A basic Convolutional Neural Network model was built using the TensorFlow framework to classify handwritten numeric digits as either odd or even and trained with the MNIST dataset. Various training and evaluation characteristics of neural net were measured and recorded. In addition various parameters of the model were tuned and the corresponding changes to the metrics were recorded.

As part of this assignment, a simple application was also developed to predict whether user inputted images of digits are odd or even using the above trained model.

Here are the various deliverables required for submission with this assignment -

Deliverable I - Custom CNN

A CNN model was constructed with the following design -

- 2 Convolutional and Pooling layers
    - Layer 1: 32 filters of kernel size 5x5
    - Layer 2: 64 filters of kernel size 5x5
    - Pooling in both layers downsampling by a factor of 2
- Dropout rate of 40%
- Using Gradient Descent optimization and a Learning Rate of 0.001
- Trained for 5 epochs with the MNIST dataset that has 55,000 samples

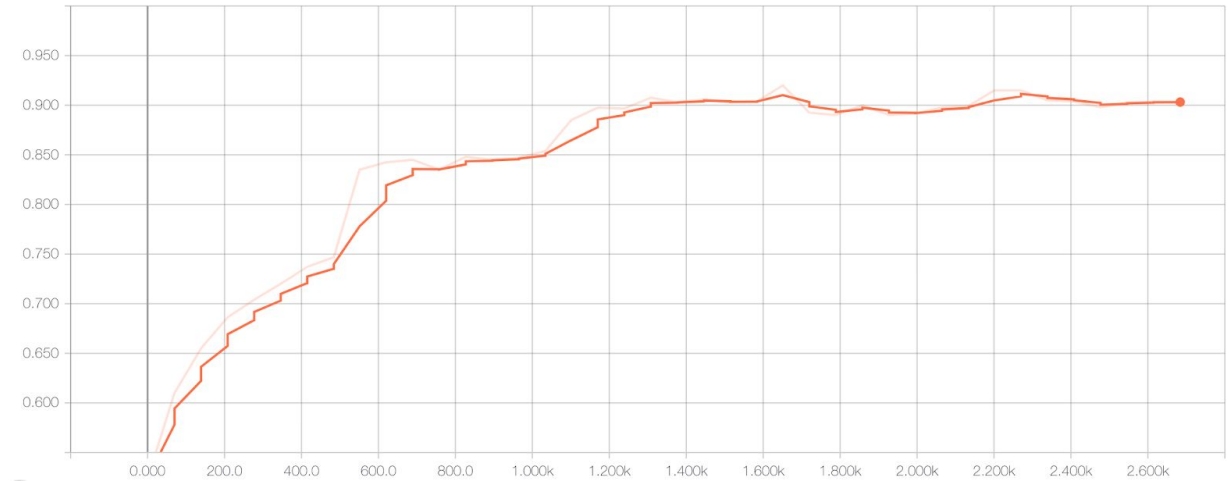
The following characteristics were measured and recorded as the model was trained and evaluated for 5 epochs (at each step during training and at each epoch for evaluation).

- Loss
- Accuracy
- Precision
- Recall

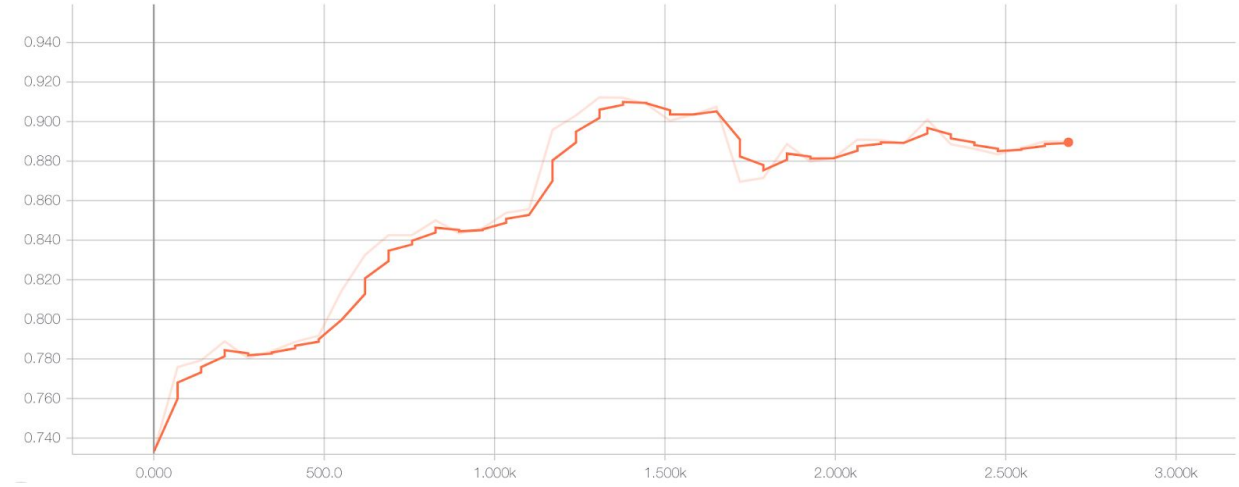
The recorded data was plotted on charts and visualized.

