Machine Learning Advanced Nanodegree

Project Report

Aditya A Kulkarni **December 26th, 2018.**

Definition

Project Overview

Tracking, detecting and recognizing faces is one of the challenging and sought after tasks in computer vision. Until a decade ago using computers to work on images and video was a difficult task, then came gpus which boosted the use of neural networks and deep learning. This lead to a major push in the ability of machines to do computer vision tasks. Facial keypoints detection belongs to one of these tasks and can be used as a building block in several applications, such as:

- tracking faces in images and video
- analysing facial expressions
- detecting dysmorphic facial signs for medical diagnosis
- biometrics / face recognition

There are several industries benefiting from this technology. Law enforcement agencies are using face recognition to keep communities safer. Retailers are preventing crime and violence. Airports are improving travelers' convenience and security. And mobile phone companies are using face recognition to provide consumers with new layers of biometric security.

Related research paper: http://cs231n.stanford.edu/reports/2016/pdfs/007_Report.pdf

Link to the data source : https://www.kaggle.com/c/facial-keypoints-detection/data

In this project I have used CNN(Convolution neural network) to build a model. CNNs are frequently used for problems involving images. The model takes in images which have 15 points marked on each face which are key facial points and can be used for facial recognition and 3D modelling of face, and it predicts where those points will be on an unknown image.

Problem Statement

The objective of this task is to predict keypoint positions on face images and I'll be treating this problem as a regression problem. Detecting facial keypoints is a very challenging problem. Facial features vary greatly from one individual to another, and even for a single individual, there is a large amount of variation due to 3D pose, size, position, viewing angle, and illumination conditions. Computer vision research has come a long way in addressing these difficulties, but there remain many opportunities for improvement.

I will make an attempt to obtain a RMSE score in the top 50% of the Public leaderboard submission.

To achieve this I'll be preprocessing the facial images, then build a CNN model to train on the training data. After that I will test it on the test set and try to further improve the model by tuning the parameters and using different techniques.

Metrics

The evaluation metric used by the competition to score the submissions is Root Mean Square Error. RMSE is very common and is a suitable general-purpose error metric. Compared to the Mean Absolute Error, RMSE punishes large errors:

$$RMSE = \sqrt{1/n \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

where y hat is the predicted value and y is the original value.

Analysis

Data Exploration

The data set for this competition was graciously provided by Dr. Yoshua Bengio of the University of Montreal. I obtained the dataset from kaggle competition:-

https://www.kaggle.com/c/facial-keypoints-detection/data

The data is in the form of csv files. The each row in the file contains values for each predicted keypoint specified by an (x,y) real-valued pair in the space of pixel indices. There are 15 key points, which represent the following elements of the face:

1. Left_eye_center
2. Right_eye_center
3. left_eye_inner_corner
4. left_eye_outer_corner
5. right_eye_inner_corner
6. right_eye_outer_corner
7. left_eyebrow_inner_end
8. left_eyebrow_outer_end
9. right_eyebrow_inner_end
10. right_eyebrow_outer_end
11. nose_tip
12. mouth_left_corner
13. mouth_right_corner
14. mouth_center_top_lip
15. mouth_center_bottom_lip

Left and right here refers to the point of view of the subject.

In some examples, some of the target keypoint positions are missing (encoded as missing entries in the csv, i.e., with nothing between two commas).

The input image is given in the last field of the data files, and consists of a list of pixels (ordered by row), as integers in (0,255). The images are 96x96 pixels.(i.e 9216 values in the image column)

Data files

- **training.csv:** list of training 7049 images. Each row contains the (x,y) coordinates for 15 keypoints, and image data as row-ordered list of pixels.
- **test.csv:** list of 1783 test images. Each row contains ImageId and image data as row-ordered list of pixels
- **submissionFileFormat.csv:** list of 27124 key points to predict. Each row contains a RowId, ImageId, FeatureName, Location. FeatureName are "left_eye_center_x," "right_eyebrow_outer_end_y," etc. Location is what you need to predict

After a brief study I think I'll be splitting the training the data to create train and validate sets. Also, at the moment I think I'll be using the keypoints which are common in most of the images. I'll be using all the examples in the dataset.

