Project: Evaluation of neural network architectures on MNIST datasets

Group Members : Yidong HUANG, Simon ROBER, Marck-Edward KEMEH

ABOUT

The MNIST dataset is a large database of handwritten digits used in many forms of image processing. This dataset contains 60,000 training images and 10,000 test images. Our aim is to design some neural networks that can recognise these hand-written digits. our projects will be based on two different networks for this task.

In [1]:
```
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras
import tensorflow.keras.layers as layers
import tensorflow.keras.models as models
import scikitplot as skplt
import matplotlib.pyplot as plt
import sklearn as skl
from sklearn.metrics import roc_curve
```

We begin by splitting the MNIST dataset into two. A set for training and another for testing. We then normalize the test set and training set by dividing with 255 so that the values can be between 0 and 1. Because we will be working with convolutional neural network, we need to change the original shape of the MNIST dataset which is (60000, 28, 28) to that of the convolutional neural network (60000, 28, 28, 1). This section is just to prepare our dataset for the network.

In [2]:
```
# dividing dataset into train and test set, also reshaping dataset to fit in
put of network
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train/255.0, x_test/255.0
print('shape before adding dimension is :' ,x_train.shape)
x_train, x_test = np.expand_dims(x_train, axis= -1), np.expand_dims(x_test,
axis = -1)
x_train = x_train.reshape(x_train.shape[0],28,28,1)
x_test = x_test.reshape(x_test.shape[0],28,28,1)
print ('shape after adding dimension is :' ,x_train.shape)
```

```
shape before adding dimension is : (60000, 28, 28)
shape after adding dimension is : (60000, 28, 28, 1)
```

First Model

Our first model is a simple network definition which will be used to train the neural network. Our input shape has to match that of the train set as did above and we are using relu as activation

In [6]:
```
#first simple model

model1 = keras.Sequential()
model1.add(layers.Input(shape = (28,28,1)))
model1.add(layers.Conv2D(32, (3,3), padding ='valid', activation = 'relu'))
model1.add(layers.MaxPool2D((2,2), (2, 2)))
model1.add(layers.Flatten())
model1.add(layers.Dense(10, activation = 'softmax'))
model1.summary()
model1.compile(optimizer = "adam", loss = "sparse_categorical_crossentropy",
                metrics = ["accuracy"])
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_2 (Conv2D)            (None, 26, 26, 32)        320       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 5408)              0         
_________________________________________________________________
dense_3 (Dense)              (None, 10)                54090     
=================================================================
Total params: 54,410
Trainable params: 54,410
Non-trainable params: 0
_________________________________________________________________
```

We now test using our test set against the model we trained to determine if the model actually learned to recognise the digits. For each epoch, we can see the loss rate is decreasing which tells us the model is actually improving on the learning.

In [7]:
```
#training
model1.fit(x_train, y_train, epochs = 4)
```

```
Train on 60000 samples
Epoch 1/4
60000/60000 [==============================] - 41s 679us/sample - loss: 0.213
3 - accuracy: 0.9391
Epoch 2/4
60000/60000 [==============================] - 35s 583us/sample - loss: 0.081
7 - accuracy: 0.9768
Epoch 3/4
60000/60000 [==============================] - 40s 659us/sample - loss: 0.062
0 - accuracy: 0.9817
Epoch 4/4
60000/60000 [==============================] - 34s 571us/sample - loss: 0.050
5 - accuracy: 0.9848
```

Out[7]:
```
<tensorflow.python.keras.callbacks.History at 0x7f38e85fed10>
```

Next we use the accuracy to determine how well our classifier classifies on the test set to make sure we are not over fitting.

