Auteurs : Alexandre Bremard, Zineb Fadili, Zihao Hua, Stefan Ristovski

Report – Machine Learning Project

Table of Contents

I. Introduction

The goal of this project, a part of the Machine Learning Module, is to create a system for face detection using deep learning, or convolutional neural networks. In this project we prepared and found data for training and testing our model, we chose an architecture for our neural network. Furthermore, we implemented the bootstrapping technique to improve the accuracy of our model. Finally, we implemented the face detection system using the sliding window approach.

In this report, we will outline the process of creating the system, the technical challenges we faced, and the results we managed to obtain by implementing and testing our system.

II. Model

II.1. Description of the model

The model is a Convolutional Neural Network that takes as an input a 36*36 image and passes the image through the following layers:

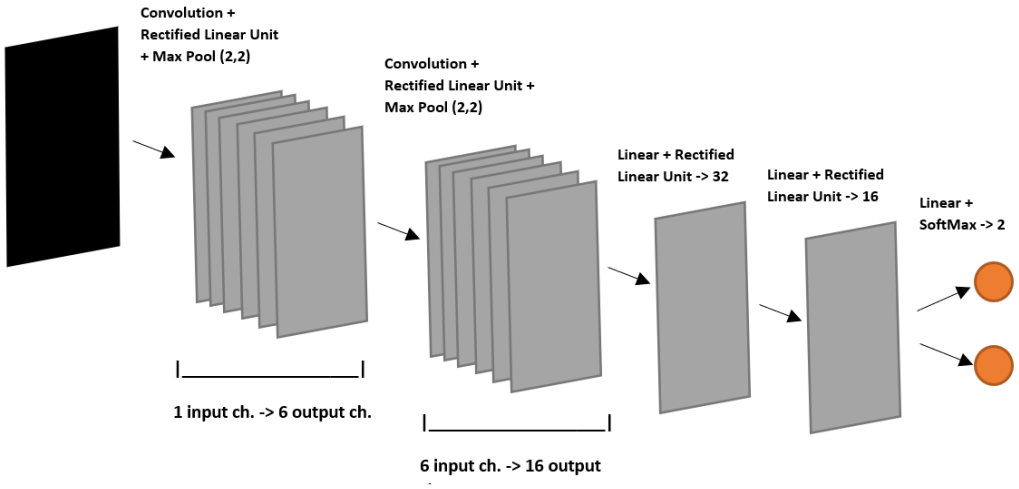

Figure 1: Schema of the Neural Network

We use a Convolutional Neural Network because it's the most appropriate for our case: image classification. CNNs have a strong ability to pick up highly specific features in images and therefore perform well on this type of project.

After each convolutional layer, we use ReLU as an activation function in order to only activate the neurons that produce positive values. This is because the CNN only learns an abstract representation of the image it is classifying, and when presented with an unknown image, doesn't know which representation is relevant. Using ReLU prevents irrelevant neurons from contributing negatively to the output of the network. Afterwards, we also perform max pooling so as to extract only sharp features and reduce the variance in computation by reducing the number of parameters of the network.

After the two convolutional layers, we perform two linear operations followed by an application of ReLU to reduce the dimensionality of our output, as we expect to have 2 classes (face 1 or no face 0).

Finally, we apply the SoftMax activation function on our last layer because we interpret the output of the network as a probability and SoftMax enables us to have values between 0 and 1 that sum up to 1.

II.2. Training the model

We have chosen to balance the given dataset for the training using the torch ImbalancedDatasetSampler to reduce the bias of our network. In fact, the given dataset is heavily imbalanced with an overrepresentation of the class 1 (almost thrice as much as 0). The sampler used makes sure that we have ~90000 representations of each class (face or no face) for the training. We have used it for the test data as well.

Although we have taken out 20% of the dataset for validation, we haven't implemented that part due to a lack of time.

We divide the training input into batches of size 32 and run the training for 4 epochs. As a criterion, we opted for the Cross Entropy Loss as it is particularly appropriate for probabilistic outputs

(since we use SoftMax). As an optimizer, we chose the Stochastic Gradient Descent with a learning rate of 0.01 established after practical experiments that have proven that this learning rate is fitting for a rather fast but stable loss decrease.