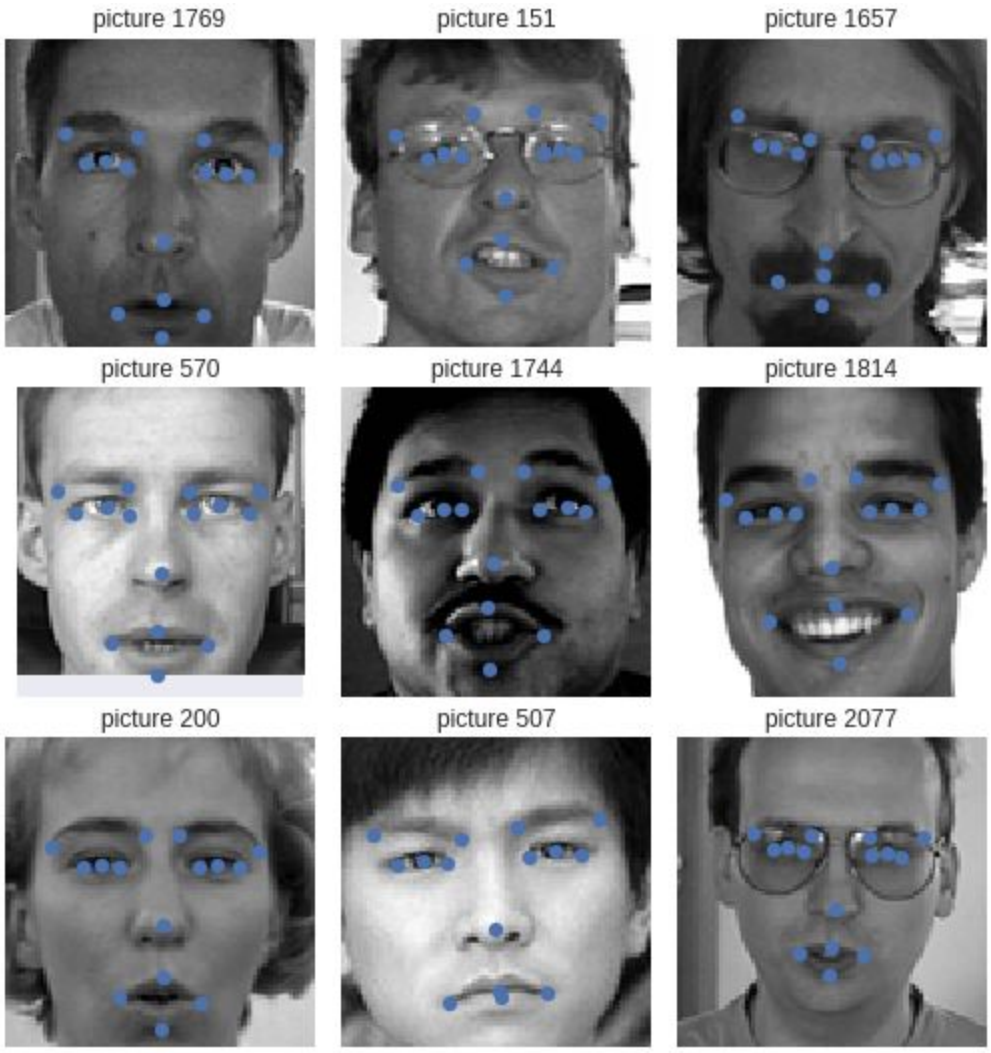
Exploratory Visualization

The below image shows the count of landmarks on each image

	index	0
0	left_eye_center_x	7039
1	left_eye_center_y	7039
2	right_eye_center_x	7036
3	right_eye_center_y	7036
4	left_eye_inner_corner_x	2271
5	left_eye_inner_corner_y	2271
6	left_eye_outer_corner_x	2267
7	left_eye_outer_corner_y	2267
8	right_eye_inner_corner_x	2268
9	right_eye_inner_corner_y	2268
10	right_eye_outer_corner_x	2268
11	right_eye_outer_corner_y	2268
12	left_eyebrow_inner_end_x	2270
13	left_eyebrow_inner_end_y	2270
14	left_eyebrow_outer_end_x	2225
15	left_eyebrow_outer_end_y	2225
16	right_eyebrow_inner_end_x	2270
17	right_eyebrow_inner_end_y	2270
18	right_eyebrow_outer_end_x	2236
19	right_eyebrow_outer_end_y	2236
20	nose_tip_x	7049
21	nose_tip_y	7049
22	mouth_left_corner_x	2269
23	mouth_left_corner_y	2269
24	mouth_right_corner_x	2270
25	mouth_right_corner_y	2270
26	mouth_center_top_lip_x	2275
27	mouth_center_top_lip_y	2275
28	mouth_center_bottom_lip_x	7016
29	mouth_center_bottom_lip_y	7016
30	Image	7049

Below image is the shape of the training data and the range of values of x and y.

