Machine Learning Project Report

Fabian Schröder (Exchange)

HUANG Xuanhao (Exchange)

QI Jiaqi

29/10/2023

Introduction

In this project, we aim to realize a face detection model from a binary data set, face or not face. We divide our task into two parts. In the first part, we need to train some face classifiers with different strategies. In the second part, we choose the best model and implement advanced algorithms and models to detect faces on new test pictures. In this report, we will describe what we have learned and what we have achieved.

Exploratory Data Analysis

Data visualization

- First, all training and test data are 36*36 grayscale images of defined size.

- Next, we randomly selected nine images from both the training and test sets, as shown below. We found that the images of training and test sets are relatively similar, with faces accounting for the vast majority of pixels. Therefore, the accuracy of the trained model should be good.

- In addition, we observed the diversity of the training data. For example, there are people with different skin colors, with different ages, as well as different shooting angles and lighting environments. Hence, we expect that the generalization ability of the model will be good. If we observe more detail, we can see that some people's ears are covered, some people are wearing glasses, and some people are showing their teeth or closing their eyes, etc. This information will give us some inspiration for designing the model later.

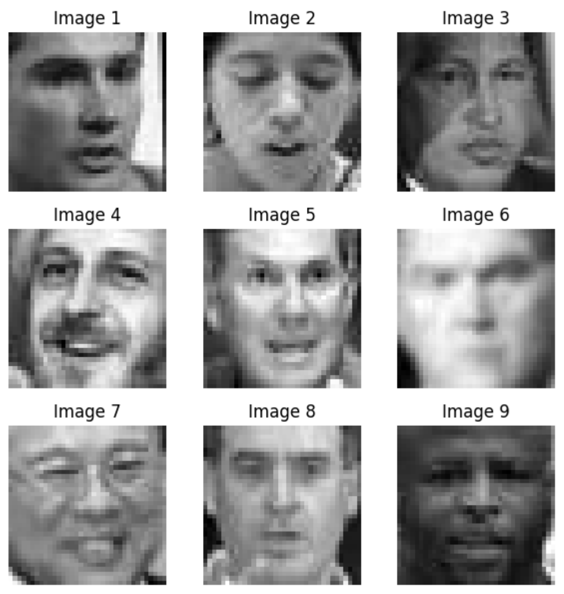


Figure 1: Training

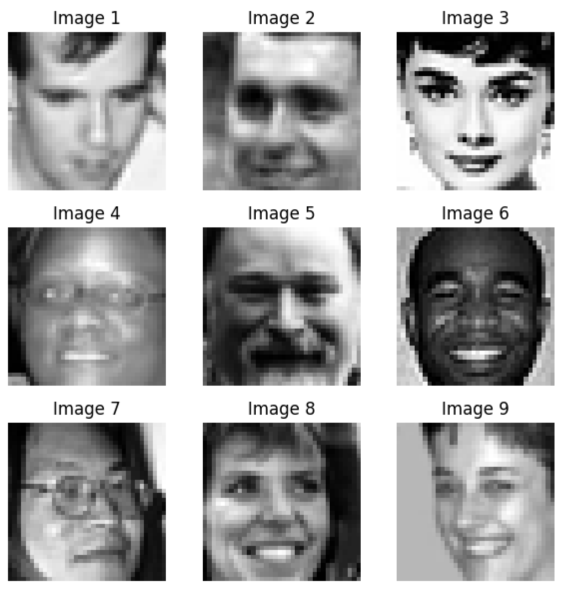


Figure 2: Test

Data augmentation

Data augmentation could improve the generalization ability of the model in order to predict new images and still maintain a considerable accuracy. Firstly, we crop the image to 0.8 to 1 times of the original size and resize it back to 36*36 size. And due to the different shooting angles of the pictures, we choose to flip the picture. Each picture has an 80% probability of being flipped left and right, as shown in the figure below. Finally, if the model performance is still average, we can add more operations. For example, between +10° and -10°, the image can be rotated by a random angle.

