Capstone Project

April 17, 2021

1 Capstone Project

1.1 Image classifier for the SVHN dataset

1.1.1 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.

1.1.2 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.

1.1.3 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.

```
In [1]: import tensorflow as tf
        from scipy.io import loadmat
```

```
In [2]: from tensorflow.keras.preprocessing import image
        import matplotlib.pyplot as plt
        import numpy as np
        import pandas as pd
        import random
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Flatten, Softmax
        from tensorflow.keras.layers import Dropout, Conv2D, MaxPooling2D, BatchNormalization
```

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.

```
In [3]: # 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.

1.2 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.

```
In [4]: #Extract training and test images and labels
        print(f'Train keys: {train.keys()}')
        print(f'Test keys: {train.keys()}')
        x_train = train['X']
        y_train = train['y']

        x_test = test['X']
        y_test = test['y']
```

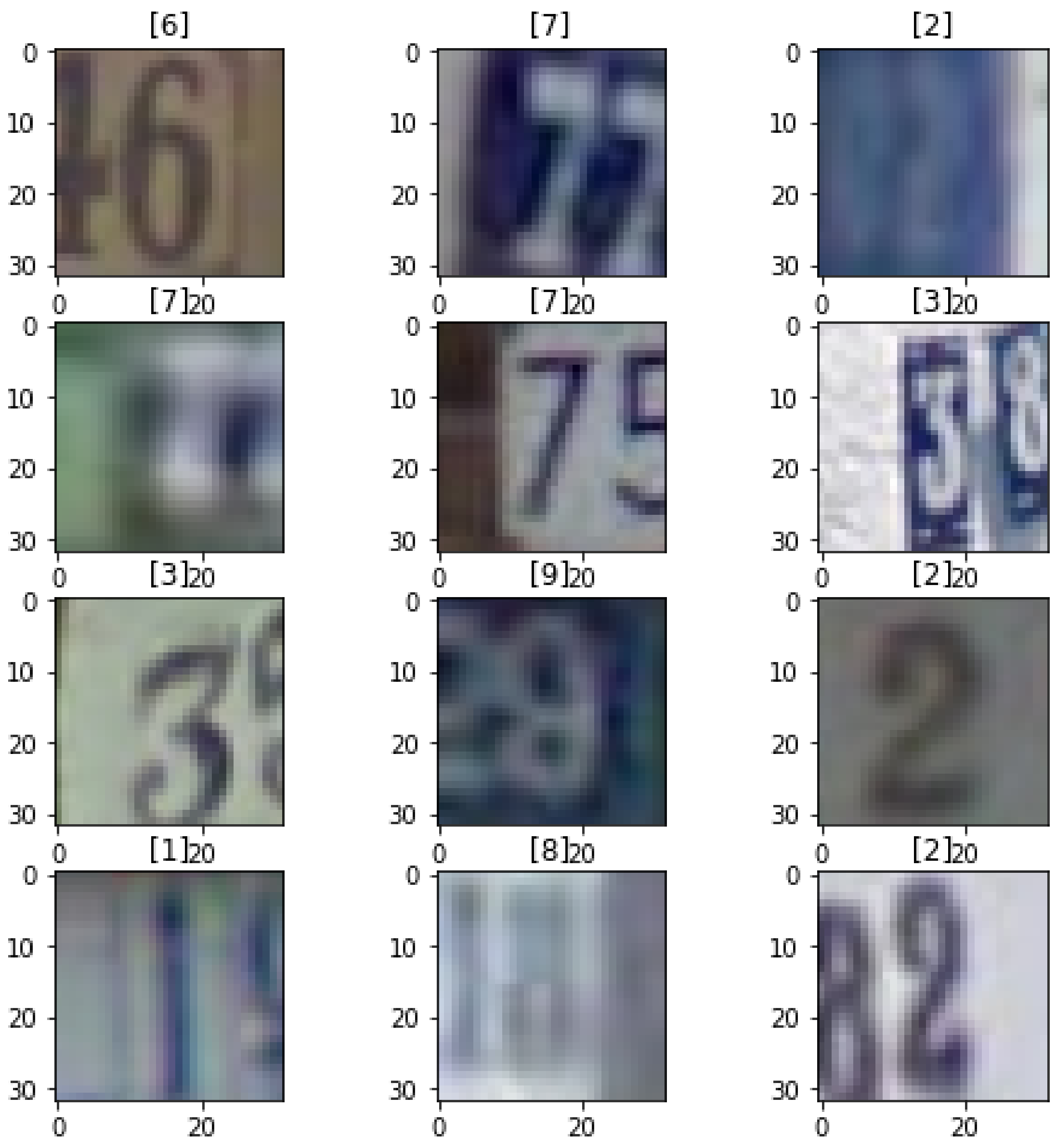
```
Train keys: dict_keys(['__header__', '__version__', '__globals__', 'X', 'y'])
Test keys: dict_keys(['__header__', '__version__', '__globals__', 'X', 'y'])
```

```
In [5]: print(f'Shape x : {x_train.shape}')

        y_train = np.where(y_train ==10 , 0, y_train)
        y_test = np.where(y_test ==10 , 0, y_test)

        print('Frequency of each label in training')
        unique, counts = np.unique(np.squeeze(y_train), return_counts=True)
        print (np.asarray((unique, counts)).T)

        print('Frequency of each label in testing')
        unique, counts = np.unique(np.squeeze(y_test), return_counts=True)
        print (np.asarray((unique, counts)).T)
```

