Capstone Project

August 20, 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
        import numpy as np
        import matplotlib.pyplot as plt
```


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')
```

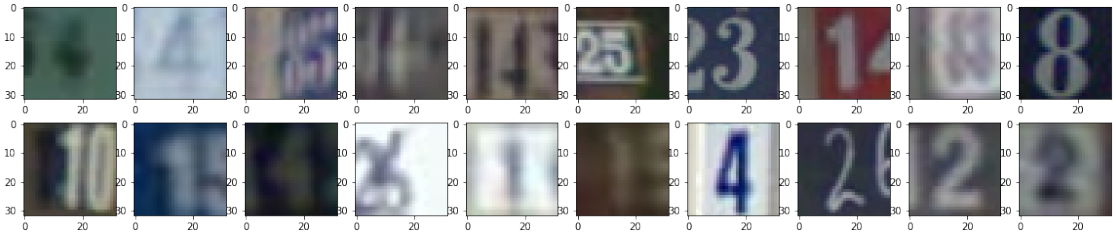
Both `train` and `test` are dictionaries with keys `X` and `y` for the input images and labels respectively.

1.2 1. Inspect and preprocess the dataset

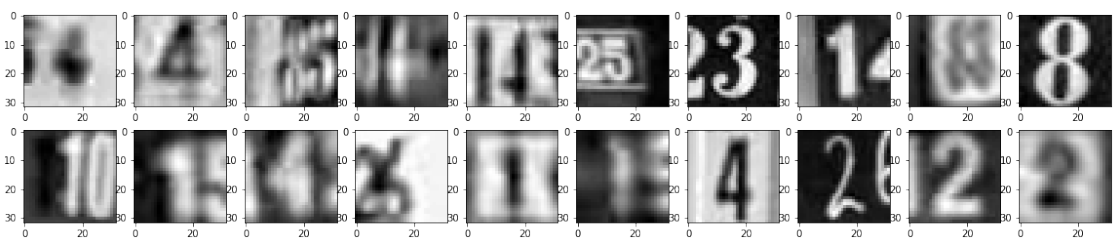
- Extract the training and testing images and labels separately from the train and test dictionaries loaded for you.
- Select a random sample of images and corresponding labels from the dataset (at least 10), and display them in a figure.
- Convert the training and test images to grayscale by taking 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]: train_X, train_y = train['X'], train['y']
        test_X, test_y = test['X'], test['y']
```

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

In [5]: train_X.shape, test_X.shape

Out[5]: ((73257, 32, 32, 3), (26032, 32, 32, 3))

In [6]: train_y = np.where(train_y==10, 0, train_y)
        test_y = np.where(test_y==10, 0, test_y)

In [7]: train_indices = np.random.choice(train_X.shape[0], 10)
        test_indices = np.random.choice(test_X.shape[0], 10)
        f, ax = plt.subplots(2, 10, figsize=(20, 4))
        for j in range(10):
            ax[0, j].imshow(train_X[train_indices[j], :, :, :])
        for j in range(10):
            ax[1, j].imshow(test_X[test_indices[j], :, :, :])
```



```
In [8]: train_X_gray = np.mean(train_X, axis=-1, keepdims=True)
        test_X_gray = np.mean(test_X, axis=-1, keepdims=True)
        train_X_gray[0].shape, test_X_gray[0].shape

Out[8]: ((32, 32, 1), (32, 32, 1))

In [9]: # train_indices = np.random.choice(train_X_gray.shape[-1], 10)
        # test_indices = np.random.choice(test_X_gray.shape[-1], 10)
        # indices = [train_indices, test_indices]
        f, ax = plt.subplots(2, 10, figsize=(20, 4))
        for j in range(10):
            ax[0, j].imshow(train_X_gray[train_indices[j], :, :, :].squeeze(axis=-1), cmap='gra
        for j in range(10):
            ax[1, j].imshow(test_X_gray[test_indices[j], :, :, :].squeeze(axis=-1), cmap='gray
```



In [ ]:

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 [10]: from tensorflow.keras.models import Sequential
         from tensorflow.keras.layers import Dense, Flatten, Conv2D, BatchNormalization, MaxPo
         from tensorflow.keras.callbacks import ModelCheckpoint
```

