# MNIST Dataset Exploration

Sarveshwaran Jayaraman

# Table of Contents

## Introduction

There are 2 Tasks in this challenge:

- Task 1 : Train a Neural Network as per architecture specification
- Task 2 : Develop an inference server to predict handwritten Digits

## Tech Stack Used

- Keras API with Tensorflow Backend
- Tensorflow
- Python 3.x
- Numpy for numerical computing
- Matplotlib for visualization and plotting functionality

## Task 1: Train a Neural Network as per architecture specification

The given architecture is not a sequential model. To develop this, I have used Keras Functional API as illustrated in the following page (Fig.1). The parameters for the specified model are tabulated in (Table 1).

## 1.1 Architecture of Network using Keras Functional API

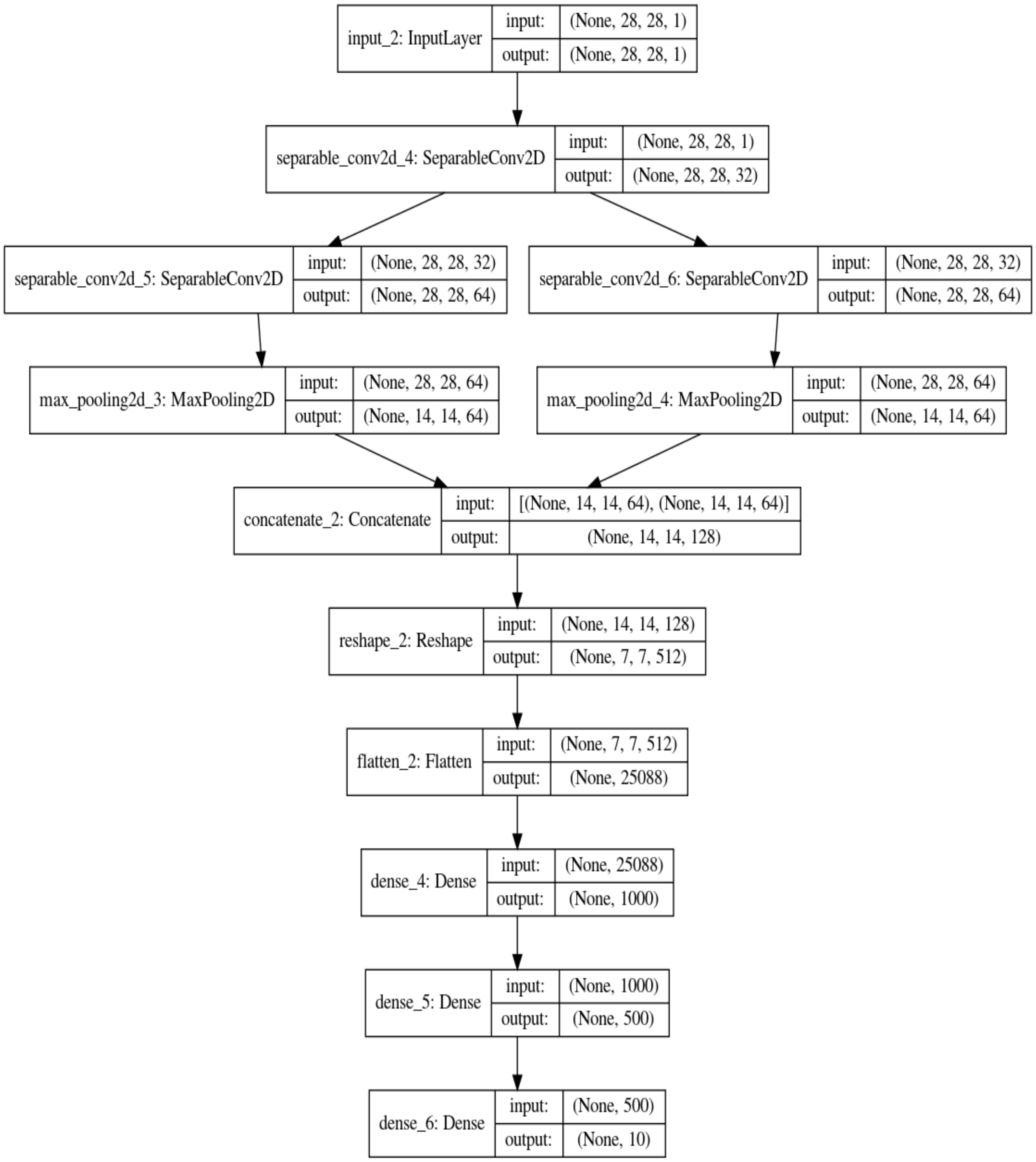
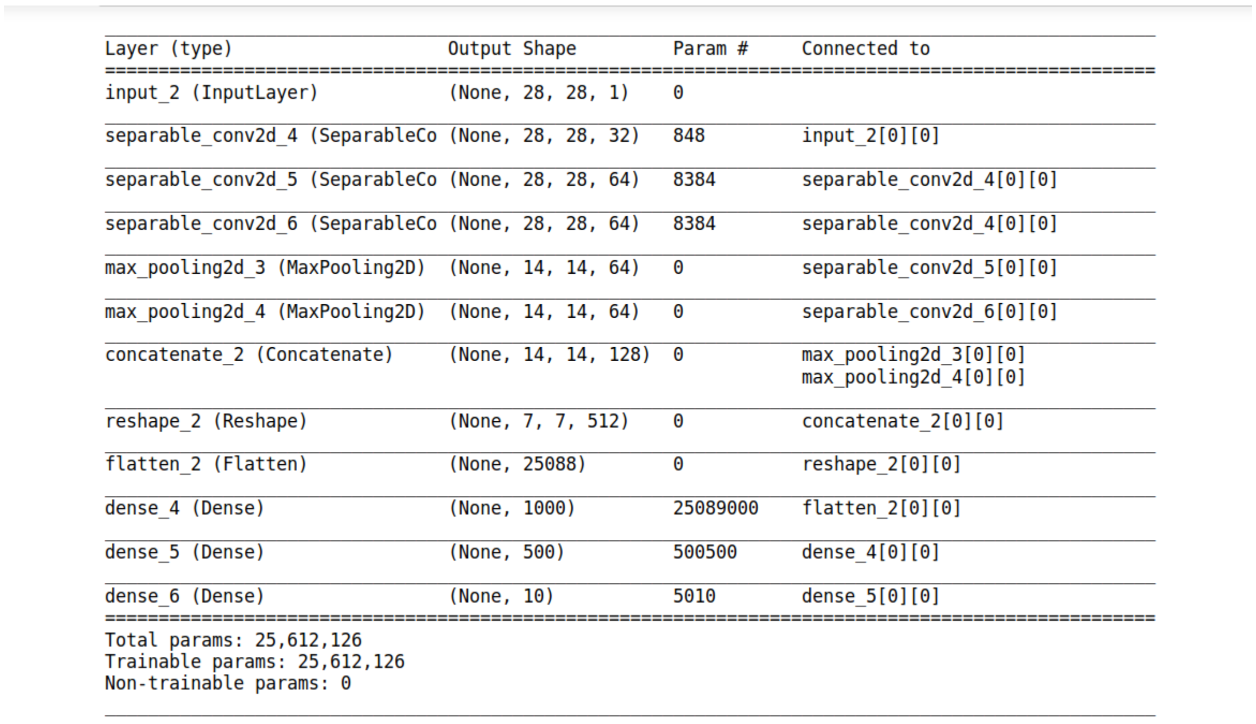

