Convolutional Neural Networks and Their Use for Face Alignment: An In-Depth Study

Jasper Lemberg

I. Summary and Justification

To complete this task, a CNN was implemented. First, because CNN is just a specific type of neural network, it is important to understand what a neural network is. In a neural network, a series of nodes applies weights to an input and then outputs the combined weight and input after going through an activation function. This is essentially a threshold applied to the output. This process is repeated for many layers of interconnected nodes until it outputs the explanatory variable (Russell and Norvig).

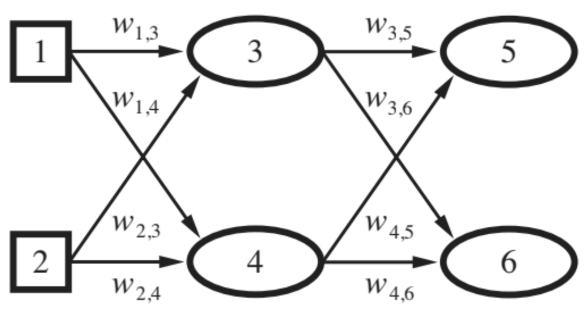

Fig. 1. An example of a basic neural network. Russell and Norvig, 2010

A CNN is a neural network that includes a combination of convolutional, pooling, and dense layers ("Machine Learning Glossary"). The convolutional layer is the most important factor in why a CNN was chosen for this task. In this layer, a filter scans over an image and takes the dot product of the pixels under that filter. This is repeated for all parts of the image and the output is a condensed version of the original image outlining certain features (IBM Cloud Education). This is done using multiple filters, which can then be used to identify different parts of the image. For example, one filter early on in the network could outline hard vertical lines. Later on, one could outline the side of a face. By the end, many individual features could be detected per layer. This makes the code very condensed compared to another method, so it is more comprehensible and easier to use.

This model consisted of the following layers:

- Two consecutive convolutional layers with thirty-two 7x7 filters. The first of these layers requires an input of size (244, 244, 3), the size of each image.
- A 3x3 pooling layer.
- Two more convolutional layers. These are identical to the first convolutional layers except with 5x5 filters instead of 7x7 filters.

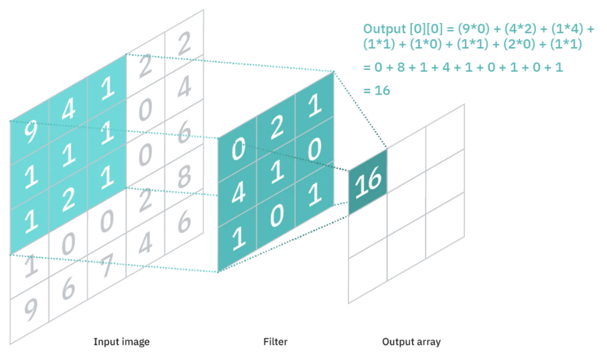
Output [0][0] = (9*0) + (4*2) + (1*4) + (1*1) + (1*0) + (1*1) + (2*0) + (1*1)
= 0 + 8 + 1 + 4 + 1 + 0 + 1 + 0 + 1
= 16
Input image
Filter
Output array

Fig. 2. Convolution of an image. IBM Cloud Education, 20 October 2020

- Another 3x3 pooling layer.
- Two final convolutional layers with 3x3 filters instead of 5x5 filters.
- A flattening layer to make the three-dimensional images into a one-dimensional output.
- A dense layer to compute eighty-four outputs, one for every x and y for the forty-two landmarks.
- A reshape layer to convert the output to an array of size (42, 2).

The standard practice is to use thirty-two or sixty-four filters per convolutional layer. Thirty-two were used because it was closer to the number of points needed for the output (forty-two). The size of the filters also decreases throughout the network. This was done so that larger features could be identified from the unprocessed image first. This would theoretically decrease processing time. Pooling layers were placed after the first two sets of convolutional layers in order to further reduce the size of the images. If this was not done, a very large array would go to each subsequent layer, which would make finding features even more difficult.

A CNN was ultimately chosen for this task because it could cycle through multiple epochs of individual features quickly and without user intervention. In another model like cascaded regression, parameters would need to be manually fed into the model and changed after each run. Considering the size of the test set and the complexity of the task, this would be painstaking and time-consuming. However, in a neural network, these weights can be automatically updated for each epoch via backpropagation. This consists of calculating the error from the output layer, moving that value back to the previous layer, adjusting the weights accordingly, and repeating until it reaches the beginning of the network (Russell and Norvig). Using this algorithm, the optimal weights can be discovered without the need for human input and, more importantly, the addition of human error in the model creation process. This becomes even easier because the model was built using Google's TensorFlow library, which trains and compiles the model with single lines of code.