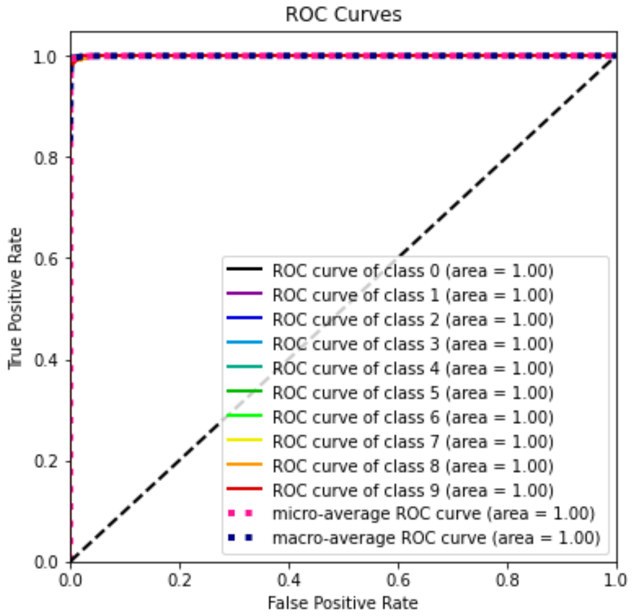
In [8]:
```
def accuracy(model, x_test, y_test):
    test_loss, test_acc = model.evaluate(x_test, y_test)
    print("test loss is : {0} - test accuracy is : {1}".format(test_loss, te
st_acc))


# Accuracy:
accuracy(model1, x_test, y_test)
```

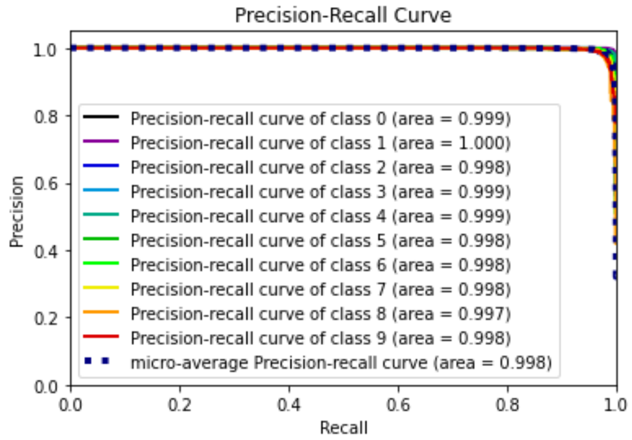
```
10000/10000 [==============================] - 2s 158us/sample - loss: 0.0488
- accuracy: 0.9837
test loss is : 0.04877334827999585 - test accuracy is : 0.9836999773979187
```

ROC

We implement the ROC curve and Precision-Recall curve to view the performance of our model on the test set

In [10]:
```
def roc_curve(model, x_test, y_test):
    y_true = y_test # Given ground truth
    y_probas = model.predict(x_test)
    skplt.metrics.plot_roc(y_true, y_probas, figsize=(6,6))     # https://sc
ikit-plot.readthedocs.io/en/stable/metrics.html
    plt.show()

# ROC:
roc_curve(model1, x_test, y_test)

def precision_recall(model, x_test, y_test):
    y_probas = model.predict(x_test)
    skplt.metrics.plot_precision_recall(y_test, y_probas)
    plt.show()

# Precision-Recall:
precision_recall(model1, x_test, y_test)
```





TODO: interpret results of Roc Curve, Precision recall!!!

Second Model

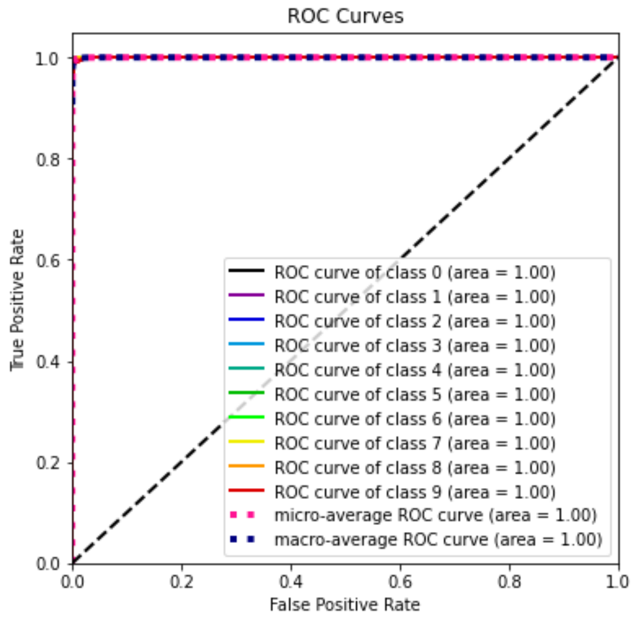
Below is our second model which follows LeNet architecture and has an input shape of (32, 32, 1). Again we define the network, train it and see how it performs in terms of accuracy, Roc curve and Precision-Recall curve.