We have tracked the loss evolution during our training and printed in every 200-mini batch through our 4 epochs:

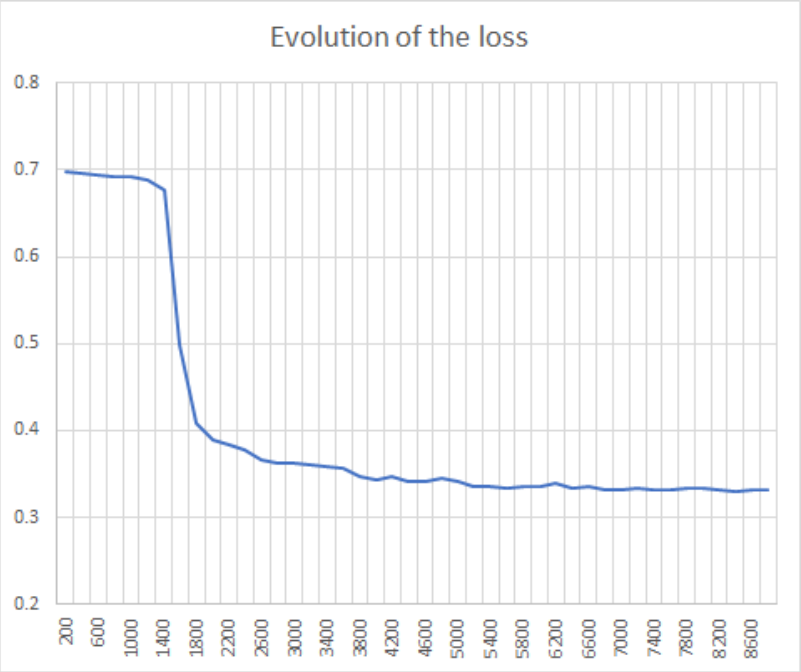

Figure 2: Evolution of the loss through the training

We arrive at an accuracy of 92% before doing the bootstrap method (see next section). We can see through the graph that the loss decreases quite well even though it starts to stagnate around the third epoch. Also, there is some noise that could be resolved through batch normalization.

III. Bootstrapping

The Bootstrap method is a sampling technique used in Machine Learning that is used to improve the stability and accuracy of ML algorithms used for classification. In our model in particular, we used bootstrapping to give better accuracy on face detection or classifying something as a face or non-face.

We started by searching for images that we can use as non-faces for the bootstrapping part, since the dataset we had in the beginning contained many more faces than non-faces. We wanted a diverse set of textured images, that will train the model to adapt to multiple different scenarios and be able to better classify faces in different conditions (lighting, background, image noise etc.) The dataset of images we used is from the Amsterdam Library of Textures, that contains 250 rough textures from materials taken in different conditions, amounting to a total of over 27,000 images that were used for bootstrapping. In order to adapt the images to the ones we already have used for training and testing previously, we had to reduce them to a size of 36 by 36 pixels, in black and white. Once our dataset was ready, we could implement the bootstrapping method in the training of our model.

Figure 3: Examples of textured images

The next step was implementing the bootstrapping process itself. Firstly, we train the model with the original dataset that was provided, using the entirety of the non-face images (around 27,000 images). We balance the dataset, so we include the same amount of face images for the training. After the initial train and test, we continue with the actual bootstrapping process involving our texture images. We run the images through our model to detect false positives. When a false positive is detected, and the probability that the model gives crosses a threshold, we inject the image into our training set. Probability in this context refers to the degree of certainty that the image contains a face according to the model. Once we run through our texture images, and we've added the false positives into the training set, we balance it again by adding more face images. We also shuffle the images before each training, to maintain the accuracy of the back propagation. We lower the threshold and run through the same process again.

For our model, we start off the bootstrapping process with the initial threshold of 0.8 (probability that there is a face in the image, values between 0 and 1). After each run, the threshold is lowered by 0.2, until we reach 0.2 on the last run. Below is a simplified pseudo-code of our bootstrapping process:

```
T ← Threshold of probability
while T > 0.2 do
        get training data
        shuffle training data
        balance the dataset
        train the model
        Ti ← get texture images
        for image in Ti
                Proba ← run image though model
                if Proba > T
```

```
                    add image to training data
                end if
            end for
            T = T - 0.2
        end while
```

Bootstrapping is essential in improving the accuracy of neural networks and classifiers. We have reached an accuracy of 94% percent after using it (an increase of 2%). According to a research paper on the topic of image transform bootstrapping, the performance of a classification system depends a lot on the data used for training, and slight variation in the scene content of an image and distracting regions may prevent the correct classification in some cases. Therefore diversifying the dataset and this method of injecting textures can improve the performance of the classifier and output more accurate results. In our own testing, the difference between the model we have simply trained on the initial dataset and the model we bootstrapped using textured images is significant, as seen in the images below.