- **Training Statistics**

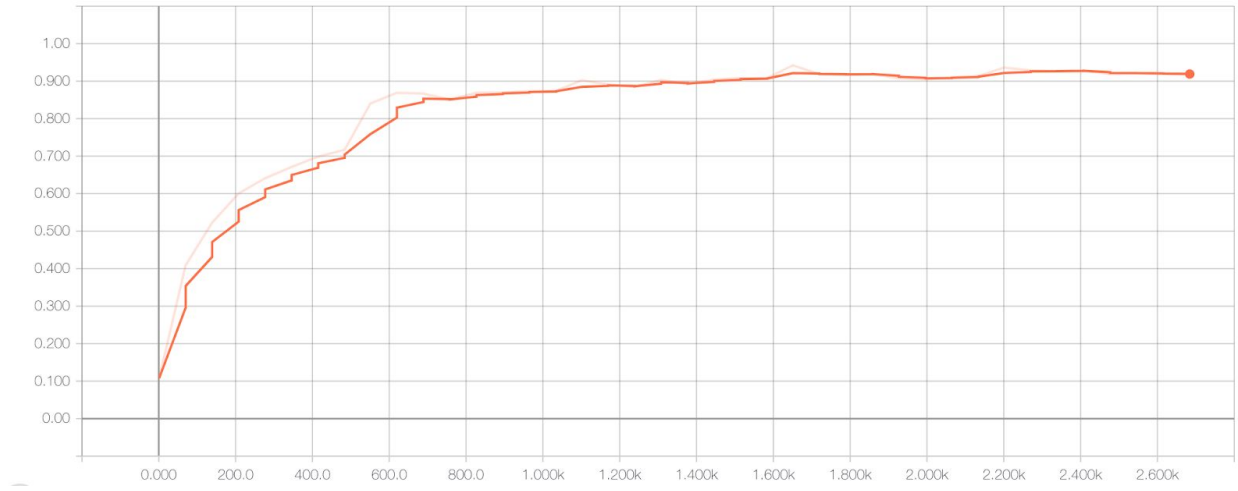
training_accuracy
tag: Training_Summaries/training_accuracy



