Capstone Project

September 4, 2020

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
        import numpy as np
        import matplotlib.pyplot as plt
        import random

        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Flatten, Dropout, Conv2D, MaxPooling2D, Bat
        from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
        from tensorflow.keras.optimizers import Adam
```

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 [2]: # 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 [3]: X_train = train['X']
        X_test = test['X']
        y_train = train['y']
        y_test = test['y']
```

```
In [4]: X_train = np.moveaxis(X_train, -1, 0)
        X_test = np.moveaxis(X_test, -1, 0)

In [5]: y_train = np.where(y_train==10, 0, y_train)
        y_test = np.where(y_test==10, 0, y_test)

In [6]: indices = random.sample( range(0,X_train.shape[0]), 10 )
        fig, ax = plt.subplots(1, 10, figsize=(10,1))

        for i in range(10):
            ax[i].set_axis_off()
            ax[i].imshow(X_train[indices[i]])
            ax[i].set_title(y_train[indices[i]])
```


```
In [6]: X_train_greyscale = np.mean(X_train, -1, keepdims=True)
        X_test_greyscale = np.mean(X_test, -1, keepdims=True)

In [7]: indices = random.sample( range(0,X_train_greyscale.shape[0]), 10 )
        fig, ax = plt.subplots(1, 10, figsize=(10,1))

        for i in range(10):
            ax[i].set_axis_off()
            ax[i].imshow(X_train_greyscale[indices[i],:,:,0], cmap='gray')
            ax[i].set_title(y_train[indices[i]])
```

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.*

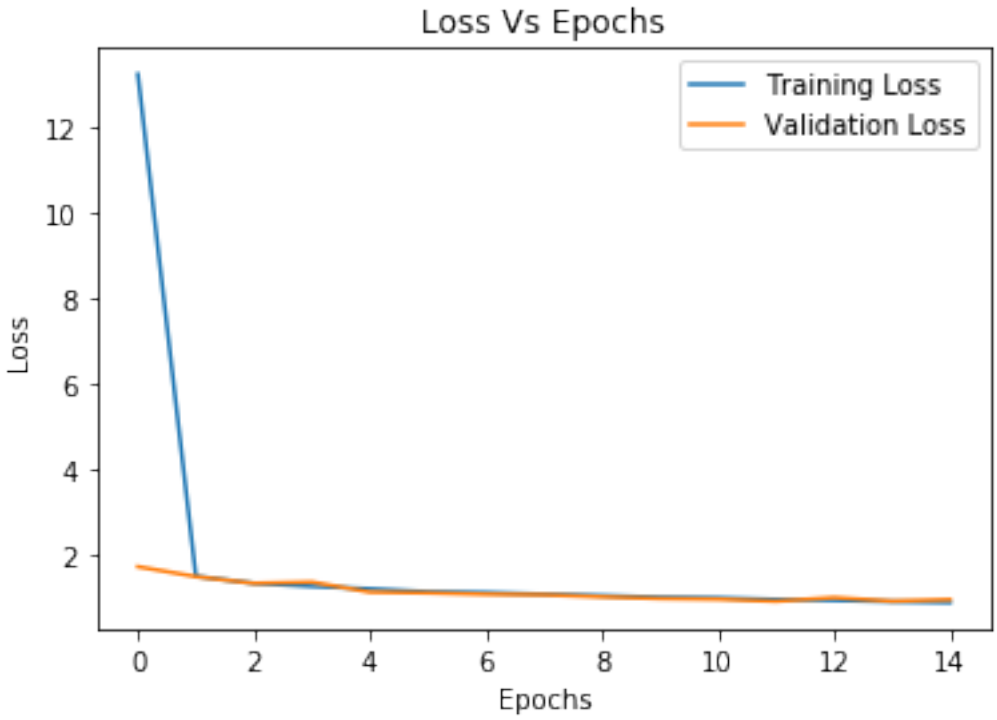
- Print out the model summary (using the summary() method)
- Compile and train the model (we recommend a maximum of 30 epochs), making use of both training and validation sets during the training run.
- Your model should track at least one appropriate metric, and use at least two callbacks during training, 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 [7]: def get_new_MLP_model():

