

## ▼ Capstone Project

### Image classifier for the SVHN dataset

#### Instructions

In this notebook, you will create a neural network that classifies real-world images digits. You will use concepts from throughout this course in building, training, testing, validating and saving your Tensorflow classifier model.

This project is peer-assessed. Within this notebook you will find instructions in each section for how to complete the project. Pay close attention to the instructions as the peer review will be carried out according to a grading rubric that checks key parts of the project instructions. Feel free to add extra cells into the notebook as required.


#### How to submit

When you have completed the Capstone project notebook, you will submit a pdf of the notebook for peer review. First ensure that the notebook has been fully executed from beginning to end, and all of the cell outputs are visible. This is important, as the grading rubric depends on the reviewer being able to view the outputs of your notebook. Save the notebook as a pdf (File -> Download as -> PDF via LaTeX). You should then submit this pdf for review.

#### Let's get started!

We'll start by running some imports, and loading the dataset. For this project you are free to make further imports throughout the notebook as you wish.

```
import tensorflow as tf
import numpy as np
from scipy.io import loadmat
```

 SVHN overview image For the capstone project, you will use the [SVHN dataset](#). This is an image dataset of over 600,000 digit images in all, and is a harder dataset than MNIST as the numbers appear in the context of natural scene images. SVHN is obtained from house numbers in Google Street View images.

- Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu and A. Y. Ng. "Reading Digits in Natural Images with Unsupervised Feature Learning". NIPS Workshop on Deep Learning and Unsupervised Feature Learning, 2011.

Your goal is to develop an end-to-end workflow for building, training, validating, evaluating and saving a neural network that classifies a real-world image into one of ten classes.

```
# Run this cell to load the dataset
```

```
train = loadmat('data/train_32x32.mat')
test = loadmat('data/test_32x32.mat')
```

Both `train` and `test` are dictionaries with keys `x` and `y` for the input images and labels respectively.

## ▼ 1. Inspect and preprocess the dataset

- Extract the training and testing images and labels separately from the `train` and `test` dictionaries loaded for you.
- Select a random sample of images and corresponding labels from the dataset (at least 10), and display them in a figure.
- Convert the training and test images to grayscale by taking the average across all colour channels for each pixel. *Hint: retain the channel dimension, which will now have size 1.*
- Select a random sample of the grayscale images and corresponding labels from the dataset (at least 10), and display them in a figure.

```
train_x = train['X']
train_y = train['y']
```

```
test_x = test['X']
test_y = test['y']
```

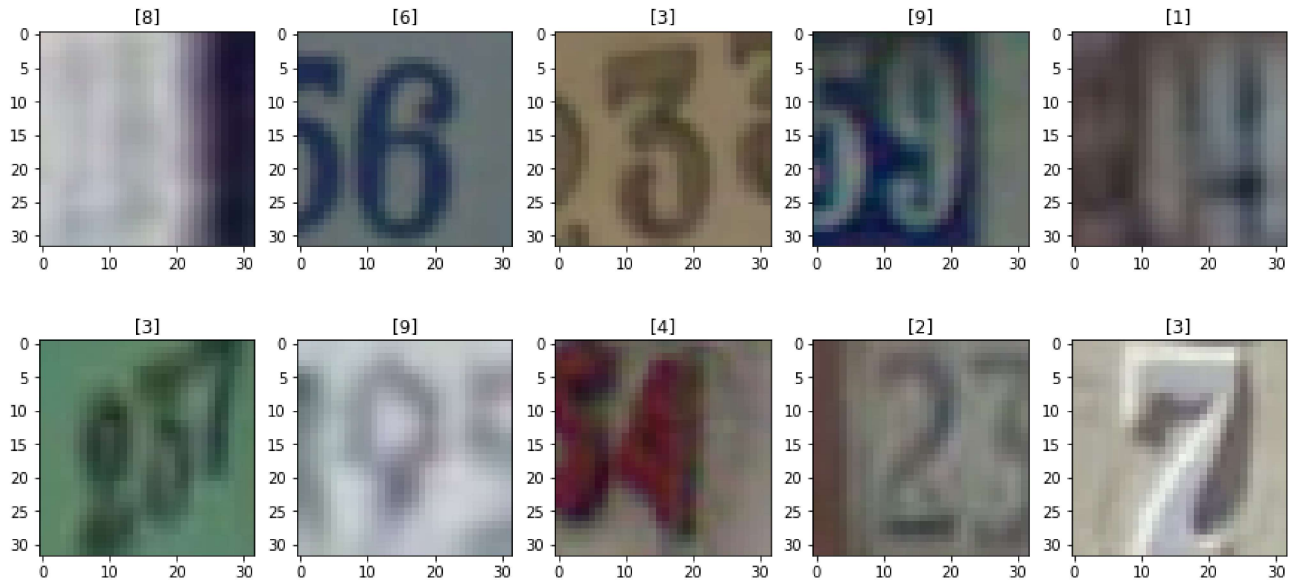
```
train_x = np.moveaxis(train_x, -1, 0)
test_x = np.moveaxis(test_x, -1, 0)
```

```
import random
images = []
for i in range(10):
    images.append(random.randint(0,train_x.shape[0]))
images
```