training_precision
tag: Training_Summaries/training_precision

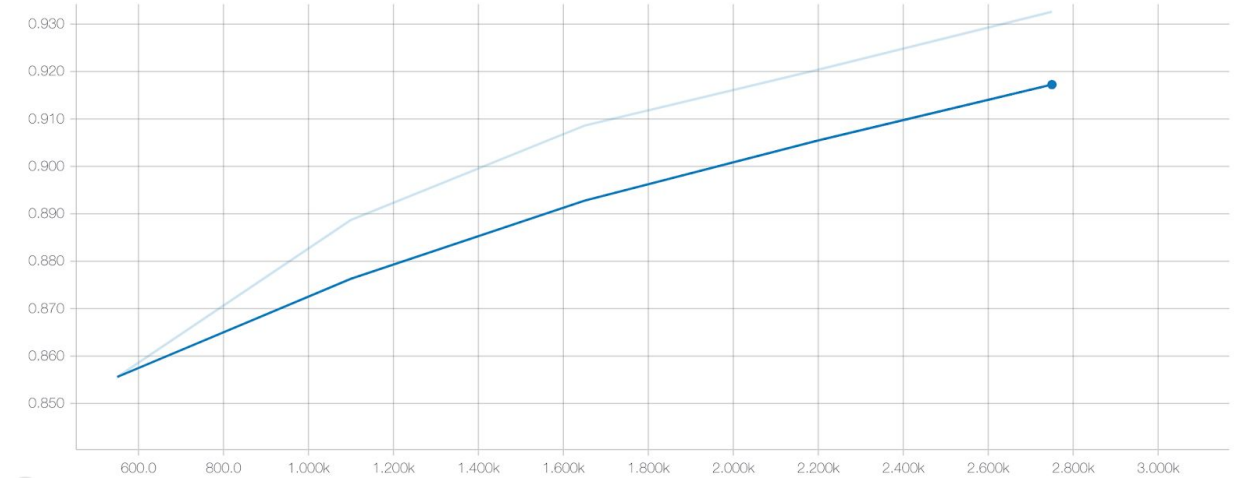


training_recall
tag: Training_Summaries/training_recall

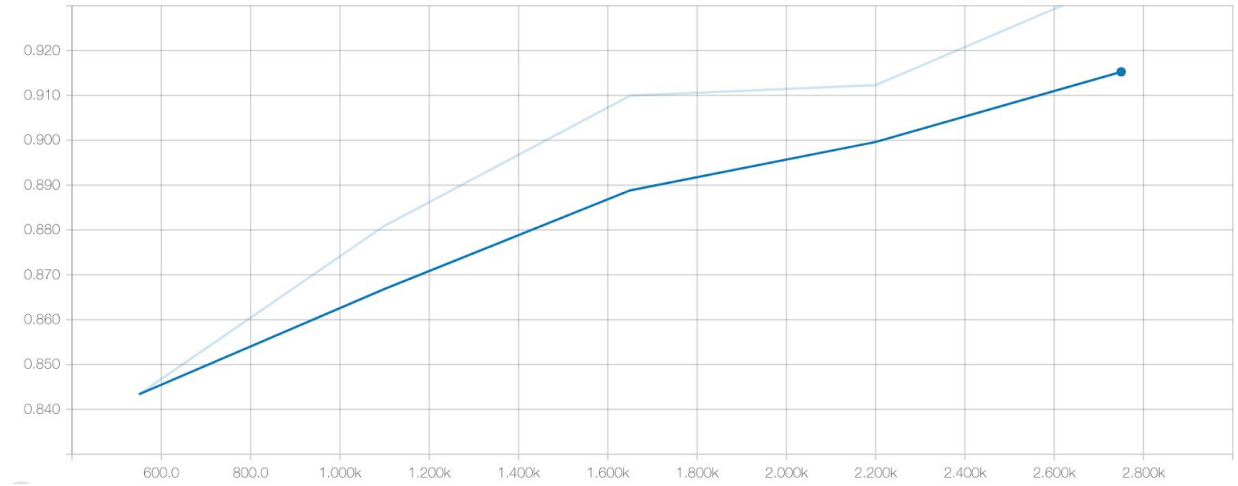


- **Evaluation Statistics**

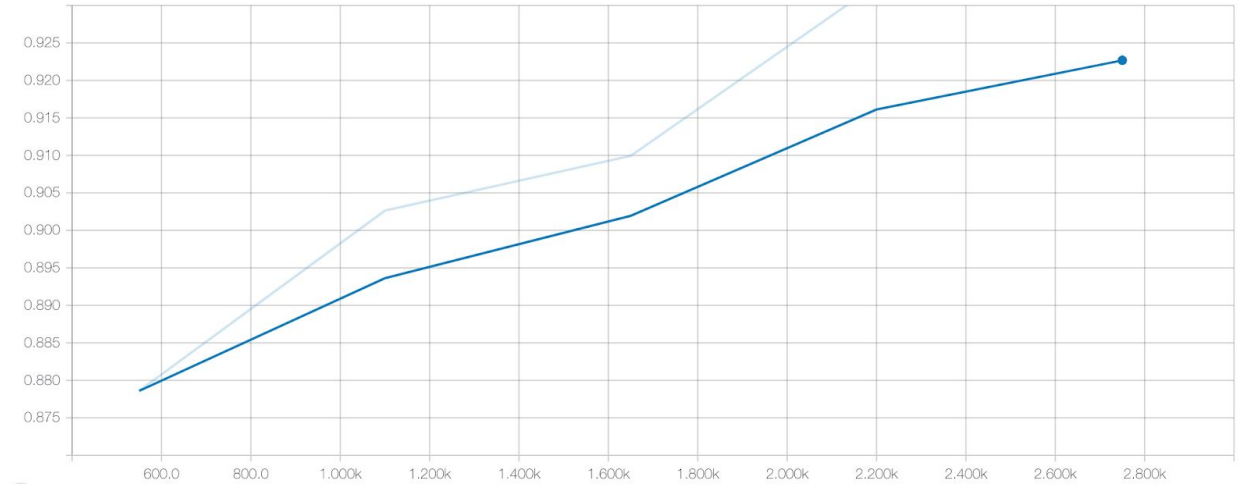
eval_accuracy



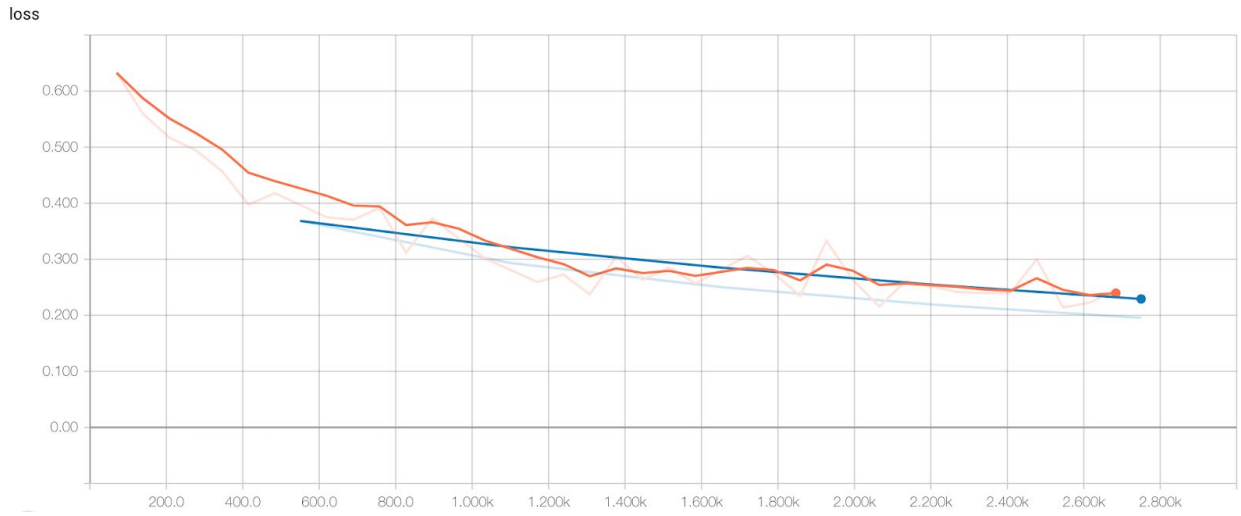
eval_precision



eval_recall



- **Training & Evaluation Loss**



Final Results

At the end of the training and evaluation process the following results were achieved interms of the above metrics -

- **Training:**
  - Accuracy: 0.9346
  - Loss: 0.18815172
  - Precision: 0.9365863
  - Recall: 0.9343713
- **Evaluation**
  - Accuracy: 0.9326m
  - Loss: 0.19545963
  - Precision: 0.93564355
  - Recall: 0.93121797