Image preprocessing was limited for this experiment, as the training, testing, and example images were already fitted to a standard size (244x244) with each face and the face's eyes roughly in the same position. Should the model not have been as effective, two methods could have been employed. First, the area around the face could be blurred using a Gaussian blur. This would limit the noise in areas besides the face, although would require extra face detection to single out the face versus the rest of the background (automaticaddison.com). This idea was ultimately scrapped because a Gaussian filter would eventually be applied to the images during convolution, so any blurring beforehand would be redundant.

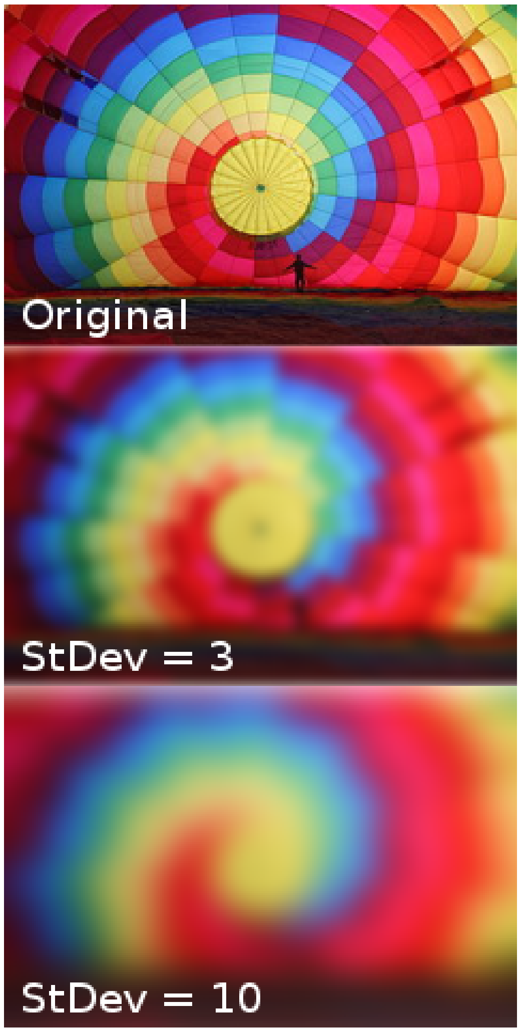

Fig. 3. An example of an image with a small and large Gaussian blur. *Automatic Addison*, 23 October 2019, https://automaticaddison.com/pros-and-cons-of-gaussian-smoothing/.

Dimensionality reduction using Principal Component Analysis (PCA), which would limit dimensions to the most important principal components, was another preprocessing option. Considering that the model has eighty-four outputs (forty-two x and y coordinates), this would be advantageous.

II. RESULTS

A. *Quantitative*

Overall, the model performed extremely well. After running for fifty epochs (planned for 100, but early stopping was implemented due to a lack of decrease in mean squared error), the MSE of the model was 19.3609 and the validation MSE was 25.3427. The MSE per epoch is shown in figure 4.

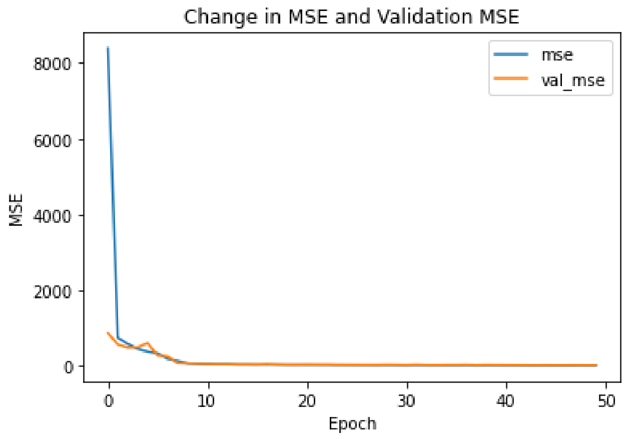

Fig. 4. MSE for the training and validation data over fifty epochs

When run on the training images, the average Euclidean distance between the predicted and ground truth points was 4.924172903245675 and 91.75% of the distances were less than 10. However, there were some outliers, with some points being over fifty off. This will be explained in later sections.