```
In [13]: def get_new_model(input_shape):
             model = Sequential([
                 Flatten(input_shape=input_shape, name='flatten'),
                 Dense(512, activation='relu', name='dense_1'),
                 Dense(512, activation='relu', name='dense_2'),
                 Dense(256, activation='relu', name='dense_3'),
                 Dense(128, activation='relu', name='dense_4'),
                 Dense(10, activation='softmax', name='dense_5'),
             ])

             return model
```

```
In [14]: model = get_new_model(train_X[0].shape)
         model.summary()
```

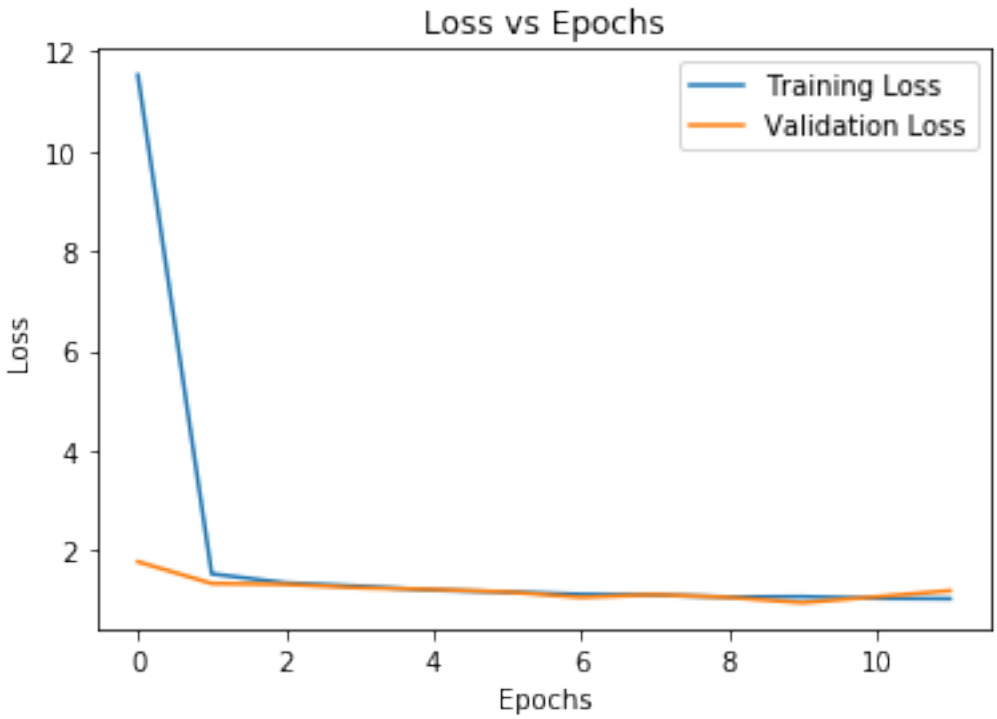
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 3072)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1573376
_________________________________________________________________
```

```
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_3 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_4 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290
=================================================================
Total params: 2,001,546
Trainable params: 2,001,546
Non-trainable params: 0
_________________________________________________________________
```

```
In [15]: model.compile(optimizer='adam',
                       loss='sparse_categorical_crossentropy',
                       metrics=['accuracy'])

In [16]: checkpoint_path = 'model_checkpoints/checkpoint'
         checkpoint = ModelCheckpoint(filepath=checkpoint_path,
                                      save_best_only=True,
                                      save_weights_only=True,
                                      verbose=0,
                                      save_freq='epoch',
                                      monitor='val_loss',
                                      mode='min')

In [17]: earlystop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', patience=

In [18]: history = model.fit(train_X, train_y, epochs=30,
                             validation_split=0.15, batch_size=64, verbose=1,
                             callbacks=[checkpoint, earlystop])

Train on 62268 samples, validate on 10989 samples
Epoch 1/30
62268/62268 [==============================] - 104s 2ms/sample - loss: 11.5300 - accuracy: 0.2
Epoch 2/30
62268/62268 [==============================] - 100s 2ms/sample - loss: 1.5221 - accuracy: 0.503
Epoch 3/30
62268/62268 [==============================] - 101s 2ms/sample - loss: 1.3407 - accuracy: 0.574
Epoch 4/30
62268/62268 [==============================] - 103s 2ms/sample - loss: 1.2747 - accuracy: 0.599
Epoch 5/30
62268/62268 [==============================] - 102s 2ms/sample - loss: 1.1956 - accuracy: 0.627
Epoch 6/30
62268/62268 [==============================] - 100s 2ms/sample - loss: 1.1590 - accuracy: 0.638
Epoch 7/30
62268/62268 [==============================] - 101s 2ms/sample - loss: 1.1098 - accuracy: 0.656
Epoch 8/30
```