            Model = Sequential([
                Flatten(input_shape=X_train[0].shape),
                Dense(512, activation='relu'),
                Dense(512, activation='relu'),
                Dense(256, activation='relu'),
                Dense(128, activation='relu'),
                Dense(10, activation='softmax')
            ])

            return Model

        model = get_new_MLP_model()
        model.summary()
```

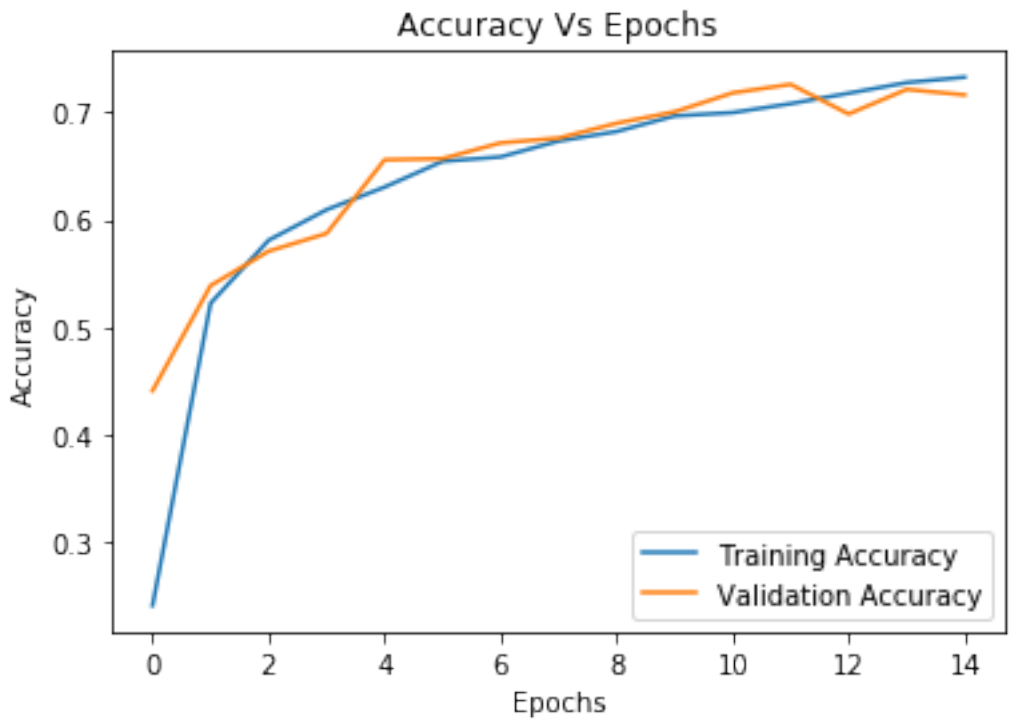
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 3072)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1573376
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1290
=================================================================
Total params: 2,001,546
Trainable params: 2,001,546
Non-trainable params: 0
_________________________________________________________________
```

```
In [30]: cp_path = 'Checkpoint/Best_model'
         cp = ModelCheckpoint(cp_path,
                              save_best_only=True,
                              save_weights_only=True,
                              verbose=False,
                              save_freq='epoch',
                              monitor='val_loss',
                              mode='min')

In [31]: EarlyStop = EarlyStopping(monitor='val_loss', mode='min', patience=3)

In [32]: model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accu

In [33]: history = model.fit(X_train, y_train, epochs=30, batch_size=64, verbose=1, validation_

Train on 65931 samples, validate on 7326 samples
Epoch 1/30
65931/65931 [==============================] - 95s 1ms/sample - loss: 13.2207 - accuracy: 0.241
Epoch 2/30
65931/65931 [==============================] - 92s 1ms/sample - loss: 1.4794 - accuracy: 0.5222
Epoch 3/30
65931/65931 [==============================] - 91s 1ms/sample - loss: 1.3154 - accuracy: 0.5808
Epoch 4/30
65931/65931 [==============================] - 94s 1ms/sample - loss: 1.2450 - accuracy: 0.6093
Epoch 5/30
65931/65931 [==============================] - 97s 1ms/sample - loss: 1.1854 - accuracy: 0.6303
Epoch 6/30
65931/65931 [==============================] - 99s 1ms/sample - loss: 1.1163 - accuracy: 0.6543
Epoch 7/30
65931/65931 [==============================] - 98s 1ms/sample - loss: 1.0987 - accuracy: 0.6584
Epoch 8/30
65931/65931 [==============================] - 98s 1ms/sample - loss: 1.0590 - accuracy: 0.6732
Epoch 9/30
65931/65931 [==============================] - 98s 1ms/sample - loss: 1.0273 - accuracy: 0.6820
Epoch 10/30
65931/65931 [==============================] - 99s 1ms/sample - loss: 0.9882 - accuracy: 0.6965
Epoch 11/30
65931/65931 [==============================] - 99s 1ms/sample - loss: 0.9737 - accuracy: 0.6996
Epoch 12/30
65931/65931 [==============================] - 99s 1ms/sample - loss: 0.9371 - accuracy: 0.7079
Epoch 13/30
65931/65931 [==============================] - 99s 1ms/sample - loss: 0.9184 - accuracy: 0.7177
Epoch 14/30
65931/65931 [==============================] - 98s 1ms/sample - loss: 0.8880 - accuracy: 0.7275
Epoch 15/30
65931/65931 [==============================] - 98s 1ms/sample - loss: 0.8670 - accuracy: 0.7326


In [34]: plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
```

```
plt.title("Loss Vs Epochs")
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['Training Loss', 'Validation Loss'], loc='upper right')
plt.show()
```



