Candidate Name: Yonggun Lee

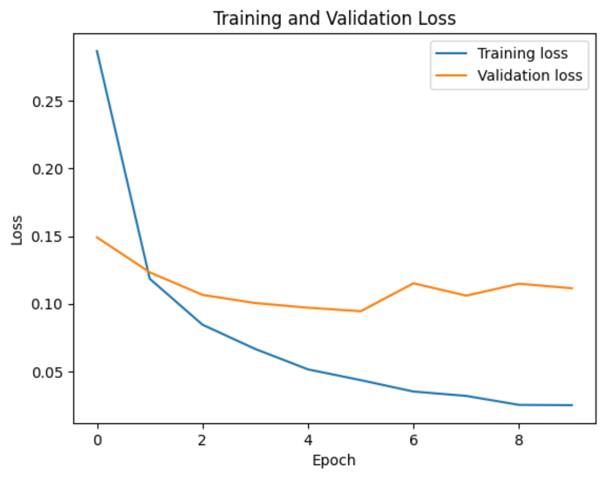
Warm up Question

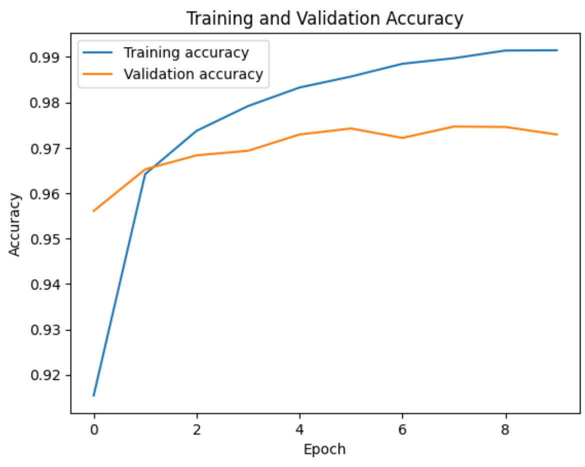
Warmup:

1. Firstly, explore and run this tensorflow-keras example for a simple convnet that does MNIST classification: https://keras.io/examples/vision/mnist_convnet/.

   a. Plot the loss and accuracy curves for the train and validation sets and explain your evaluation of the model. How many parameters does the model have and what is the total memory footprint of your model?

Answer: I trained the model with MNIST and firstly split training and testing set. When I perform the training, I additionally set 20% of training set as validation set. Plots are attached below

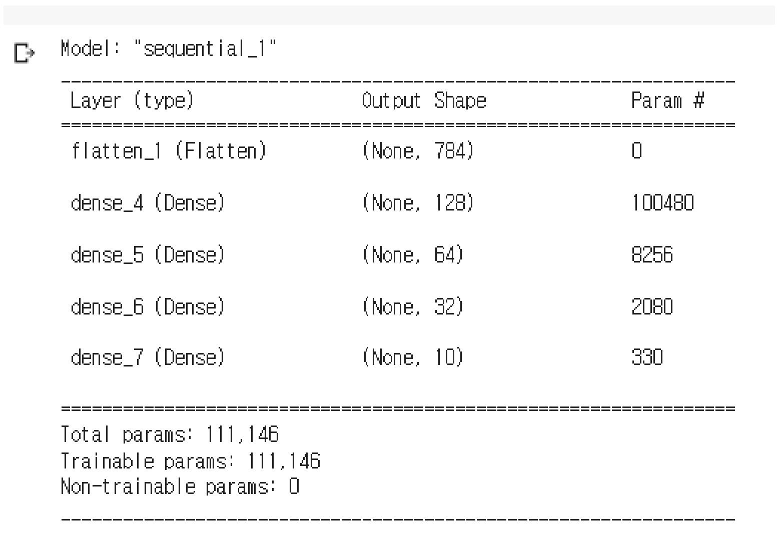




I trained for 10 epochs (10 iterations) and details are below attached. Training accuracy reached to 99.15% while validation accuracy reached 97.29%