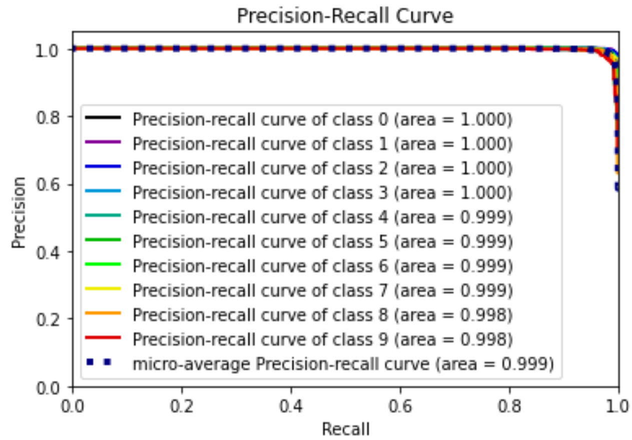
In [9]:
```
model2 = keras.Sequential([
        layers.Conv2D(filters=6, kernel_size=(3, 3), activation='relu', inpu
t_shape=(28,28,1)),
        layers.AveragePooling2D(),
        layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'),
        layers.AveragePooling2D(),
        layers.Flatten(),
        layers.Dense(units=120, activation='relu'),
        layers.Dense(units=84, activation='relu'),
        layers.Dense(units=10, activation = 'softmax')
    ])


model2.summary()
model2.compile(optimizer = "adam", loss = "sparse_categorical_crossentropy",
                metrics = ["accuracy"])

# x_train = np.pad(x_train, ((0,0),(2,2),(2,2),(0,0)), 'constant')
model2.fit(x_train, y_train, epochs = 4)
# Accuracy:
accuracy(model2, x_test, y_test)
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 6)         60        
_________________________________________________________________
average_pooling2d_2 (Average (None, 13, 13, 6)         0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 11, 11, 16)        880       
_________________________________________________________________
average_pooling2d_3 (Average (None, 5, 5, 16)          0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 400)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 120)               48120     
_________________________________________________________________
dense_5 (Dense)              (None, 84)                10164     
_________________________________________________________________
dense_6 (Dense)              (None, 10)                850       
=================================================================
Total params: 60,074
Trainable params: 60,074
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples
Epoch 1/4
60000/60000 [==============================] - 148s 2ms/sample - loss: 0.2297
- accuracy: 0.9309
Epoch 2/4
60000/60000 [==============================] - 145s 2ms/sample - loss: 0.0758
- accuracy: 0.9767
Epoch 3/4
60000/60000 [==============================] - 148s 2ms/sample - loss: 0.0535
- accuracy: 0.9835
Epoch 4/4
60000/60000 [==============================] - 146s 2ms/sample - loss: 0.0434
- accuracy: 0.9861
10000/10000 [==============================] - 3s 324us/sample - loss: 0.0370
- accuracy: 0.9883
test loss is : 0.037002476275642404 - test accuracy is : 0.9883000254631042
```

In [12]:
```
roc_curve(model2, x_test, y_test)
precision_recall(model2, x_test, y_test)
```

```
10000/10000 [==============================] - 2s 159us/sample - loss: 0.0459
- acc: 0.9841
test loss is : 0.04586207606535172 - test accuracy is : 0.9840999841690063
```





TODO: interpret results of Roc Curve, Precision recall!!! + Add comments to the model.

