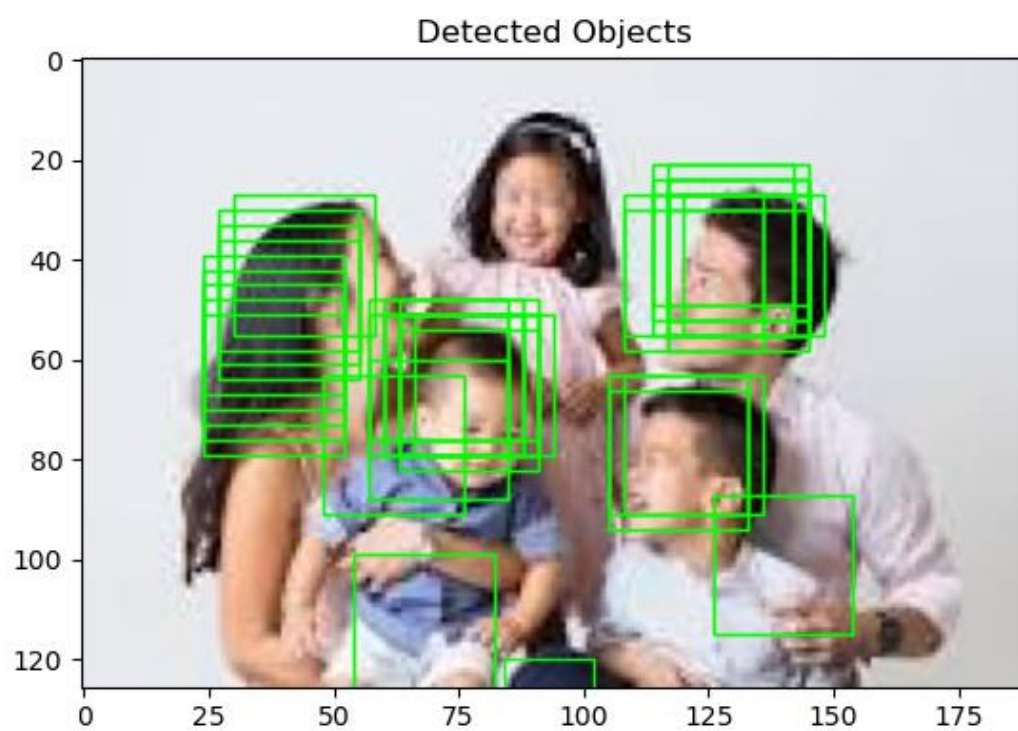
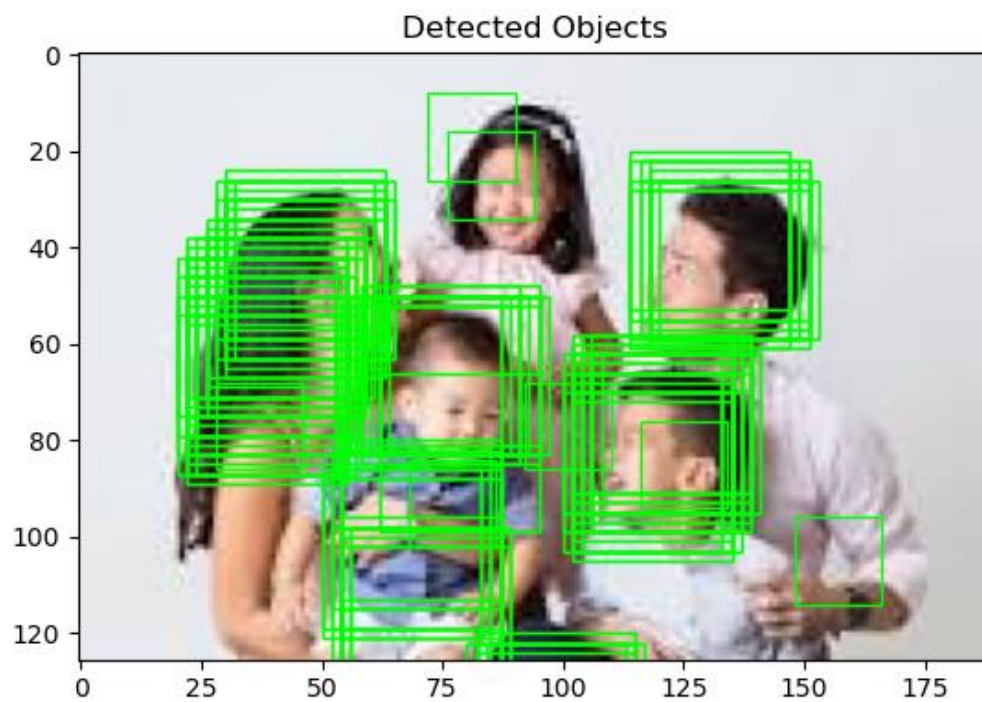
        torchvision.transforms.RandomApply(nn.ModuleList([torchvision.transforms.RandomAffine(degrees=30, translate=(0.05, 0.05), shear=10)], p=0.2),
        torchvision.transforms.RandomApply(nn.ModuleList([torchvision.transforms.GaussianBlur(kernel_size=5)], p=0.24),
        torchvision.transforms.RandomApply(nn.ModuleList([torchvision.transforms.ColorJitter(brightness=(0.5, 1.5), contrast=(1, 1.5), saturation=(0.5, 1.5), hue=(-0.1, 0.1)),], p=0.2),
        torchvision.transforms.RandomApply([torchvision.transforms.GaussianBlur((1, 9), (0.2, 0.3))], p=0.2),
        transforms.Grayscale(),
        transforms.ToTensor(),
        transforms.Normalize(mean=(0.5, ), std=(0.5, ))]) #
self.data_loader = data_loader(transform)

```

## Detecting the multiple faces on the picture

### Interface:





Two different result of sliding window detection best face recognition models.

### **Sliding Window:**

"Sliding Window" is a technique for object detection in images by sliding a window of a specified size and analysing its content using a classifier. Windows are moved with a fixed displacement along the image, and when the classifier detects an object, its location is recorded. Subsequently, to detect larger objects, the window is enlarged, and the detection process is repeated.

### **YOLO v8 – state of the art object detector**

<https://github.com/ultralytics/ultralytics> - github page repository

Yolo is a fast architecture family specializing in various tasks. Most recent edition YOLO v8 have proven to be powerful object detector fully suited to face recognition task.

Full architecture chart done be the GitHub user “RangeKing” can be seen below. Yolo is the type of the network that uses specific data annotation format. Each picture has a corresponding txt file containing annotations. Each annotation consist of class id (information about how object is classified) and a horizontal bounding box location (X and Y of central point of an object and its height and width in relation to height and width of an image).

**Face detection dataset:** <https://www.kaggle.com/datasets/fareselmenshawii/face-detection-dataset>  
the dataset contains 16000 annotated pictures of human faces in YOLO format.