[1940, 45289, 72026, 843, 18906, 65886, 45857, 40519, 43121, 32137]

```
import matplotlib.pyplot as plt
%matplotlib inline
fig,a = plt.subplots(2,5,figsize = (15,7))
for i in range(5):
    a[0][i].imshow(train_x[images[i],:,:], cmap=plt.get_cmap('gray'))
    a[0][i].set_title(train_y[images[i]])
    a[1][i].imshow(train_x[images[i+5],:,:], cmap=plt.get_cmap('gray'))
    a[1][i].set_title(train_y[images[i+5]])
plt.show()
```



```
train_x = np.mean(train_x,axis = -1)
test_x = np.mean(test_x,axis = -1)
```

```
train_x = train_x.reshape(train_x.shape[0],train_x.shape[1],train_x.shape[2],1)
test_x = test_x.reshape(test_x.shape[0],test_x.shape[1],test_x.shape[2],1)
```

```
train_y = train_y - 1
test_y = test_y - 1
train_y = tf.keras.utils.to_categorical(train_y,num_classes = 10)
test_y = tf.keras.utils.to_categorical(test_y,num_classes = 10)
```

```
train_y
```



```
array([[1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       ...,
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.]], dtype=float32)
```

```
images = []
for i in range(10):
    images.append(random.randint(0,test_x.shape[0]))
images
```