As you can see we now use all the previous defined functions to build the second model. These first two model were our first experience playing around with Keras and with the CNN. In the next section we will look into the parameters in a more systematic maner. We will build on the second model described above and alter one parameter everytime

Comparison model one and model two

In [ ]:

Learning Rate

By default the learning rate of the model with Adam is 0.001. Lets increase and decease the learning rate slightly to see what the effect is.

In [13]:
```
model2_increase_lr = keras.Sequential([
        layers.Conv2D(filters=6, kernel_size=(3, 3), activation='relu', inpu
t_shape=(28,28,1)),
        layers.AveragePooling2D(),
        layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'),
        layers.AveragePooling2D(),
        layers.Flatten(),
        layers.Dense(units=120, activation='relu'),
        layers.Dense(units=84, activation='relu'),
        layers.Dense(units=10, activation = 'softmax')
    ])

model2_increase_lr.compile(optimizer = tf.keras.optimizers.adam(learning_rat
e=0.01)
, loss = "sparse_categorical_crossentropy", metrics = ["accuracy"])

model2_increase_lr.fit(x_train, y_train, epochs = 4)

accuracy(model2_increase_lr, x_test, y_test)
roc_curve(model2_increase_lr, x_test, y_test)
precision_recall(model2_increase_lr, x_test, y_test)
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_4 (Conv2D)            (None, 26, 26, 6)         60        
_________________________________________________________________
average_pooling2d_2 (Average (None, 13, 13, 6)         0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 11, 11, 16)        880       
_________________________________________________________________
average_pooling2d_3 (Average (None, 5, 5, 16)          0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 400)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 120)               48120     
_________________________________________________________________
dense_6 (Dense)              (None, 84)                10164     
_________________________________________________________________
dense_7 (Dense)              (None, 10)                850       
=================================================================
Total params: 60,074
Trainable params: 60,074
Non-trainable params: 0
_________________________________________________________________
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-13-a8d7acc7e26a> in <module>
     11 
     12 model2_increase_lr.summary()
---> 13 model2_increase_lr.compile(optimizer = tf.keras.optimizers.adam(learn
ing_rate=0.01)
     14 , loss = "sparse_categorical_crossentropy", metrics = ["accuracy"])
     15 

~/.local/lib/python3.6/site-packages/tensorflow/python/util/deprecation_wrapp
er.py in __getattr__(self, name)
    104     if name.startswith('_dw_'):
    105       raise AttributeError('Accessing local variables before they are
created.')
--> 106     attr = getattr(self._dw_wrapped_module, name)
    107     if (self._dw_warning_count < _PER_MODULE_WARNING_LIMIT and
    108         name not in self._dw_deprecated_printed):

AttributeError: module 'tensorflow.python.keras.api._v1.keras.optimizers' has
no attribute 'adam'
```

In [ ]:
```
model2_decrease_lr = keras.Sequential([
        layers.Conv2D(filters=6, kernel_size=(3, 3), activation='relu', inpu
t_shape=(28,28,1)),
        layers.AveragePooling2D(),
        layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'),
        layers.AveragePooling2D(),
        layers.Flatten(),
        layers.Dense(units=120, activation='relu'),
        layers.Dense(units=84, activation='relu'),
        layers.Dense(units=10, activation = 'softmax')
    ])

model2_decrease_lr.compile(optimizer = tf.keras.optimizers.adam(learning_rat
e=0.0001)
, loss = "sparse_categorical_crossentropy", metrics = ["accuracy"])

model2_decrease_lr.fit(x_train, y_train, epochs = 4)

accuracy(model2_decrease_lr, x_test, y_test)
roc_curve(model2_decrease_lr, x_test, y_test)
precision_recall(model2_decrease_lr, x_test, y_test)
```

The vanilla second model had an accuracy of about 0.9875. By increasing the learning rate times 10 the accuracy drops slightly to about 0.9839 . When decreasing the learning rate through a division of 10 the accuracy also drops, but now even worse to about 0.9646 .

