

# Machine Learning Engineer Nanodegree

## Capstone Project

---

Nicholas Low  
April 14, 2017

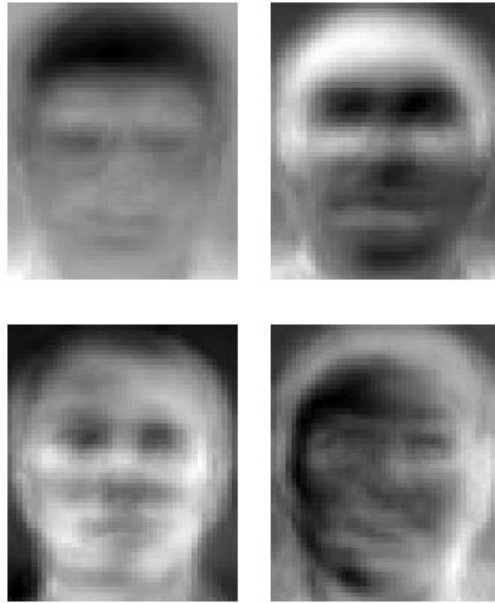
## I. Definition

---

### Project Overview

Facial keypoints detection is a field of study that has been relevant for many years due to its potential in creating a non-invasive method of identifying points on a face that can be used to determine any number of details about a person. However, facial features can differ greatly on individuals due to many variations of conditions in gathering images of individuals; therefore, determining accurate information from these features can prove to be difficult.

A well-known beginning to facial recognition systems is due to Tuevo Kohonen, a Finnish academic, who explained that a neural network could perform facial recognition only on aligned and normalized face images utilizing eigenvectors eventually becoming known as eigenfaces seen in Figure 1 [1]. These eigenfaces are representations of what the average face may look like based on a large set of data and allow a computer to determine what parts of the face generally look like. In modern facial keypoints detection, eigenfaces still exist as a primary method of identifying parts of the face although the science has become much more advanced.



*Figure 1: Example of Eigenfaces*

By improving the method of facial keypoints detection systems, a number of solutions can occur. People may utilize these systems for lie detectors, medical diagnosis, biometrics, etc. On a bigger scale, every person may eventually have their own Sherlock Holmes, without the personality accompanying it (At least in the show "Sherlock"), to deduce the truth behind the many clues that exist within one's face.

## **Problem Statement**

The problem to be solved is to predict keypoint positions/locations on face images. The goal is to predict the areas and parts where the mouth, eyes, ears, and nose are for all images with high accuracy. By determining the positions of these keypoints, a machine can gather information about the face. This will be a regression supervised learning problem as the data set includes labeled training data that uses continuous values in the form of pixels to learn from. A potential way to solve the facial keypoints detection problem is to utilize neural networks. The strategy to achieve the desired solution will be:

1. Attempt to filter unnecessary pixels in the image
2. Preprocess outliers by determining quartiles
3. Apply linear regression and determine errors for benchmark model
4. Apply neural network and determine errors to compare to benchmark

5. Utilize k-fold and Grid search cross-validation to ensure the model generalizes to new data in the most optimal way

## Metrics

The evaluation metric that will be used to quantify performance of both models is going to be the root mean squared error. The root mean square error is relevant for this problem because it works very well for a model whose main purpose is to predict. The root mean square error is the average distance of a data point from the fitted line which will help determine the generalization of the models to new data.

There will be 30 errors displayed for each model representing the spread in the values of the output compared to the model. The errors will be found by determining each of the predicted keypoints' deviation from their actual values, squaring these values, averaging the squared values, and then taking the square root. Mathematically, each of the errors will be determined by the equation:

$$RMSE_{errors} = \sqrt{\frac{\sum_{i=1}^n (\hat{y}_i - y_i)^2}{n}}$$

*Figure 2: Root Mean Square Error Equation*

## II. Analysis

---

### Data Exploration

The details of the dataset to be used in this model were taken from Kaggle's Facial Keypoints Detection competition [2]. The 15 keypoints, to be expected as outputs, being considered that represent elements in the face where left and right refer to the point of view of the subject are:

left\_eye\_center, right\_eye\_center, left\_eye\_inner\_corner, left\_eye\_outer\_corner,  
right\_eye\_inner\_corner, right\_eye\_outer\_corner, left\_eyebrow\_inner\_end,  
left\_eyebrow\_outer\_end, right\_eyebrow\_inner\_end, right\_eyebrow\_outer\_end, nose\_tip,  
mouth\_left\_corner, mouth\_right\_corner, mouth\_center\_top\_lip,  
mouth\_center\_bottom\_lip.

