Udacity Computer Vision Report: Facial Keypoint Detection

Arief Ramadhan

May 2020

1 Problem Definition

This project aims to build a facial keypoint detector that draws 68 keypoints given an image of a single person. Facial keypoints are small magenta dots on each of the faces as shown in figure 1. They mark important areas of the face: the eyes, corners of the mouth, the nose, etc. Keypoints are relevant for a variety of tasks, such as face filters, emotion recognition, and pose recognition.

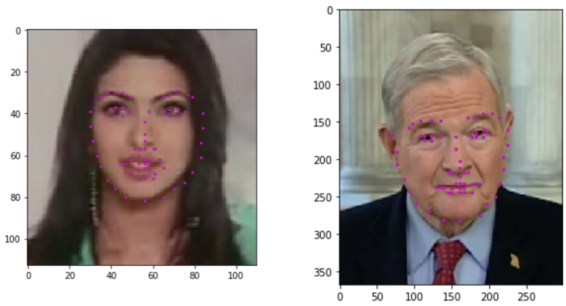

Figure 1: Facial Keypoints Examples [1]

1.1 Problem Statement

Given images of a person, locate 68 key points of the person's face. Figure 2 displays the 68 keypoints of a person's face.

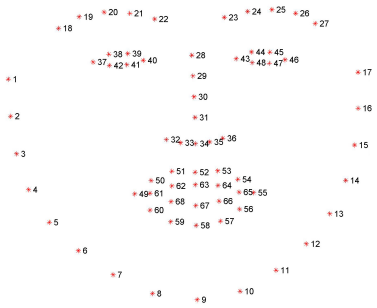

Figure 2: 68 Keypoints [1]

1.2 Metrics

We evaluate the performance of our facial keypoint detector by looking at its Mean Squared Error (MSE) and visual inspection. MSE is the most popular loss function for regression.

It is the mean of the squared differences between true and predicted values, as show in this formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i')^2$$

Where:

- n = number of data
- y = actual value
- y' = predicted value

Figure 2 shows that each keypoint is given a number from the range of 1 to 68. In a single image, each predicted keypoint is compared againts the actual keypoint. For example, predicted keypoint number 1 will be compared to actual keypoint number 1 to calculate the loss.

2 Datasets and Inputs

The data was extracted from the Youtube Faces Dataset, which includes faces of people in YouTube videos. In total, there are 5770 color images in the dataset: 3462 for training and 2308 for testing. The information about the images and their keypoints are summarized in the provided CSV files. Each image contains a single face and 68 keypoints locations in x and y coordinates.

3 Methods

3.1 Preprocessing

We first convert our image into a grayscale, as color does not play an important role in presenting facial keypoints. We also resize the image and crop it towards its center, to minimize memory use and speed up the model training.

3.2 GPU

Compared to CPU (Central Processing Unit), a GPU (Graphics Processing Unit) is better at fetching large amounts of memory. This makes GPU better than CPU in training a deep learning model. In this project, we use pytorch GPU library in jupyter notebook. The GPU that we used was provided by Udacity in the Computer Vision course workspace.

3.3 Model building: Neural Network

Our Neural Network model consists of four types of layers: convolutional, max-pooling, dropout, and fully connected. The model was built through trial and error. Figure 3 displays our model architecture. Each CNN (Convolutional Neural Network) layers has ReLU (Rectified Linear Unit) activation function and is followed by a max-pooling layer. The last FCN (Fully Connected Layers) is the output layer which has 136 values or 68 pairs of keypoints coordinates (x,y).