Let us take a look at the different curves. The vanilla version had an almost perfect precision-recall curve, where half of the labels were labeled correctly all the time and the other labels 99% of the times. Yet both increasing and decreasing the learning rate deteriorates the good results by serveral percentages. Here again the decreasing model performs worse then the increasing model.

TODO: ROC Curve

When the learning rate is too large, then the algorithm might overshoot its goal, but when the learning rate is too small it might never reach the local minimum (or at least take to much time). Our initial learning rate seems to lie in the desired interval, since increasing or decreasing the learning rare reduces performance. The interval is still quite large: from 0.01 to 0.0001. An imporovement to the model would be to further narrow down this interval to optimise the choice for the learning rate.

Batch Size

Batch size describe the number of datapoints used in gradient descent, a larger batch size can help you train the model more quickly however at the cost of accuracy. A smaller batch size causes more noise but can reduce generalization error thus have a higher accuracy. Another adventage is that when using GPU, smaller batch size can allow paralle execution since a smaller batch can fit easier in the memory unit.

In [3]:
```
#batch size is defaut to 32

model2_batch_size3 = keras.Sequential([
        layers.Conv2D(filters=6, kernel_size=(3, 3), activation='relu', inpu
t_shape=(28,28,1)),
        layers.AveragePooling2D(),
        layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'),
        layers.AveragePooling2D(),
        layers.Flatten(),
        layers.Dense(units=120, activation='relu'),
        layers.Dense(units=84, activation='relu'),
        layers.Dense(units=10, activation = 'softmax')
    ])


model2_batch_size3.summary()
model2_batch_size3.compile(optimizer = "adam", loss = "sparse_categorical_cr
ossentropy",
                metrics = ["accuracy"])

# x_train = np.pad(x_train, ((0,0),(2,2),(2,2),(0,0)), 'constant')
model2_batch_size3.fit(x_train, y_train, batch_size=8, epochs = 4)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 26, 26, 6)         60        
_________________________________________________________________
average_pooling2d (AveragePo (None, 13, 13, 6)         0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 16)        880       
_________________________________________________________________
average_pooling2d_1 (Average (None, 5, 5, 16)          0         
_________________________________________________________________
flatten (Flatten)            (None, 400)               0         
_________________________________________________________________
dense (Dense)                (None, 120)               48120     
_________________________________________________________________
dense_1 (Dense)              (None, 84)                10164     
_________________________________________________________________
dense_2 (Dense)              (None, 10)                850       
=================================================================
Total params: 60,074
Trainable params: 60,074
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples
Epoch 1/4
60000/60000 [==============================] - 25s 418us/sample - loss: 0.175
0 - accuracy: 0.9466
Epoch 2/4
60000/60000 [==============================] - 25s 414us/sample - loss: 0.066
0 - accuracy: 0.9796
Epoch 3/4
60000/60000 [==============================] - 25s 414us/sample - loss: 0.045
6 - accuracy: 0.9854
Epoch 4/4
60000/60000 [==============================] - 25s 410us/sample - loss: 0.034
6 - accuracy: 0.9887
```

Out[3]:
```
<tensorflow.python.keras.callbacks.History at 0x1616441d0>
```

In [4]:
```
model2_batch_size2 = keras.Sequential([
        layers.Conv2D(filters=6, kernel_size=(3, 3), activation='relu', inpu
t_shape=(28,28,1)),
        layers.AveragePooling2D(),
        layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'),
        layers.AveragePooling2D(),
        layers.Flatten(),
        layers.Dense(units=120, activation='relu'),
        layers.Dense(units=84, activation='relu'),
        layers.Dense(units=10, activation = 'softmax')
    ])


model2_batch_size2.summary()
model2_batch_size2.compile(optimizer = "adam", loss = "sparse_categorical_cr
ossentropy",
                metrics = ["accuracy"])

# x_train = np.pad(x_train, ((0,0),(2,2),(2,2),(0,0)), 'constant')
model2_batch_size2.fit(x_train, y_train, batch_size=16, epochs = 4)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 26, 26, 6)         60
_________________________________________________________________
average_pooling2d_2 (Average (None, 13, 13, 6)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 11, 11, 16)        880
_________________________________________________________________
average_pooling2d_3 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_3 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_4 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_5 (Dense)              (None, 10)                850
=================================================================
Total params: 60,074
Trainable params: 60,074
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples
Epoch 1/4
60000/60000 [==============================] - 16s 271us/sample - loss: 0.187
9 - accuracy: 0.9420
Epoch 2/4
60000/60000 [==============================] - 16s 265us/sample - loss: 0.065
8 - accuracy: 0.9794
Epoch 3/4
60000/60000 [==============================] - 16s 265us/sample - loss: 0.046
6 - accuracy: 0.9859
Epoch 4/4
60000/60000 [==============================] - 16s 265us/sample - loss: 0.035
9 - accuracy: 0.9889
```