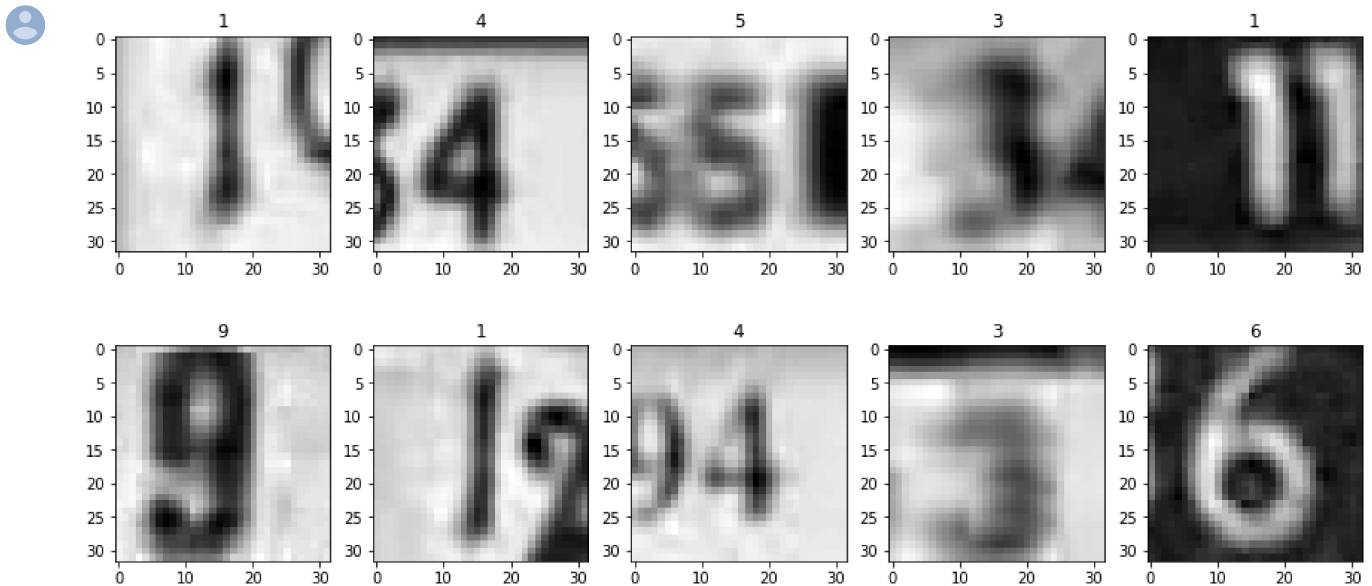
```
[2460, 16196, 623, 7177, 21941, 22531, 23312, 12518, 22209, 11740]
```

```
import matplotlib.pyplot as plt
%matplotlib inline
fig1.a1 = plt.subplots(2.5,figsize = (15,7))
```

```

for i in range(5):
    a1[0][i].imshow(test_x[images[i],:,:,0], cmap=plt.get_cmap('gray'))
    a1[0][i].set_title(np.argmax(test_y[images[i]])+1)
    a1[1][i].imshow(test_x[images[i+5],:,:,0], cmap=plt.get_cmap('gray'))
    a1[1][i].set_title(np.argmax(test_y[images[i+5]])+1)
plt.show()

```



## 2. MLP neural network classifier

- Build an MLP classifier model using the Sequential API. Your model should use only Flatten and Dense layers, with the final layer having a 10-way softmax output.
- You should design and build the model yourself. Feel free to experiment with different MLP architectures. *Hint: to achieve a reasonable accuracy you won't need to use more than 4 or 5 layers.*
- Print out the model summary (using the summary() method)
- Compile and train the model (we recommend a maximum of 30 epochs), making use of both training and validation sets during the training run.
- Your model should track at least one appropriate metric, and use at least two callbacks during training, one of which should be a ModelCheckpoint callback.
- As a guide, you should aim to achieve a final categorical cross entropy training loss of less than 1.0 (the validation loss might be higher).
- Plot the learning curves for loss vs epoch and accuracy vs epoch for both training and validation sets.
- Compute and display the loss and accuracy of the trained model on the test set.

```
train x = train x/255.0
```

```
test_x = test_x/255.0
```

```
def createMLP():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Flatten(input_shape = (32,32,1)),
        tf.keras.layers.Dense(64,activation = 'relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dense(128,activation = 'relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dense(256,activation = 'relu'),
        tf.keras.layers.Dense(256,activation = 'relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dense(10,activation = 'softmax')
    ])
    return model

model = createMLP()
model.compile(loss = 'categorical_crossentropy',optimizer = 'adam',metrics = ['accuracy'])
model.summary()
```



Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 64)	65600
batch_normalization (Batch Normalization)	(None, 64)	256
dense_1 (Dense)	(None, 128)	8320
batch_normalization_1 (Batch Normalization)	(None, 128)	512
dense_2 (Dense)	(None, 256)	33024
dense_3 (Dense)	(None, 256)	65792
batch_normalization_2 (Batch Normalization)	(None, 256)	1024
dense_4 (Dense)	(None, 10)	2570
Total params: 177,098		
Trainable params: 176,202		
Non-trainable params: 896		

```
call1 = tf.keras.callbacks.EarlyStopping(monitor = 'val_accuracy',patience = 5)
call2 = tf.keras.callbacks.ModelCheckpoint('checkpoints_best_onlyMLP',save_weights_only =

history = model.fit(train_x,train_y,epochs = 30,validation_split = 0.2,callbacks = [call1,
print(history.epoch,history.history['accuracy'][-1])
```



Train on 58605 samples, validate on 14652 samples

Epoch 1/30

58605/58605 [=====] - 62s 1ms/sample - loss: 1.3672 - accuracy: 0.7685

Epoch 2/30

58605/58605 [=====] - 58s 986us/sample - loss: 1.0242 - accuracy: 0.8255

Epoch 3/30

58605/58605 [=====] - 58s 986us/sample - loss: 0.9176 - accuracy: 0.8561

Epoch 4/30

58605/58605 [=====] - 58s 988us/sample - loss: 0.8561 - accuracy: 0.8775

Epoch 5/30

58605/58605 [=====] - 57s 979us/sample - loss: 0.8255 - accuracy: 0.8918

Epoch 6/30

58605/58605 [=====] - 58s 983us/sample - loss: 0.7918 - accuracy: 0.9176

Epoch 7/30

58605/58605 [=====] - 58s 984us/sample - loss: 0.7775 - accuracy: 0.9325

Epoch 8/30

58605/58605 [=====] - 57s 978us/sample - loss: 0.7618 - accuracy: 0.9429

Epoch 9/30

58605/58605 [=====] - 57s 981us/sample - loss: 0.7429 - accuracy: 0.9518

Epoch 10/30

58605/58605 [=====] - 57s 978us/sample - loss: 0.7325 - accuracy: 0.9581

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9] 0.768501

```
import matplotlib.image as mpimg
```

```
import matplotlib.pyplot as plt
```

```
acc=history.history['accuracy']
```

```
val_acc=history.history['val_accuracy']
```

```
loss=history.history['loss']
```

```
val_loss=history.history['val_loss']
```

```
epochs=range(len(acc)) # Get number of epochs
```

```
plt.plot(epochs, acc, 'r', "Training Accuracy")
```

```
plt.plot(epochs, val_acc, 'b', "Validation Accuracy")
```

```
plt.title('Training and validation accuracy')
```

```
plt.figure()
```

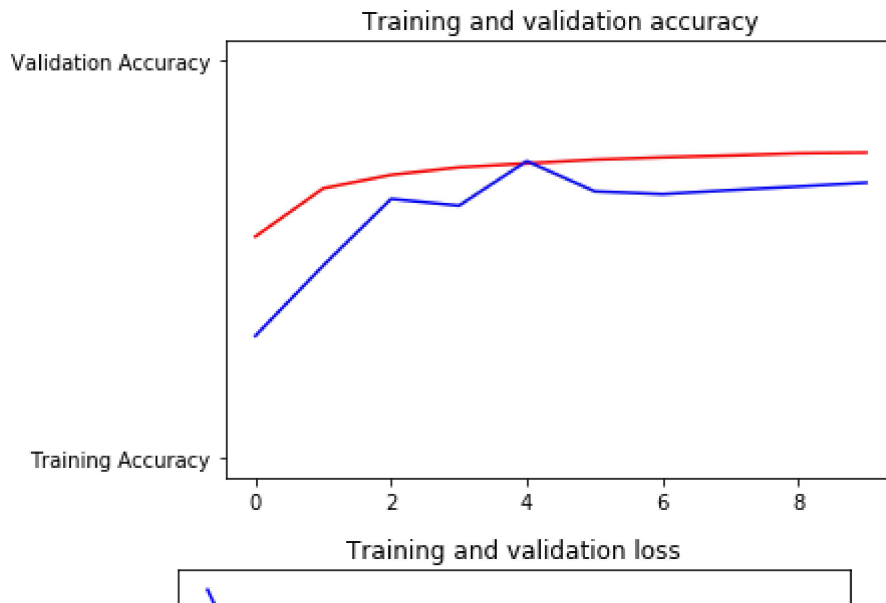
```
plt.plot(epochs, loss, 'r', "Training Loss")
```

```
plt.plot(epochs, val_loss, 'b', "Validation Loss")
```

```
plt.title('Training and validation loss')
```



```
Text(0.5, 1.0, 'Training and validation loss')
```



```
loss,acc = model.evaluate(test_x,test_y)
```

```
26032/1 [=====
```

```
|  \  |
print('loss = ',loss)
```

```
loss = 1.0749247797662376
```

```
Training Loss |
print('acc = ',acc)
```

```
acc = 0.6663722
```