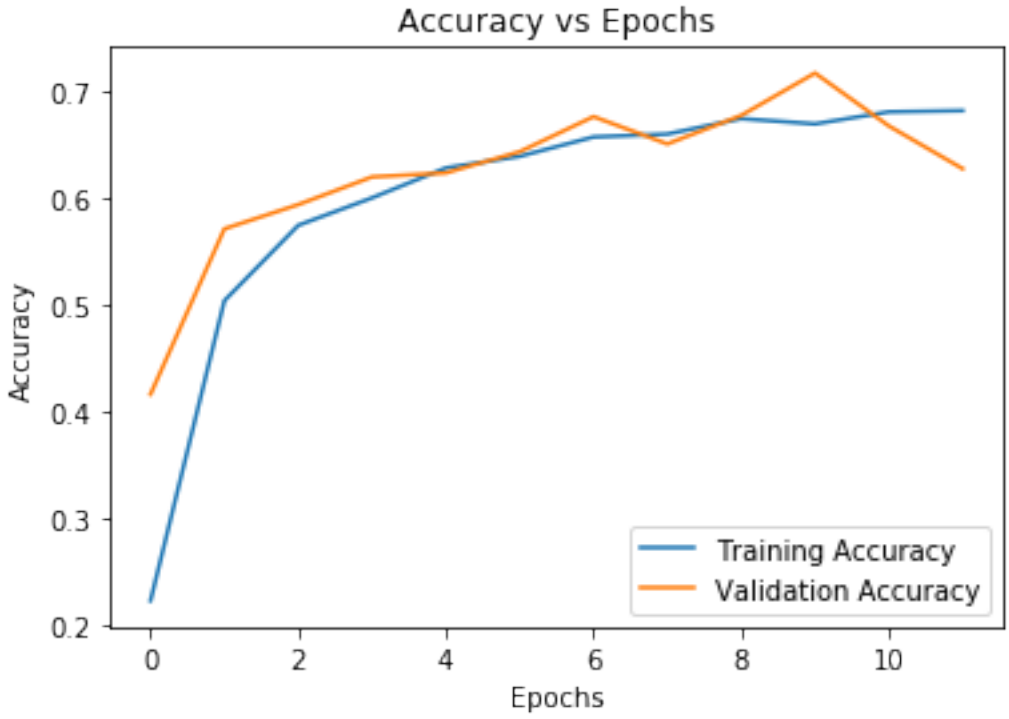
```
62268/62268 [==============================] - 101s 2ms/sample - loss: 1.0975 - accuracy: 0.659
Epoch 9/30
62268/62268 [==============================] - 101s 2ms/sample - loss: 1.0584 - accuracy: 0.674
Epoch 10/30
62268/62268 [==============================] - 100s 2ms/sample - loss: 1.0650 - accuracy: 0.669
Epoch 11/30
62268/62268 [==============================] - 100s 2ms/sample - loss: 1.0310 - accuracy: 0.680
Epoch 12/30
62268/62268 [==============================] - 100s 2ms/sample - loss: 1.0239 - accuracy: 0.681
```

In [19]:
```
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 [20]:
```
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 [21]: test_loss, test_acc = model.evaluate(test_X, test_y, verbose=0)
         print("Test loss: {:.3f}\nTest accuracy: {:.2f}%".format(test_loss, 100 * test_acc))
```