Out[4]:
```
<tensorflow.python.keras.callbacks.History at 0x10d565150>
```

In [5]:
```
model2_batch_size = keras.Sequential([
        layers.Conv2D(filters=6, kernel_size=(3, 3), activation='relu', inpu
t_shape=(28,28,1)),
        layers.AveragePooling2D(),
        layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'),
        layers.AveragePooling2D(),
        layers.Flatten(),
        layers.Dense(units=120, activation='relu'),
        layers.Dense(units=84, activation='relu'),
        layers.Dense(units=10, activation = 'softmax')
    ])


model2_batch_size.summary()
model2_batch_size.compile(optimizer = "adam", loss = "sparse_categorical_cro
ssentropy",
                metrics = ["accuracy"])

# x_train = np.pad(x_train, ((0,0),(2,2),(2,2),(0,0)), 'constant')
model2_batch_size.fit(x_train, y_train, batch_size=64, epochs = 4)
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_4 (Conv2D)            (None, 26, 26, 6)         60        
_________________________________________________________________
average_pooling2d_4 (Average (None, 13, 13, 6)         0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 11, 11, 16)        880       
_________________________________________________________________
average_pooling2d_5 (Average (None, 5, 5, 16)          0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 400)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 120)               48120     
_________________________________________________________________
dense_7 (Dense)              (None, 84)                10164     
_________________________________________________________________
dense_8 (Dense)              (None, 10)                850       
=================================================================
Total params: 60,074
Trainable params: 60,074
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples
Epoch 1/4
60000/60000 [==============================] - 8s 125us/sample - loss: 0.3139
- accuracy: 0.9053
Epoch 2/4
60000/60000 [==============================] - 7s 120us/sample - loss: 0.0998
- accuracy: 0.9688
Epoch 3/4
60000/60000 [==============================] - 7s 121us/sample - loss: 0.0701
- accuracy: 0.9780
Epoch 4/4
60000/60000 [==============================] - 7s 117us/sample - loss: 0.0546
- accuracy: 0.9830
```

Out[5]:
```
<tensorflow.python.keras.callbacks.History at 0x143dd70d0>
```

In [6]:
```
model2_batch_size4 = keras.Sequential([
        layers.Conv2D(filters=6, kernel_size=(3, 3), activation='relu', inpu
t_shape=(28,28,1)),
        layers.AveragePooling2D(),
        layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'),
        layers.AveragePooling2D(),
        layers.Flatten(),
        layers.Dense(units=120, activation='relu'),
        layers.Dense(units=84, activation='relu'),
        layers.Dense(units=10, activation = 'softmax')
    ])


model2_batch_size4.summary()
model2_batch_size4.compile(optimizer = "adam", loss = "sparse_categorical_cr
ossentropy",
                metrics = ["accuracy"])

# x_train = np.pad(x_train, ((0,0),(2,2),(2,2),(0,0)), 'constant')
model2_batch_size4.fit(x_train, y_train, batch_size=128, epochs = 4)
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_6 (Conv2D)            (None, 26, 26, 6)         60        
_________________________________________________________________
average_pooling2d_6 (Average (None, 13, 13, 6)         0         
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 11, 11, 16)        880       
_________________________________________________________________
average_pooling2d_7 (Average (None, 5, 5, 16)          0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 400)               0         
_________________________________________________________________
dense_9 (Dense)              (None, 120)               48120     
_________________________________________________________________
dense_10 (Dense)             (None, 84)                10164     
_________________________________________________________________
dense_11 (Dense)             (None, 10)                850       
=================================================================
Total params: 60,074
Trainable params: 60,074
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples
Epoch 1/4
60000/60000 [==============================] - 6s 98us/sample - loss: 0.4148
- accuracy: 0.8827
Epoch 2/4
60000/60000 [==============================] - 6s 93us/sample - loss: 0.1197
- accuracy: 0.9642
Epoch 3/4
60000/60000 [==============================] - 6s 94us/sample - loss: 0.0855
- accuracy: 0.9740
Epoch 4/4
60000/60000 [==============================] - 6s 94us/sample - loss: 0.0691
- accuracy: 0.9794
```