### 3. CNN neural network classifier

- Build a CNN classifier model using the Sequential API. Your model should use the Conv2D, MaxPool2D, BatchNormalization, Flatten, Dense and Dropout layers. The final layer should again have a 10-way softmax output.
- You should design and build the model yourself. Feel free to experiment with different CNN architectures. *Hint: to achieve a reasonable accuracy you won't need to use more than 2 or 3 convolutional layers and 2 fully connected layers.*
- The CNN model should use fewer trainable parameters than your MLP model.
- Compile and train the model (we recommend a maximum of 30 epochs), making use of both training and validation sets during the training run.
- Your model should track at least one appropriate metric, and use at least two callbacks during training, one of which should be a ModelCheckpoint callback.
- You should aim to beat the MLP model performance with fewer parameters!
- Plot the learning curves for loss vs epoch and accuracy vs epoch for both training and validation sets.
- Compute and display the loss and accuracy of the trained model on the test set.

```
def createCNN():
    modelCNN = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(16,(2,2),activation = 'relu',input_shape = (32,32,1),padding = 'same'),
        tf.keras.layers.MaxPool2D(2,2),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Conv2D(32,(2,2),activation = 'relu',padding = 'same'),
        tf.keras.layers.MaxPool2D(2,2),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(64,activation = 'relu'),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(10,activation = 'softmax')
    ])
    return modelCNN
```

```
modelCNN = createCNN()
modelCNN.compile(loss = 'categorical_crossentropy',optimizer = 'adam',metrics = ['accuracy'])
modelCNN.summary()
```



Model: "sequential\_1"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 16)	80
max_pooling2d (MaxPooling2D)	(None, 16, 16, 16)	0
batch_normalization_3 (Batch Normalization)	(None, 16, 16, 16)	64
conv2d_1 (Conv2D)	(None, 16, 16, 32)	2080
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 32)	0
dropout (Dropout)	(None, 8, 8, 32)	0
batch_normalization_4 (Batch Normalization)	(None, 8, 8, 32)	128
flatten_1 (Flatten)	(None, 2048)	0
dense_5 (Dense)	(None, 64)	131136
dropout_1 (Dropout)	(None, 64)	0
dense_6 (Dense)	(None, 10)	650
Total params: 134,138		
Trainable params: 134,042		
Non-trainable params: 96		

```
call3 = tf.keras.callbacks.ModelCheckpoint('checkpoints_best_onlyCNN',save_weights_only = True)
class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('loss')<0.2):
            print("\nReached 20% loss so cancelling training!")
```



```

print('Reached 20% loss so cancelling training: ')
self.model.stop_training = True
call4 = myCallback()

```

```

history = modelCNN.fit(train_x,train_y,epochs = 30,validation_split = 0.2,callbacks = [cal
print(history.epoch,history.history['accuracy'][-1])