Each data point for these elements is specified by an (x,y) real-valued pair in the space of pixel indices. Data points that are missing are left blank. The input image is displayed

in the last field of the datasets consisting of a list of pixels (ordered by row), as integers between (0,255). A sample of this data set is described in Table 1:

*Table 1: Sample of Data Features*

left_eye_center_x	left_eye_center_y	Image
66.03356	39.00227	... 238 236 237 238 240 240 239 241 241 243 240 239 231 212 190 173 148 122 ...

There are 7049 images in the training set and 1783 images in the testing set and each of these images are 96x96 pixels.

The statistics that were calculated to examine the dataset are the minimum, maximum, mean, median, and standard deviation of the 15 keypoints' x and y values. The statistics will be displayed as in Figure 3:

```
Statistics for left_eye_center_x
Minimum pixels: 22.7633446452
Maximum pixels: 94.68928
Mean pixels: 66.3590212448
Median pixels: 66.4975659574
Standard deviation of pixels: 3.447988302
```

*Figure 3: Sample of Statistics Data*

The rest of the statistics can be viewed in the code.ipynb file associated with this project under the Data Exploration section. Based on the calculated statistics, there must be outliers in all the keypoints considering standard deviation is fairly small for all the keypoints and the maximum and minimum seem fairly outside the range of the mean. The median and mean are fairly equal for all keypoints. The largest difference between the median and mean is for right\_eye\_outer\_corner\_y with a difference of approximately 0.1664; therefore, the data is neither skewed to the right nor the left.

There are also 4909 missing values in the dataset which may make the row of data irrelevant where there is a missing value unless that missing value is also missing its x,y pair.

## Exploratory Visualization

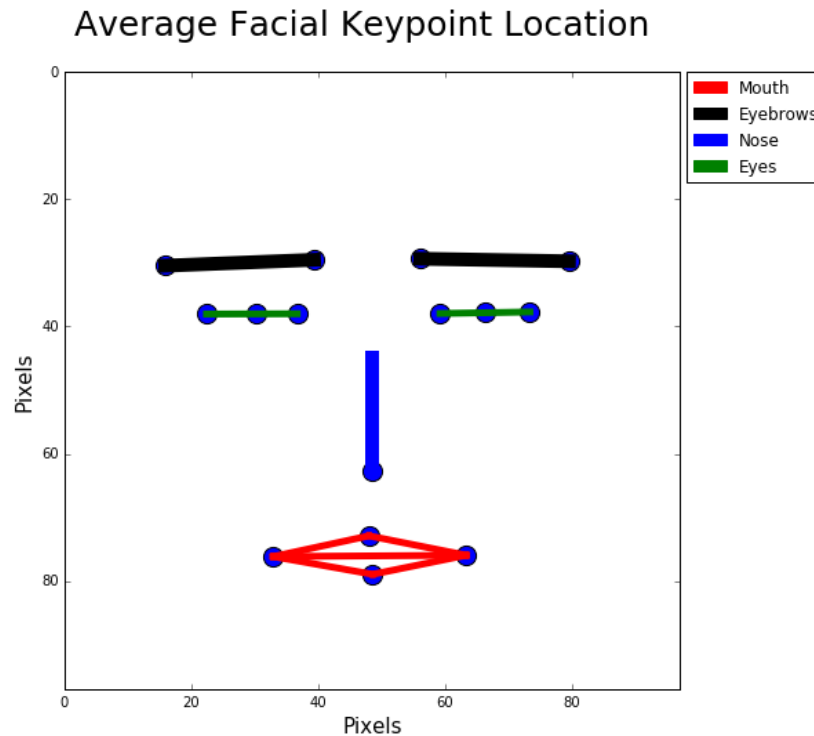


Figure 4: Visualization of Facial Keypoints

Figure 4 depicts a visualization of an average person's facial keypoints locations. Lines were drawn to visualize the keypoints better and give an idea of what they would look like on an image. This current iteration uses all the data points which consist of possible outliers.