In [35]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title("Accuracy Vs Epochs")
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['Training Accuracy', 'Validation Accuracy'], loc='lower right')
plt.show()
```




```
In [36]: test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=False)
         print(f"Test Loss is {test_loss}")
         print(f"Test Accuracy is {test_accuracy}")
```

```
Test Loss is 1.020696268795309
Test Accuracy is 0.7009065747261047
```

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 [8]: def get_new_CNN_model():

            Model = Sequential([
                Conv2D(16, (3,3), padding='same', activation='relu', input_shape=X_train[0].sha
                BatchNormalization(),
                MaxPooling2D((2,2)),
                Conv2D(16, (3,3), padding='same', activation='relu'),
                BatchNormalization(),
                MaxPooling2D((2,2)),
                Conv2D(16, (3,3), padding='same', activation='relu'),
                BatchNormalization(),
                MaxPooling2D((2,2)),
                Flatten(),
                Dense(256, activation='relu'),
                Dropout(0.2),
                Dense(128, activation='relu'),
                Dropout(0.2),
                Dense(10, activation='softmax')
            ])

            return Model

        CNN_model = get_new_CNN_model()
        CNN_model.summary()
```

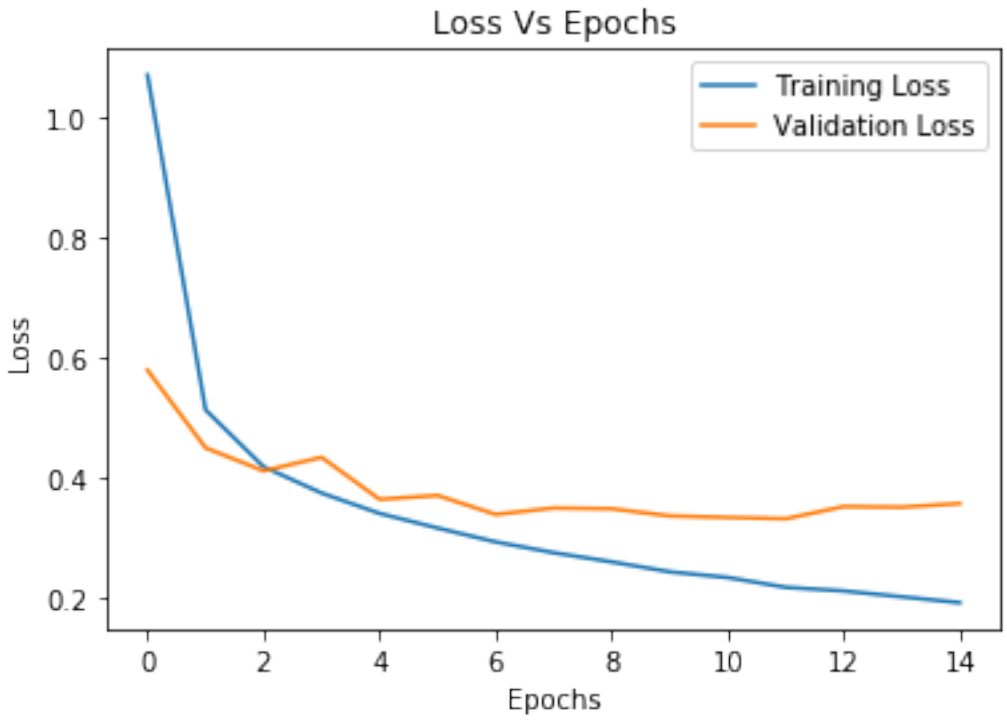
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 16)        448
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 16)        64
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 16)        2320
_________________________________________________________________
batch_normalization_1 (Batch (None, 16, 16, 16)        64
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 16)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 16)          2320
_________________________________________________________________
batch_normalization_2 (Batch (None, 8, 8, 16)          64
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 16)          0
```