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
Epoch 1/10
1500/1500 [==============================] - 7s 4ms/step - loss: 0.2869 - accuracy: 0.9154 - val_loss: 0.1491 - val_accuracy: 0.9561
Epoch 2/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.1185 - accuracy: 0.9642 - val_loss: 0.1232 - val_accuracy: 0.9653
Epoch 3/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0845 - accuracy: 0.9737 - val_loss: 0.1065 - val_accuracy: 0.9683
Epoch 4/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.0665 - accuracy: 0.9792 - val_loss: 0.1005 - val_accuracy: 0.9693
Epoch 5/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0514 - accuracy: 0.9833 - val_loss: 0.0970 - val_accuracy: 0.9729
Epoch 6/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.0435 - accuracy: 0.9857 - val_loss: 0.0945 - val_accuracy: 0.9743
Epoch 7/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0350 - accuracy: 0.9885 - val_loss: 0.1151 - val_accuracy: 0.9722
Epoch 8/10
1500/1500 [==============================] - 5s 3ms/step - loss: 0.0318 - accuracy: 0.9897 - val_loss: 0.1060 - val_accuracy: 0.9747
Epoch 9/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0252 - accuracy: 0.9914 - val_loss: 0.1148 - val_accuracy: 0.9746
Epoch 10/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0250 - accuracy: 0.9915 - val_loss: 0.1115 - val_accuracy: 0.9729
313/313 [==============================] - 1s 1ms/step - loss: 0.1086 - accuracy: 0.9734
Test accuracy: 0.9733999967575073
```

To calculate the foot print, I print out the model summary.



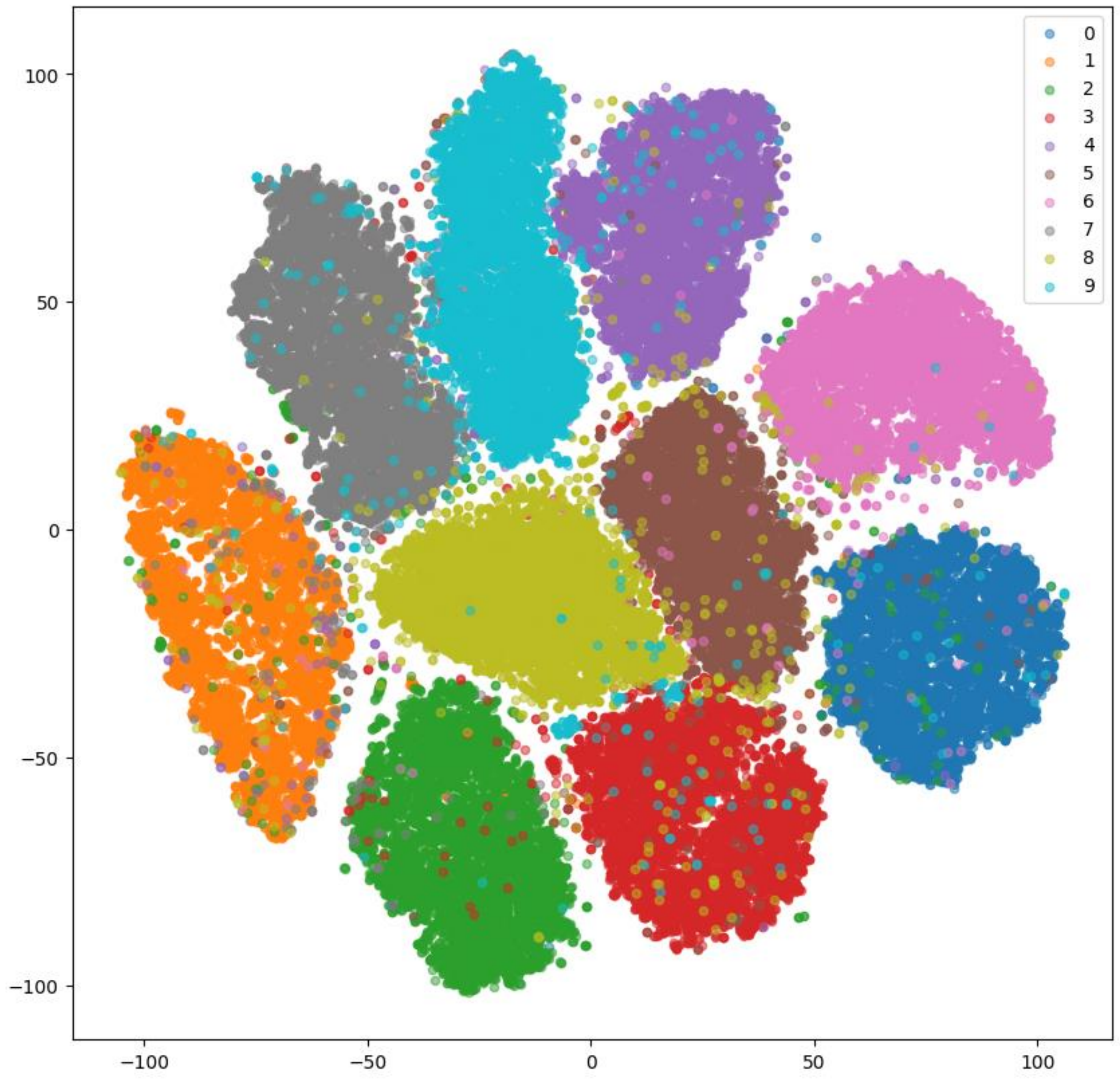
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 784)               0

 dense_4 (Dense)             (None, 128)               100480

 dense_5 (Dense)             (None, 64)                8256

 dense_6 (Dense)             (None, 32)                2080

 dense_7 (Dense)             (None, 10)                330

=================================================================
Total params: 111,146
Trainable params: 111,146
Non-trainable params: 0
_________________________________________________________________
```