input_2: InputLayer | input: (None, 28, 28, 1) | output: (None, 28, 28, 1)
separable_conv2d_4: SeparableConv2D | input: (None, 28, 28, 1) | output: (None, 28, 28, 32)
separable_conv2d_5: SeparableConv2D | input: (None, 28, 28, 32) | output: (None, 28, 28, 64)
separable_conv2d_6: SeparableConv2D | input: (None, 28, 28, 32) | output: (None, 28, 28, 64)
max_pooling2d_3: MaxPooling2D | input: (None, 28, 28, 64) | output: (None, 14, 14, 64)
max_pooling2d_4: MaxPooling2D | input: (None, 28, 28, 64) | output: (None, 14, 14, 64)
concatenate_2: Concatenate | input: [(None, 14, 14, 64), (None, 14, 14, 64)] | output: (None, 14, 14, 128)
reshape_2: Reshape | input: (None, 14, 14, 128) | output: (None, 7, 7, 512)
flatten_2: Flatten | input: (None, 7, 7, 512) | output: (None, 25088)
dense_4: Dense | input: (None, 25088) | output: (None, 1000)
dense_5: Dense | input: (None, 1000) | output: (None, 500)
dense_6: Dense | input: (None, 500) | output: (None, 10)

**Fig 1. Illustration of NN architecture**

## 1.2 Parameters of the Model



```
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            (None, 28, 28, 1)    0
__________________________________________________________________________________________________
separable_conv2d_4 (SeparableCo (None, 28, 28, 32)   848         input_2[0][0]
__________________________________________________________________________________________________
separable_conv2d_5 (SeparableCo (None, 28, 28, 64)   8384        separable_conv2d_4[0][0]
__________________________________________________________________________________________________
separable_conv2d_6 (SeparableCo (None, 28, 28, 64)   8384        separable_conv2d_4[0][0]
__________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 14, 14, 64)   0           separable_conv2d_5[0][0]
__________________________________________________________________________________________________
max_pooling2d_4 (MaxPooling2D)  (None, 14, 14, 64)   0           separable_conv2d_6[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 14, 14, 128)  0           max_pooling2d_3[0][0]
                                                                 max_pooling2d_4[0][0]
__________________________________________________________________________________________________
reshape_2 (Reshape)             (None, 7, 7, 512)    0           concatenate_2[0][0]
__________________________________________________________________________________________________
flatten_2 (Flatten)             (None, 25088)        0           reshape_2[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 1000)         25089000    flatten_2[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                 (None, 500)          500500      dense_4[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                 (None, 10)           5010        dense_5[0][0]
==================================================================================================
Total params: 25,612,126
Trainable params: 25,612,126
Non-trainable params: 0
__________________________________________________________________________________________________
```

**Table 1. Model parameters**

**NOTE:** I have used separable 2D convolutions instead of normal 2D convolution, to speed up the training time.

## Task 2: Develop an inference server to predict handwritten Digits

To develop a robust Inference Server, following steps were performed:

1. Develop a robust Neural Network in Keras using the architecture as specified in the first part
2. Transform the Keras model to Tensorflow-TensorRT model for accelarating inference

## 2 Robust Neural Network in Keras

The following sections outline the process of designing and optimizing our neural network model

### 2.1 Train/Test Split

- Complete MNIST Dataset was split into train and test datasets.
- Further, train dataset was split into train and validation/hold-out set.

### 2.2 NN-Parameter Estimation

Using the validation set, following parameters were estimated, using accuracy of validation set as metric of performance:

- **Batch Size and Number of Epochs**
- **Optimal rate for the optimization algorithm (ADAM)**

The best(empirically determined) batch size and no. of epochs are used to further select an appropriate learning rate for the ADAM optimization algorithm

### 2.2.1 Batch Size and Number of Epochs

Number of Epochs and Batch Size were empirically determined as shown below. Experiments were conducted to find the optimal epoch and training batch size for maximal validation set performance.

We note that maximum accuracy on validation set is obtained using the following configuration:

**Batch size = 32 & Number of Epochs = 3**

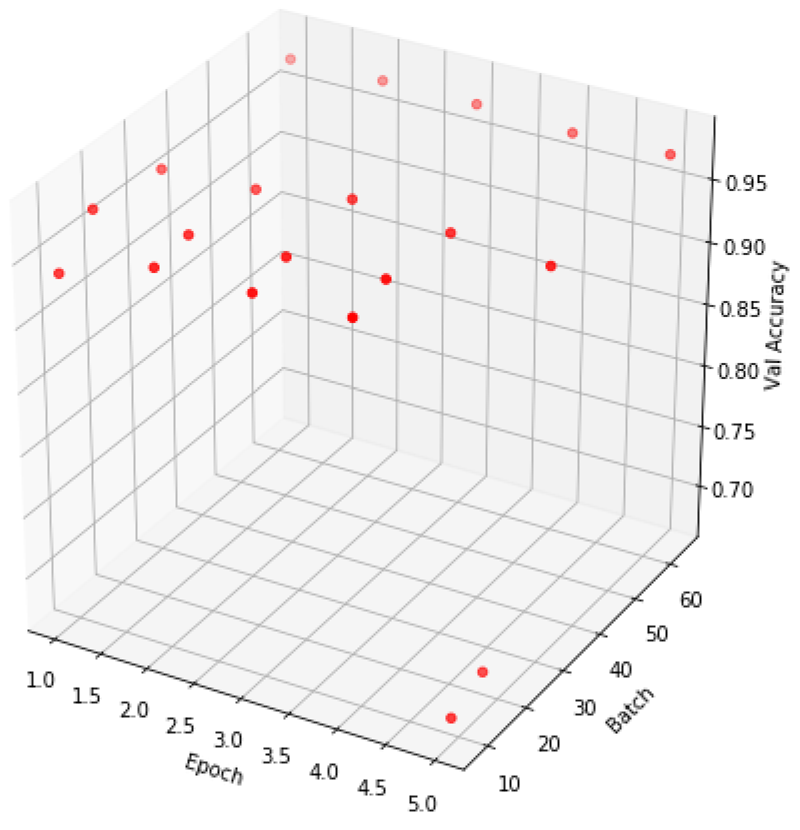


**Fig 2. No. of epochs vs. Batch size vs. Validation Accuracy**

### 2.2.2 Optimal rate for the optimization algorithm (ADAM)

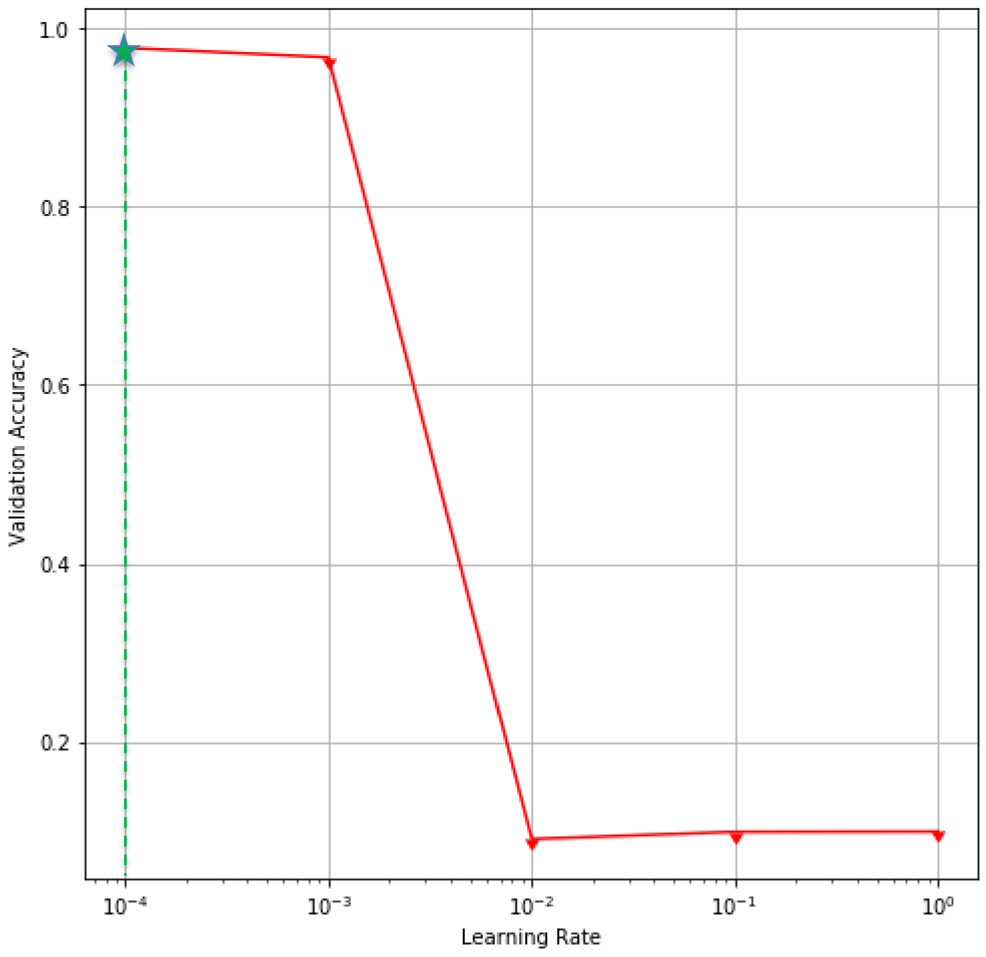


**Fig 3. Learning Rate vs. Validation Accuracy**

**(for the Batch size and no. of epochs selected in 2.3.1)**

**Observation:** We see that Validation set accuracy is highest when learning rate is **0.0001**. We will be using this value of learning rate for further experiments.

## 2.3 Performance Tuning using Dropout and Batch Normalization

In order to avoid overfitting and improve the accelerating the learning process Neural network, we explore dropout and batch normalization. We observe that we have a reduction in performance on validation set using Batch normalization and dropout.

| **Method** | **Results on Validation Set** |
|---|---|
| Plain Neural Network with learnt hyperparameters | 0.97808 |
| Using Batch Normalization | 0.9408 |
| Using Dropout in Fully Connected Layers | 0.9417 |
| Batch Normalization + Dropout | 0.9048 |