Out[6]:
```
<tensorflow.python.keras.callbacks.History at 0x132e44f10>
```

Number of epochs

An epoch refers to one full cycle through the training data. Until this point we have been training with four epochs. This means that the classifier looks at every training example four times in total to train the network. As seen above when fitting the network, we get the following output:

```
Epoch 1/4
60000/60000 [==============================] - 20s 338us/sample - loss: 0.2497 - ac
c: 0.9249
Epoch 2/4
60000/60000 [==============================] - 19s 311us/sample - loss: 0.0817 - ac
c: 0.9747
Epoch 3/4
60000/60000 [==============================] - 21s 343us/sample - loss: 0.0558 - ac
c: 0.9823
Epoch 4/4
60000/60000 [==============================] - 20s 334us/sample - loss: 0.0430 - ac
c: 0.9865
```

Notice how each epoch requires about the same amount of time, but the more epochs you use, the more time it requires to train the network. Also notice how the loss decreases and the accuracy increases with each extra epoch. This reveals a certain trade-off; accuracy or shorter training time. Of course we cannot keep increasing the amount of epochs for ever and expect an continuing increase of the accuracy. At a ceratain point the loss will start to increase because of overfitting. Let us look for the amount of epochs where the loss starts to rise again. This is the optimal in terms of loss, since it is the local minimum.

Now to save you the time of running the blow 15 epochs, the results will be displayed after the code.

In [ ]:
```
model2_epochs = keras.Sequential([
        layers.Conv2D(filters=6, kernel_size=(3, 3), activation='relu', inpu
t_shape=(28,28,1)),
        layers.AveragePooling2D(),
        layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'),
        layers.AveragePooling2D(),
        layers.Flatten(),
        layers.Dense(units=120, activation='relu'),
        layers.Dense(units=84, activation='relu'),
        layers.Dense(units=10, activation = 'softmax')
    ])


model2_epochs.compile(optimizer = "adam", loss = "sparse_categorical_crossen
tropy",
                metrics = ["accuracy"])

# x_train = np.pad(x_train, ((0,0),(2,2),(2,2),(0,0)), 'constant')
model2_epochs.fit(x_train, y_train, epochs = 15)
```

```
Epoch 1/30
60000/60000 [==============================] - 26s 438us/sample - loss: 0.2179 - ac
c: 0.9363
Epoch 2/30
60000/60000 [==============================] - 23s 390us/sample - loss: 0.0717 - ac
c: 0.9779
Epoch 3/30
60000/60000 [==============================] - 24s 396us/sample - loss: 0.0518 - ac
c: 0.9840
Epoch 4/30
60000/60000 [==============================] - 24s 401us/sample - loss: 0.0416 - ac
c: 0.9869
Epoch 5/30
60000/60000 [==============================] - 29s 483us/sample - loss: 0.0340 - ac
c: 0.9895
Epoch 6/30
60000/60000 [==============================] - 27s 456us/sample - loss: 0.0283 - ac
c: 0.9912
Epoch 7/30
60000/60000 [==============================] - 28s 460us/sample - loss: 0.0229 - ac
c: 0.9926
Epoch 8/30
60000/60000 [==============================] - 28s 468us/sample - loss: 0.0195 - ac
c: 0.9941
Epoch 9/30
60000/60000 [==============================] - 34s 560us/sample - loss: 0.0179 - ac
c: 0.9942
Epoch 10/30
60000/60000 [==============================] - 39s 643us/sample - loss: 0.0151 - ac
c: 0.9953
Epoch 11/30
60000/60000 [==============================] - 33s 549us/sample - loss: 0.0143 - ac
c: 0.9953
Epoch 12/30
60000/60000 [==============================] - 33s 554us/sample - loss: 0.0106 - ac
c: 0.9964
Epoch 13/30
60000/60000 [==============================] - 29s 490us/sample - loss: 0.0108 - ac
c: 0.9963
```