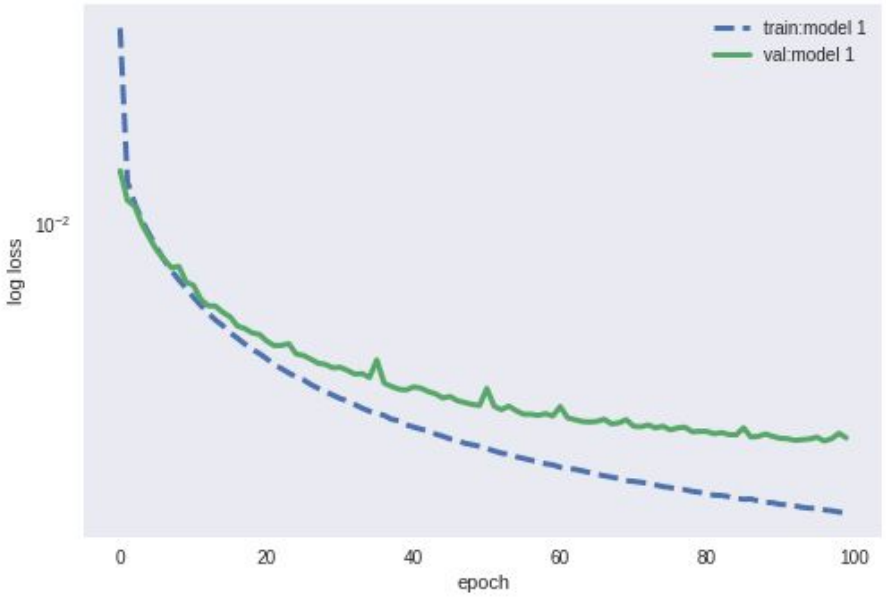
```
X.shape == (2140, 9216); X.min == 0.000; X.max == 1.000
y.shape == (2140, 30); y.min == -0.920; y.max == 0.996
```