```
Shape x : (32, 32, 3, 73257)
Frequency of each label in training
[[    0  4948]
 [    1 13861]
 [    2 10585]
 [    3  8497]
 [    4  7458]
 [    5  6882]
 [    6  5727]
 [    7  5595]
 [    8  5045]
 [    9  4659]]
Frequency of each label in testing
[[   0 1744]
 [   1 5099]
 [   2 4149]
 [   3 2882]
 [   4 2523]
 [   5 2384]
 [   6 1977]
 [   7 2019]
 [   8 1660]
 [   9 1595]]
```

```
In [6]: #Display a sample of images and its labels
        n_l = random.sample(range(1, len(y_train)), 12)
```

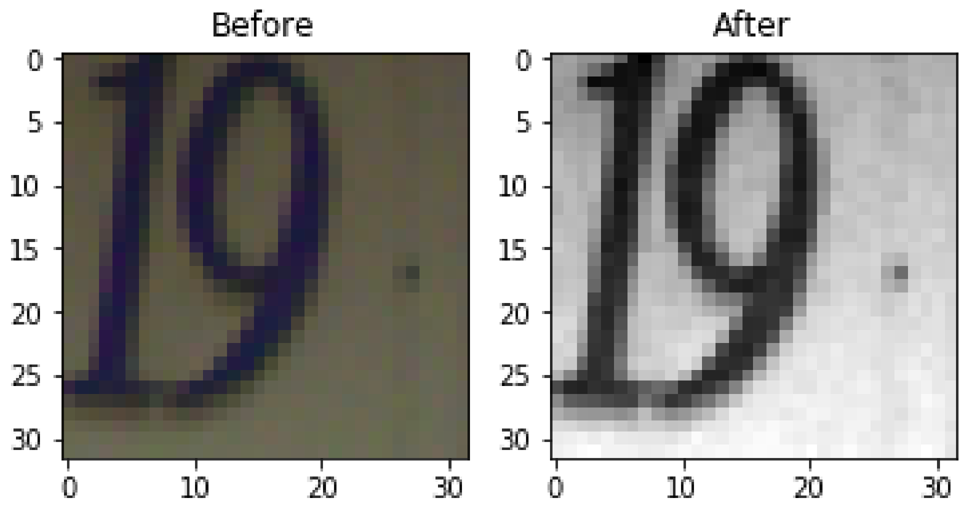
```
fig=plt.figure(figsize=(8, 8))
columns = 3
rows = 4
for i in range(1, columns*rows +1):
    img = x_train[:,:,:,n_l[i-1]]
    ax = fig.add_subplot(rows, columns, i)
    plt.imshow(img)
    ax.set_title(y_train[n_l[i-1]])
```



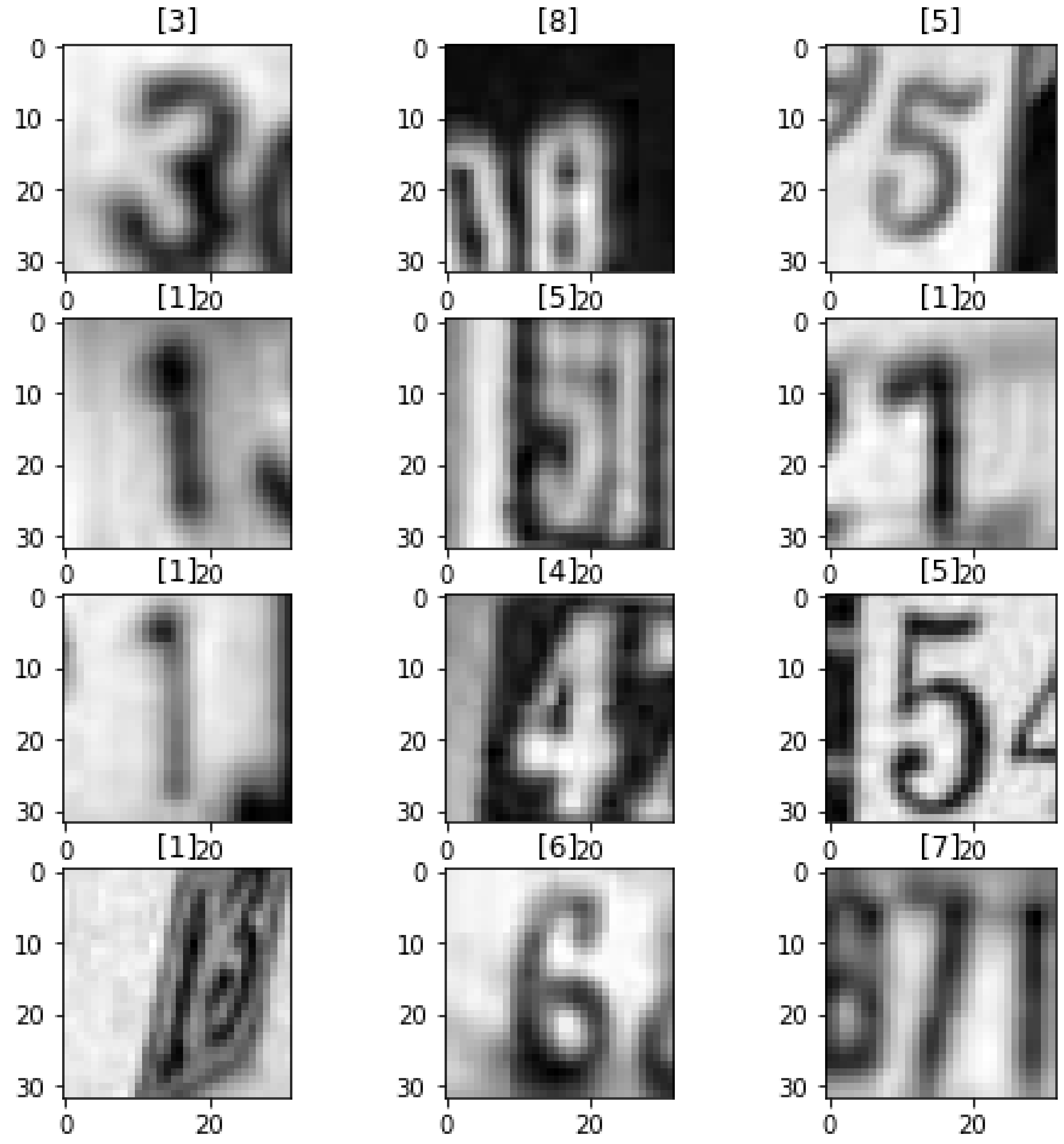
In [7]:
```
#Convert the training and test images to grayscale by taking the average across all co
x_train_grayscale = np.mean(x_train, axis=2, keepdims=True)
```