Since the total # of params are 111146, I can calculate the memory footprint by 111146 x 32 / 8 / 1024 / 1024. That will give me 0.42 MB.

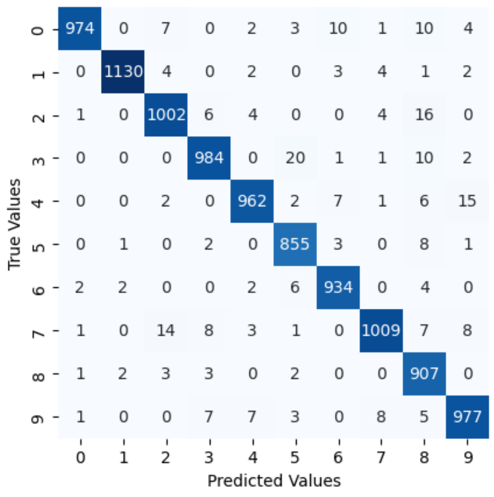
Technical Analysis Question 1.

1. Analyze the MNIST dataset using any technique(s) of your choice. Explain your interpretations of the data and your choice of technique(s).

For this it looks like the question wants me to observe the how the MNIST dataset is. To observe it, I can work with visualization to see the how it cluster or showing something interesting. In this work, I will be using t-SNE to visualize how each class cluster (Projecting in 2 Dimensions). PCA were popular in the past, but unlike PCA is able to account for non-linear relationships.



Other than this, we can also visualize with plotting out confusion matrix. I will be making Logistic Regression model to predict and get the confusion matrix.