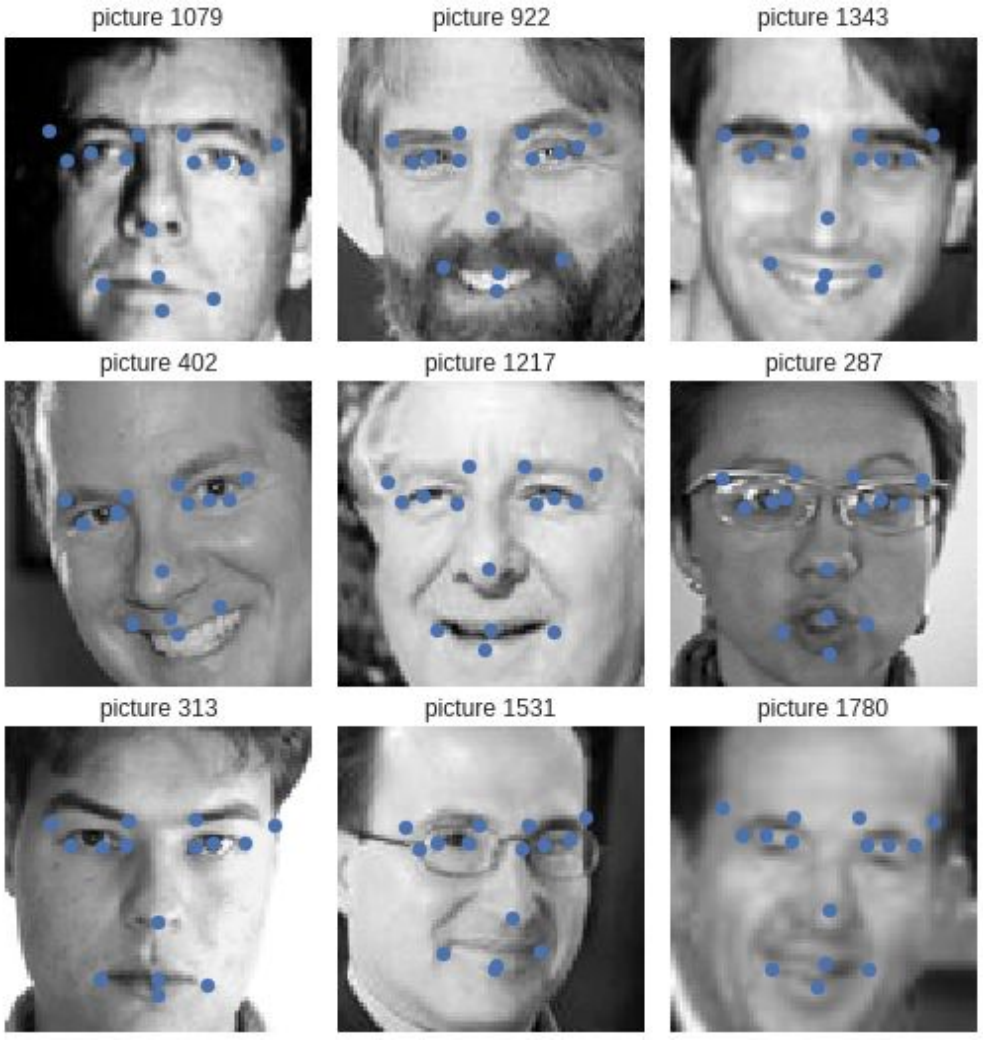


The above images are part of the training data. This is what is fed into the model as input, an image with x, y coordinates of landmarks on the face.

The below images are :-

- The training graph of the simple CNN for 100 epochs
- Performance of the simple CNN model on test data

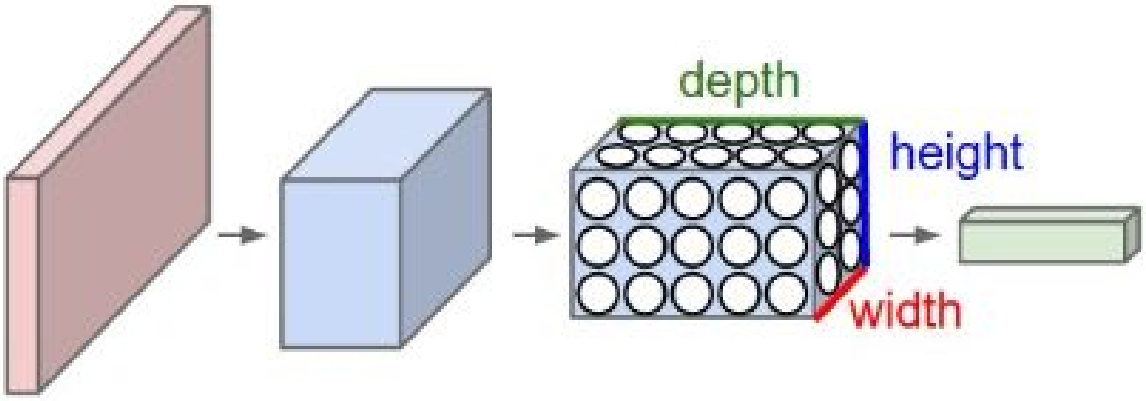

train:model 1
val:model 1
log loss
10^{-2}
0
20
40
60
80
100
epoch


picture 1079
picture 922
picture 1343
picture 402
picture 1217
picture 287
picture 313
picture 1531
picture 1780

Algorithms and Techniques

- I have used Convolution Neural Network for this project as it involves working on images. First I used a simple neural network model which barely did anything good. The second model has three Convolution layers in it connected to two dense layers which are connected to output layer. I have used SGD optimizer. The models are trained with and without dropout, and before and after data augmentation.
- For the input to the model load2d() function is used.
- Convolution Neural Networks(CNN)
    - CNN are inspired by the biological process in that the connectivity pattern between neurons resembles the organization of the animal visual cortex.
    - Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.
    - CNN is similar to ordinary neural network. They have neurons which have learnable weights and biases.
    - Each neuron receives some input, performs a dot product and optionally follows it with a non-linearity.
    - Regular neural networks don't scale well to full images. CNN take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way.
    - In particular, unlike regular neural network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height and depth.
    - CNN architectures consists of Convolution layer, Pooling layer and Fully connected layer. We will stack these layers to form a full CNN architecture.
    - Convolution layer :- It is the core building block of the Convolution network that does most of the computational heavy lifting.
    - Convolution layer is the first layer to extract the features from the images.
    - Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.
    - Pooling layer :- This reduces the number of parameters when images are too large.
    - Spatial pooling also called as subsampling or downsampling which reduces the dimensionality of each map but retains the important information.

- Spatial pooling can be of different types :- Max pooling, Average pooling.
- Fully connected layer :- To feed the input into fully connected layer we flattened our matrix into vector and feed it into fully connected layer like a neural network.



Benchmark

The competition's overview page has a getting started with R section which provides a benchmark model using the evaluation metric mentioned below.

The score it 3.758999 for the benchmark model and I'll attempt build a model and improvise on this metric score.

The link for the benchmark model page is:-

https://www.kaggle.com/c/facial-keypoints-detection#getting-started-with-r

I have submitted model with four different approaches:-

- CNN model with data augmentation with dropout.
- CNN model with data augmentation without dropout.
- CNN model with dropout.
- CNN model without dropout.