```
        x_test_grayscale = np.mean(x_test, axis=2, keepdims=True)
        fig = plt.figure()
        a = fig.add_subplot(1, 2, 1)
        imgplot = plt.imshow(x_train[:,:,:,1])
        a.set_title('Before')
        a = fig.add_subplot(1, 2, 2)
        imgplot = plt.imshow(np.squeeze(x_train_grayscale[:,:,:,1]),cmap='gray')
        a.set_title('After')

        print(x_train_grayscale.shape)
```

```
(32, 32, 1, 73257)
```



```
In [8]: #Display a sample of  grayscale images and its labels
        n_l = random.sample(range(1, len(y_train)), 12)
        fig=plt.figure(figsize=(8, 8))
        columns = 3
        rows = 4
        for i in range(1, columns*rows +1):
            img = np.squeeze(x_train_grayscale[:,:,:,n_l[i-1]])

            ax = fig.add_subplot(rows, columns, i)
            plt.imshow(img ,cmap='gray')
            ax.set_title(y_train[n_l[i-1]])
```



1.3 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.

```
In [9]: model = Sequential([
            Flatten(input_shape = x_train_grayscale[:,:,:,0].shape),
            Dense(64, activation='relu'),
            Dense(64, activation='relu'),
            Dense(64, activation='relu'),
            Dense(64, activation='relu'),
            Dense(64, activation='relu'),
            Dense(10, activation='softmax')
            ])
```

```
In [10]: #Print model summary
         model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 64)                65600
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_3 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_4 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_5 (Dense)              (None, 10)                650
=================================================================
Total params: 82,890
Trainable params: 82,890
Non-trainable params: 0
_________________________________________________________________
```

```
In [11]: from tensorflow.keras.callbacks import Callback
         from tensorflow.keras.callbacks import ModelCheckpoint
         #Define callbacks
         class myCallback(Callback):
             def on_train_begin(self, logs = None):
                 print('Starting training')
```

```
            def on_epoch_begin(self, epoch, logs = None):
                print(f'Starting epoch {epoch}')
            def on_epoch_end(self, epoch, logs = None):
                print(f'Ending epoch {epoch}')
            def on_train_end(self, logs = None):
                print('Finished Training')

        checkpoint_path = 'model_checkpoints/checkpoint'
        checkpoint = ModelCheckpoint(filepath = checkpoint_path,
                                     frequency = 'epoch',
                                     save_weights_only = True,
                                     verbose = 1,
                                     save_best_only = True)
```

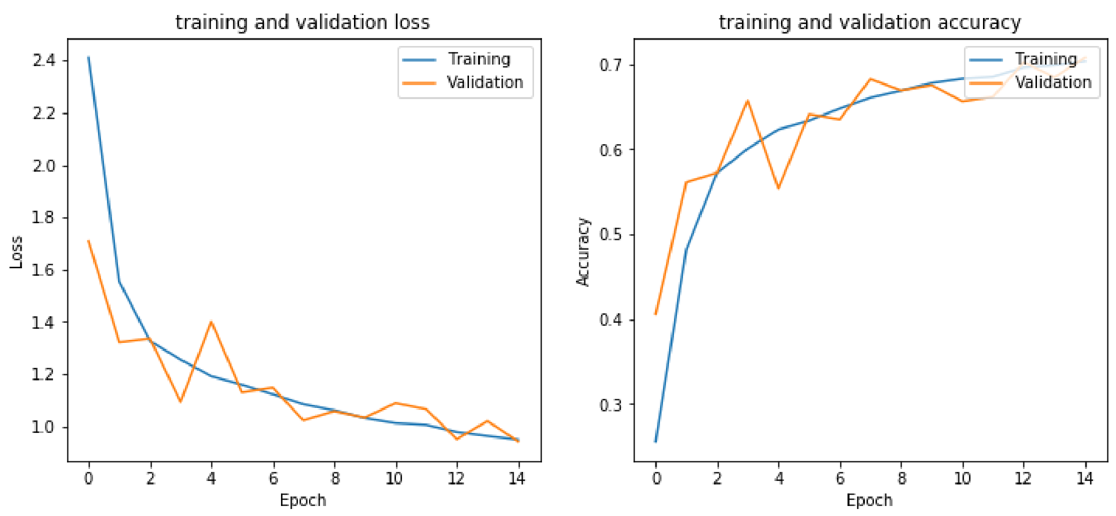
```
In [12]: #Compile the model
         model.compile(optimizer = 'adam',
                       loss = 'sparse_categorical_crossentropy',
                       metrics = ['accuracy'])