```



Train on 58605 samples, validate on 14652 samples

```

Epoch 1/30
58605/58605 [=====] - 233s 4ms/sample - loss: 1.0103 - accur
Epoch 2/30
58605/58605 [=====] - 216s 4ms/sample - loss: 0.5900 - accur
Epoch 3/30
58605/58605 [=====] - 221s 4ms/sample - loss: 0.5138 - accur
Epoch 4/30
58605/58605 [=====] - 215s 4ms/sample - loss: 0.4744 - accur
Epoch 5/30
58605/58605 [=====] - 219s 4ms/sample - loss: 0.4432 - accur
Epoch 6/30
58605/58605 [=====] - 228s 4ms/sample - loss: 0.4209 - accur
Epoch 7/30
58605/58605 [=====] - 226s 4ms/sample - loss: 0.4057 - accur
Epoch 8/30
58605/58605 [=====] - 225s 4ms/sample - loss: 0.3844 - accur
Epoch 9/30
58605/58605 [=====] - 230s 4ms/sample - loss: 0.3703 - accur
Epoch 10/30
58605/58605 [=====] - 225s 4ms/sample - loss: 0.3651 - accur
Epoch 11/30
58605/58605 [=====] - 223s 4ms/sample - loss: 0.3541 - accur
Epoch 12/30
58605/58605 [=====] - 221s 4ms/sample - loss: 0.3432 - accur
Epoch 13/30
58605/58605 [=====] - 222s 4ms/sample - loss: 0.3379 - accur
Epoch 14/30
58605/58605 [=====] - 222s 4ms/sample - loss: 0.3328 - accur
Epoch 15/30
58605/58605 [=====] - 224s 4ms/sample - loss: 0.3309 - accur
Epoch 16/30
58605/58605 [=====] - 227s 4ms/sample - loss: 0.3150 - accur
Epoch 17/30
58605/58605 [=====] - 212s 4ms/sample - loss: 0.3183 - accur
Epoch 18/30
58605/58605 [=====] - 220s 4ms/sample - loss: 0.3052 - accur
Epoch 19/30
58605/58605 [=====] - 231s 4ms/sample - loss: 0.3041 - accur
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18] 0.9035407

```

```

import matplotlib.image as mpimg
import matplotlib.pyplot as plt

```

```

acc=history.history['accuracy']
val_acc=history.history['val_accuracy']
loss=history.history['loss']
val_loss=history.history['val_loss']