The aim is reduce the score and the models scored less than the benchmark model.

5 submissions for **Aditya Kulkarni** Sort by Most recent

All Successful Selected

Submission and Description	Private Score	Public Score	Use for Final Score
Facial keypoints Detection (version 2/2) a day ago by Aditya Kulkarni cnn model 3 with dropout	3.09400	3.16970	☐
Facial keypoints Detection (version 2/2) a day ago by Aditya Kulkarni cnn model 3 without dropout	3.09400	3.16970	☐
Facial keypoints Detection (version 2/2) a day ago by Aditya Kulkarni cnn model 2 with dropout	3.10652	3.22886	☐
Facial keypoints Detection (version 2/2) a day ago by Aditya Kulkarni cnn model 2 without dropout	3.10124	3.21218	☐

Methodology

Data Preprocessing

- The dataset is 285 MB in size and contains 7049 images for training and 1783 images for testing. Images are grayscale and of size 96x96. The images are in string format where the 96x96 matrix is a single row of 9216 values.
- The load function returns a 2D numpy array with shape (nb_samples, rows x columns) for the image and returns a 2D numpy array with shape (nb_samples, nb_landmarks x 2) for the x,y coordinates of landmarks.
- When using TensorFlow as backend, Keras CNNs require a 4D array(4D tensor) as input with shape :-

  (nb_samples, rows, columns, channels)

  where nb_samples corresponds to total number of images(or samples), and rows, columns and channels correspond to number of rows, columns and channels for each image respectively.
- The load2d() function uses the load() function's output and reshapes it into the following format :-

  (nb_samples, 96, 96, 1)
- The image is 96x96 and it's a grayscale image thus the number of channels is 1.

Implementation

- The dataset was downloaded from kaggle competition(link is mentioned above). The training and test csv files were extracted from zip files and stored in FTRAIN and FTEST, while the idlookup file is stored in FIdLookup.
- The train data consists of 7049 images and the coordinates for landmarks, the test data consists of 1783 images and the coordinates for landmarks. For training purpose the data was split into 80% of FTRAIN for training and 20% for validation.
- Some general purpose functions are defined with the help of the benchmark kernel. These functions help load, plot and convert data into proper format.
- The data is in the form of 2D matrix, it is converted into a 4D tensor using the load2d function.

- The load2d function uses the load function. Load function converts the images which is a string of numbers into matrix form and returns the 2D arrays. The load 2d function uses this as input and reshapes the data into 4D tensor.
- First a simple CNN model is built and trained on the dataset.
- Then a CNN model is built and data is passed into it.
- The architecture of the CNN model is as follows :-
    - The first layer is convolution input layer, 32 feature maps with a 3x3 kernel size and Rectified Linear Unit i.e ReLU as the activation function. The input size is 96x96x1.
    - The second layer is a maxpooling layer with kernel size of 2x2. IT is used to reduce the spatial size which results in reduction of number of parameters and hence the computation is reduced.
    - There is an optional dropout layer after every maxpooling layer. It is a regularization technique used to reduce overfitting.
    - The third layer is convolution input layer, 64 feature maps with a 3x3 kernel size and Rectified Linear Unit i.e ReLU as the activation function.
    - The fourth layer is a maxpooling layer with kernel size of 2x2.
    - The fifth layer is convolution input layer, 128 feature maps with a 2x2 kernel size and Rectified Linear Unit i.e ReLU as the activation function.
    - The sixth layer is a maxpooling layer with kernel size of 2x2.
    - The seventh layer is a flatten layer.
    - The eighth and ninth layers are fully connected dense layers with 300 units and ReLU activation function.
    - The tenth layer is a dense layer with 30 units. The units are 30 because each of 15 landmark points have x and y coordinate, i.e 15 x 2 = 30
    - The model is compiled with SGD optimizer which has learning rate of 0.01 and momentum of 0.9.
- This model is then trained with and without dropout on augmented and non-augmented data.
- For augmentation the data is converted using a custom function as the package function only flips the images, which is good for image classification problem but not here. Since we also need the landmark points on the image this function is used.
- Plotting is done to compare training for models.
- Submission function is created to create a submission file for kaggle

- Complication occurred during the training of the models, despite a GPU it took me a lot of time to train the models and trying different parameters became difficult.
- Also more data augmentation could be done to build a better model or models which did not happen because of the computation power required.