```
_________________________________________________________________
flatten_1 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               65792
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1290
=================================================================
Total params: 105,258
Trainable params: 105,162
Non-trainable params: 96
_________________________________________________________________
```

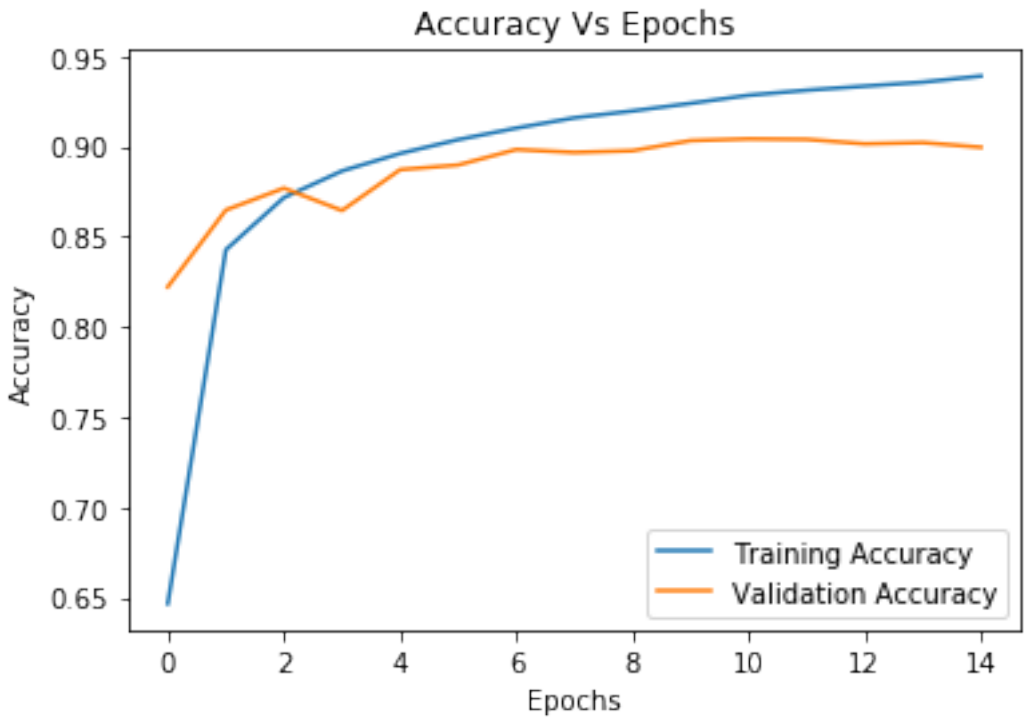
```
In [7]: cp_path = 'CNN_Checkpoint/Best_model'
        cp = ModelCheckpoint(cp_path,
                             save_best_only=True,
                             save_weights_only=True,
                             verbose=False,
                             save_freq='epoch',
                             monitor='val_loss',
                             mode='min')