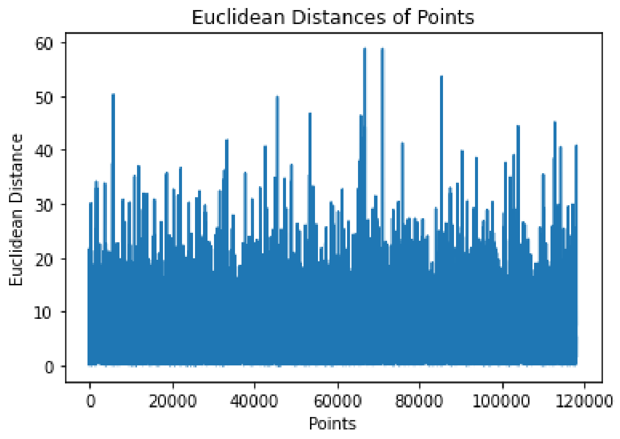

Fig. 5. Euclidean distance between predicted and ground truth points

The cumulative density of the predicted points matches that of the actual points fairly well, meaning that they are distributed fairly evenly.

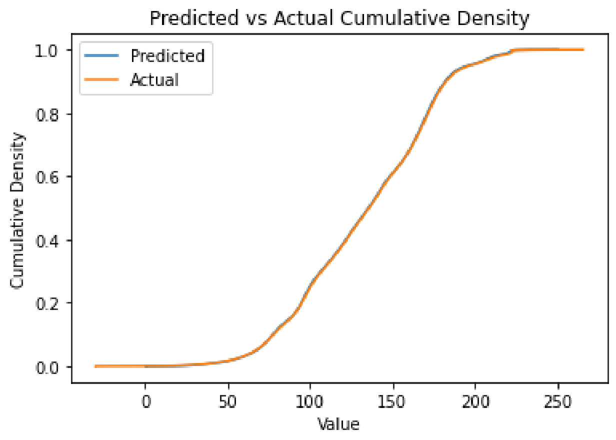

Fig. 6. Euclidean distance between predicted and ground truth points

While the predicted points were distributed similarly compared to the ground truth points, there were some discrepancies. In general, there were slightly more predictions made on the top and left sides of each image versus the bottom and right sides. This may have to do with the face orientations and the presence of hair, which could obstruct the top part of the face and cause predictions to be pushed higher than they should.

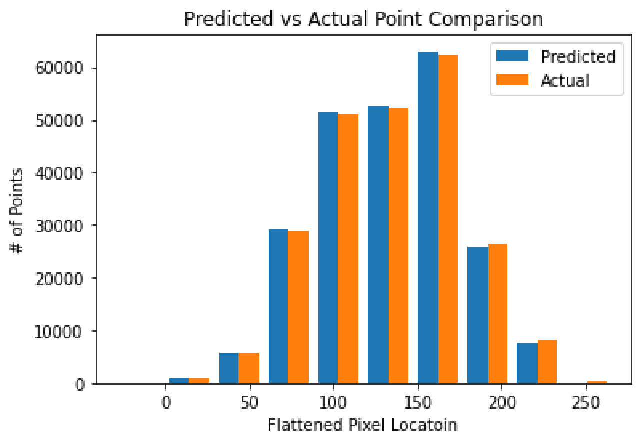

Fig. 7. Euclidean distance between predicted and ground truth points

B. *Qualitative*

Figures 8-13 consist of the model's results for the example data. This consists of images of a variety of different races, genders, and orientations relative to the camera, and expressions. In general, the model is best at locating the eyes (points 18-21). This could be because of the significant change in color in many examples. It fails in two notable examples. First, in figure 11, the person is making an unusually round expression with their mouth. This is not picked up well because it is unlike the fairly horizontal straight faces and smiles in other examples. Augmenting the data to include more expressions like this (or just this image rotated in certain ways) could limit this problem. Also, when the face is not directly looking at the camera like in figure 12, the edge of the face may not be detected as accurately. Including a point descriptor such as SIFT in the model could help here because it would then account for point angle and orientation.

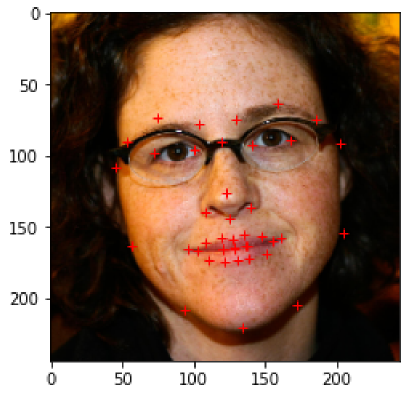

Fig. 8.

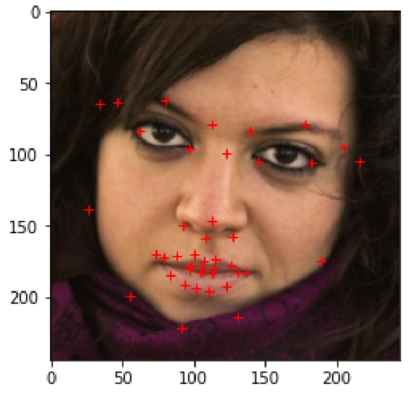

Fig. 9.