Refinement

The initial model had a validation loss above 0.001. I aimed to get it under 0.001 and to get there I flipped the images in the training data. Keras has a ImageDataGenerator class but it does not change the landmarks on the image. Thus I used a custom built function. The model reached a validation loss of 0.00099 which is what I wanted to achieve.
The model uses SGD optimizer with a learning rate of 0.01 because a high learning rate did not converge to a global minima and a smaller learning rate was too slow. Already knowing that the learning rate values need to be in small enough to not miss the minima but also large enough to have a steady pace I started with 0.01 as the value and it worked well for me so there was not much of hyper parameter testing.
SGD has a problem that it takes too much time to approach the target under the influence of a large gradient, and momentum can help speed up this process. So the value of momentum is 0.9 and nesterov accelerated gradient a improved version of momentum is used in the model. It allows to detect the gradient ahead of time.

Results

Model Evaluation and Validation

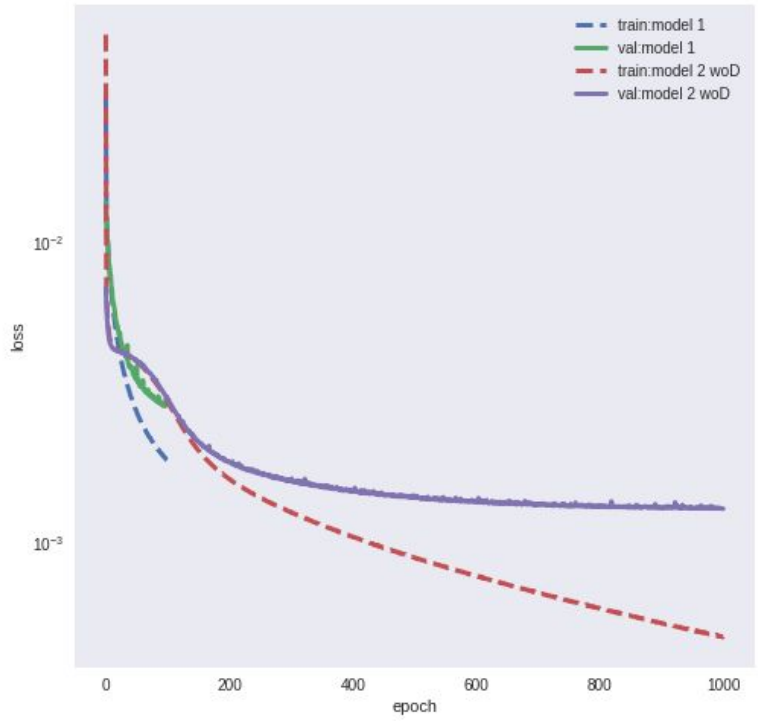
The validation data was 20% of the training data and it is used to evaluate model performance during its development. To resolve overfitting there is a dropout layer added after every convolution layer.

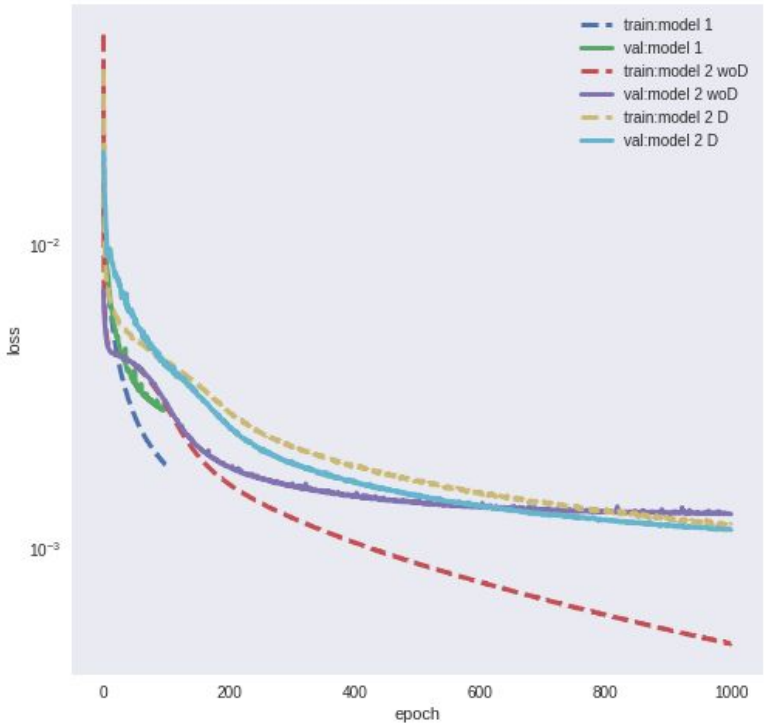
I found that without data augmentation, the model performs worse with dropout added.