The average mouth looks to be in an area that proportionally fits where a human mouth might be. The upper lip (47.98, 72.91) is above the lower lip (48.57, 78.97) and the corner of the mouth are to the right (32.90, 76.18) and left (63.29, 75.97) of the upper/lower lip. The tip of nose is at (48.37, 62.72) pixels which in Figure 4 is in a probable location for a nose. The center of the left eye is at (66.36, 37.65), the inner corner of the left eye is at (59.16, 37.94), and the outer corner of the left eye is at (73.33, 37.70). The right eye is in similar areas except mirrored on the opposite side; however, the face is not completely symmetrical.

## Algorithms and Techniques

The algorithm this project will utilize to solve the problem is neural networks seen in Figure 3. A neural network is a machine learning model where inputs get sent through a set of neurons and output based on an activation function. The activation function is determined during the training phase which can be done in a number of ways. Specifically, the sklearn library has an MLPClassifier that allows for the integration of a

neural network that trains utilizing backpropagation. Backpropagation is a technique that sends an input through the network. The output of this network is then compared to the true output and an error value is computed for the neurons in the output layer. These errors then are propelled backwards until all the neurons have an error value to describe its contribution to the output.

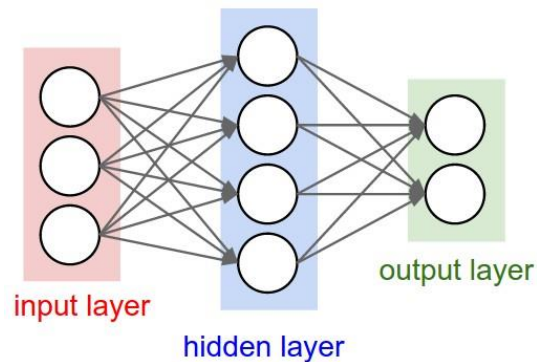


Figure 5: Neural Network Example

The classifier has a large number of default parameters that will be used that are associated with this classifier are solver, alpha, hidden\_layer\_sizes, and random\_state. These parameters are defined in the sklearn library [3] as:

**hidden\_layer\_sizes** : tuple, length = n\_layers - 2, default (100,)

The ith element represents the number of neurons in the ith hidden layer.

**activation** : {'identity', 'logistic', 'tanh', 'relu'}, default 'relu'

Activation function for the hidden layer.

**solver** : {'lbfgs', 'sgd', 'adam'}, default 'adam'

The solver for weight optimization.

**batch\_size** : int, optional, default 'auto'

Size of minibatches for stochastic optimizers.

**learning\_rate** : {'constant', 'invscaling', 'adaptive'}, default 'constant'

Learning rate schedule for weight updates.

**max\_iter** : int, optional, default 200

Maximum number of iterations.

**random\_state** : int or RandomState, optional, default None

State or seed for random number generator.

**shuffle** : bool, optional, default True

Whether to shuffle samples in each iteration.

**tol** : float, optional, default 1e-4

Tolerance for the optimization.

**learning\_rate\_init** : double, optional, default 0.001

The initial learning rate used. It controls the step-size in updating the weights.

**power\_t** : double, optional, default 0.5

The exponent for inverse scaling learning rate.

**verbose** : bool, optional, default False

Whether to print progress messages to stdout.

**warm\_start** : bool, optional, default False

When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution.

**momentum** : float, default 0.9

Momentum for gradient descent update.

**early\_stopping** : bool, default False

Whether to use early stopping to terminate training when validation score is not improving.

**validation\_fraction** : float, optional, default 0.1

The proportion of training data to set aside as validation set for early stopping.

**beta\_1** : float, optional, default 0.9

Exponential decay rate for estimates of first moment vector in adam.

**beta\_2** : float, optional, default 0.999

Exponential decay rate for estimates of second moment vector in adam.

**epsilon** : float, optional, default 1e-8

Value for numerical stability in adam.

After the initial training and error testing using the root mean squared error equation discussed in the Metrics section, the solution will be maximized by testing with k-fold and Grid search cross-validation. K-fold Cross Validation will test the model's ability to generalize by partitioning the dataset into k different sets of training and testing samples. Then the technique will apply the neural network model to each of the k sets and average the errors to find whether the model generalizes to an independent data set well.

Grid search cross-validation will be a technique to find the best set of parameters to use in the neural network by iterating through all possible combinations of parameters. The technique will then apply the neural network model to each combination and determine the combination with the least error.