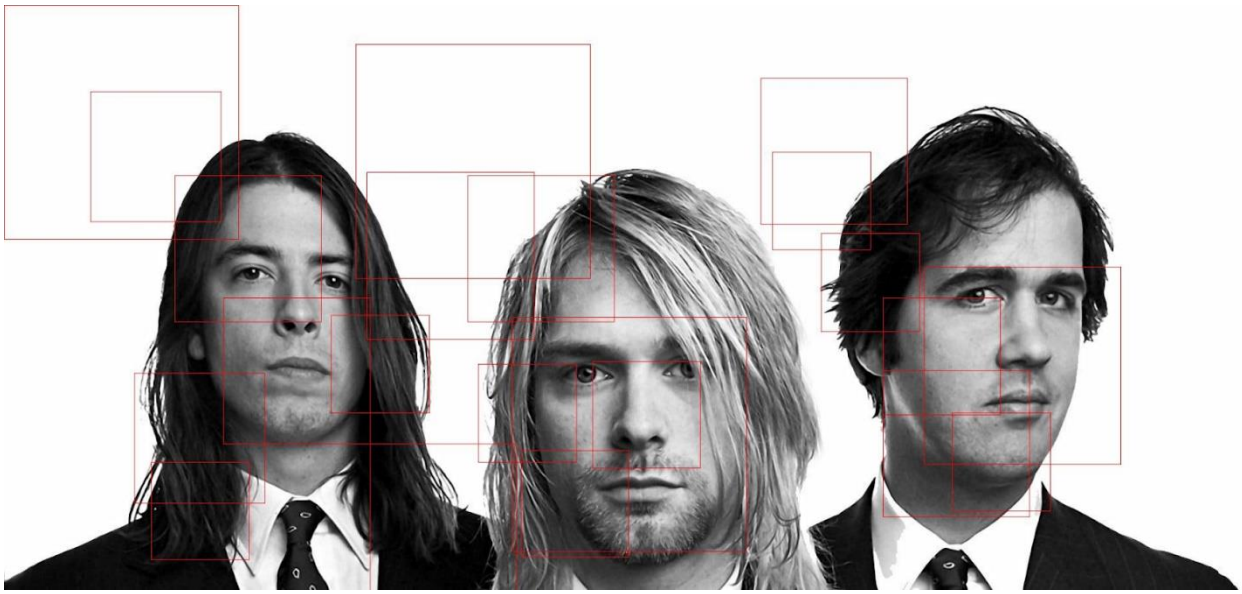

Figure 4: Results from running the non-bootstrapped model on an image

Figure 5: Results from running the bootstrapped model on an image

A red square represents a face that was detected by the model with a probability higher than 0.9. You can find more information about how this was achieved in the "Face Detection on a big image" chapter. The difference in accuracy is significant. The bootstrapped model is better at detecting facial features, and mostly avoids problematic areas around the hair, or in the background. In contrast to that, the non-bootstrapped model, it is harder to come to conclusions about the faces detected in the image, nor discern any pattern or features in most of the detected areas.

To conclude, the bootstrapping method and the injection of textured images has greatly improved our model's accuracy and its ability to distinguish facial features from other textures and backgrounds. It is possible to imagine expanding this solution by including even more images containing not just textures, but objects and backgrounds, so the classifier can distinguish better between them and a person's face.

IV. Face Detection on a big image

IV.1. Sliding window approach

Now that we have a trained model for face detection on a 36 x 36 image, we can apply it to a bigger image using the sliding window approach. The idea is to slide a 36 x 36 window across the entire image, moving n pixels (n will be referred as the step size) to the right each time. In our experiment we set the step size to 6 pixels (one sixth of the window width/height) to make sure we don't miss any parts on the face.