         #Change dimension to: (batch, h, w, c)

         history = model.fit(np.transpose(x_train_grayscale, (3, 0, 1, 2)),
                             y_train, epochs = 15, validation_split = 0.1, batch_size = 64,
                             verbose = 1 , callbacks=[myCallback(), tf.keras.callbacks.EarlySto
```

```
Train on 65931 samples, validate on 7326 samples
Starting training
Starting epoch 0
Epoch 1/15
65856/65931 [============================>.] - ETA: 0s - loss: 2.4121 - accuracy: 0.2556Ending

Epoch 00001: val_loss improved from inf to 1.70847, saving model to model_checkpoints/checkpoin
65931/65931 [==============================] - 24s 370us/sample - loss: 2.4113 - accuracy: 0.25
Starting epoch 1
Epoch 2/15
65664/65931 [============================>.] - ETA: 0s - loss: 1.5541 - accuracy: 0.4819Ending

Epoch 00002: val_loss improved from 1.70847 to 1.32175, saving model to model_checkpoints/check
65931/65931 [==============================] - 22s 332us/sample - loss: 1.5536 - accuracy: 0.48
Starting epoch 2
Epoch 3/15
65600/65931 [============================>.] - ETA: 0s - loss: 1.3274 - accuracy: 0.5719Ending

Epoch 00003: val_loss did not improve from 1.32175
65931/65931 [==============================] - 22s 337us/sample - loss: 1.3269 - accuracy: 0.57
Starting epoch 3
Epoch 4/15
65792/65931 [============================>.] - ETA: 0s - loss: 1.2550 - accuracy: 0.6005Ending
```

```
Epoch 00004: val_loss improved from 1.32175 to 1.09327, saving model to model_checkpoints/checl
65931/65931 [==============================] - 23s 342us/sample - loss: 1.2548 - accuracy: 0.60
Starting epoch 4
Epoch 5/15
65920/65931 [============================>.] - ETA: 0s - loss: 1.1918 - accuracy: 0.6234Ending

Epoch 00005: val_loss did not improve from 1.09327
65931/65931 [==============================] - 22s 331us/sample - loss: 1.1917 - accuracy: 0.62
Starting epoch 5
Epoch 6/15
65728/65931 [============================>.] - ETA: 0s - loss: 1.1584 - accuracy: 0.6339Ending

Epoch 00006: val_loss did not improve from 1.09327
65931/65931 [==============================] - 22s 334us/sample - loss: 1.1588 - accuracy: 0.63
Starting epoch 6
Epoch 7/15
65728/65931 [============================>.] - ETA: 0s - loss: 1.1234 - accuracy: 0.6480Ending

Epoch 00007: val_loss did not improve from 1.09327
65931/65931 [==============================] - 22s 341us/sample - loss: 1.1227 - accuracy: 0.64
Starting epoch 7
Epoch 8/15
65792/65931 [============================>.] - ETA: 0s - loss: 1.0845 - accuracy: 0.6610Ending

Epoch 00008: val_loss improved from 1.09327 to 1.02282, saving model to model_checkpoints/checl
65931/65931 [==============================] - 23s 342us/sample - loss: 1.0846 - accuracy: 0.66
Starting epoch 8
Epoch 9/15
65728/65931 [============================>.] - ETA: 0s - loss: 1.0611 - accuracy: 0.6694Ending

Epoch 00009: val_loss did not improve from 1.02282
65931/65931 [==============================] - 22s 336us/sample - loss: 1.0616 - accuracy: 0.66
Starting epoch 9
Epoch 10/15
65856/65931 [============================>.] - ETA: 0s - loss: 1.0316 - accuracy: 0.6783Ending

Epoch 00010: val_loss did not improve from 1.02282
65931/65931 [==============================] - 22s 337us/sample - loss: 1.0315 - accuracy: 0.67
Starting epoch 10
Epoch 11/15
65664/65931 [============================>.] - ETA: 0s - loss: 1.0130 - accuracy: 0.6831Ending

Epoch 00011: val_loss did not improve from 1.02282
65931/65931 [==============================] - 22s 338us/sample - loss: 1.0123 - accuracy: 0.68
Starting epoch 11
Epoch 12/15
65728/65931 [============================>.] - ETA: 0s - loss: 1.0053 - accuracy: 0.6856Ending
```