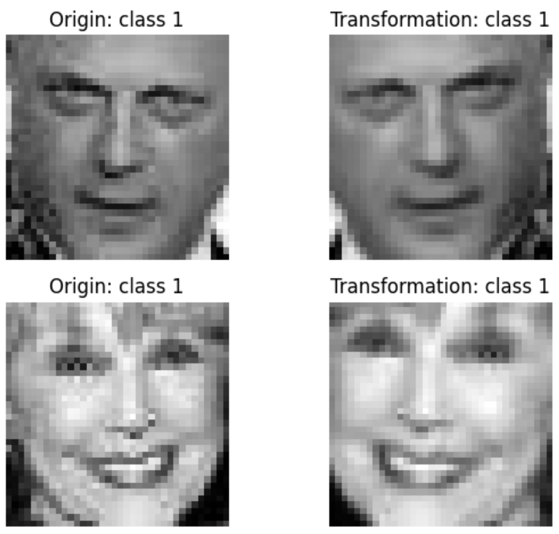


Figure 3: Data augmentation

Distribution Analysis

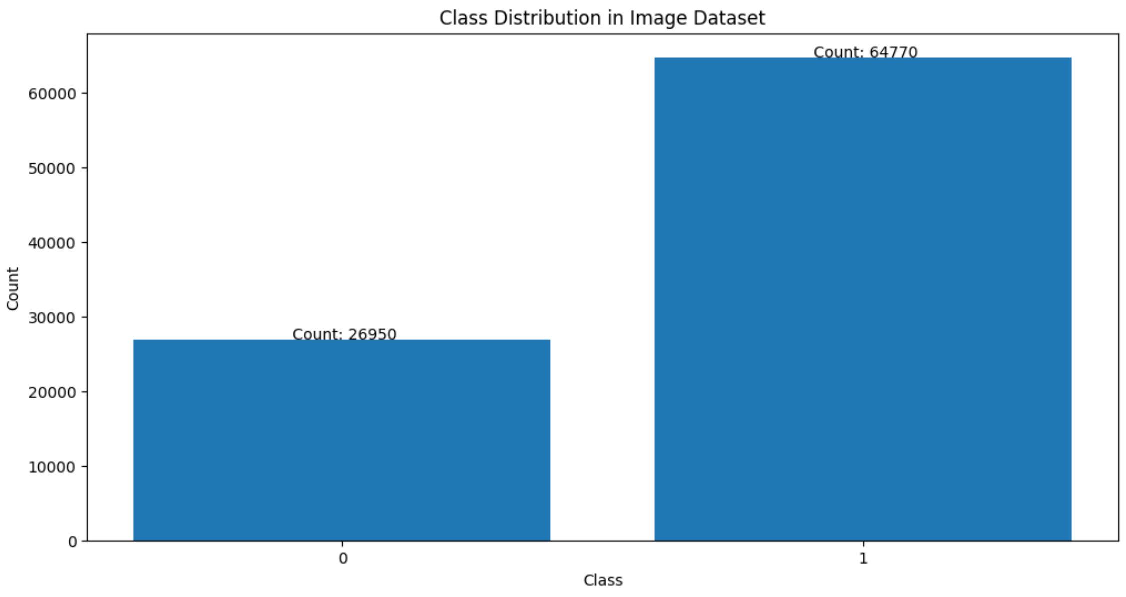


Figure 4: Data distribution

Unbalanced datasets can lead to uneven performance. In other words, it performs better on majority categories, but performs worse on a minority of categories, favoring the prediction of the majority category. To address this issue, we can weight the loss function, e.g, for the non-face and face categories, we could modify the share of their loss. We can also change the evaluation factors such as F1-score, to calculate the precision and recall. Finally, we choose another package, the Imbalanced DatasetSampler. In contrast to the classical oversampling and undersampling, it can estimate the sampling weights automatically.

Model Building

Face Classification Part

In this part, different models and strategies are trained and compared, and then we select the optimal model to achieve face detection in the second part.

For the data set, we take 80% of the data as the training set to train the model, and the remaining 20% of the data as the validation set to select hyperparameters. For example, K-fold cross validation or grid search traverses hyperparameter combinations. However, this is not the focus of this project, so we directly use the original data set for training, except for individual declared strategies. Finally, we compare the accuracy on the test set to select the optimal model.

For all classifiers, we use the CrossEntropy loss function to calculate the loss. Initially, we set the learning rate to 0.001 and did 10 iterations. After some attempts, the learning rate was set to 0.006 and did 15 iterations. In addition, we also applied the mini-batch strategy and set the batch size to 32. It introduces randomness to help the model escape from local minima better. Because using the entire dataset for gradient descent can cause the model to get stuck in a local minimum. Not only that, gradient descent calculations using the entire dataset can be very slow. mini-batch speeds up the training process by dividing the data set into small batches and using only 32 pieces of data at a time to calculate gradients.