## Benchmark

The benchmark model is a linear regression model utilizing the sklearn library. The error values display the accuracy of the predicted values where 0 is the absolute best possible value. Considering most values are not near 0 in Table 2, this model is clearly not the best model and can be used as a benchmark to improve upon. The average root mean squared error of all the values is approximately 2.6. This value will be useful in



comparing possible solutions as neural network's validation loss is outputted in average mean squared error.

*Table 2: Benchmark Error Values*

Keypoint	Mean-Squared Error Value (x,y)
left_eye_center	3.35655638587, 3.00659826681
right_eye_center	3.62549863274, 2.62479747181
left_eye_inner_corner	3.21890866112, 2.43413717571
left_eye_outer_corner	5.46501014269, 3.76696884129
right_eye_inner_corner	3.51918267712, 2.26374336594
right_eye_outer_corner	6.23334070135, 3.38509566459
left_eyebrow_inner_end	6.30298251755, 5.79883191317
left_eyebrow_outer_end	8.3075473728, 9.53307994484
right_eyebrow_inner_end	5.49655021744, 5.98976093614
right_eyebrow_outer_end	10.6152351573, 8.40806984996
nose_tip	6.4053908771, 12.4011882545
mouth_left_corner	9.72771687866, 10.1476079771
mouth_right_corner	8.83732425413, 11.3415232284
mouth_center_top_lip	4.51902474406, 14.0610461972
mouth_center_bottom_lip	5.2222874231, 16.7996997835

## III. Methodology

---

### Data Preprocessing

Figure 6 depicts the code utilized to preprocess the data. First, the data had several issues regarding its usability. Null values exist for 4909 values in the data set. Data could not be fitted without dealing with the null values. Considering transforming the null values into another value would affect results, the best option was to just remove them using dropna().

The data also needed to convert training image data, X\_train, into something the sklearn Linear Regression Model could recognize. Therefore, the data was expanded to split the string for each image into separate columns using the expand option in str.split(), converted all the strings into floats with astype(float), and then converted into a numpy matrix with as\_matrix(). X\_test was much easier as the contents were recognized as integers by Python, so the values were just extracted with Pandas' values function.

The data set had to be split into training and test sets as the test set provided by Kaggle does not have any true results. 1/3 of the training set was converted into a test set in order to provide a large enough data set for training the model.

```
#Drop missing value data
df = df.dropna()

#Breaks the string apart with ' ', expands each pixel value into its own column, converts to a float32, converts to a numpy
#array and divides by 255.0 to normalize the values between 0->1
X = df['Image'].str.split(' ', expand=True).astype(np.float32).as_matrix()/255.0

#Extract the output values
Y = df.drop('Image',1).values

#Break up data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size = 0.33, random_state=65)
```

*Figure 6: Preprocessing Code*

## Implementation

The implementation of the solution to the problem was done through the use of neural networks. The neural network was created utilizing the Keras library. This neural network has three layers total: input, hidden, and output similar to Figure 7. The input layer would receive the training data and learn over 400 iterations which was determined after multiple attempts to find where it converges to its best root mean squared error value. Figure 8 depicts the code and the initialization of the neural network. This input layer was set up to have 1 input node to take in the 9216 pixels with a rectifier activation function. The hidden layer has 15 neurons with the rectifier activation function. Finally, the output layer has 30 neurons for each possible output and uses a linear activation function.

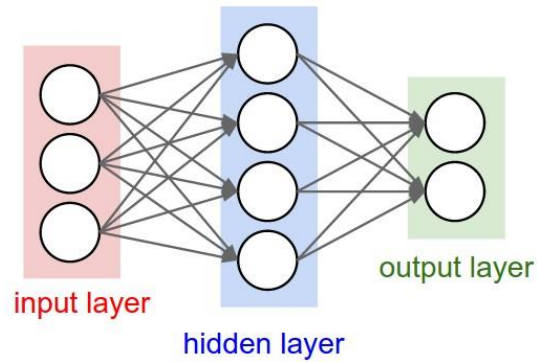


Figure 7: Example of Neural Network Layers

```
#Dense model
model_Dense = Sequential()
model_Dense.add(Dense(1, input_dim=9216, activation='relu'))
model_Dense.add(Dense(15, activation='relu'))
model_Dense.add(Dense(30, activation='linear'))
adam_Dense = optimizers.adam(lr=0.05, decay=2.5e-5)
model_Dense.compile(loss='mean_squared_error', optimizer= adam_Dense)
```