```
Epoch 00012: val_loss did not improve from 1.02282
65931/65931 [==============================] - 22s 335us/sample - loss: 1.0054 - accuracy: 0.68
Starting epoch 12
Epoch 13/15
65792/65931 [============================>.] - ETA: 0s - loss: 0.9775 - accuracy: 0.6966Ending

Epoch 00013: val_loss improved from 1.02282 to 0.95071, saving model to model_checkpoints/check
65931/65931 [==============================] - 22s 337us/sample - loss: 0.9779 - accuracy: 0.69
Starting epoch 13
Epoch 14/15
65792/65931 [============================>.] - ETA: 0s - loss: 0.9628 - accuracy: 0.6990Ending

Epoch 00014: val_loss did not improve from 0.95071
65931/65931 [==============================] - 22s 338us/sample - loss: 0.9631 - accuracy: 0.69
Starting epoch 14
Epoch 15/15
65664/65931 [============================>.] - ETA: 0s - loss: 0.9490 - accuracy: 0.7039Ending

Epoch 00015: val_loss improved from 0.95071 to 0.94217, saving model to model_checkpoints/check
65931/65931 [==============================] - 22s 337us/sample - loss: 0.9491 - accuracy: 0.70
Finished Training
```

```
In [13]: #Plot the learning curves for loss vs epoch and accuracy vs epoch for both training a
         fig = plt.figure(figsize=(12, 5))

         fig.add_subplot(121)

         plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
         plt.title('training and validation loss')
         plt.ylabel('Loss')
         plt.xlabel('Epoch')
         plt.legend(['Training', 'Validation'], loc='upper right')

         fig.add_subplot(122)

         plt.plot(history.history['accuracy'])
         plt.plot(history.history['val_accuracy'])
         plt.title('training and validation accuracy')
         plt.ylabel('Accuracy')
         plt.xlabel('Epoch')
         plt.legend(['Training', 'Validation'], loc='upper right')

         plt.show()
```



```
In [14]: #Compute and display the loss and accuracy of the trained model on the test set
         model.evaluate(np.transpose(x_test_grayscale, (3, 0, 1, 2)), y_test, verbose = 2)

26032/1 - 3s - loss: 1.0537 - accuracy: 0.6848


Out[14]: [1.05807182154336, 0.68484944]
```

1.4 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.

```
In [15]: cnn_model = Sequential([
             Conv2D(filters=16, input_shape=x_train_grayscale[:,:,:,0].shape, kernel_size=(3,
                           activation='relu', name='conv_1'),
             BatchNormalization(),
             Conv2D(filters=8, kernel_size=(3, 3), activation='relu', name='conv_2'),
             BatchNormalization(),
             MaxPooling2D(pool_size=(4, 4), name='pool_1'),
             Flatten(name='flatten'),
             Dense(units=64, activation='relu', name='dense_1'),
             Dropout(0.1),
             BatchNormalization(),
             Dense(units=32, activation='relu', name='dense_2'),
             BatchNormalization(),
             Dense(units=10, activation='softmax', name='softmax')
             ])


         cnn_checkpoint_path = 'cnn_model_checkpoints/checkpoint'
         checkpoint = ModelCheckpoint(filepath = cnn_checkpoint_path,
                                      frequency = 'epoch',
                                      save_weights_only = True,
                                      verbose = 1,
                                      save_best_only = True)
```