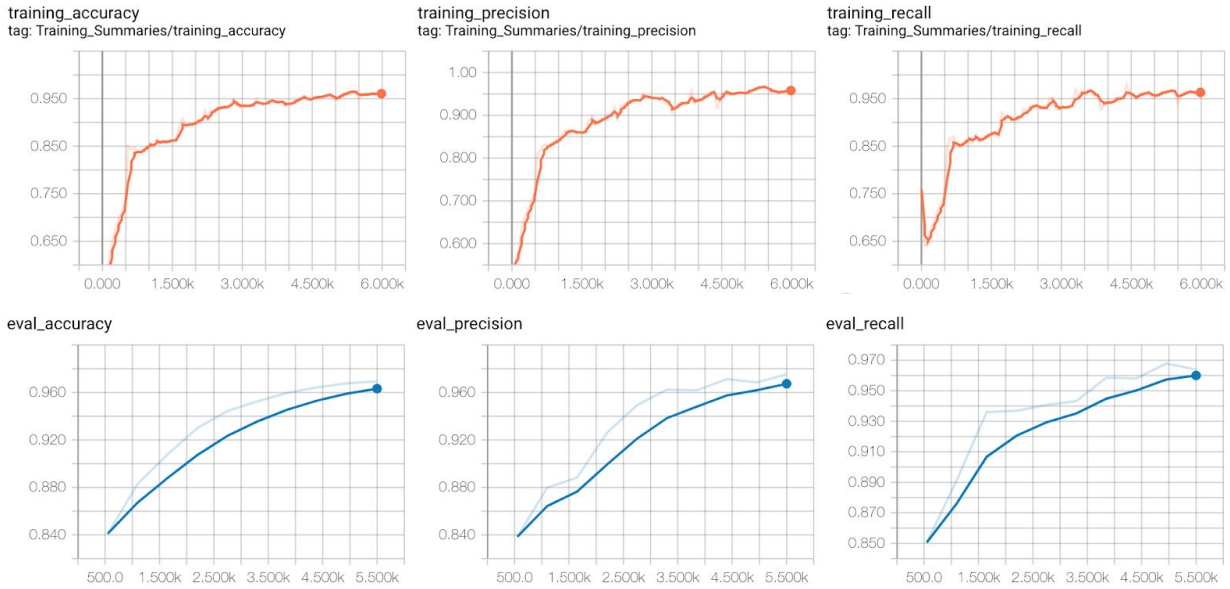
Source Code - The source code is available under the `/src` folder of the submission and the trained models are under the `/models` directory.

Deliverable III - Parameter Tuning

The following experiments were done by changing different parameters of the CNN model and the performance of the model was observed.

Modifying the architecture

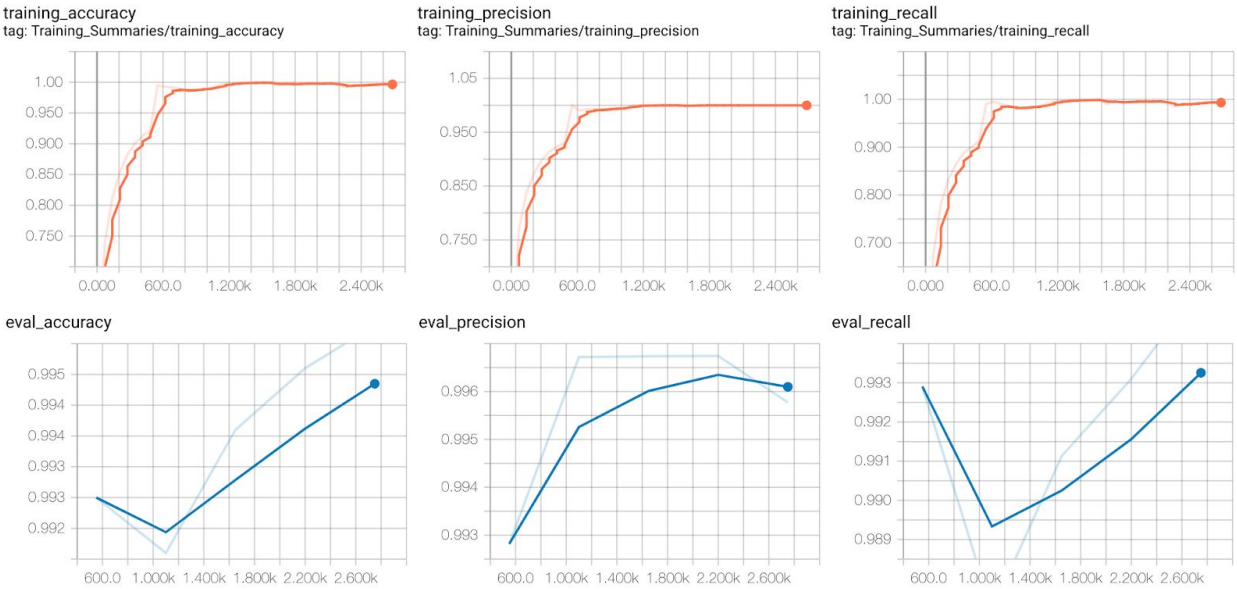
Just adding an extra convolutional layer to the network improved the results at the end of 5 epochs of training, resulting in accuracy, recall and precision to be over 95% which is almost a 5 point average increase across the board from the standard architecture used previously.



