Name: Brandon Eyongherok

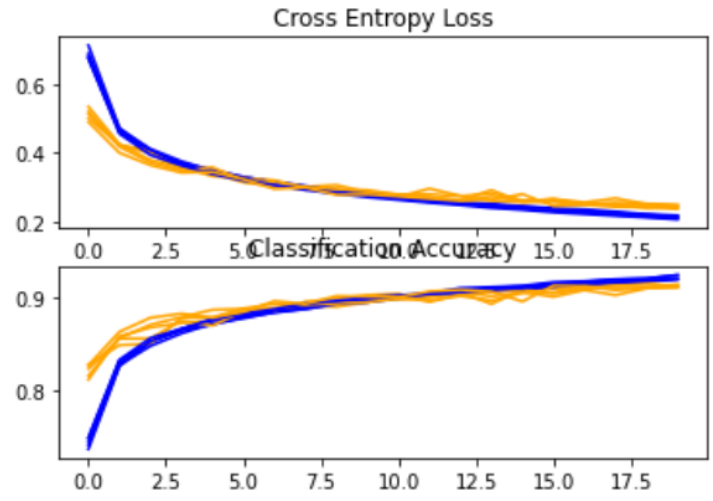
1. Implement Normalization technique to achieve
   zero mean and zero variance and One-hot encoding if required

2.

i. Report all the layers that you considered in your model and illustrate the functionality of each layer. Report the results obtained for various learning rates that you considered. Also, illustrate all the hyper parameters (e.g., number of filters, Dimensions of Filter, Stride, Padding, Activation Function,etc.). Make sure to elaborate the reason for considering specific parameters with mathematical calculation wherever required. (15p)

Two models were chosen as the final models and their entire codes were submitted. All the criterias chosen for the final model were based on the results of other test models and parameters.

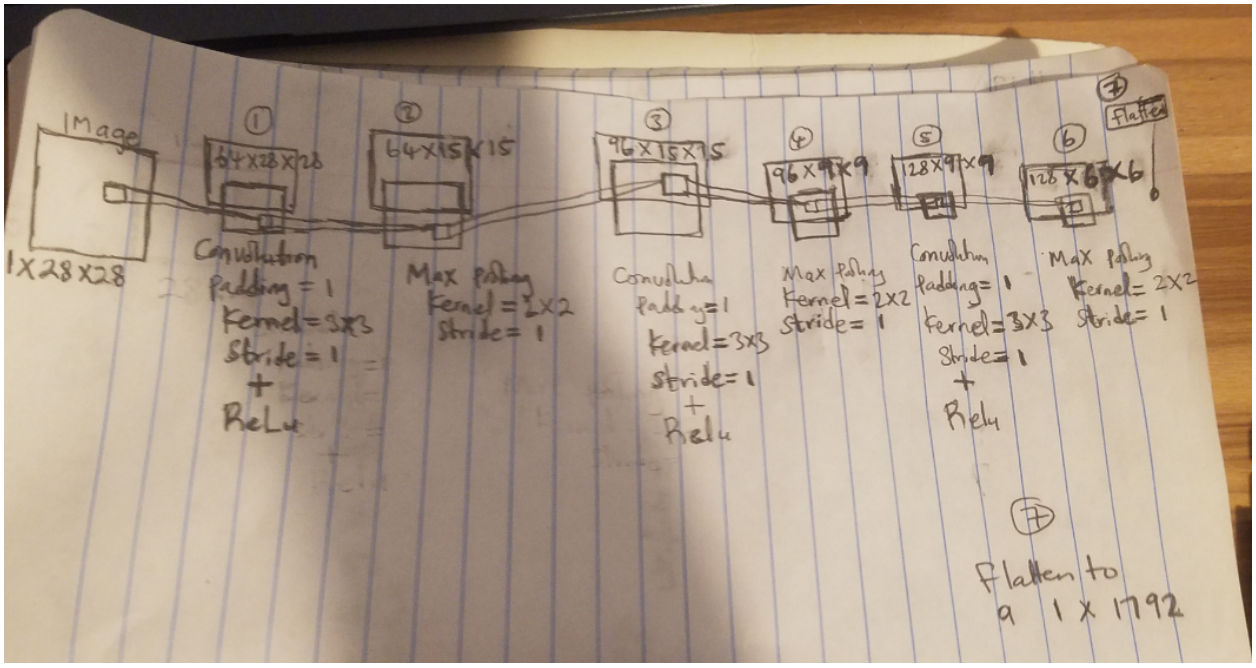
The first model consists of 64 feature maps, pooling of kernel size of 2, 96 feature map, pooling of kernel size of 2, and 128 feature map, poolin of kernel size of 2, and then flatten. Example shown below:

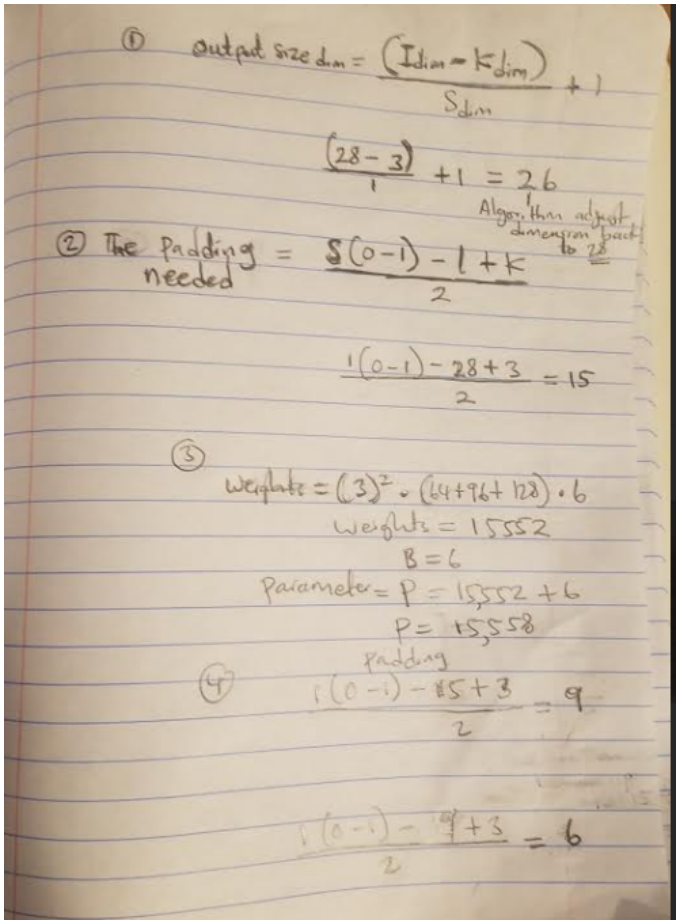



```
Accuracy: mean=91.285 std=0.118, n=5
```

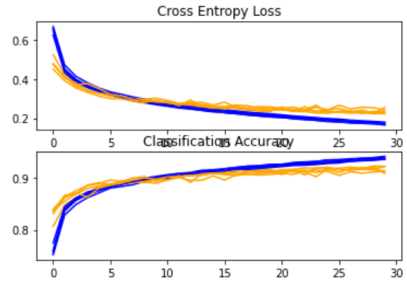
The loss and accuracy shows the model had little to no overfitting. The drawbacks include the time it took to run those many layers for 20 epoch. It took about 2 hours to complete running.

Calculations:



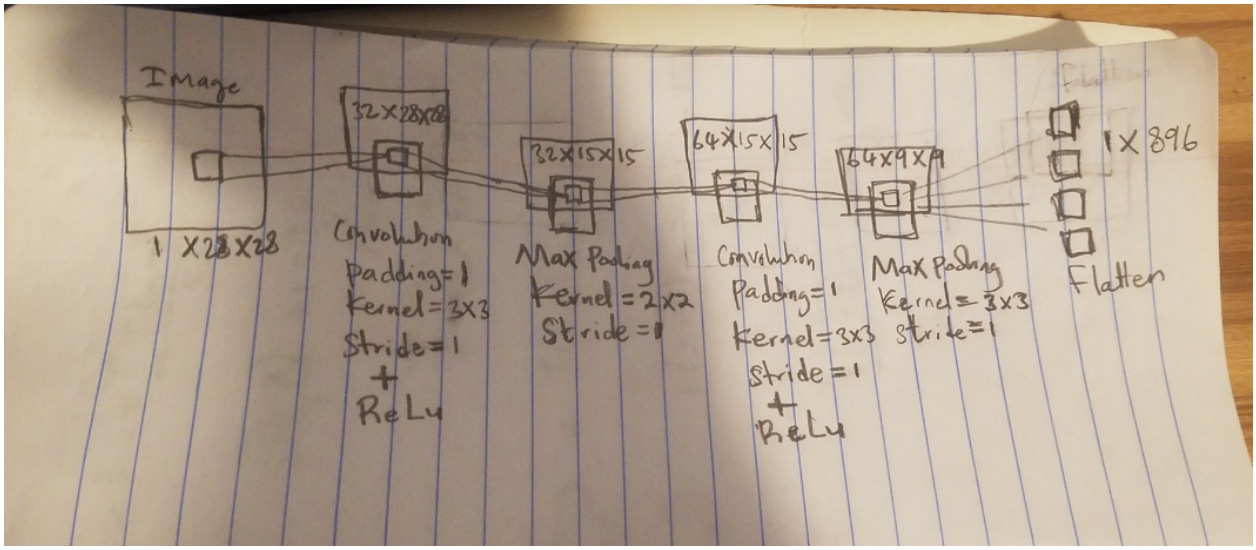


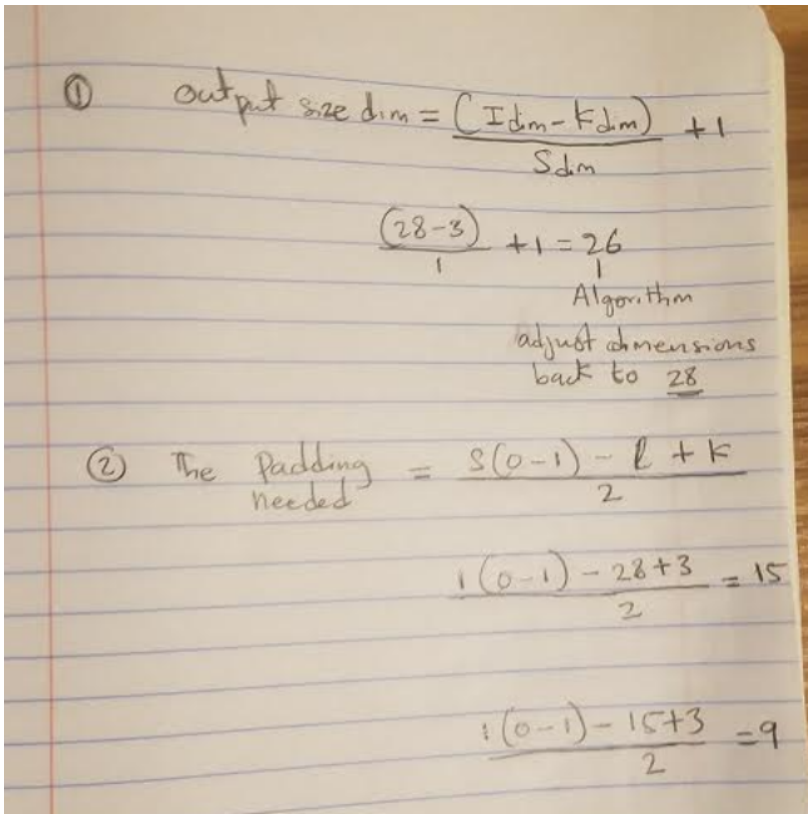
The second model consists of 32 feature maps, pooling of kernel size of 2, 64 feature maps, pooling of kernel size of 2, and flatten. Example shown below:




```
Accuracy: mean=91.472 std=0.534, n=5
```

The number of layers is less in this model, but the training epoch is increased to 30. There is a slight overfit in this model than the first model, but still miniscule. The drawback also includes the time it takes to run. It took about an hour.
Calculations:

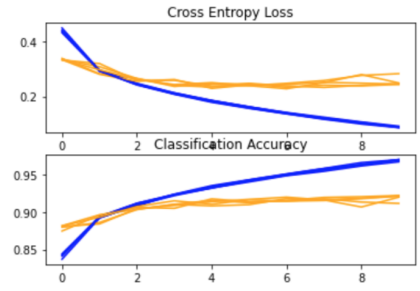




Learning rate of 0.05, 0.01, and 0.001 were considered but only 0.01 is used.
The same model was used for all the learning rates considered. Two convolutional layers, one with 32 feature maps and another with 64 feature maps.
An example with a learning rate of 0.05

```
41
42 # define cnn model
43 def define_model():
44     model = Sequential()
45     model.add(Conv2D(32, (3, 3), padding='same', activation='relu', kernel_initializer='he_uniform', input_shape
46     model.add(MaxPooling2D((2, 2)))
47     model.add(Conv2D(64, (3, 3), padding='same', activation='relu', kernel_initializer='he_uniform'))
48     model.add(Flatten())
49     model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
50     model.add(Dense(10, activation='softmax'))
51     opt = SGD(lr=0.05)
52     model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
53     return model
54
```