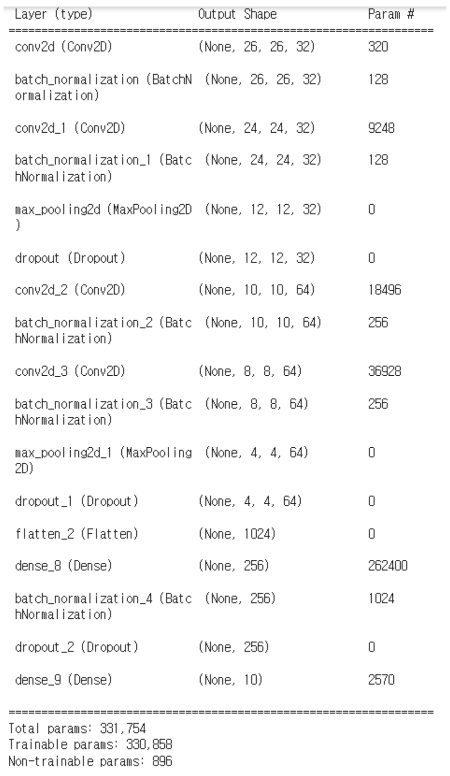
	0	1	2	3	4	5	6	7	8	9
0	974	0	7	0	2	3	10	1	10	4
1	0	1130	4	0	2	0	3	4	1	2
2	1	0	1002	6	4	0	0	4	16	0
3	0	0	0	984	0	20	1	1	10	2
4	0	0	2	0	962	2	7	1	6	15
5	0	1	0	2	0	855	3	0	8	1
6	2	2	0	0	2	6	934	0	4	0
7	1	0	14	8	3	1	0	1009	7	8
8	1	2	3	3	0	2	0	0	907	0
9	1	0	0	7	7	3	0	8	5	977

With confusion matrix, we can see how the each label has correlation to different class.

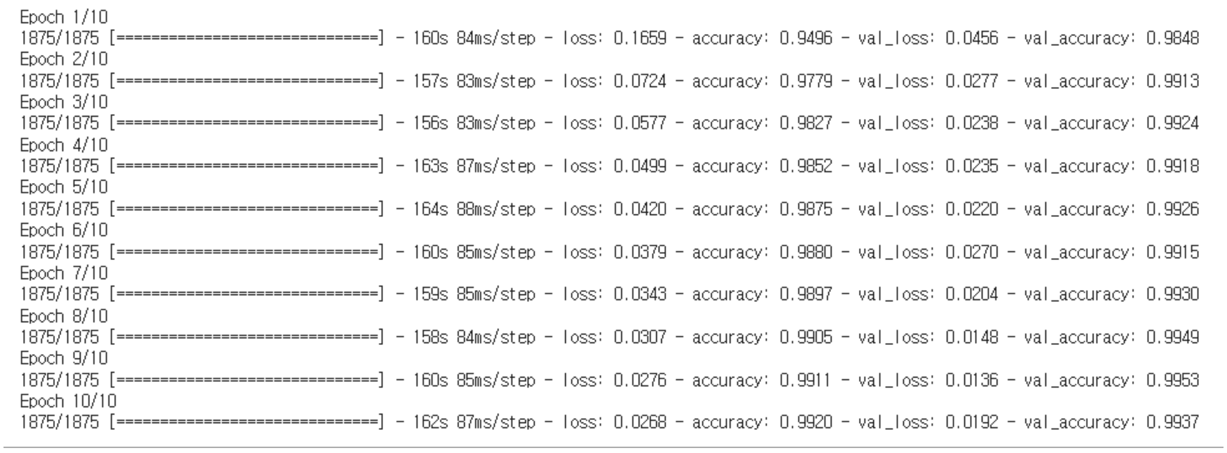
Technical Analysis Question 2.

2. Make the above keras model bigger (bulkier with more layers of your choice) while retaining test accuracy >=98%. Explain your interpretation of the results and your choice of layers.