```
Test loss: 1.323
Test accuracy: 60.91%
```

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.

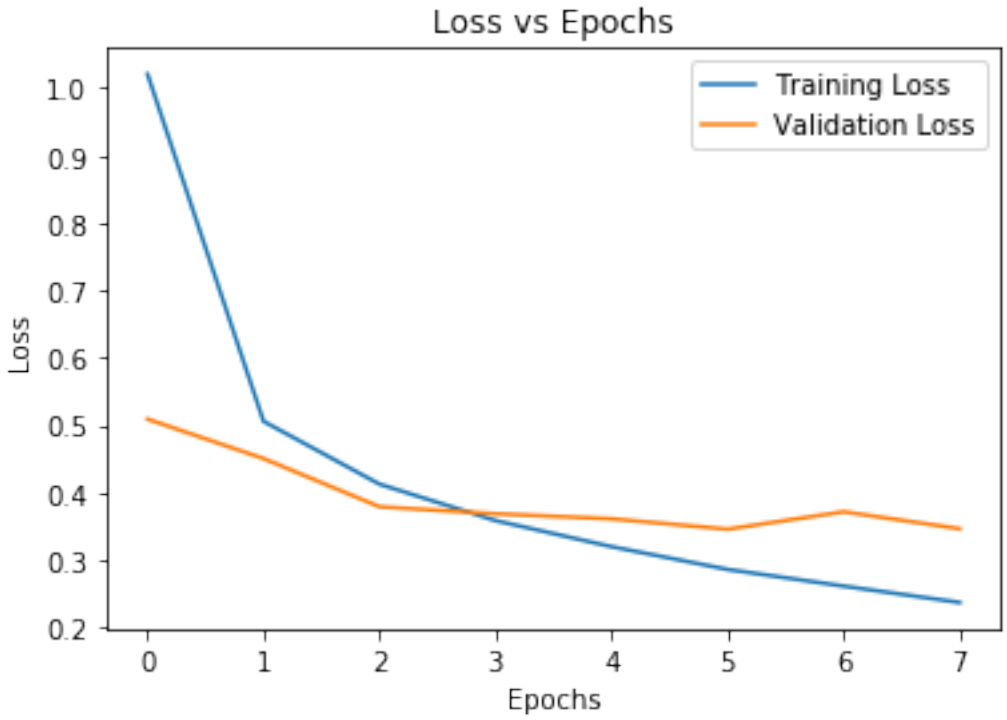
```
In [22]: def get_new_cnn_model(input_shape):
             model = Sequential([
                 Conv2D(8, kernel_size=3, activation='relu', padding='same', input_shape=input_
                 BatchNormalization(name='batch_norm_1'),
                 MaxPooling2D(pool_size=2, name='pool_1'),
                 Conv2D(16, kernel_size=3, activation='relu', padding='same', name='conv_2'),
                 BatchNormalization(name='batch_norm_2'),
                 MaxPooling2D(pool_size=2, name='pool_2'),
                 Conv2D(32, kernel_size=3, activation='relu', padding='same', name='conv_3'),
                 BatchNormalization(name='batch_norm_3'),
                 MaxPooling2D(pool_size=2, name='pool_3'),
                 Flatten(name='flatten'),
                 Dense(256, activation='relu', name='dense_1'),
                 Dropout(0.2),
                 Dense(128, activation='relu', name='dense_2'),
                 Dropout(0.2),
                 Dense(10, activation='softmax', name='dense_3'),
             ])

             return model

In [23]: cnn_model = get_new_cnn_model(train_X[0].shape)
         cnn_model.summary()

Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_1 (Conv2D)              (None, 32, 32, 8)         224
_________________________________________________________________
batch_norm_1 (BatchNormaliza (None, 32, 32, 8)         32
_________________________________________________________________
pool_1 (MaxPooling2D)        (None, 16, 16, 8)         0
_________________________________________________________________
conv_2 (Conv2D)              (None, 16, 16, 16)        1168
_________________________________________________________________
batch_norm_2 (BatchNormaliza (None, 16, 16, 16)        64
_________________________________________________________________
pool_2 (MaxPooling2D)        (None, 8, 8, 16)          0
_________________________________________________________________
conv_3 (Conv2D)              (None, 8, 8, 32)          4640
_________________________________________________________________
batch_norm_3 (BatchNormaliza (None, 8, 8, 32)          128
_________________________________________________________________
```