```
In [16]: #Print model summary
         cnn_model.summary()
```

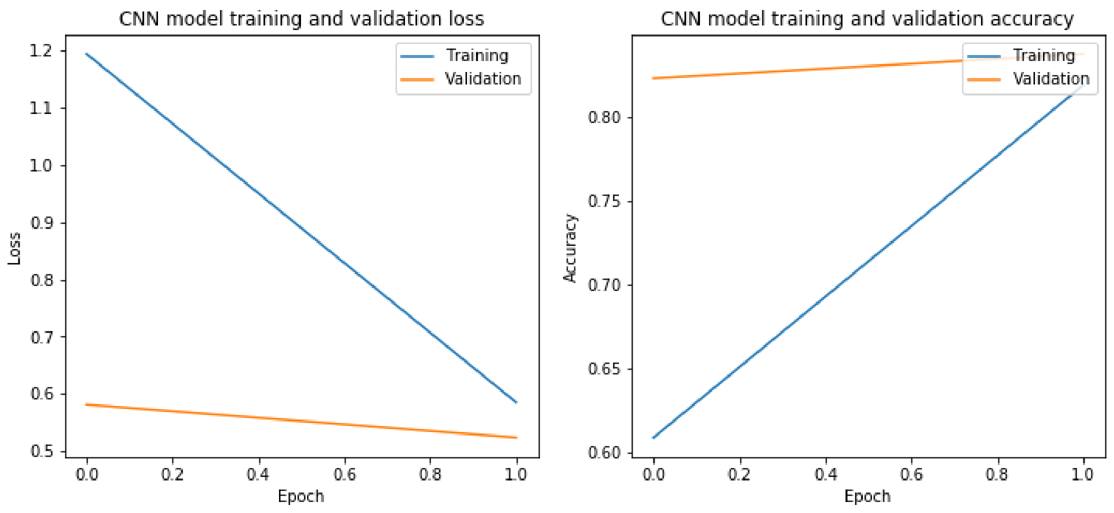
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_1 (Conv2D)              (None, 30, 30, 16)        160
_________________________________________________________________
batch_normalization (BatchNo (None, 30, 30, 16)        64
_________________________________________________________________
conv_2 (Conv2D)              (None, 28, 28, 8)         1160
_________________________________________________________________
batch_normalization_1 (Batch (None, 28, 28, 8)         32
_________________________________________________________________
pool_1 (MaxPooling2D)        (None, 7, 7, 8)           0
_________________________________________________________________
flatten (Flatten)            (None, 392)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                25152
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
batch_normalization_2 (Batch (None, 64)                256
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
batch_normalization_3 (Batch (None, 32)                128
_________________________________________________________________
softmax (Dense)              (None, 10)                330
=================================================================
Total params: 29,362
Trainable params: 29,122
Non-trainable params: 240
_________________________________________________________________
```

```
In [17]: #Compile the model
         cnn_model.compile(optimizer = 'adam',
                       loss = 'sparse_categorical_crossentropy',
                       metrics = ['accuracy'])

         #Change dimension to: (batch, h, w, c)

         cnn_history = cnn_model.fit(np.transpose(x_train_grayscale, (3, 0, 1, 2)),
                             y_train, epochs = 2, validation_split = 0.1, batch_size = 64,
                             verbose = 1 , callbacks=[myCallback(), tf.keras.callbacks.EarlySto
```

```
Train on 65931 samples, validate on 7326 samples
Starting training
Starting epoch 0
Epoch 1/2
65920/65931 [============================>.] - ETA: 0s - loss: 1.1940 - accuracy: 0.6089Ending

Epoch 00001: val_loss improved from inf to 0.58077, saving model to cnn_model_checkpoints/chec
65931/65931 [==============================] - 370s 6ms/sample - loss: 1.1940 - accuracy: 0.608
Starting epoch 1
Epoch 2/2
65920/65931 [============================>.] - ETA: 0s - loss: 0.5847 - accuracy: 0.8185Ending

Epoch 00002: val_loss improved from 0.58077 to 0.52330, saving model to cnn_model_checkpoints/c
65931/65931 [==============================] - 364s 6ms/sample - loss: 0.5847 - accuracy: 0.818
Finished Training
```

```
In [18]: #Plot the learning curves for loss vs epoch and accuracy vs epoch for both training a
         fig = plt.figure(figsize=(12, 5))

         fig.add_subplot(121)

         plt.plot(cnn_history.history['loss'])
         plt.plot(cnn_history.history['val_loss'])
         plt.title('CNN model training and validation loss')
         plt.ylabel('Loss')
         plt.xlabel('Epoch')
         plt.legend(['Training', 'Validation'], loc='upper right')
```

```
fig.add_subplot(122)

plt.plot(cnn_history.history['accuracy'])
plt.plot(cnn_history.history['val_accuracy'])
plt.title('CNN model training and validation accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')

plt.show()
```



```
In [19]: #Compute and display the loss and accuracy of the trained model on the test set
         cnn_model.evaluate(np.transpose(x_test_grayscale, (3, 0, 1, 2)), y_test, verbose = 2)
```

```
26032/1 - 43s - loss: 0.5060 - accuracy: 0.8243
```

```
Out[19]: [0.5774817038567871, 0.8243316]
```

1.5 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.

```
In [20]: model2 = Sequential([
             Flatten(input_shape = x_train_grayscale[:,:,:,0].shape),
             Dense(64, activation='relu'),
             Dense(64, activation='relu'),
             Dense(64, activation='relu'),
             Dense(64, activation='relu'),
             Dense(64, activation='relu'),
             Dense(10, activation='softmax')
             ])
         model2.load_weights(checkpoint_path)
```

```
Out[20]: <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7fddec4055c0>
```