Figure 8: Dense Neural Net Code

## Refinement

The initial attempt at improving the neural network was to manipulate parameters to run more efficiently and accurately. I used Grid search cross-validation to check for a few parameters; however, the project was limited due to memory restraints. For the traditional neural network the parameters used were epochs and batch size. The Grid Search tested for [50,200] epochs and [50,200,600] batch sizes. Based on the Grid Search, the best traditional neural network had the greatest results at 50 epochs and 600 batch size as defined in Figure 9.

Best Traditional Neural Network: 52.697220 using {'nb\_epoch': 50, 'batch\_size': 600}

Figure 9: Grid Search Traditional Neural Network

```
Epoch 47/50
0s - loss: 10.2205 - val_loss: 10.0375
Epoch 48/50
0s - loss: 10.2150 - val_loss: 10.0349
Epoch 49/50
0s - loss: 10.2125 - val_loss: 10.0341
Epoch 50/50
0s - loss: 10.2102 - val_loss: 10.0341
```

*Figure 10: Traditional Neural Network Result*

However, the traditional neural network did not result in a better root mean squared error score than the benchmark model with score of 3.17 as seen by taking the square root of the validation loss score in Figure 10. After researching the topic of utilizing images in neural networks, convolutional neural networks or CNNs came up as an altered neural network that creates models based off of image inputs more efficiently. The reason CNNs are better for images is that they make the assumption that inputs are images which introduces properties that are encoded into the architecture. These properties are generally the width, height, and channels of the image. This allows the function to be more efficient and reduce the amount of parameters in the network.

Figure 11 is the code implementation of a 2D convolutional neural network. The code consists of reshaping the data to a (1,1,96,96) form in the format (samples, channel, row, col), then running it through a set of convolutional layers that include pooling layers to reduce the size of data to process. Convolutional neural networks are much more memory intensive than traditional neural networks. Therefore, in using Grid search cross-validation, I could only test for variable batch sizes. The tested batch sizes were 200 and 600. Figure 11 shows the results of the grid search cross-validation where the 600 batch size was the most optimal netting an average of 2.13 root mean squared error. This root mean squared error is much better than the average of the traditional neural network by a large margin.

```

#Convolution Neural Network
model_CNN = Sequential()
#Had to reshape data to format: (samples, channel, row, col)
X_train_CNN = X_train.reshape(-1,1,96,96)
X_test_CNN = X_test.reshape(-1,1,96,96)
model_CNN.add(Convolution2D(32, 3, 3, input_shape=(1,96,96)))
model_CNN.add(MaxPooling2D(pool_size=(2, 2)))
model_CNN.add(Convolution2D(64, 2, 2))
model_CNN.add(MaxPooling2D(pool_size=(2, 2)))
model_CNN.add(Flatten())
model_CNN.add(Dense(30))
adam = optimizers.adam(lr=0.008, decay=2.5e-5)
model_CNN.compile(loss='mean_squared_error', optimizer= adam)

```

Figure 11: Convolution Neural Net Code

```

Best Convolutional Neural Network: 4.555775 using {'batch_size': 600}

```

Figure 12: Grid Search Convolutional Neural Net

## IV. Results

### Model Evaluation and Validation

Utilizing the linear regression benchmark model and comparing between both neural network models, the convolutional neural network was determined to be the best model.

The final parameters for the model were determined after trial and error attempts, as well as, grid search cross-validation. Due to the memory constraints of the computer, the grid search cross-validation could only look through two different possibilities of batch size without crashing. The grid search cross-validation found that a batch size of 600 is the best as seen in Figure 11. Therefore, the final parameters are seen in Figure 12. The 200 epochs allow for some convergence of the validation results. This amount of epochs was also close to maximum amount allowed by the memory constraints as by 300 to 400 epochs, the computer would crash frequently.