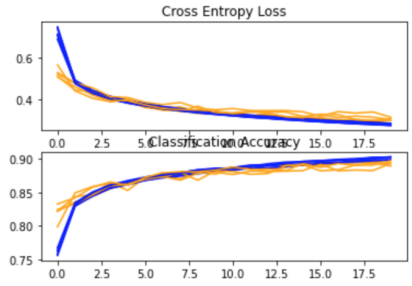
Learning rate of 0.05:




```
Accuracy: mean=91.907 std=0.389, n=5
```

The learning of 0.05 learns too quickly, so the model overfit. Thus this learning rate is eliminated.

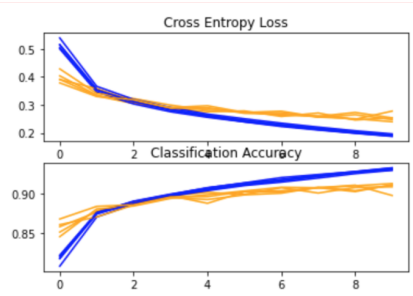
Learning rate of 0.001:




```
Accuracy: mean=89.305 std=0.251, n=5
```

The learning rate of 0.001 does well as is observable in the loss and accuracy graph, but it took 20 epochs to get to a good loss and accuracy graph.

Learning rate of 0.01:

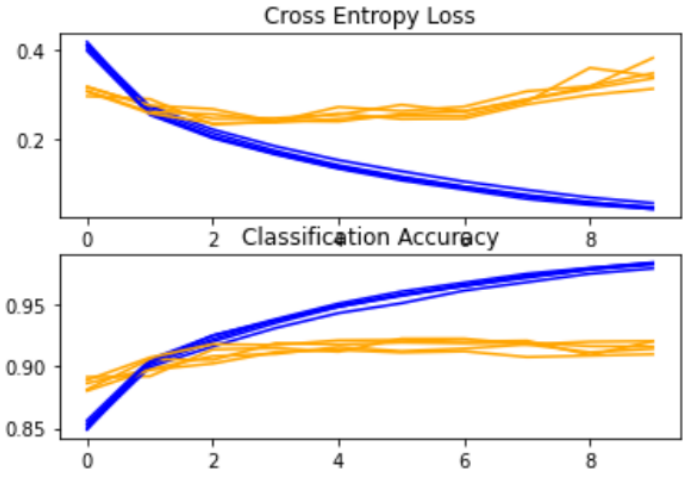



```
Accuracy: mean=90.842 std=0.542, n=5
```

The learning rate of 0.01 does not do as well because there is more over fitting than the learning rate of 0.001, but if you take into consideration that you can drop some of the weights using a Dropout in keras. Then, this is the ideal learning rate because you can learn as quickly as the learning rate of 0.05 while also regularizing the model using Dropout.

In two of the models tried out, was a 32 feature map feeding to a 64 feature map and in another model a 64 feature map feeding into a 32 feature map. See below:

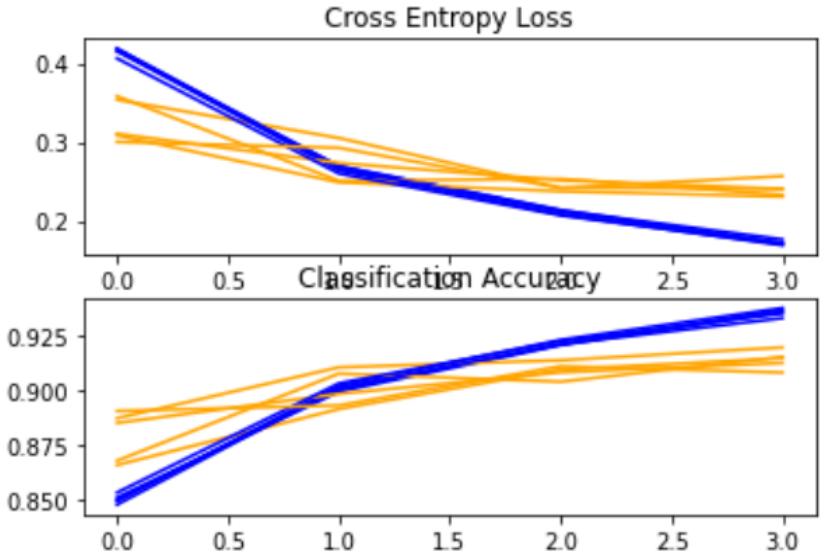
```python
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(64, (3, 3), padding='same', activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
    model.add(Conv2D(32, (3, 3), padding='same', activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(lr=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```