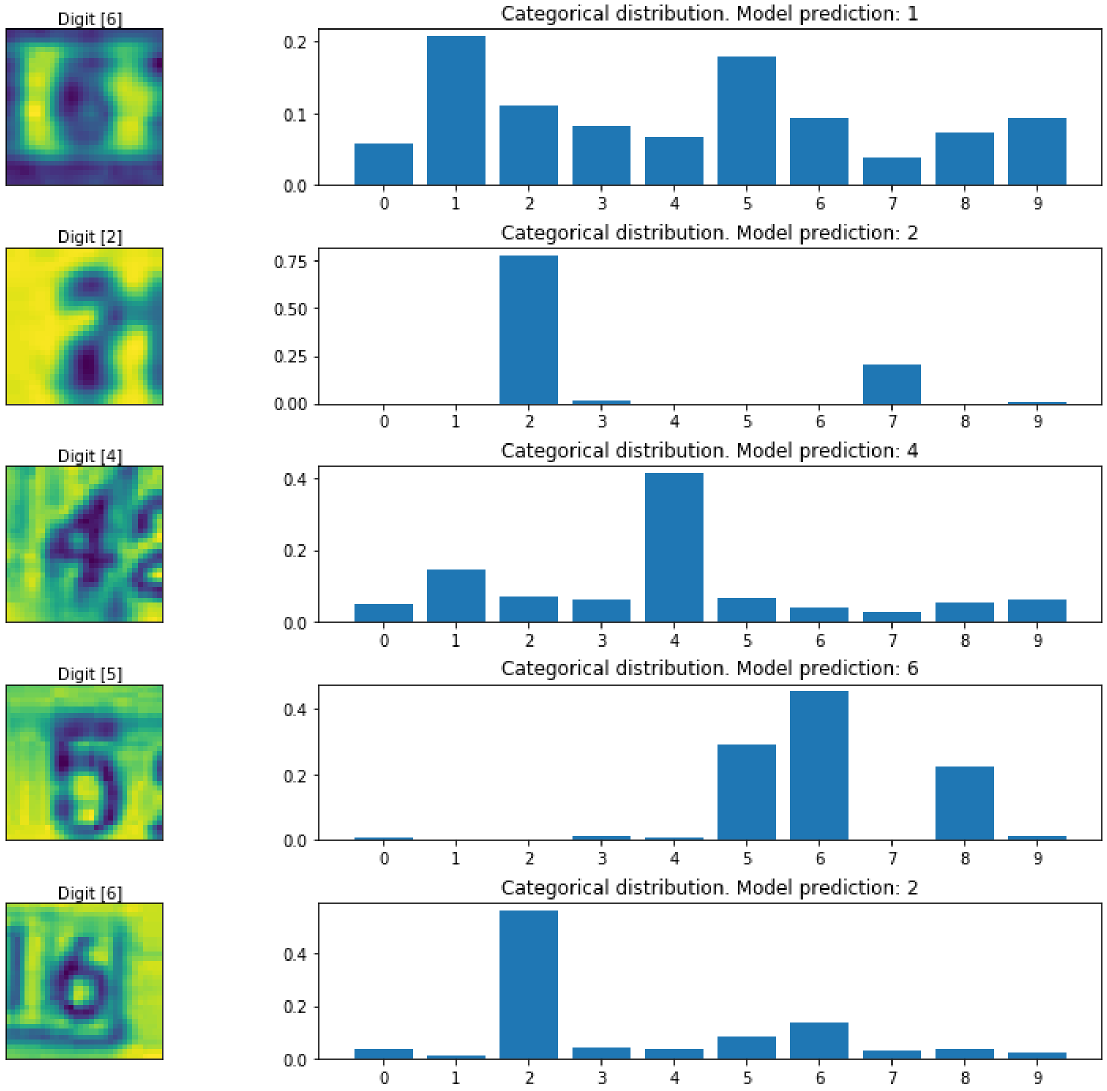
```
In [21]: cnn_model2 = Sequential([
             Conv2D(filters=16, input_shape=x_train_grayscale[:,:,:,0].shape, kernel_size=(3, 3
                         activation='relu', name='conv_1'),
             BatchNormalization(),
             Conv2D(filters=8, kernel_size=(3, 3), activation='relu', name='conv_2'),
             BatchNormalization(),
             MaxPooling2D(pool_size=(4, 4), name='pool_1'),
             Flatten(name='flatten'),
             Dense(units=64, activation='relu', name='dense_1'),
             BatchNormalization(),
             Dense(units=32, activation='relu', name='dense_2'),
             BatchNormalization(),
             Dense(units=10, activation='softmax', name='softmax')
             ])
         cnn_model2.load_weights(cnn_checkpoint_path)
```

```
WARNING:tensorflow:Inconsistent references when loading the checkpoint into this object graph.

Two checkpoint references resolved to different objects (<tensorflow.python.keras.layers.norma
WARNING:tensorflow:Inconsistent references when loading the checkpoint into this object graph.

Two checkpoint references resolved to different objects (<tensorflow.python.keras.layers.core.
WARNING:tensorflow:Inconsistent references when loading the checkpoint into this object graph.

Two checkpoint references resolved to different objects (<tensorflow.python.keras.layers.norma
```