The following is the summary of the model used for training.

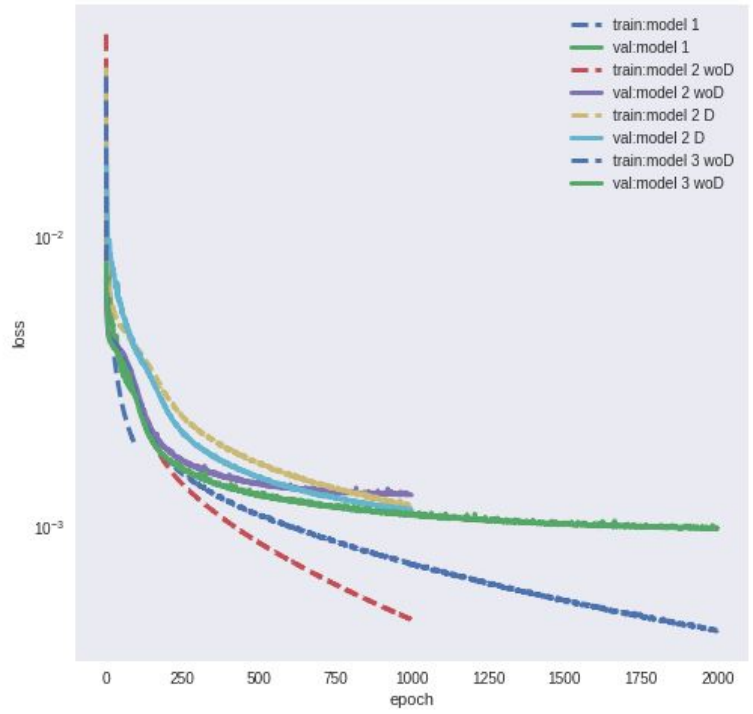
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 94, 94, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 47, 47, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 45, 45, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 22, 22, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 21, 21, 128)       32896
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 10, 10, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 12800)             0
_________________________________________________________________
dense_4 (Dense)              (None, 300)               3840300
_________________________________________________________________
dense_5 (Dense)              (None, 300)               90300
_________________________________________________________________
dense_6 (Dense)              (None, 30)                9030
=================================================================
Total params: 3,991,342
Trainable params: 3,991,342
Non-trainable params: 0
_________________________________________________________________
```

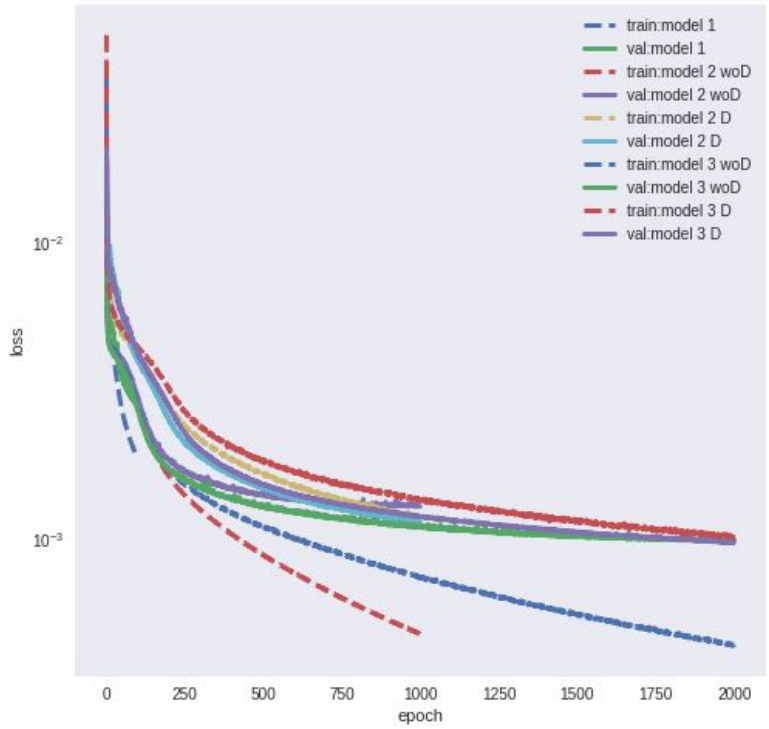
The following graph shows the training and validation accuracy of the simple cnn model compared with the above model without any dropout and then with drop out.





The following graph shows the same model used above with and without dropout but trained on augmented data.





The above models gave the same score i.e the model when trained on augmented data performed the same irrespective of whether we add dropout or not.

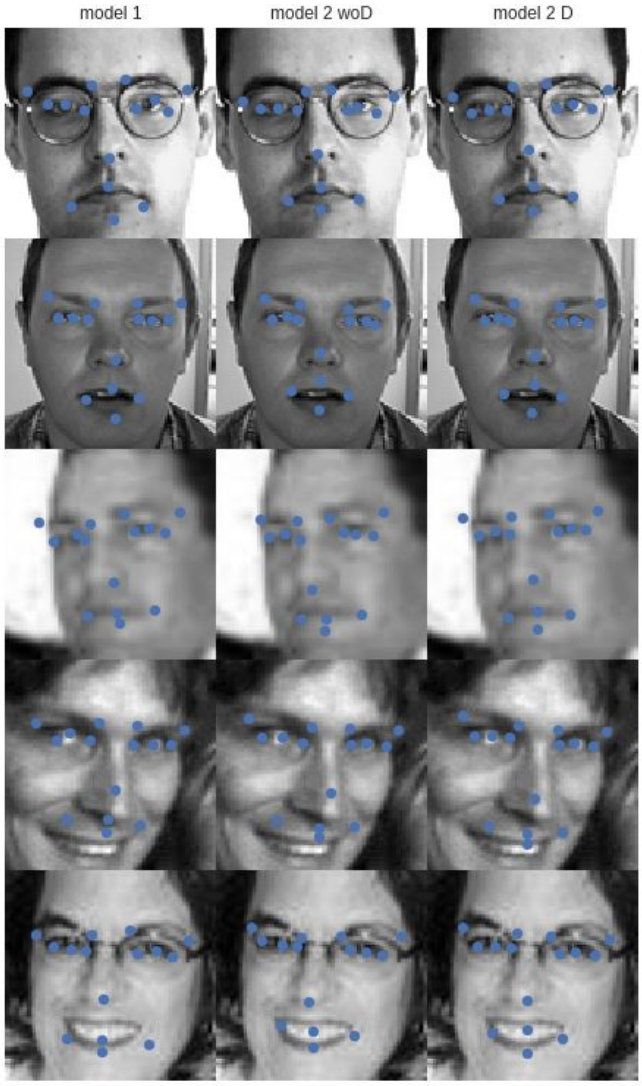
This is a simultaneous comparison of all the models on test data at the same time and it can seen that though they give a decent output they are still overfitting a little and the landmarks are not that accurate. I feel if the model was trained on data which was augmented more than it is now it would do a better job at it. This model can be trusted as it performs quite well and ranks in the top 60 of the competition, but it will not perform exceptional on unseen data.

Justification

The initial models give public score of 3.22886, 3.21218 while after augmentation the score is reduced below 3.2 which is 3.16970. All three scores outperform my benchmark model score which is 3.75. The public leaderboard rank is under top 60.