```
Accuracy: mean=91.600 std=0.398, n=5
```

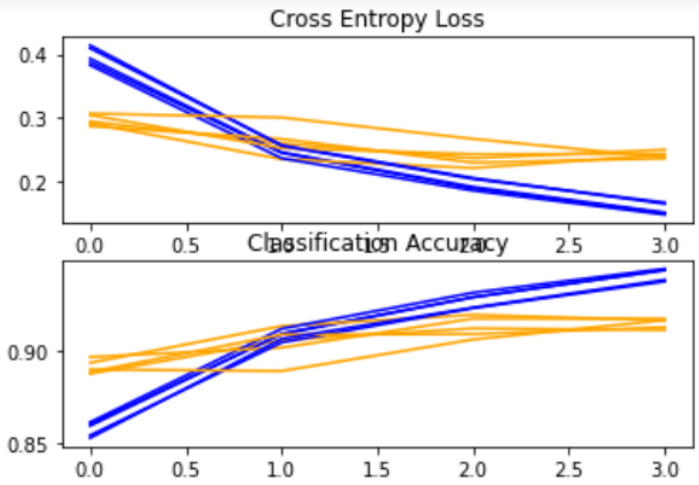
Adding an additional layer from 64 feature map to 32, meant that the convolutional neural network lost some details because the size of the filter was shrinked. In the graph, the **blue represents the training set and the yellow represents the validation set**. The **cross entropy loss graph** shows that in the case of this additional layer with 32 filters, the training epoch should have been stopped at about the fourth epoch. The training error starts diverging from the validation error at about the fourth epoch. The training is getting a much better prediction than the validation, which is a classic case of the convolutional neural network **overfitting** during training. This is also noticeable is the **Classification Accuracy graph**, where the blue line, the training set diverges from the validation set and gets better accuracy, meanwhile the validation set is flatline around the 90 percent mark.

Stop it the convolutional neural network above in the fourth epoch results:



It is observable that the results are much better, no overfitting.

Switch the 64 filter with the 32 filter, because the layers are increasing from 32 to 64, less details should be lost than previously going from a 64 to 32 filter.




```
Accuracy: mean=91.535 std=0.246, n=5
```

Got about the same result even after switching the layers, which means that not relevant details or details that could change the prediction scores were captured from the 32 filter to the 64 filter. So widening the convolution neural network from a 32 filter to a 64 filter, which theoretically would mean it is capturing more detail during filtering, did not make a significant difference in the prediction during the validation test. As it is observable by the mean accuracy score and the cross entropy loss graph and Classification Accuracy graph.

ii.

Show the type of optimizers used out of various options during the compilation of the model and type of cross entropy for classifying multiclasses. Decay the learn rate. Describe your parameter decay approach and comment on your results training on number of epochs. (10p)

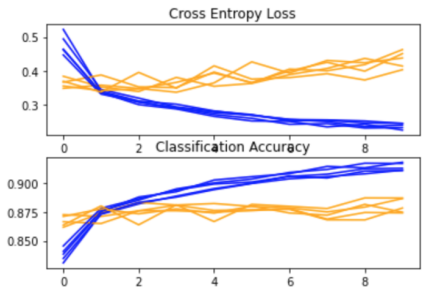
The Adam optimizer and the GDS optimizers are used. The Adam optimizer used with the same layer as the GDS shows that the GDS optimizer performs better, so the GDS optimizer was explored further and Adam optimizer was not.

We have two optimizers in this scenario, the GDS and Adam optimers. The layers pooling and neurons are all kept the same, so in both cases there are two layers, a 32 feature map into a 64 feature map. So, the only difference in between the two algorithms are their optimizers. The SGD optimizer performed better during validation than the Adam optimizer. The results were also noticeable with just one layer of 64 feature maps which was not demonstrated below like the other two examples.