The final model was also tested with a testing set to see how well it generalizes to new data. The test's result is noted in Figure 13 as val\_loss which displays a value in the form of the mean squared error. At 200 epochs, the root mean squared error of the test set is 2.06 which is quite an improvement over the 2.6 of the benchmark model.

```
history_CNN = model_CNN.fit(X_train_CNN, y_train, nb_epoch=200, batch_size = 600, verbose = 2,  
                             validation_data=(X_test_CNN, y_test))
```

*Figure 13: Final Parameters of CNN*

```
Epoch 196/200  
16s - loss: 6.8267 - val_loss: 6.1986  
Epoch 197/200  
15s - loss: 5.2058 - val_loss: 4.3095  
Epoch 198/200  
16s - loss: 4.1161 - val_loss: 4.5742  
Epoch 199/200  
16s - loss: 4.4235 - val_loss: 4.6806  
Epoch 200/200  
16s - loss: 4.3427 - val_loss: 4.2736
```

*Figure 14: Mean Squared Error of CNN*

## Justification

Out of the two models present in this paper, convolutional neural networks is the only one that beats the benchmark model which was the requirement to being a useful final solution. Also, the traditional neural network had an average root mean squared error of 3.17, as seen by the mean squared error score in Figure 9. Meanwhile, the convolutional neural network had an average root mean squared error score of 2.06. The final result of the model measured by the root mean squared error metric is 2.06 which is much better than the 2.6 average error of the linear regression model.

Figure 14 justifies the reasoning behind choosing the convolutional neural network as the best solution to the problem. The first plot displays the train loss vs validation loss of the traditional neural network. The plot has a much smoother learning curve that converges well; however, it only converges at approximately 10.0 mean squared error or 3.16 root mean squared error which is 0.56 worse than the benchmark model. On the other hand the second plot depicting the convolutional neural network has much more erratic learning curve; however, it converges at a much lower root mean squared of 2.06.