CNN (Convolutional Neural Network)

First, we use a classic CNN framework. It has two convolutional layers and each layer has a pooling layer. Then, it's followed by three fully connected layers, and the activation function we choose is the ReLu function. Compared with no activation function or linear activation function, ReLu adds the ability of nonlinear mapping to the model, allowing the model to cope with more complex situations; compared with functions such as Sigmoid, ReLu changes all negative values to 0. The positive value remains unchanged. So there is no Vanishing Gradient Problem. However, the problem of neuron death may occur, and it can be improved to functions such as Leaky-ReLU.

For this framework, we trained two models using two different optimizers. The first is SGD (Stochastic Gradient Descent), and the second is ADAM (Adaptive Moment Estimation), which uses Adaptive Learning Rate. That is to say, the information of past gradients is used to determine the learning rate in each direction. The flatter the function, the faster the update.

Finally, for these two models we obtained the following accuracy on the test set:

- SGD: 94.6775 % with 7222 / 7628
- ADAM: 95.9885 % with 7322 / 7628

CNN with drop out (new architecture)

First, regarding the convolutional layer, we only added one more convolutional layer with 16 input and output channels. Because we think 16 important facial features is already a lot. At the same time, we also changed the size of the convolution kernel from 5*5 to 3*3 because we want to extract more detailed features. In addition, we also retain the number of pooling layers, because each time pooling is performed, the image size will be reduced exponentially. Through multiple convolutional layers, the model can gradually extract and combine higher-level features, helping us to retain more information.

Then regarding the fully connected layer, we only added a fully connected layer with 64 neurons, because it was directly reduced from 16*6*6 = 576 to 32, so we wanted to add a layer of transition. In addition, according to the specific convolution algorithm, we use the following formula (rounded down) to calculate the image size after convolution.

output_size = (input_size - kernel_size + 1) / stride

Finally, regarding regularization, for each fully connected layer, we apply the Drop out strategy. That is to say, during forward propagation, we randomly set the output of some neurons to 0, and during back propagation, only the gradients of activated neurons will be updated. During testing, this strategy is not applied. Drop out is a powerful strategy to solve overfitting and prevent the model from overly relying on individual neurons. In addition, we also added the L2 regularization term to the loss function, and we can accept a little accuracy reduction to improve the generalization ability of the model.

We obtained the following accuracy on the test set:

- Drop out: 96.6046 % with 7369 / 7628

CNN with early stopping (new architecture)

For this model, we also use the new architecture. The difference is that we use 80% of the previously defined data set to train the model, and the remaining 20% is used as a validation set to monitor the performance of the model, and stop immediately once it exceeds our tolerance. This is also a strategy to prevent overfitting. We implemented a very basic version, where we stopped training when the loss on the validation set did not fall below the previous best loss for more than three iterations.

For the model we obtained the following accuracy on the test set:

- Early stopping: 96.2769 % with 7344 / 7628

We end this section with a visualization of the training loss, with learning rates of 0.0001 and 0.01 and a total of 10 iterations. We found that the SGD optimizer does converge slower than ADAM. We also observed that when the learning rate is large, the model converges faster, and the loss in the validation set and training set does not change much, successfully triggering early stopping.