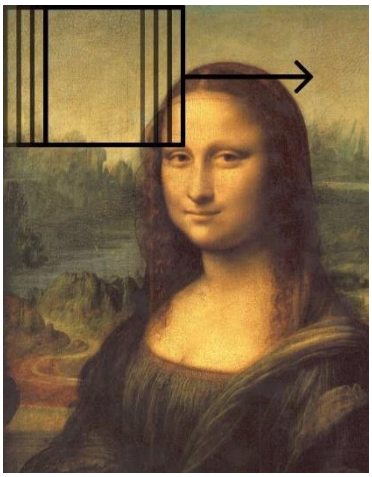

Figure 6: Sliding window approach
(Step size and window sizes are not at scale)

However, we cannot apply this method to a big image (1000 x 1000 for example) directly because a face might have a size of 200 x 180 pixels which will never fit our sliding window, so we need to resize the original image by a scale s. Our rescaling formula is as follow:

$$S = \frac{h}{36 * i}$$

h is the original image height

i is a face/image ratio that we iterate between 1.5 and 6. For example i=6 means that the face's height is one sixth of the image height. (The Mona Lisa painting above has a ratio of ~3)

To make sure that we capture all faces on the image, we need to run a sliding window scan at different scales (which is why we iterate over multiple face/image ratio) and then combine the results. As we can see on *Figure 5: Results from running the bootstrapped model* on an image above, the red squares have different sizes depending on the scale we used.

IV.2. Non-maximum suppression technique

After detecting some credible face images, we noticed that most of them overlapped with each other. To eliminate these redundant results, we applied non-maximum suppression (NMS), a technique to filter the predictions of an object detector, especially suitable for our case, which used a sliding window approach.

As its name implies, its algorithm is based on excluding other images that have lower confidence to make the result more accurate. It takes three inputs, a list of coordinates of images **B**, the corresponding confidence scores **S** and an overlap threshold **N** which is a constant between 0,3 and 0,5. And finally returns a list of filtered images **D** which is initially empty.

The implementation steps are as follows:

1. Find the most "confident" image in the input set, remove it from **B** then put it into **D**.

2. Calculate IoU (Intersection over Union) of this selected image with every other image in **B**. If the value of IoU is greater than the threshold **N**, remove the related image in the set **B**.
3. Repeat steps 1 and 2. Find the image having highest confidence score in the rest of set **B**, recalculate their IoU and dispatch them by comparing IoU and the threshold. Loop until the set **B** becomes empty.

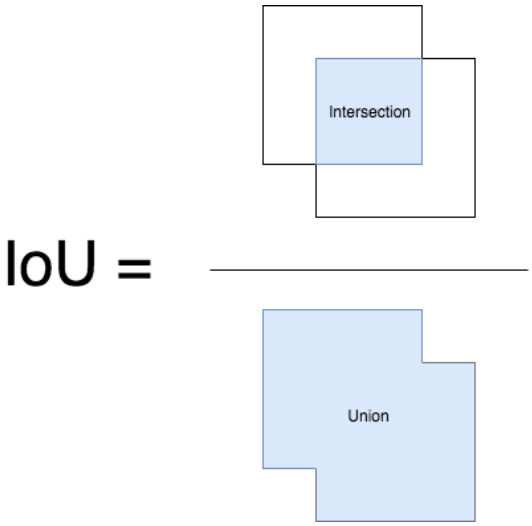

Figure 7: Intersection Over Union IOU

In our case, we'll take the data representing "is face" at the output of the neural network as the confidence score. They take values in the range between 0.95 and 1, and there is not a significant difference between them. This leads to the fact that the final output of NMS may still have errors, but there has been a significant improvement in the elimination of redundant results.

V. Further Improvements

We can implement multiple changes and improvements to our face detector in order to improve the accuracy of our model. Here are some suggestions on what could be improved:

- General accuracy of the model, running more epochs on training.
- The model seems to detect faces on dark zones of the image. We noticed that the original training dataset consists of grey faces on a white background so maybe the model interprets dark/bright contrasts as faces. The hypothesis needs to be verified
- Optimize bootstrapping threshold. Currently, the threshold starts at 0.8 and decreases to 0.2 by increments of 0.2. It may be useful to do smaller increments, such as: 0.8→0.6→0.45→0.35→0.3→0.25→0.2
- Add other datasets for bootstrapping to increase diversity.
- Add a data augmentation layer to the training image with faces to improve performance by generalization.
- Use a part of the training dataset (around 20%) for validation during testing.

VI. Conclusion

In conclusion, this project has allowed us to create a concrete example from the theoretical knowledge we gained through the lectures on Machine Learning and Neural Networks. We have built a Convolutional Neural Network, trained it, and tested it. This project was also a way for us to learn techniques that help improve the accuracy of a network such as bootstrapping and approaches to apply the neural network's functionality in various manners, i.e., not only detecting a face at the center of an image but exploring an image at different scales.