After 12 epochs the loss show a small increase. So the optimal amount of epochs for this CNN is 12. Now when appling more epochs we see that the loss starts to fluctuate and decreases gently. This is expected, but overfitting makes the classifier unreliable.

Note that running this code again might result in some different values. Yet the same reasoning holds.

Kernel size

In this section, we would like to see the results of varied kernel sizes on our model. As seen from the roiginial second model with kernel sizes (3, 3) at each layer, we get an accuracy of 0.984 on the test set as defined below:

```
10000/10000 [==============================] - 2s 159us/sample - loss: 0.0459 - ac
c: 0.9841
test loss is : 0.04586207606535172 - test accuracy is : 0.9840999841690063
```

We set the first layer of the network with kernel size of (5, 5) and the second remains same at (3, 3) and we get an accuracy of 0.983 on the test set as shown below. This is an improvement on the original model with both kernels at (3, 3).

```
10000/10000 [==============================] - 3s 307us/sample - loss: 0.0525 - acc
uracy: 0.9835
test loss is : 0.05253282631125767 - test accuracy is : 0.9835000038146973
```

Changing the kernal size of first layer to (6, 6) gives an accuracy of 0.986 and (2, 2) gives an accuracy of 0.986.

```
10000/10000 [==============================] - 3s 339us/sample - loss: 0.0408 - acc
uracy: 0.9862
test loss is : 0.0407773371128249 - test accuracy is : 0.9861999750137329
```

In [16]:

```
model2 = keras.Sequential([
        layers.Conv2D(filters=6, kernel_size=(5, 5), activation='relu', inpu
t_shape=(28,28,1)),
        layers.AveragePooling2D(),
        layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'),
        layers.AveragePooling2D(),
        layers.Flatten(),
        layers.Dense(units=120, activation='relu'),
        layers.Dense(units=84, activation='relu'),
        layers.Dense(units=10, activation = 'softmax')
    ])


#model2.summary()
model2.compile(optimizer = "adam", loss = "sparse_categorical_crossentropy",
                metrics = ["accuracy"])

# x_train = np.pad(x_train, ((0,0),(2,2),(2,2),(0,0)), 'constant')
model2.fit(x_train, y_train, epochs = 4)
# Accuracy:
accuracy(model2, x_test, y_test)
```

```
Train on 60000 samples
Epoch 1/4
60000/60000 [==============================] - 131s 2ms/sample - loss: 0.2081
- accuracy: 0.9390
Epoch 2/4
60000/60000 [==============================] - 141s 2ms/sample - loss: 0.0692
- accuracy: 0.9790
Epoch 3/4
60000/60000 [==============================] - 146s 2ms/sample - loss: 0.0496
- accuracy: 0.9854
Epoch 4/4
60000/60000 [==============================] - 143s 2ms/sample - loss: 0.0382
- accuracy: 0.9880
10000/10000 [==============================] - 4s 376us/sample - loss: 0.0328
- accuracy: 0.9896
test loss is : 0.03284961047746474 - test accuracy is : 0.9896000027656555
```

we can see changing the kernel size alone does not significantly increase the accuracy of the model

NOTE

rerunning th above code might give different results

In [ ]:

```

```

In [ ]:

```

```