```

```

epochs=range(len(acc)) # Get number of epochs

```

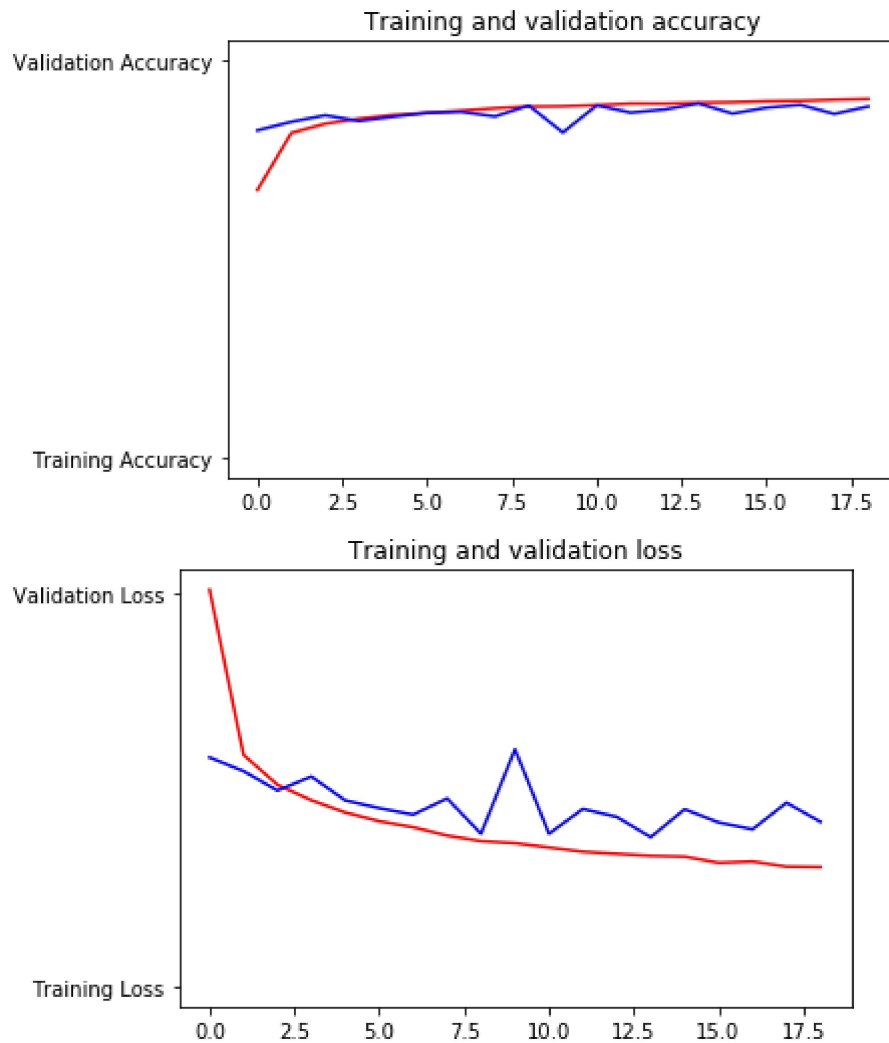
```
plt.plot(epochs, acc, 'r', "Training Accuracy")
plt.plot(epochs, val_acc, 'b', "Validation Accuracy")
plt.title('Training and validation accuracy')
plt.figure()
```

```
plt.plot(epochs, loss, 'r', "Training Loss")
plt.plot(epochs, val_loss, 'b', "Validation Loss")
```

```
plt.title('Training and validation loss')
```



```
Text(0.5, 1.0, 'Training and validation loss')
```



```
loss,acc = modelCNN.evaluate(test_x,test_y)
```



```
26032/1 [=====]
```

```
print('loss = ',loss)
```



```
loss = 0.46608391826366746
```

```
print('acc = ',acc)
```



```
acc = 0.8691776
```

## ▼ 4. Get model predictions

- Load the best weights for the MLP and CNN models that you saved during the training run.
- Randomly select 5 images and corresponding labels from the test set and display the images with their labels.
- Alongside the image and label, show each model's predictive distribution as a bar chart, and the final model prediction given by the label with maximum probability.

```
MLP = createMLP()
CNN = createCNN()
```

```
MLP.load_weights('checkpoints_best_onlyMLP')
CNN.load_weights('checkpoints_best_onlyCNN')
```

 <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7fde83c78898>

```
import random
images = []
for i in range(5):
    images.append(random.randint(0,test_x.shape[0]))
images
```

 [13633, 10433, 3624, 8136, 10313]

```
fig,a = plt.subplots(5,5,figsize=(20,20))
for i in range(5):
    a[i][0].imshow(test_x[images[i],:,:0], cmap = plt.get_cmap('gray'))
    a[i][0].set_title(np.argmax(test_y[images[i]])+1)
    img = test_x[images[i]]
    img = np.expand_dims(img,axis=0)
    predMLP = MLP.predict(img)
    predCNN = CNN.predict(img)
    a[i][1].bar(np.arange(1,11),predMLP[0],color='blue')
    a[i][1].set_title('MLP')
    a[i][2].text(0.5,0.5,np.argmax(predMLP)+1,fontsize = 30)
    a[i][3].bar(np.arange(1,11),predCNN[0],color='green')
    a[i][3].set_title('CNN')
    a[i][4].text(0.5,0.5,np.argmax(predCNN)+1,fontsize = 30)
plt.show()
```