We used Adam optimizer and Mean Square Error loss function. Adam is the combination of RMSprop and momentum [2]. It uses the squared gradients to scale the learning rate like RMSprop, and takes advantages of the exponentially decaying average of past gradients similar to momentum [2]. Adam is a current trend in Machine Learning, as it is straightforward to implement and computationally efficient. In its original paper, Adam was shown

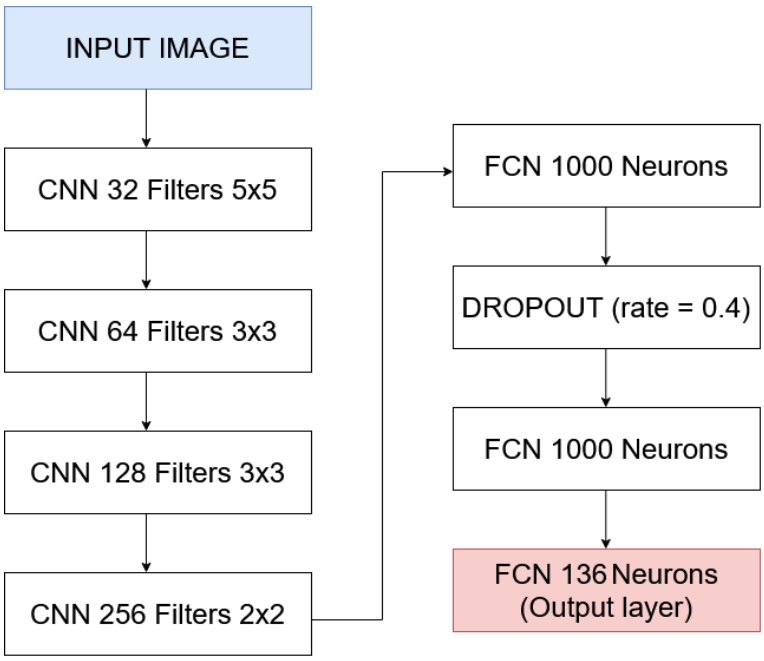

Figure 3: Network Architectures

to outperform other optimization technique (such as, AdaGrad, RMSProp, SGDNesterov) in terms of speed of convergence [2].

We chose Mean Square Error (MSE) because we dealt with a regression task. MSE is the most commonly used loss function for regression [3]. It is sensitive towards outliers and is commonly used when the input is normally distributed [3].

3.4 Haar Cascade

We utilized OpenCV's pre-trained Haar Cascade classifiers to have a complete pipeline of keypoint detection. For further references of this Haar Cascade model see this documentation.

Haar Cascade is commonly used for face detection. It works by applying multiple filters that detect distinctive features of faces, such as eyes, mouth, and nose. Figure 4 shows how a Haar Cascade works. In our case, we first detected all the faces within an image using the Haar Cascade model, before drawing the keypoints using our trained Neural Network model.

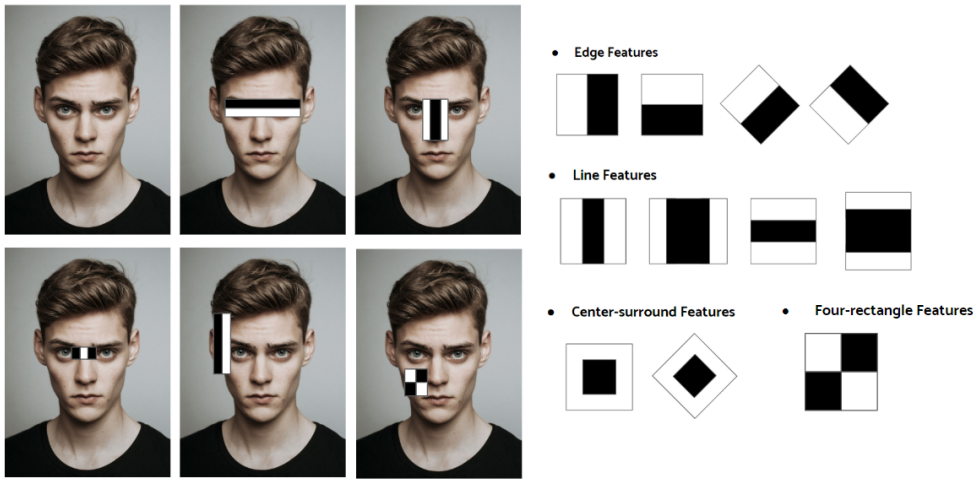

Figure 4: Haar Cascade for Face Detection [4]

3.5 Complete Pipeline of Facial Keypoint Detector

Our facial keypoint detector consisted of a Haar Cascade model that detect faces in an image and a neural network model that detect and draw keypoints.