### **Changes in the code from GitHub:**

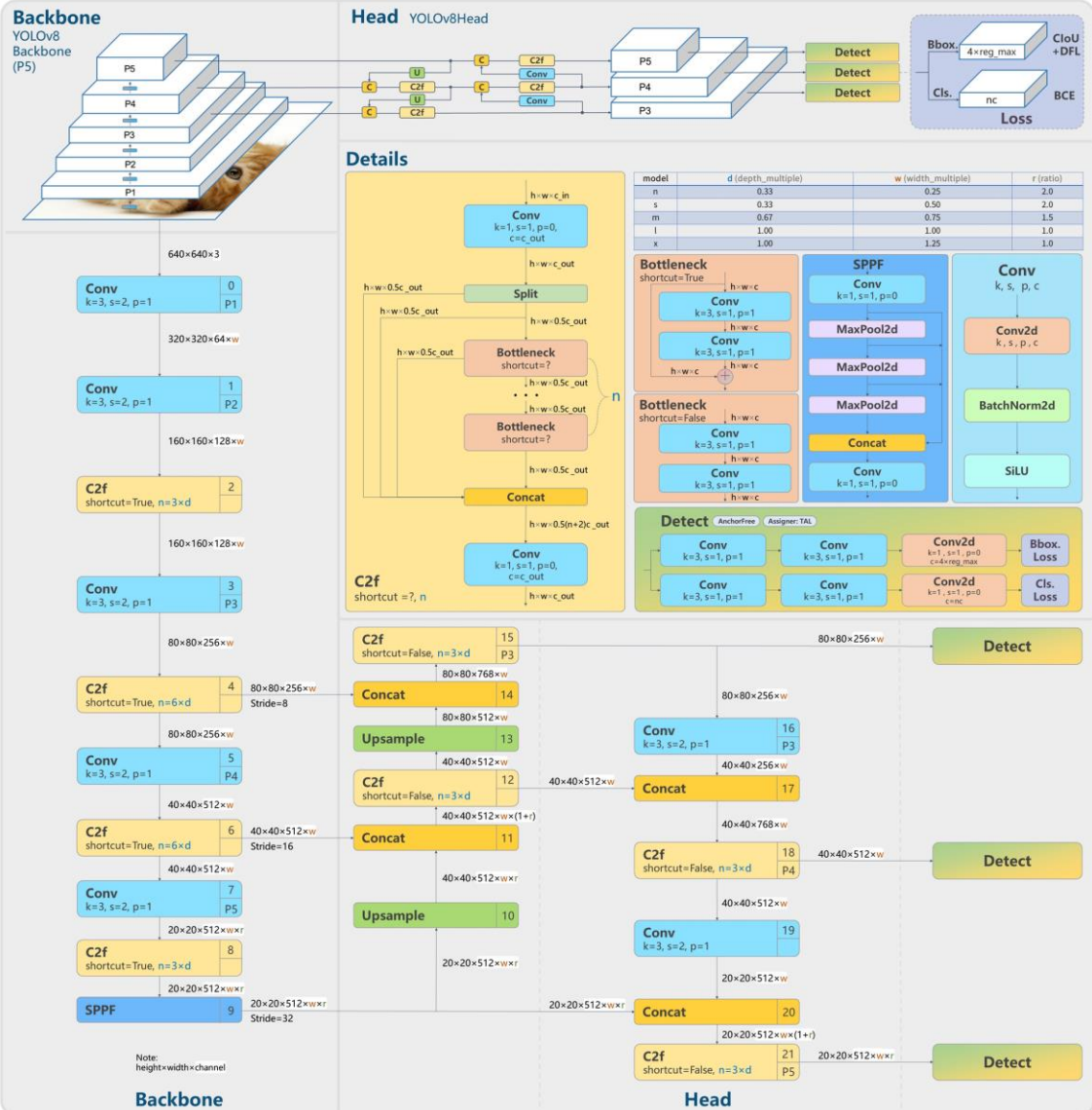
1. Added the fdd.yaml dataset description file
2. main.py – training and validation and prediction code for the YOLO v8 detector

### **Results Bellow**

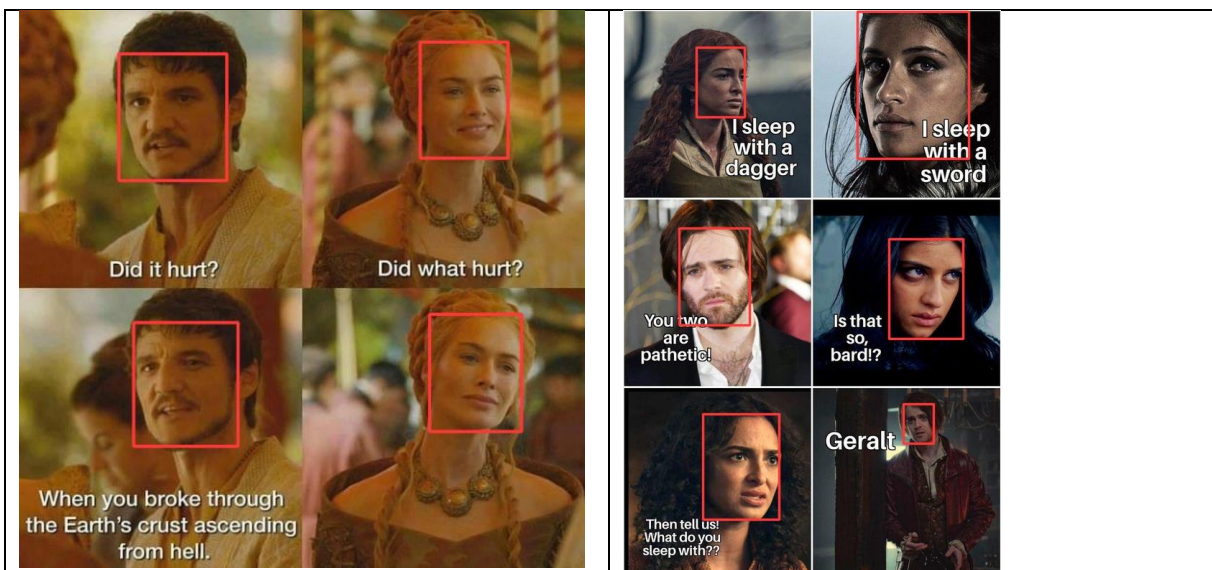