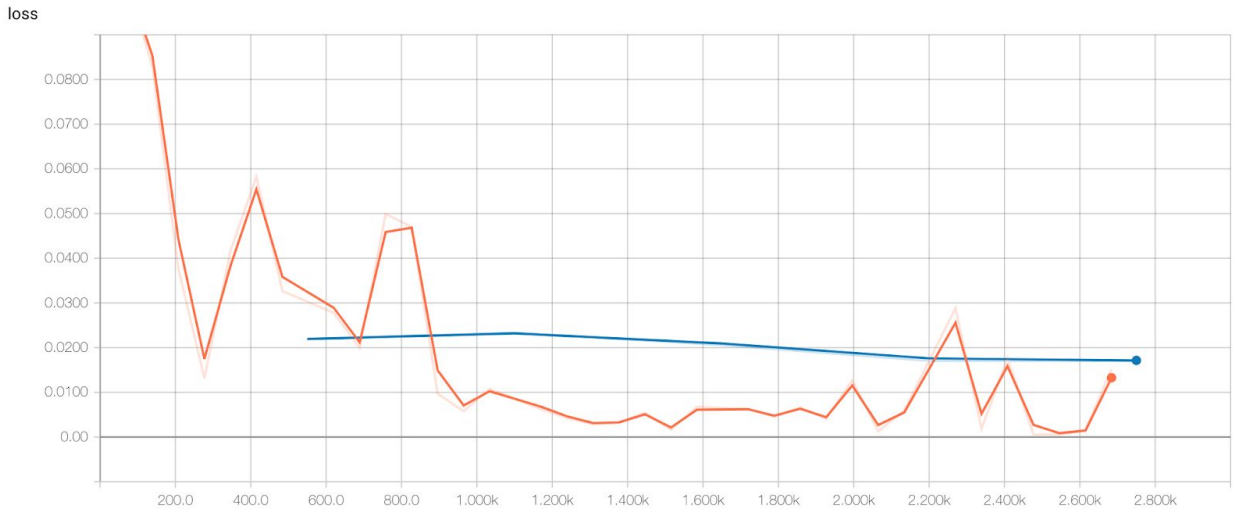
This suggests adding layers can improve accuracy but it could lead to overfitting if not done carefully after a process of trial and error.

Changing the Optimizer and loss function

Gradient Descent optimizer was used during the original training. This was changed to use the Adam Optimizer (described here - https://arxiv.org/abs/1412.6980) which resulted in a dramatic increase in performance. At the end of the training the training accuracy was 0.9962 and evaluation accuracy was 0.9953. The following are the characteristics observed during training for 5 epochs.