Answer:



```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 batch_normalization (BatchN  (None, 26, 26, 32)       128
 ormalization)

 conv2d_1 (Conv2D)           (None, 24, 24, 32)        9248

 batch_normalization_1 (Batc  (None, 24, 24, 32)       128
 hNormalization)

 max_pooling2d (MaxPooling2D  (None, 12, 12, 32)       0
 )

 dropout (Dropout)           (None, 12, 12, 32)        0

 conv2d_2 (Conv2D)           (None, 10, 10, 64)        18496

 batch_normalization_2 (Batc  (None, 10, 10, 64)       256
 hNormalization)

 conv2d_3 (Conv2D)           (None, 8, 8, 64)          36928

 batch_normalization_3 (Batc  (None, 8, 8, 64)         256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 4, 4, 64)          0

 flatten_2 (Flatten)         (None, 1024)              0

 dense_8 (Dense)             (None, 256)               262400

 batch_normalization_4 (Batc  (None, 256)              1024
 hNormalization)

 dropout_2 (Dropout)         (None, 256)               0

 dense_9 (Dense)             (None, 10)                2570

=================================================================
Total params: 331,754
Trainable params: 330,858
Non-trainable params: 896
```



```
Epoch 1/10
1875/1875 [==============================] - 160s 84ms/step - loss: 0.1659 - accuracy: 0.9496 - val_loss: 0.0456 - val_accuracy: 0.9848
Epoch 2/10
1875/1875 [==============================] - 157s 83ms/step - loss: 0.0724 - accuracy: 0.9779 - val_loss: 0.0277 - val_accuracy: 0.9913
Epoch 3/10
1875/1875 [==============================] - 156s 83ms/step - loss: 0.0577 - accuracy: 0.9827 - val_loss: 0.0238 - val_accuracy: 0.9924
Epoch 4/10
1875/1875 [==============================] - 163s 87ms/step - loss: 0.0499 - accuracy: 0.9852 - val_loss: 0.0235 - val_accuracy: 0.9918
Epoch 5/10
1875/1875 [==============================] - 164s 88ms/step - loss: 0.0420 - accuracy: 0.9875 - val_loss: 0.0220 - val_accuracy: 0.9926
Epoch 6/10
1875/1875 [==============================] - 160s 85ms/step - loss: 0.0379 - accuracy: 0.9880 - val_loss: 0.0270 - val_accuracy: 0.9915
Epoch 7/10
1875/1875 [==============================] - 159s 85ms/step - loss: 0.0343 - accuracy: 0.9897 - val_loss: 0.0204 - val_accuracy: 0.9930
Epoch 8/10
1875/1875 [==============================] - 158s 84ms/step - loss: 0.0307 - accuracy: 0.9905 - val_loss: 0.0148 - val_accuracy: 0.9949
Epoch 9/10
1875/1875 [==============================] - 160s 85ms/step - loss: 0.0276 - accuracy: 0.9911 - val_loss: 0.0136 - val_accuracy: 0.9953
Epoch 10/10
1875/1875 [==============================] - 162s 87ms/step - loss: 0.0268 - accuracy: 0.9920 - val_loss: 0.0192 - val_accuracy: 0.9937
```

- Input layer: a convolutional layer with 32 filters and a 3x3 kernel, followed by batch normalization and ReLU activation. The input shape is (28, 28, 1), corresponding to a grayscale image.
- Convolutional layer: another convolutional layer with 32 filters and a 3x3 kernel, followed by batch normalization and ReLU activation.