In [8]: EarlyStop = EarlyStopping(monitor='val_loss', mode='min', patience=3)

In [9]: CNN_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['a

In [10]: history = CNN_model.fit(X_train, y_train, epochs=30, batch_size=64,
                                 verbose=1, validation_split=0.1, callbacks=[cp, EarlyStop])

Train on 65931 samples, validate on 7326 samples
Epoch 1/30
65931/65931 [==============================] - 338s 5ms/sample - loss: 1.0690 - accuracy: 0.646
Epoch 2/30
65931/65931 [==============================] - 329s 5ms/sample - loss: 0.5134 - accuracy: 0.842
Epoch 3/30
65931/65931 [==============================] - 333s 5ms/sample - loss: 0.4191 - accuracy: 0.871
Epoch 4/30
65931/65931 [==============================] - 334s 5ms/sample - loss: 0.3753 - accuracy: 0.886
Epoch 5/30
65931/65931 [==============================] - 336s 5ms/sample - loss: 0.3415 - accuracy: 0.896
Epoch 6/30
65931/65931 [==============================] - 336s 5ms/sample - loss: 0.3173 - accuracy: 0.903
Epoch 7/30
```

```
65931/65931 [==============================] - 336s 5ms/sample - loss: 0.2943 - accuracy: 0.910
Epoch 8/30
65931/65931 [==============================] - 338s 5ms/sample - loss: 0.2763 - accuracy: 0.915
Epoch 9/30
65931/65931 [==============================] - 339s 5ms/sample - loss: 0.2611 - accuracy: 0.919
Epoch 10/30
65931/65931 [==============================] - 337s 5ms/sample - loss: 0.2446 - accuracy: 0.923
Epoch 11/30
65931/65931 [==============================] - 336s 5ms/sample - loss: 0.2352 - accuracy: 0.928
Epoch 12/30
65931/65931 [==============================] - 337s 5ms/sample - loss: 0.2189 - accuracy: 0.931
Epoch 13/30
65931/65931 [==============================] - 334s 5ms/sample - loss: 0.2129 - accuracy: 0.933
Epoch 14/30
65931/65931 [==============================] - 334s 5ms/sample - loss: 0.2032 - accuracy: 0.935
Epoch 15/30
65931/65931 [==============================] - 330s 5ms/sample - loss: 0.1935 - accuracy: 0.939
```

```
In [11]: plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
         plt.title("Loss Vs Epochs")
         plt.ylabel('Loss')
         plt.xlabel('Epochs')
         plt.legend(['Training Loss', 'Validation Loss'], loc='upper right')
         plt.show()
```



```
In [12]: plt.plot(history.history['accuracy'])
         plt.plot(history.history['val_accuracy'])
         plt.title("Accuracy Vs Epochs")
         plt.ylabel('Accuracy')
         plt.xlabel('Epochs')
         plt.legend(['Training Accuracy', 'Validation Accuracy'], loc='lower right')
         plt.show()
```




```
In [13]: test_loss, test_accuracy = CNN_model.evaluate(X_test, y_test, verbose=False)
         print(f"Test Loss is {test_loss}")
         print(f"Test Accuracy is {test_accuracy}")
```

```
Test Loss is 0.397798109098985
Test Accuracy is 0.8935156464576721
```

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 [9]: Best_MLP = get_new_MLP_model()
        Best_MLP.load_weights('Checkpoint/Best_model')
```