Overall, the complete pipeline of our facial keypoint detector is:

1. Detect all the faces in an image using OpenCV pretrained Haar Cascade model
2. Preprocess the face images: convert to crop, resize, and convert them to grayscale to fit our trained model input
3. Use our trained model to detect facial keypoints on the the detected face images

Figure 5 visually explains the pipeline of our facial keypoint detector.

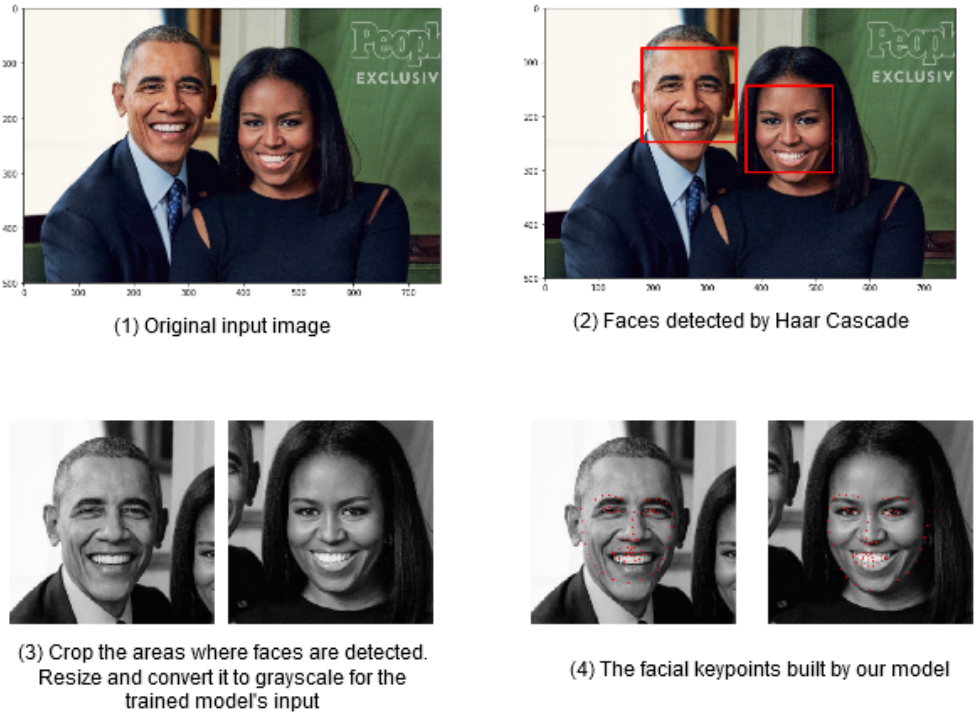

Figure 5: Facial Keypoint Detector Pipeline

4 Results and Discussions

Figure 6 displays our predicted keypoints in three test image. We considered this result to be acceptable, because the predicted keypoints closely resemble the actual keypoints. Numerically, we obtained a loss value of 0.03 after training the model for 40 epochs.

Figure 6: Actual (Green) vs Predicted (Purple) Keypoints

5 Conclusions

In this project, we built a facial keypoint detector using deep learning methodology and a pretrained Haar Cascade model. Our deep learning model consisted of convolutional layers and fully connected layers as displayed by Figure 3. We obtained a loss value of 0.03 which visually, as shown in Figure 6, indicate that our model was able to generate keypoints that closely resembled its actual position.

References

[1] C. Camacho. (2018) P1 facial keypoints. [Online]. Available: https://github.com/udacity/P1_Facial_Keypoints

[2] V. Bushaev. (2018) Adam — latest trends in deep learning optimization. [Online]. Available: https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c

[3] Peltarion. (2020) Mean squared error. [Online]. Available: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/mean-squared-error

[4] J. Jeong. (2019) Computer vision for beginners: Part 3. [Online]. Available: https://towardsdatascience.com/computer-vision-for-beginners-part-3-79de62dbeef7