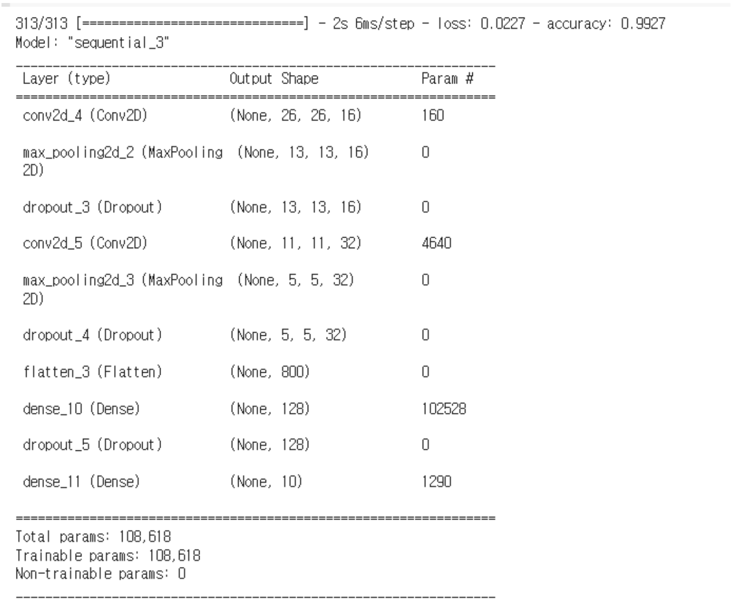
- Max pooling layer: a max pooling layer with a 2x2 pool size, to reduce the spatial dimensions of the feature maps by half.
- Dropout layer: a dropout layer with a rate of 0.25, to randomly drop out 25% of the activations and reduce overfitting.
- Convolutional layer: a convolutional layer with 64 filters and a 3x3 kernel, followed by batch normalization and ReLU activation.
- Convolutional layer: another convolutional layer with 64 filters and a 3x3 kernel, followed by batch normalization and ReLU activation.
- Max pooling layer: another max pooling layer with a 2x2 pool size.
- Dropout layer: another dropout layer with a rate of 0.25.
- Flatten layer: a flatten layer to convert the 2D feature maps into a 1D vector.
- Dense layer: a fully connected layer with 256 units and ReLU activation,
- followed by batch normalization and dropout with a rate of 0.5.
- Last Dense layer output much be 10 because it is 10 digit classes from MNIST

Comparing to the previous model, validation accuracy already hit 98% above from the first iteration. However, model is bulkier. Which means it needs more time to train and computational power will be needed more. To make it faster, using high spec of CUDA machine will be one of good solution.

Technical Analysis Question 3.

3. Make the above keras model smaller (remove your choice of layers) while retaining test accuracy >=98%. Explain your interpretation of the results and your choice of layers.

For the last model (Removed layer model), from 2nd epoch, it started to show 98% above validation accuracy. Model summary is below



```
313/313 [==============================] - 2s 6ms/step - loss: 0.0227 - accuracy: 0.9927
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 26, 26, 16)        160

 max_pooling2d_2 (MaxPooling  (None, 13, 13, 16)       0
 2D)

 dropout_3 (Dropout)         (None, 13, 13, 16)        0

 conv2d_5 (Conv2D)           (None, 11, 11, 32)        4640

 max_pooling2d_3 (MaxPooling  (None, 5, 5, 32)         0
 2D)

 dropout_4 (Dropout)         (None, 5, 5, 32)          0

 flatten_3 (Flatten)         (None, 800)               0

 dense_10 (Dense)            (None, 128)               102528

 dropout_5 (Dropout)         (None, 128)               0

 dense_11 (Dense)            (None, 10)                1290

=================================================================
Total params: 108,618
Trainable params: 108,618
Non-trainable params: 0
_________________________________________________________________
```

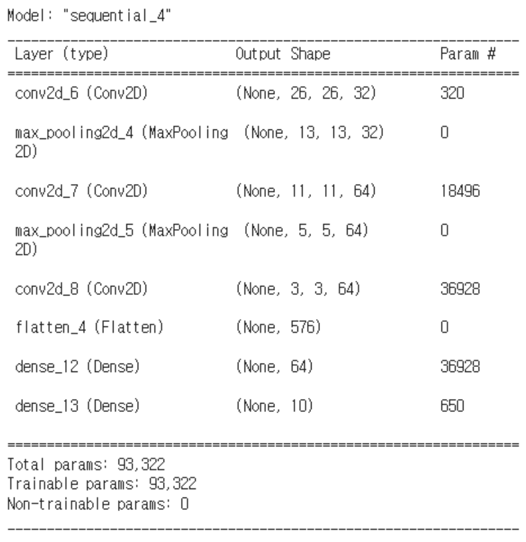
- Input layer: a convolutional layer with 16 filters and a 3x3 kernel, followed by ReLU activation. The input shape is (28, 28, 1), corresponding to a grayscale image.
- Max pooling layer: a max pooling layer with a 2x2 pool size, to reduce the spatial dimensions of the feature maps by half.
- Dropout layer: a dropout layer with a rate of 0.25, to randomly drop out 25% of the activations and reduce overfitting.
- Convolutional layer: another convolutional layer with 32 filters and a 3x3 kernel, followed by ReLU activation.
- Max pooling layer: another max pooling layer with a 2x2 pool size.
- Dropout layer: another dropout layer with a rate of 0.25.
- Flatten layer: a flatten layer to convert the 2D feature maps into a 1D vector.
- Dense layer: a fully connected layer with 128 units and ReLU activation, followed by dropout with a rate of 0.5.
- Output layer: a dense layer with 10 units and softmax activation, corresponding to the 10 possible MNIST digits.