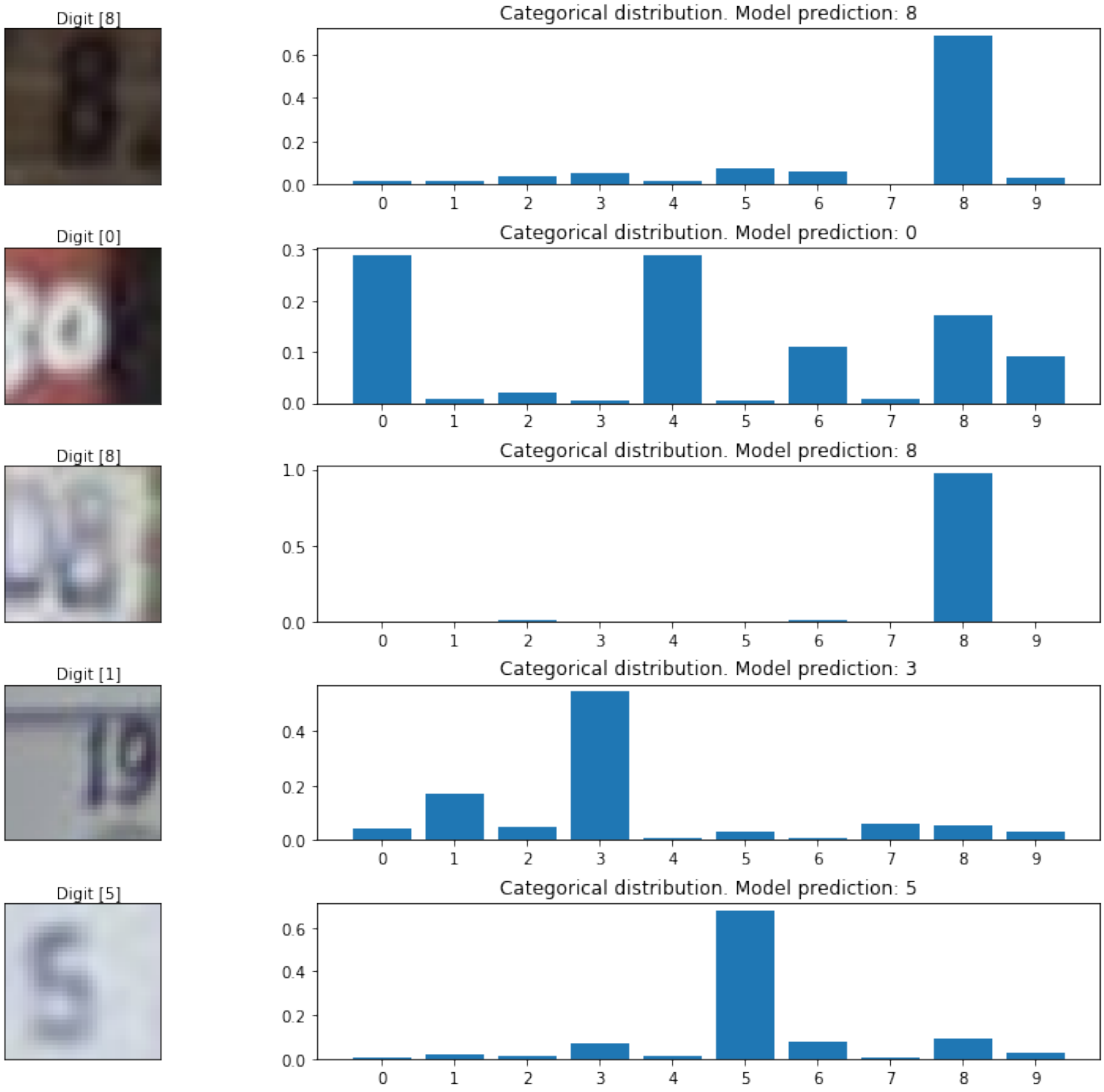
```
Out[9]: <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f3218ff8dd8>
```

```
In [10]: Best_CNN = get_new_CNN_model()
         Best_CNN.load_weights('CNN_Checkpoint/Best_model')
```