**Table 2. Tuning methods and effect on validation accuracy**

**Observation:** Therefore, we will not be using batch normalization or dropout in the final trained model.

## 2.4 Evaluation on MNIST Test Set:

On using the neural network as trained above on MNIST test set, we obtain the following result:

**Accuracy: 97.87%**

## 3 Improving Inference with TensorRT

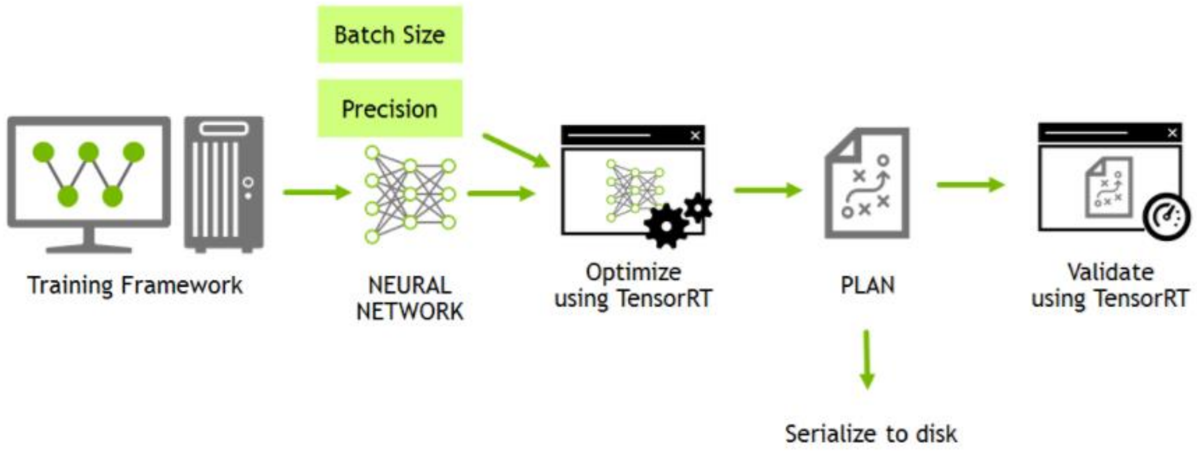


**Fig 4. Sample pipeline for TensorRT optimization**

To further optimize our implementation for production deployment, we can utilize the TensorRT modules from NVIDIA.

A brief overview of inference model optimization using TRT is as follows:

- TRT is a NVIDIA software solution for generating optimized models for production deployment of Deep Learning Models.
- A trained network (hyperparameters, model architecture) and other parameters like inference batch size and required precision are taken as input
- TensorRT performs the necessary implementation optimizations and builds an "execution plan" which can be used as-is or serialized and saved to disk for later use.

### References (TensorRT):

- Low Precision Inference with TensorRT - https://towardsdatascience.com/low-precision-inference-with-tensorrt-6eb3cda0730b
- sampleUffMNIST example - https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html#mnist_uff_sample