Technical Question (Bonus)

1. Visualize the feature maps and filters from one of the convolutional layers and explain your interpretation of what they represent.

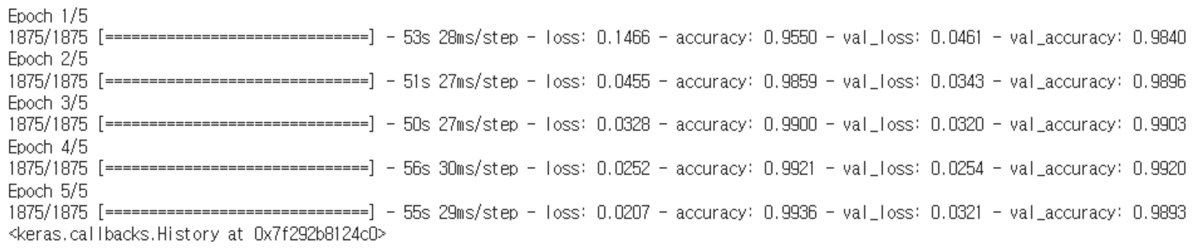
This question is more likely I guess asking me if I know if kernel is same as learnable filter and how the actual training process can be visualized. I am moving model into Keras format now and will plot out my first layer.

This is model summary that I built for this task



```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_4 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 conv2d_7 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_8 (Conv2D)           (None, 3, 3, 64)          36928

 flatten_4 (Flatten)         (None, 576)               0

 dense_12 (Dense)            (None, 64)                36928

 dense_13 (Dense)            (None, 10)                650

=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
```

Below is training process with loss&accuracy by epochs.



```
Epoch 1/5
1875/1875 [==============================] - 53s 28ms/step - loss: 0.1466 - accuracy: 0.9550 - val_loss: 0.0461 - val_accuracy: 0.9840
Epoch 2/5
1875/1875 [==============================] - 51s 27ms/step - loss: 0.0455 - accuracy: 0.9859 - val_loss: 0.0343 - val_accuracy: 0.9896
Epoch 3/5
1875/1875 [==============================] - 50s 27ms/step - loss: 0.0328 - accuracy: 0.9900 - val_loss: 0.0320 - val_accuracy: 0.9903
Epoch 4/5
1875/1875 [==============================] - 56s 30ms/step - loss: 0.0252 - accuracy: 0.9921 - val_loss: 0.0254 - val_accuracy: 0.9920
Epoch 5/5
1875/1875 [==============================] - 55s 29ms/step - loss: 0.0207 - accuracy: 0.9936 - val_loss: 0.0321 - val_accuracy: 0.9893
<keras.callbacks.History at 0x7f292b8124c0>
```

Since my Kernel size is (3, 3) (You can refer to my script), the plots shows as 3 x 3 color matrix. Those learnable filters (Kernel) will try to detect on the image (actually this is mathematic calculation in reality). Each filter is a 3 x 3 matrix of weights that is convolved with the input image to produce a feature map. The visualized filters represent the patterns that the first convolutional layer is trying to detect in the input images. The brighter regions of the filters indicate the regions.