**Keypoints**

**[ ABOUT ]**

**Method Used**

The purpose of this portion of the project was to train a machine learning model to identify 15 keypoints on human faces: center of left eye, tip of nose, etc. To achieve this, we used a Convolutional Neural Network design recommended by [Peter Skvarenina](https://towardsdatascience.com/detecting-facial-features-using-deep-learning-2e23c8660a7a) and implemented with the Keras open source neural network library running on top of the TensorFlow backend.

**Dataset**

We used a dataset provided on kaggle.com, for their [Facial Keypoints Detection](https://www.kaggle.com/c/facial-keypoints-detection/data) competition. The dataset consisted of 8,832 images – 7049 with identified keypoints for training and 1783 images for testing. However, upon investigation we discovered that nearly 5000 of the training images had only 5 keypoints identified. Some users of the dataset have filled in the missing information with averages from the other images, but we thought this would introduce error and chose to use only the 2000 images with 15 keypoints identified.

[ **PROCESS** ]

**CNN Implementation in Keras**

We implemented Skvarenina’s model in Keras with the structure shown in the diagram below. In this structure the images are preprocessed to grayscale and converted to an array of 96x96 numbers between -1 and 1 as the input to the model. The output of the model are 30 floating point numbers giving the x and y coordinates of each facial keypoint.

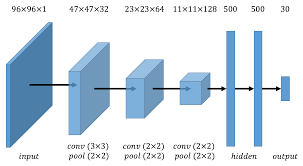


Diagram: Facial keypoints detection using Neural Network, by Zhang & Meng, Stanford University

**Network Architecture**

The training processing for the model was done in Google’s Colab notebook environment using their GPU processor for acceleration (approximately 10x faster than it ran in Colab without a GPU). We ran the training for 100 epochs, taking just over 3 hours.

**Performance**

The results from our training process are shown below. After 100 epochs the improvement had leveled off to a point that additional epochs would not provide better accuracy without changing the method. Our model got to .7 accuracy for the training, but only .3 accuracy for the test pass.



Skvarenina reports that “with some trivial tricks” he can achieve 80-90% validation accuracy with 30 epochs. The details of these tricks are not provided. However, upon visual inspection of our results we felt that the accuracy was fairly good in terms of keypoint identification that was close to the training locations. In the image below the manually identified keypoints are shown in green while our model’s predicted keypoints are shown in red.



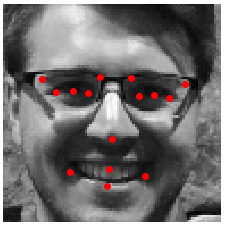
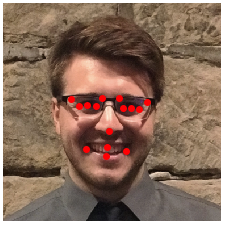
We felt this placement accuracy was adequate to continue with the application demonstration portion of this section of the project

**Application Demonstration**

To demonstrate the usefulness of models that can locate keypoints on a facial image, we decided to create an application that would accept a photo of a face, used the trained model to locate the keypoints on the face, and finally use that information to place an overlay image on the appropriate area of the face.

We found that the model did not work well on standard portraits. This was because the training images were cropped to only include the face. We used OpenCV’s Viola-Jones detector based on Haar cascades to find the bounding box for the face and used that for cropping. The model was much better at predicting the keypoints from cropped images.

The results of the application demonstration are shown below.

Original image Cropped and grayscale Model predicted keypoints Original image with keypoints

Several overlay images using the various predicted keypoints for appropriate placement.

[ **LEARNING**]

Datasets merit careful inspection. Fortunately, there are many datasets freely available on the internet that are useful for machine learning applications. Unfortunately, the completeness and quality must be inspected. In our case, although the source of the dataset (Kaggle) was credible, it was not exactly as described. This may have been intentional as a problem to be solved, but would trip up an unwary user. Also the quality of images within the dataset was inconsistent with many blurred images and some illustrations (not actual photographs). This image variety could be useful for some applications where this type of image may be encountered. For our purposes, they likely reduced the possible accuracy of the model.

Extreme accuracy may not be necessary for many applications of machine learning. Setting a ‘good enough’ standard for outputs will likely reduce model complexity and processing time required to achieve the desired results.

Colab is a useful environment for accessing GPU performance with a notebook programming format.