Conclusion

Free-Form Visualization



- The first image is the prediction of simple CNN and the convolution model with and without dropout on the test data.
- The second image is the Data augmentation done on the images. It can be seen the function written in the code flips the image and also the positions of trademarks.

- This is the comparison between all the prediction of all models on the test data.

Reflection

Summarizing the entire end to end problem solution as follows :-

- The data was downloaded from kaggle and it was preprocessed into proper shape.

- The competition has a getting started section which has a benchmark model and target rank was defined.
- An initial simple CNN model was built.
- A second model was built from scratch and trained with and without dropout.
- The above step was done on a model with same architecture after data augmentation(flipped the images and landmarks).
- The scores for the four models are :-
    1. cnn model 3 with dropout
       Private score :- 3.09400
       Public score :-3.16970

    2. cnn model 3 without dropout
       Private score :- 3.09400
       Public score :-3.16970

    3. cnn model 2 with dropout
       Private score :- 3.10652
       Public score :-3.22886

    4. cnn model 2 without dropout
       Private score :- 3.10124
       Public score :- 3.21218

The training part was difficult as it required a lot of computation and takes a lot of time despite the using gpu. Interesting part was seeing my model perform this well despite it was created from scratch.

Improvement

- The current CNN can be improved by using its architecture and building models for every landmark, eg: one CNN model for left eye, one for right eye, etc.
- We can also use the techniques we use in object detection and create bounding boxes on every landmark.
- The number of layers and the nodes can be increased to improve the accuracy of the above techniques.

References

CNN :- http://cs231n.github.io/convolutional-networks/

https://en.wikipedia.org/wiki/CNN

The benchmark model :- https://www.kaggle.com/c/facial-keypoints-detection#deep-learning-tutorial

https://elix-tech.github.io/ja/2016/06/02/kaggle-facial-keypoints-ja.html