III. LIP COLOR MODIFICATION

The lip_recolor() takes in three parameters: the image, its landmark points, and a color. Before doing anything else, two copies of the image are created: one so that the image from the dataset is not modified and another to be used as an overlay. The method then uses two steps to change the color of lips on the overlay while maintaining realism. First, the fillPoly() method from OpenCV fills in a polygon on the image using the model's landmark points. Then, the addWeighted() method, also from OpenCV, combines the overlay with the original image with an alpha value of 0.25. This value determines the opacity of the overlay, which should be limited to maintain the the image's texture underneath the overlay. This produces very consistent results, as shown in figure 14.

The main issue with this method is that if there are any objects in front of the mouth (e.g. a finger, hair), that will also be colored in. To fix this, a GAN or similar model could be used first to detect if there is anything obstructing the mouth.

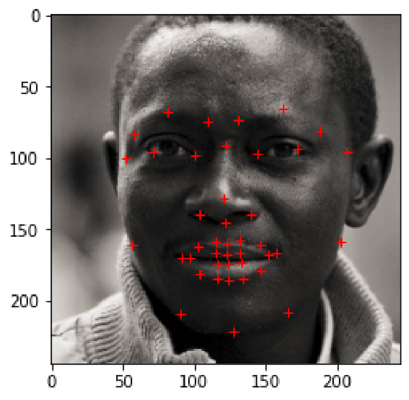

Fig. 10.

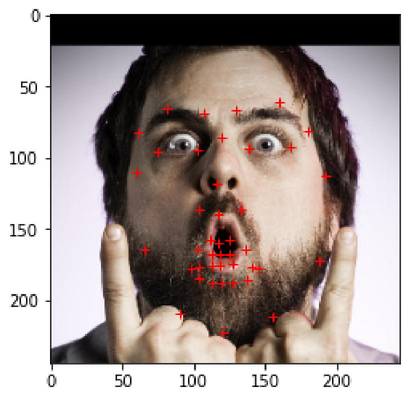

Fig. 11.

References

[1] IBM Cloud Education. "What Are Convolutional Neural Networks?" IBM, 20 Oct. 2020, https://www.ibm.com/cloud/learn/convolutional-neural-networks.

[2] "Machine Learning Glossary ." *Google Developers*, Google, https://developers.google.com/machine-learning/glossary/.

[3] "Pros and Cons of Gaussian Smoothing." *Automatic Addison*, 23 Oct. 2019, https://automaticaddison.com/pros-and-cons-of-gaussian-smoothing/.

[4] Russell, Stuart J., and Peter Norvig. "Artificial Neural Networks." *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Upper Saddle River, 2010, pp. 727–737.

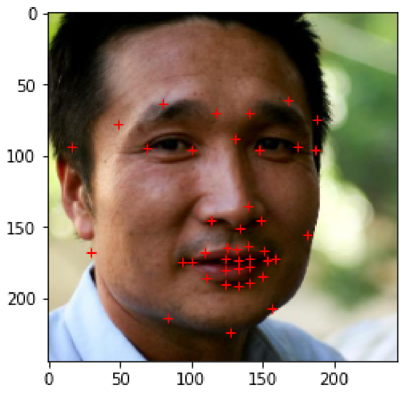

Fig. 12.

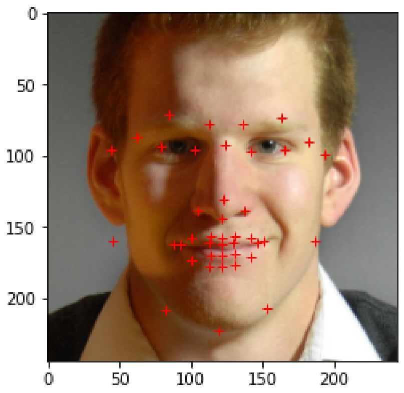

Fig. 13.

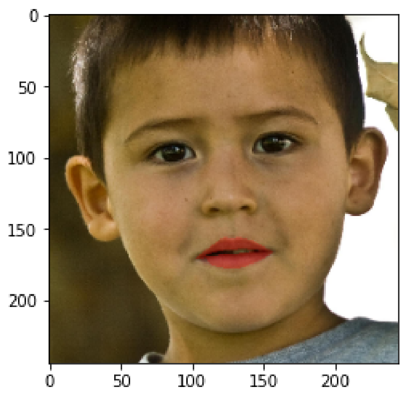

Fig. 14. Image with modified lip color