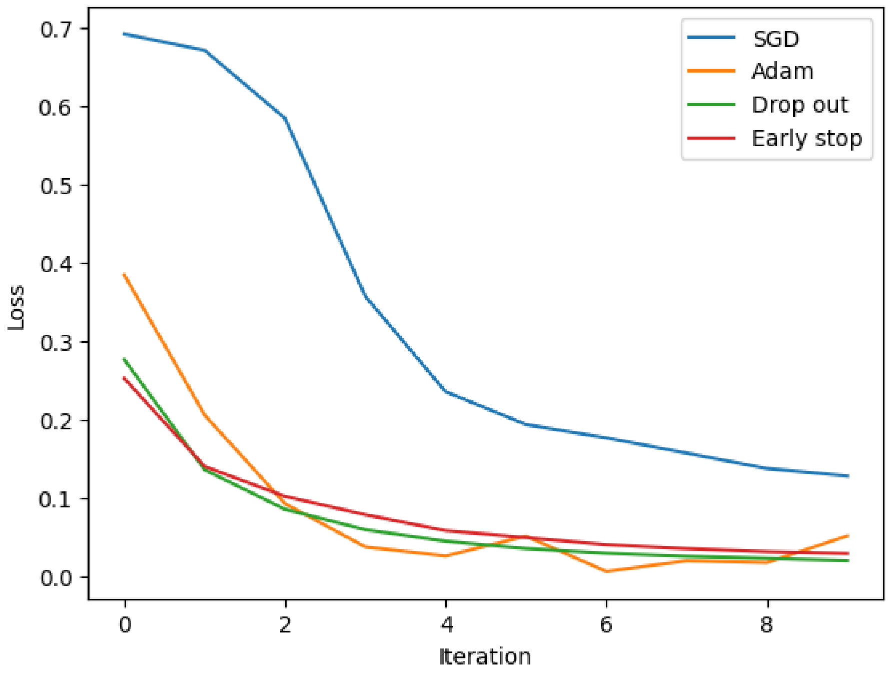


Figure 5: Training loss with lr = 0.0001

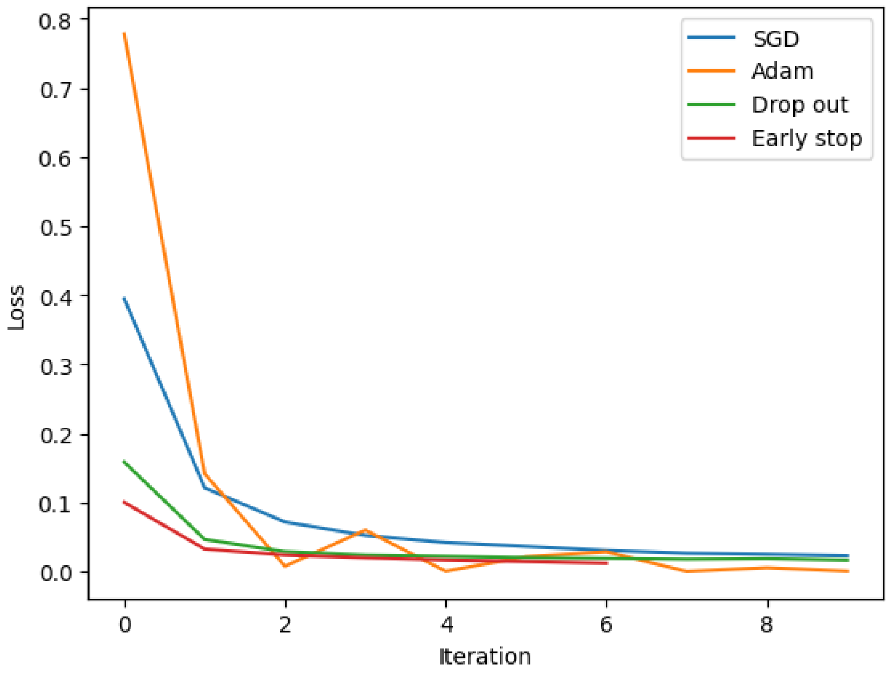


Figure 6: Training loss with lr = 0.001

We selected the Drop out model with the highest accuracy for the detection part. In this part we will understand, implement and compare different detection algorithms and models.

Sliding window & image pyramid

This is an exhaustive search, so it is relatively inefficient. In order to better understand how the algorithm works, we choose to implement it ourselves and get the following visualization results. It works as we expected, but due to display reasons, the window size changes and the image size is constant, actually the opposite is true.