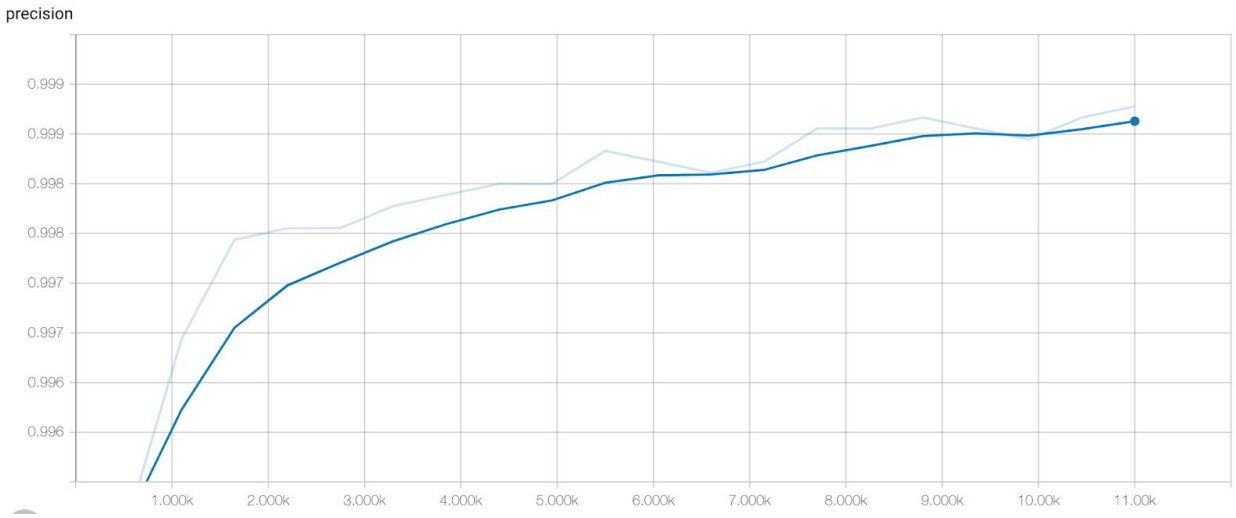


Training and evaluation loss was also minimized as observed below comparing to the earlier charts -



Training for more epochs

The same model was trained for 20 epochs with the MNIST dataset with a batch size of 100 and it resulted in a much better performing model as seen in the following evaluation precision chart.

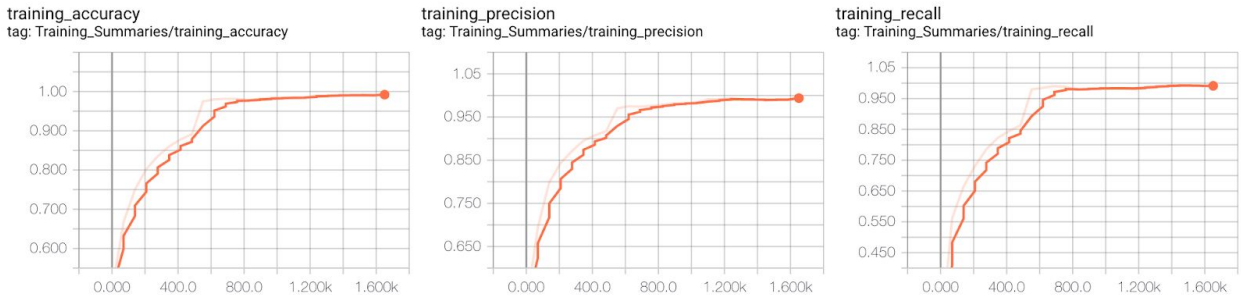


Tweaking the stride parameter

As the stride parameter was increased, it led to faster training but there was a drop in the accuracy. Lowering the stride to 1 increased the time it took to train the model but also resulted in a minor bump in accuracy. This is inline with what how the stride parameter is supposed to affect the training process of a CNN.