```python

# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), padding='same', activation='relu', kernel_initializer='he_uniform', input_shape
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), padding='same', activation='relu', kernel_initializer='he_uniform'))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    opt = Adam(lr=0.01)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model

```

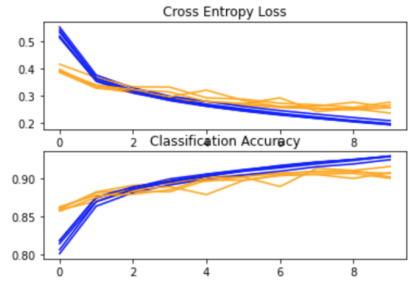



```
Accuracy: mean=88.015 std=0.561, n=5
```

Can see the SGD optimizer performed better during validation. So the SGD was further explored for the final models chosen.

```python
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), padding='same', activation='relu', kernel_initializer='he_uniform', input_shape
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), padding='same', activation='relu', kernel_initializer='he_uniform'))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    opt = SGD(lr=0.01)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model

```

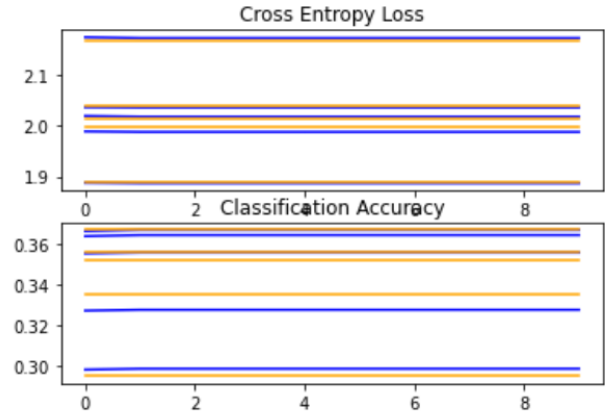



```
Accuracy: mean=90.675 std=0.544, n=5
```

The decaying learning rate is not practical in this algorithm because the epochs are too small, and remember the epochs are small because they are tied to large data samples and computational power of the computer. Thus the decaying example as demonstrated

below was not explored further. The learning rate starts at 0.01 and decays every 2 epochs at a decaying rate of 0.05 with a total epoch of ten.

```
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(64, (3, 3), padding='same', activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    lr_schedule = schedules.ExponentialDecay(
    initial_learning_rate=1e-2,
    decay_steps=2,
    decay_rate=0.05)
    opt = SGD(learning_rate=lr_schedule)
    #opt = SGD(lr=lr_schedule)
    #opt = SGD(lr=0.01, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```




```
Accuracy: mean=34.103 std=2.497, n=5
```

The epochs are too small for the learning rate to take effect sooner, so a decaying learning rate is most likely not ideal for this scenario as shown above with the vertical line for both the loss and accuracy graph.

The ReLu Activation is used during the convolution to keep the values from becoming negative throughout the filtering process. The ReLu activation outputs a zero if the values are negative. The ReLu activation function is also used in the beginning of back propagation to also ensure that the values are not negative. Softmax activation function

is used in the end with ten neurons to ensure that the convolutional neural network outputs probabilities of the ten different labels of clothing items. Since one encoding is used, only one probability, the label with the largest probability as the inferred clothing item.
The categorical_crossentropy is used as the loss function for the convolution neural network because anytime one encoding is used, categorical_crossentropy must also be used, it is required by keras.

iii.

Plot the graph for training and validation curves. Report the classification report (e.g., F1_score, precision, Recall, contingency table etc.). (10pt)
The plots are shown above with their explanations

iii.

Provide a report to validate that your model has good predicting capability over the classes(5pts)

The first model:

```
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(64, (3, 3), padding='same', activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(96, (3, 3), padding='same', activation='relu', kernel_initializer='he_uniform'))
    model.add(Dropout(0.2))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(128, (3, 3), padding='same', activation='relu', kernel_initializer='he_uniform'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    opt = SGD(lr=0.01)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

Has a validation mean of 91.285 percent accuracy during training for its validation set, but on the test set the accuracy is 88.770 percent, which is slightly below that during the training, but still relatively close.

The second model:

```
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), padding='same', activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (3, 3), padding='same', activation='relu', kernel_initializer='he_uniform'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    opt = SGD(lr=0.01)
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

Has a validation mean of 91.472 percent accuracy. However, the test set came with an accuracy mean of 87.480 percent and we also know the model slightly overfit.

Overall the first model is the best of all the models, because the validation mean accuracy more closely matches the test accuracy mean than the second model. The first model can also be improved because the training epoch can be increase to see how far the model gets before it starts overfitting.