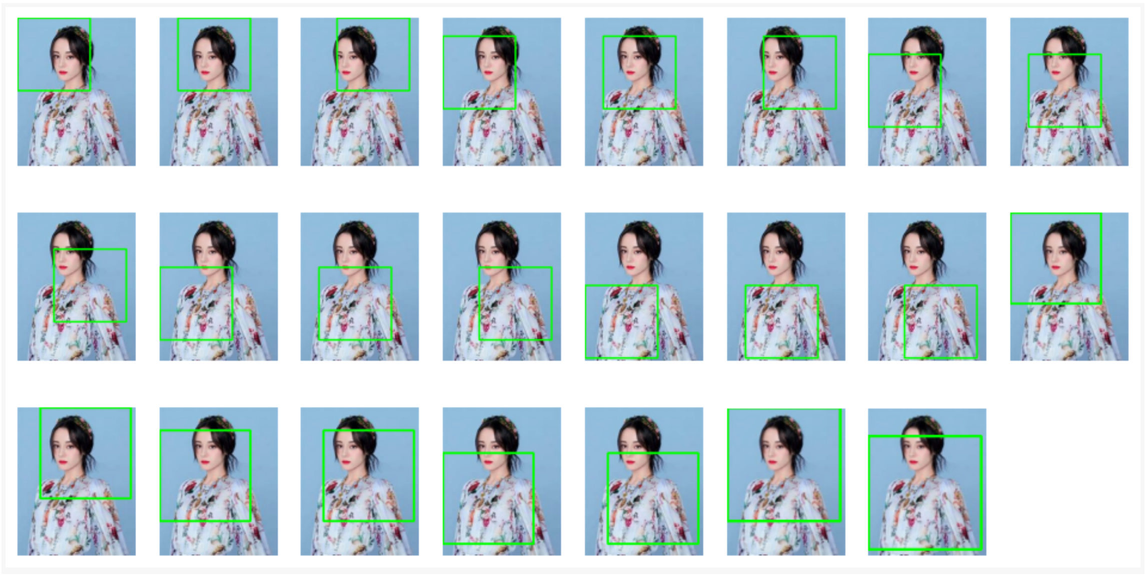
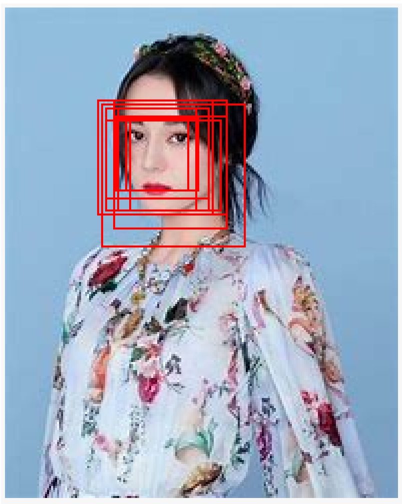


Figure 7: Sliding window visualization

Now we apply our trained classifier to detect faces and get the following results:



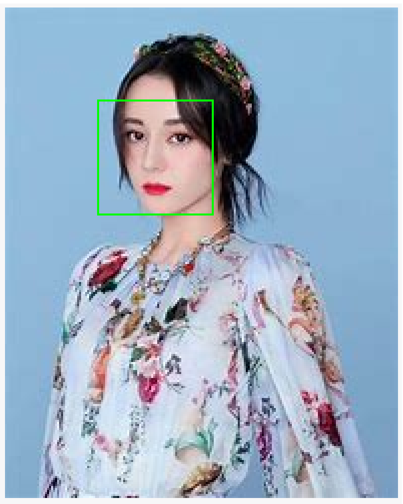


Figure 8: Sliding window detection Figure 9: NMS

NMS (non-maximum suppression) & IoU (Intersection over Union)

We also implement the NMS algorithm ourselves to facilitate our better understanding of its principles. It can reduce overlapping bounding boxes and improve the accuracy of target detection. See the code for the algorithm and detailed implementation. For a single face, we get a pretty good result. Next we test in multi-person pictures, the results are as follows:

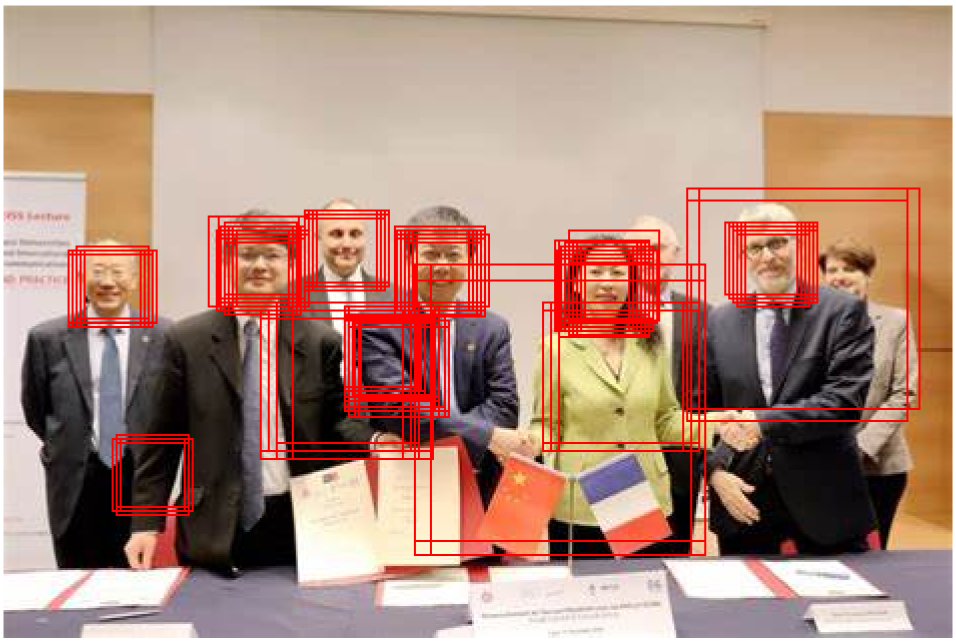


Figure 10: Sliding window detection

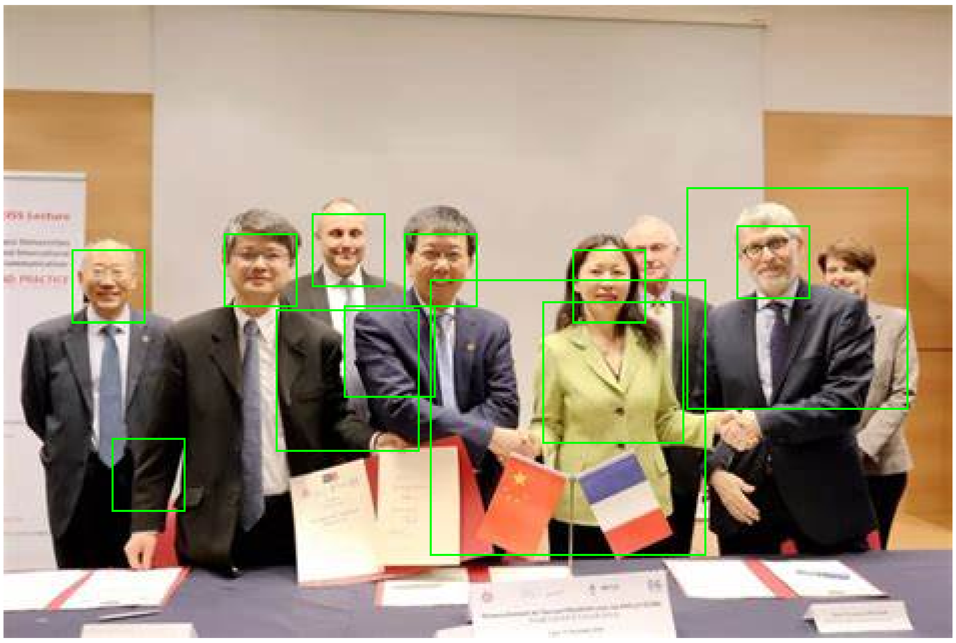


Figure 11: NMS

We found that except for the occluded faces, the rest were recognized. But we also observed some inaccurate windows, such as too small face-to-window ratio and false detections. We see that some errors originate from wrinkles in clothes, which may be similar to important facial features in grayscale images, such as the nose. So it is detected as a face. For windows with inappropriate proportions, we can add hard negatives. This means we can include this category training in the dataset. A face that we consider to be the wrong size is not a face either.

Selective search (Region proposal)

Selective search is used to generate proposals for regions that may contain objects. It generates potential regions of interest (RoIs) by merging image regions of different scales, colors, textures, and shapes. We directly called selective search and the OpenCV package to implement. We will use it to build the subsequent R-CNN model, which is to perform face detection on the recommended area. Compared with a sliding window, it will be faster and will avoid some false positive situations, because we only perform detection in the proposed area.