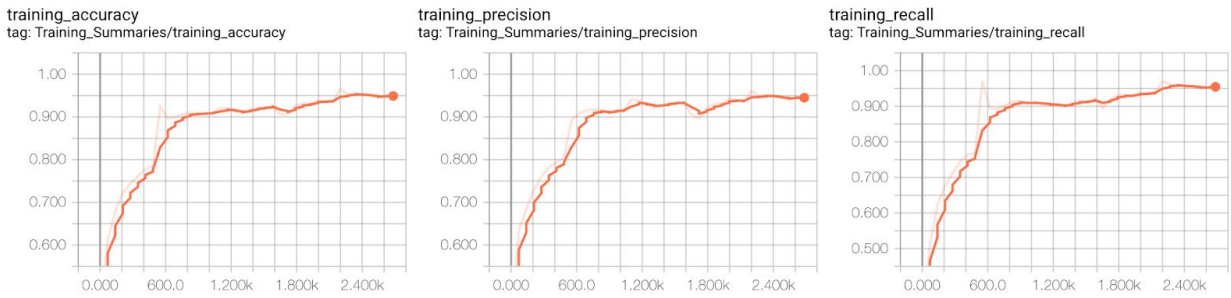
Tweaking the learning rate

The learning rate was increased to 0.05 from 0.001 and that resulted in a very quick convergence of the model, specifically at about 750 steps which is just over an epoch. But other studies do show Gradient Descent optimizer can perform poorly with higher learning rates. (ref - https://medium.com/octavian-ai/which-optimizer-and-learning-rate-should-i-use-for-deep-learning-5acb418f9b2)



Using different initializers

The original model training was initialized using `fast_conv` initializers. When switched to `he_rec`, the resulting performance was mostly the same as before but then the model started converging much earlier at about 800 steps or close to 2 epochs. This can be observed in the following charts.



Deliverable III - The Application

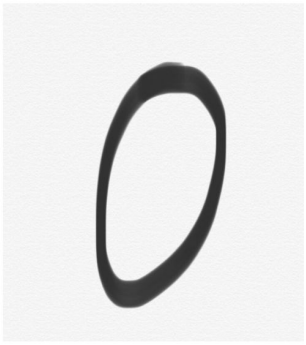
An application was developed using the model trained above to predict user inputted images of handwritten digits. The following steps were performed to read, process and predict the input image -

1. Read the image as grayscale using - `cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)`
2. Resize the image into a `28x28` one so that it matches the dimensions with which the model was originally trained.
3. Apply inverse binary thresholding using `cv2.THRESH_BINARY_INV` so that any pixel that has a value less than 127 is transformed to white (255) and pixels over 127 are transformed to black (0) so that the written digit stands out in white pixels in a black background, again matching the MNIST dataset using which the model was trained.
4. The binary image is flattened and each pixel is divided by 255 to scale it down to a value between 0 and 1 and the result is an array of shape `(784,)`.
5. Now this vector is passed in as input to the model to perform the prediction and the result is inferred printed to the user.

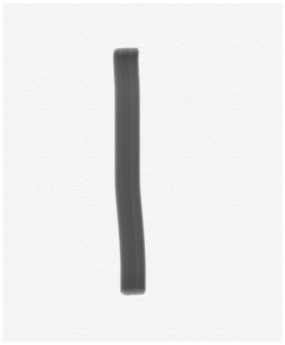
Manual - Run the cnn_test.py file using the command `*python3 cnn_test.py*` and the program will print out a menu. Entering option “1” will ask for the path to the image file. Upon providing a valid path, the program will open up the given image and the binary version in an OpenCV window and also print out whether its odd or even. To predict another image, focus on the OpenCV windows and press any key and the program will resume back to the menu options.

Test Results

The model was tested with the following test images and it performed with an accuracy of 80% (with the sample set used). It must be noted that training the model to 20 epochs improved the accuracy over the following sample set to 100%. Below are the images used for testing.



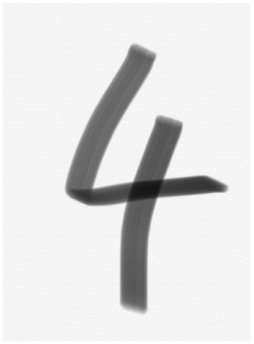
Prediction - EVEN



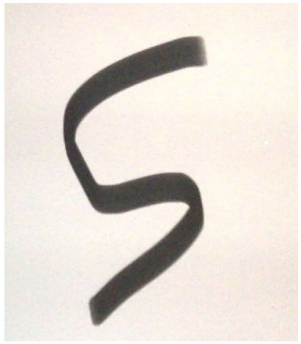
Prediction - ODD

Prediction - EVEN

Prediction - ODD



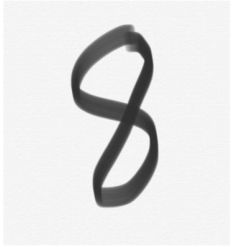
Prediction - EVEN



Prediction - EVEN

Prediction - EVEN

Prediction - ODD



Prediction - EVEN

Prediction - ODD