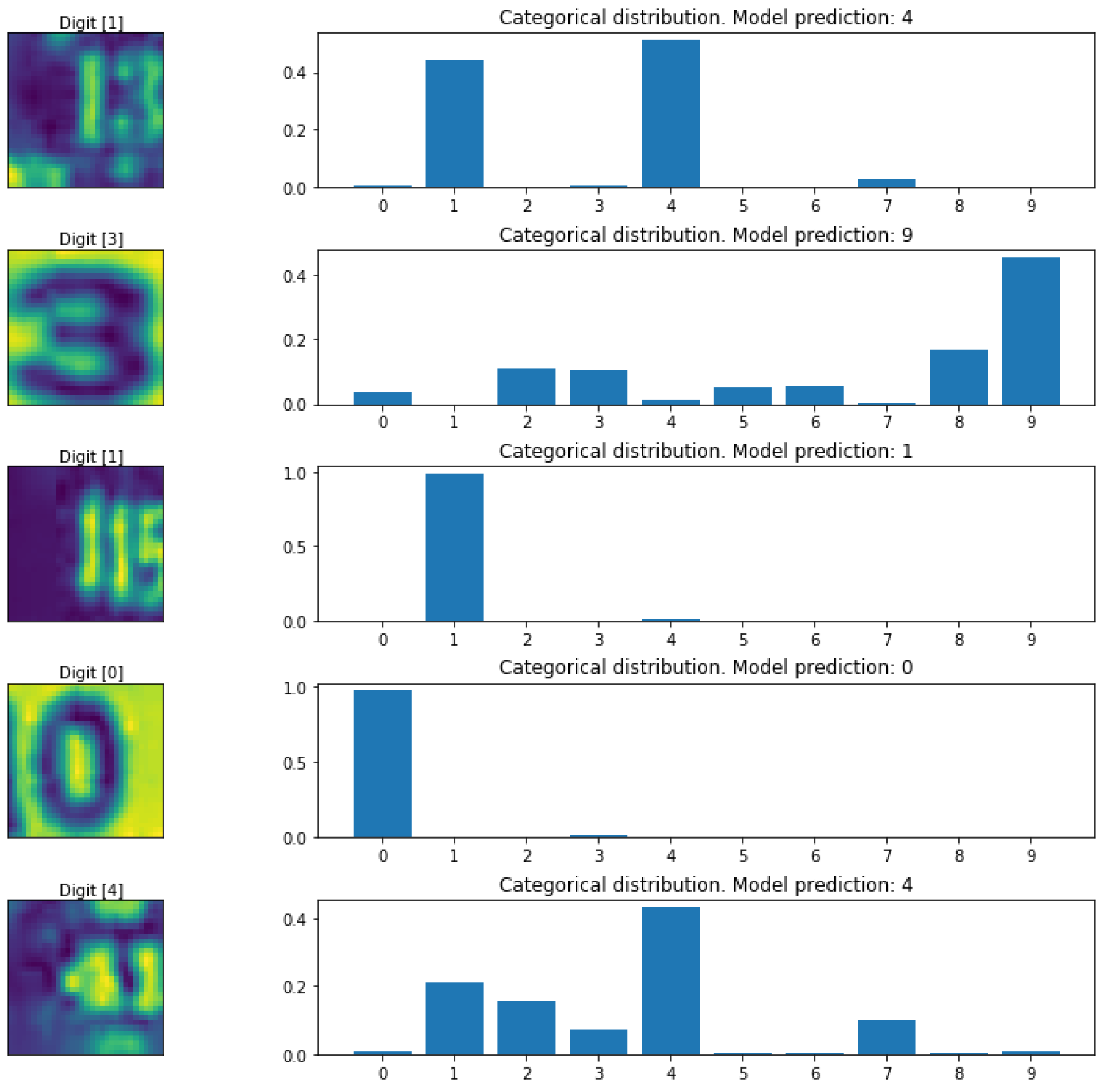
```
Out[21]: <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7fddec28e198>
```

```
In [22]: #MLP model
         #Alongside the image and label, show each models predictive distribution as a bar cha

         num_test_images = x_test_grayscale.shape[3]

         random_inx = np.random.choice(num_test_images, 5)
         random_test_images = x_test_grayscale[:,:, :,random_inx]
         random_test_images = np.transpose(random_test_images, (3, 0, 1, 2))
         random_test_labels = y_test[ random_inx, ]
```

```
predictions = model2.predict(random_test_images)

fig, axes = plt.subplots(5, 2, figsize=(16, 12))
fig.subplots_adjust(hspace=0.4, wspace=-0.2)

for i, (prediction, image, label) in enumerate(zip(predictions, random_test_images, ra
    axes[i, 0].imshow(np.squeeze(image))
    axes[i, 0].get_xaxis().set_visible(False)
    axes[i, 0].get_yaxis().set_visible(False)
    axes[i, 0].text(10., -1.5, f'Digit {label}')
    axes[i, 1].bar(np.arange(len(prediction)), prediction)
    axes[i, 1].set_xticks(np.arange(len(prediction)))
    axes[i, 1].set_title(f"Categorical distribution. Model prediction: {np.argmax(pred

plt.show()
```



```
In [23]: #CNN model
         #Alongside the image and label, show each models predictive distribution as a bar cha

         num_test_images = x_test_grayscale.shape[3]

         random_inx = np.random.choice(num_test_images, 5)
         random_test_images = x_test_grayscale[:,:, :,random_inx]
         random_test_images = np.transpose(random_test_images, (3, 0, 1, 2))
         random_test_labels = y_test[ random_inx, ]
         predictions = cnn_model2.predict(random_test_images)

         fig, axes = plt.subplots(5, 2, figsize=(16, 12))
         fig.subplots_adjust(hspace=0.4, wspace=-0.2)

         for i, (prediction, image, label) in enumerate(zip(predictions, random_test_images, ra
             axes[i, 0].imshow(np.squeeze(image))
             axes[i, 0].get_xaxis().set_visible(False)
             axes[i, 0].get_yaxis().set_visible(False)
             axes[i, 0].text(10., -1.5, f'Digit {label}')
             axes[i, 1].bar(np.arange(len(prediction)), prediction)
             axes[i, 1].set_xticks(np.arange(len(prediction)))
             axes[i, 1].set_title(f"Categorical distribution. Model prediction: {np.argmax(pred

         plt.show()
```


Digit [1]
Categorical distribution. Model prediction: 4
Digit [3]
Categorical distribution. Model prediction: 9
Digit [1]
Categorical distribution. Model prediction: 1
Digit [0]
Categorical distribution. Model prediction: 0
Digit [4]
Categorical distribution. Model prediction: 4

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```