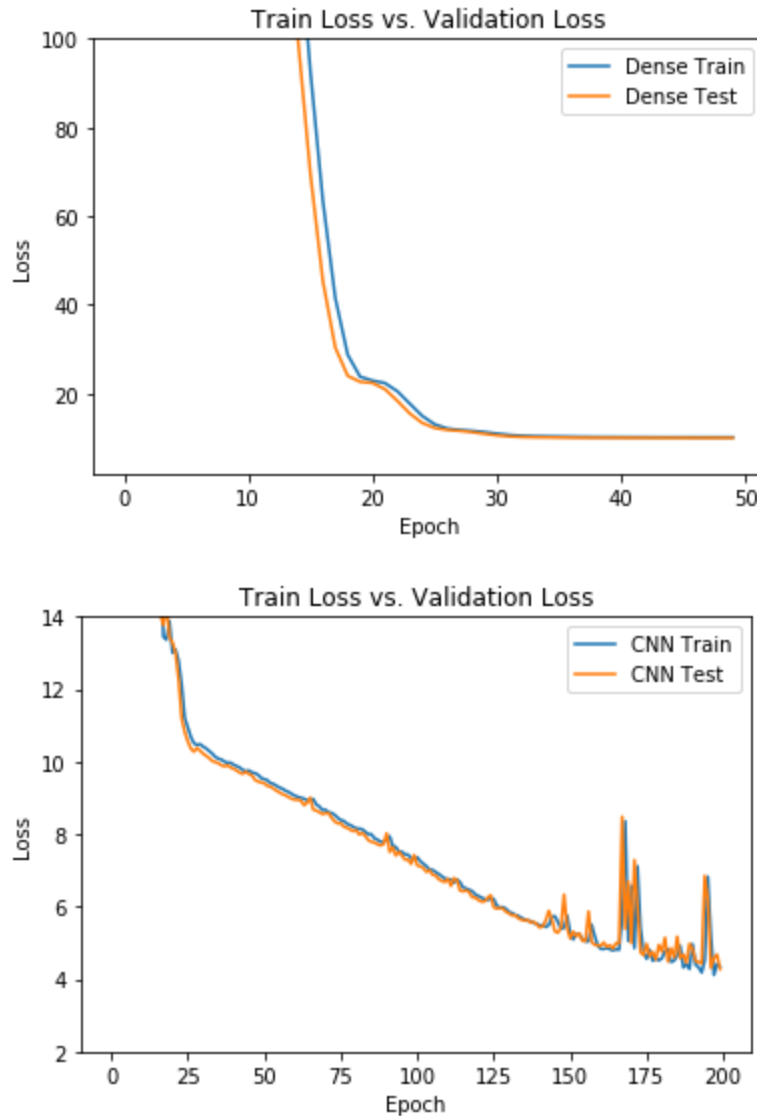


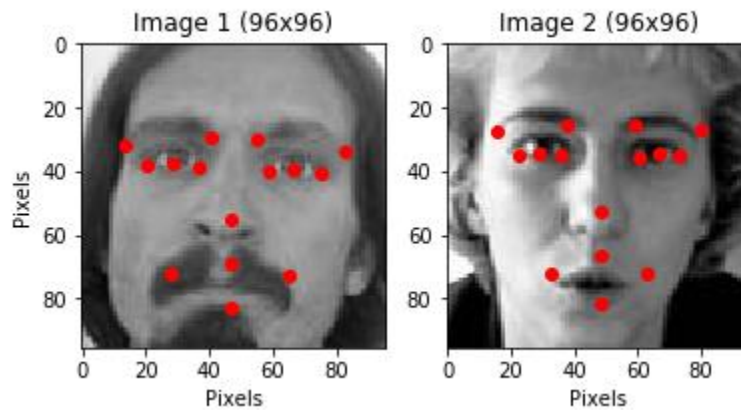
Figure 15: Train Loss vs Validation Loss Graphs

## V. Conclusion

### Free-Form Visualization

Figure 15 provides two images to represent the usefulness of the final model. These images were created by using the final model to predict points with the images as inputs. These predicted points were then placed back on the image to visualize an idea of how the model would be utilized to solve the problem of facial keypoints detection. The predicted points are not completely accurate; however, they give a good prediction as to where each point should be. This characteristic is evident as the root mean

squared error is 2.06 and not 0.00. The current state of the model looks capable of being a solution to the problem; however, there could be many improvements to the model.



*Figure 16: Sample Model Predictions*

## Reflection

The process for the project can be listed as followed:

1. Defining the problem statement, possible solutions, and determining the metric in which to determine how well the solution performs.
2. Preprocessed data by removing nans.
3. A linear regression benchmark model was created.
4. A traditional and convolutional neural network regressor was created.
5. A convolutional neural network regressor was created, trained, and tested.
6. A grid search cross validation was applied to determine an optimal set up for both models and find that the convolutional neural network was the better solution.
7. Both model were trained and tested using the optimal set up found by grid search cross validation.

Step 5 was both the most interesting and difficult aspect of the project. As the subject of convolutional neural networks was new, the difficulty was due to researching, determining, and creating a convolutional neural network with the Keras library and what each layer does to the dataset.



Learning the small and big nuances that define what a convolutional neural network is compared to a traditional neural network was the most interesting part of the project. The neural network's defining characteristics were the ability to input an image's specifications in the form of (samples, channel, row, columns) and utilize pooling filters to reduce the amount of computations the more complicated neural network needed to make.

## Improvement

The project could use many improvements. These improvements could include getting a better computer that can process more data allowing the model to run higher iterations and test for more parameters in the grid search cross validation. They may also include gathering more data to prevent overfitting and to allow for other preprocessing techniques such as outlier detection. Another possible improvement could be to utilize a different algorithm such as support vector machines to predict facial keypoints; however, through research convolutional neural networks seem to be at the forefront of image recognition techniques.

However, an improvement to the algorithm that could possibly increase the potential of the model is the optimizer. Currently, the model is using the 'adam' optimizer with a learning rate of 0.08 and a decay of  $2.5e-5$ . These parameters were determined to properly compare the traditional neural network and the convolutional neural network. However, by lowering the learning rate or changing the optimizer, the model could improve by having a steadier learning curve than Figure 15 depicts and could also improve accuracy.

## Work Cited

- [1] T. Choudhury, "History of face recognition," 2000. [Online]. Available: <http://vismod.media.mit.edu/tech-reports/TR-516/node7.html>. Accessed: Jan. 10, 2017.
- [2] Kaggle, "Data – Facial Keypoints Detection," 2017. [Online]. Available: <https://www.kaggle.com/c/facial-keypoints-detection/data>. Accessed: Jan 11, 2017.
- [3] "sklearn.neural\_network.MLPClassifier," *sklearn.neural\_network.MLPClassifier — scikit-learn 0.18.1 documentation*. [Online]. Available: [http://scikit-learn.org/stable/modules/generated/sklearn.neural\\_network.MLPClassifier.html](http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html). [Accessed: 01-Feb-2017].