# YOLOv8

RangeKing



## Detection results:



Experiment detailed:

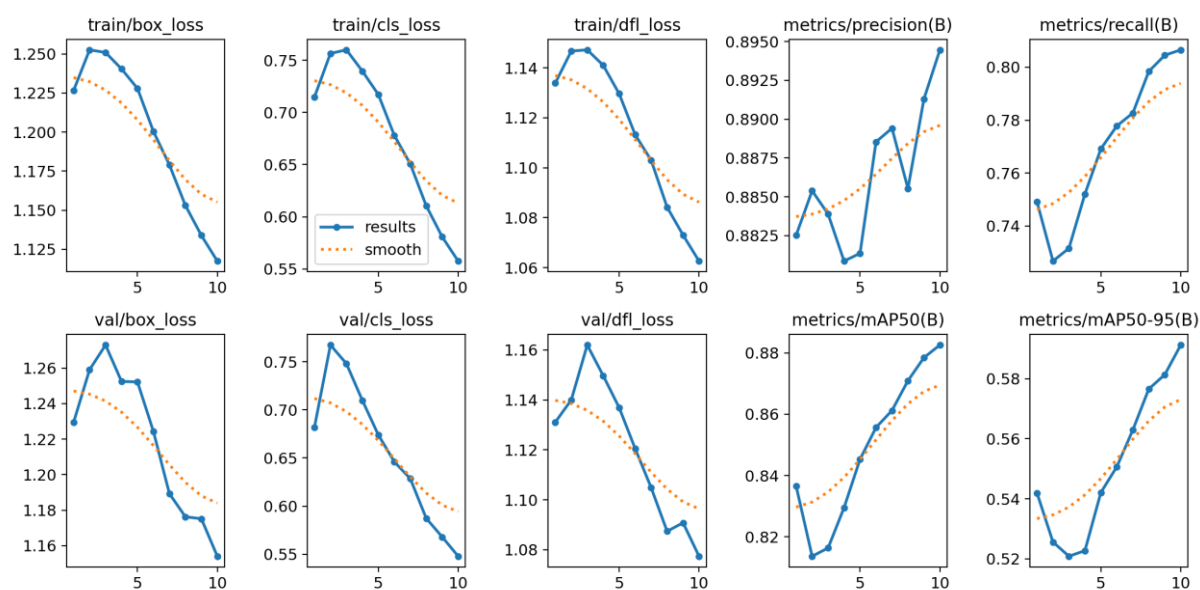
Hyperparameters:

Number of epoch: 10

Batch size: 16

Cosine learning rate: True (decreasing)

Optimizer: AdamW Learning rate: 0.002



Validation score:

- mAP50-95: 0.591
- mAP50: 0.883