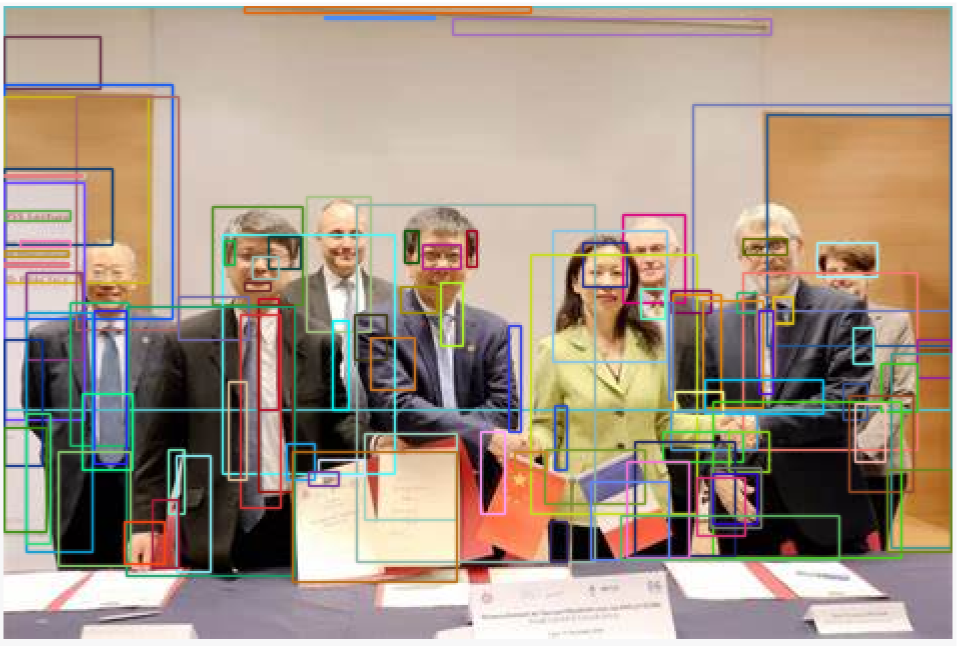


Figure 12: Selective search visualization

R-CNN (Region based CNN)

Since the sizes and shapes of regions vary, we do not want to directly resize the image or crop the image so as to lose information. We still apply a sliding window to each region for detection. Additionally, we removed areas with unreasonable aspect ratios. The result obtained is as follows:

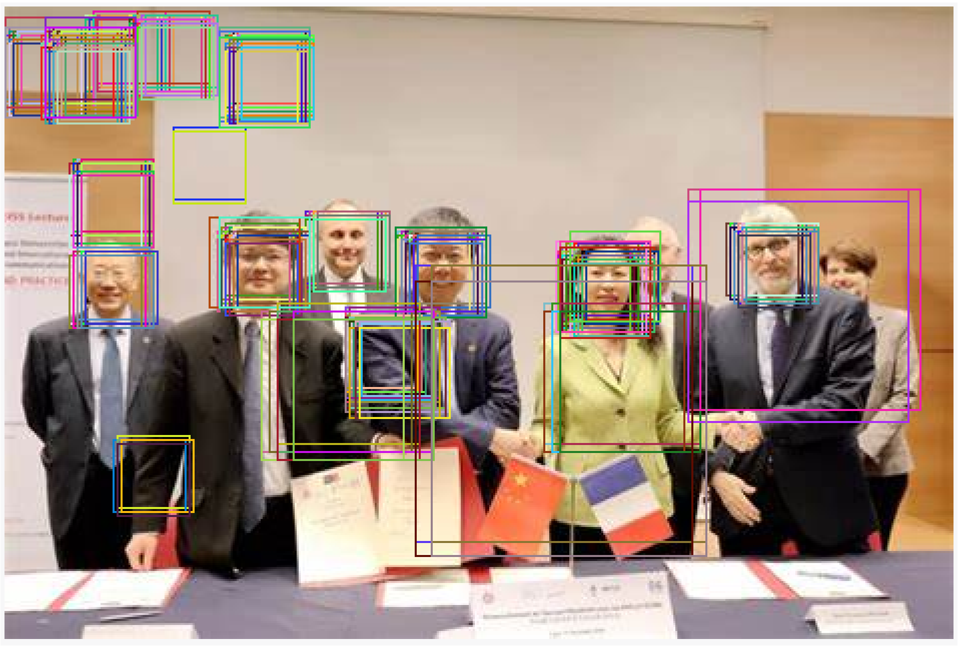


Figure 13: R-CNN detection

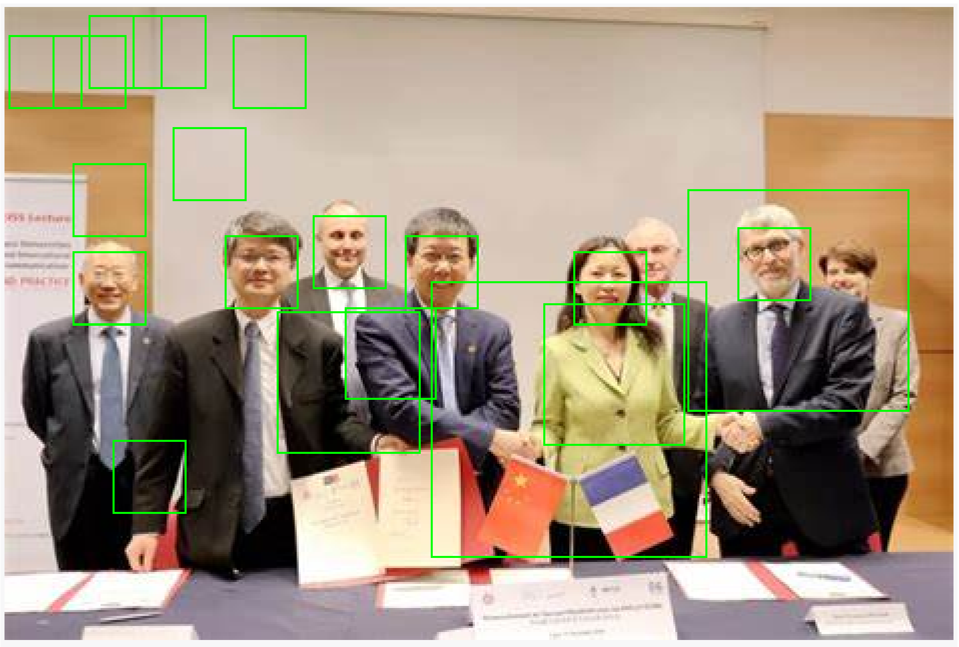


Figure 14: NMS

Overall, the effect is good, but we found that the window in the upper left corner is very strange. Our classifier considers almost solid-colored walls to be faces. This may be that the image itself contains some noise, which is more obvious in grayscale images, making it difficult for the classifier to classify. For example, 51% of the probability is a face and 49% is not a face. The specific reasons still need to be further explored.

FCN (Fully convolutional networks)

FCN replaces the full connection after the convolutional layer with a deconvolutional layer. The basic idea is that the convolution layer extracts image features, and the deconvolution layer performs image segmentation on the original image size. It accepts the entire image as input and assigns each pixel in the image to a different object category. Skip connections are used in FCN, which connect the feature maps in the convolutional layer with the feature maps in the deconvolutional layer. This helps to better restore details in pictures. We also added Batch normalization, it's used to normalize the input of a layer by adjusting its mean and variance. It helps stabilize and accelerate training of neural networks. In addition, it is worth mentioning that nn.ConvTranspose2d() in the Pytorch framework The function can be regarded as a deconvolution operation, but it does not actually calculate the transpose of the matrix. So the formula for calculating image size is slightly different:

$$output = (input - 1) * stride + outputpadding - 2 * padding + kernelsize$$

Unfortunately, we have already written the architecture of FCN, but due to time constraints, we were unable to find a suitable training set for training. So we apply GAP (Global Average Pooling) to transform it into an FCN classifier. This way we can apply our dataset for training. In short, GAP actually averages all pixel values of each channel map, and then obtains a new 1*1 channel map. Finally, the results obtained on the test set are as follows. But it’s even worse than 50%, so there is lots of amelioration to do:

- FCN (GAP): 34.5831 % with 2638 / 7628

Conclusion

In theory, we tried to apply all the theoretical knowledge we learned from lectures to our program as much as possible. In practice, we have gained a deeper understanding of how machine learning and neuronal networks work.

When it comes to aspects that can be improved, in addition to improving detection results. In the classification part, maybe we can combine several models learned in class or implement some other models such as XGBoost. In the detection part, we also want to try more Region Proposal strategies such as objectness. Also some new models like U-Net, which is similar to FCN, and more advanced single-stage detection, such as SSD and YOLO.

To sum up, our knowledge level and practical ability have been greatly improved in this project. Let us have a better understanding of computer vision.