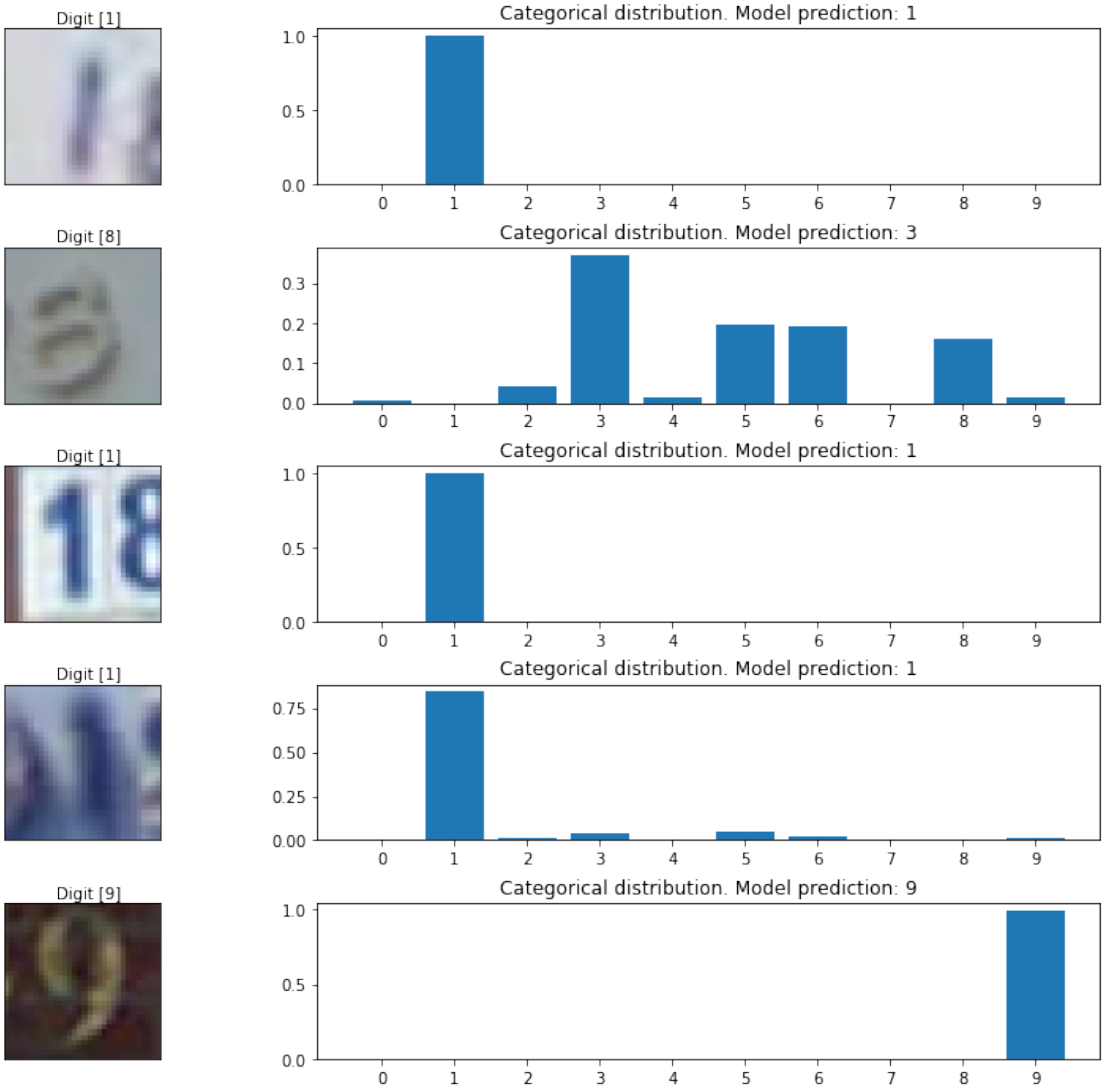
```
Out[10]: <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f32040b20b8>
```

```
In [11]: def show_predictive_distribution(model):

             num_test_images = X_test.shape[0]

             random_inx = np.random.choice(num_test_images, 5)
             random_test_images = X_test[random_inx, ...]
             random_test_labels = y_test[random_inx, ...]

             predictions = model.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
                 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

             plt.show()
```

```
In [15]: show_predictive_distribution(Best_MLP)
```



In [14]: `show_predictive_distribution(Best_CNN)`


Digit [1]
Categorical distribution. Model prediction: 1
Digit [8]
Categorical distribution. Model prediction: 3
Digit [1]
Categorical distribution. Model prediction: 1
Digit [1]
Categorical distribution. Model prediction: 1
Digit [9]
Categorical distribution. Model prediction: 9