```
pool_3 (MaxPooling2D)        (None, 4, 4, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 171,770
Trainable params: 171,658
Non-trainable params: 112
_________________________________________________________________
```

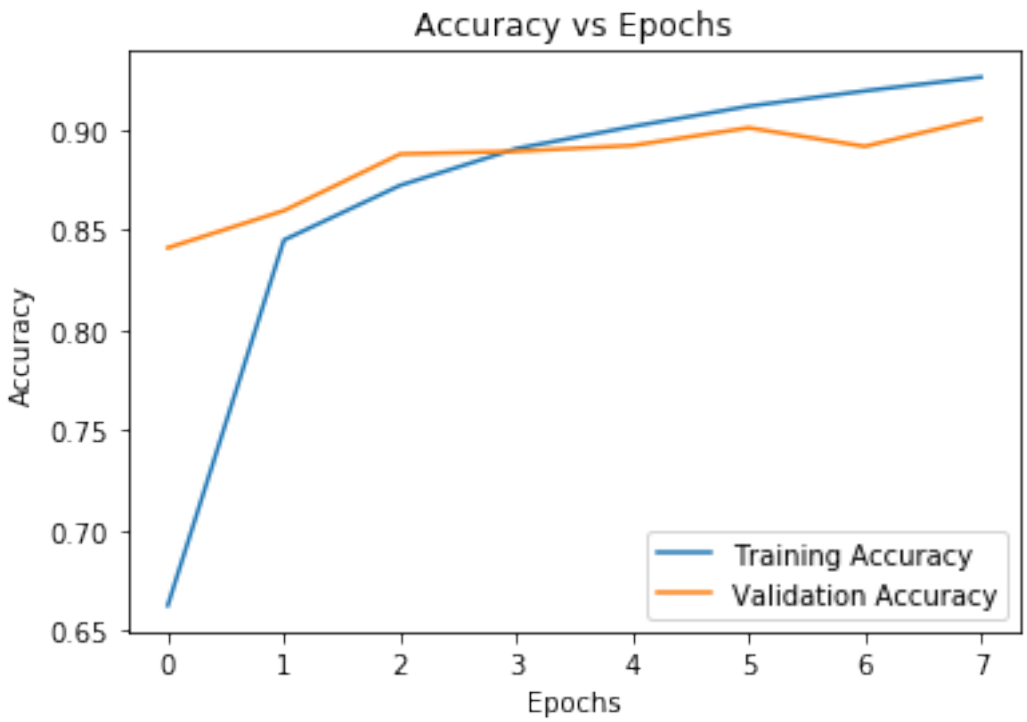
```
In [24]: cnn_checkpoint_path = 'cnn_model_checkpoints/checkpoint'
         checkpoint = ModelCheckpoint(filepath=cnn_checkpoint_path,
                                      save_best_only=True,
                                      save_weights_only=True,
                                      verbose=0,
                                      save_freq='epoch',
                                      monitor='val_loss',
                                      mode='min')

In [25]: earlystop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', patience=

In [26]: cnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
                           metrics=['accuracy'])

In [27]: history = cnn_model.fit(train_X, train_y, epochs=30,
                                 validation_split=0.15, batch_size=64, verbose=1,
                                 callbacks=[checkpoint, earlystop])

Train on 62268 samples, validate on 10989 samples
Epoch 1/30
62268/62268 [==============================] - 310s 5ms/sample - loss: 1.0206 - accuracy: 0.662
Epoch 2/30
62268/62268 [==============================] - 303s 5ms/sample - loss: 0.5060 - accuracy: 0.845
Epoch 3/30
62268/62268 [==============================] - 313s 5ms/sample - loss: 0.4128 - accuracy: 0.872
Epoch 4/30
62268/62268 [==============================] - 311s 5ms/sample - loss: 0.3588 - accuracy: 0.890
Epoch 5/30
62268/62268 [==============================] - 303s 5ms/sample - loss: 0.3202 - accuracy: 0.901
```

```
Epoch 6/30
62268/62268 [==============================] - 310s 5ms/sample - loss: 0.2864 - accuracy: 0.911
Epoch 7/30
62268/62268 [==============================] - 304s 5ms/sample - loss: 0.2617 - accuracy: 0.919
Epoch 8/30
62268/62268 [==============================] - 339s 5ms/sample - loss: 0.2374 - accuracy: 0.926
```

In [28]:
```
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 [29]:
```
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 [30]: test_loss, test_acc = cnn_model.evaluate(test_X, test_y, verbose=0)
         print("Test loss: {:.3f}\nTest accuracy: {:.2f}%".format(test_loss, 100 * test_acc))
```

```
Test loss: 0.377
Test accuracy: 89.55%
```

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.

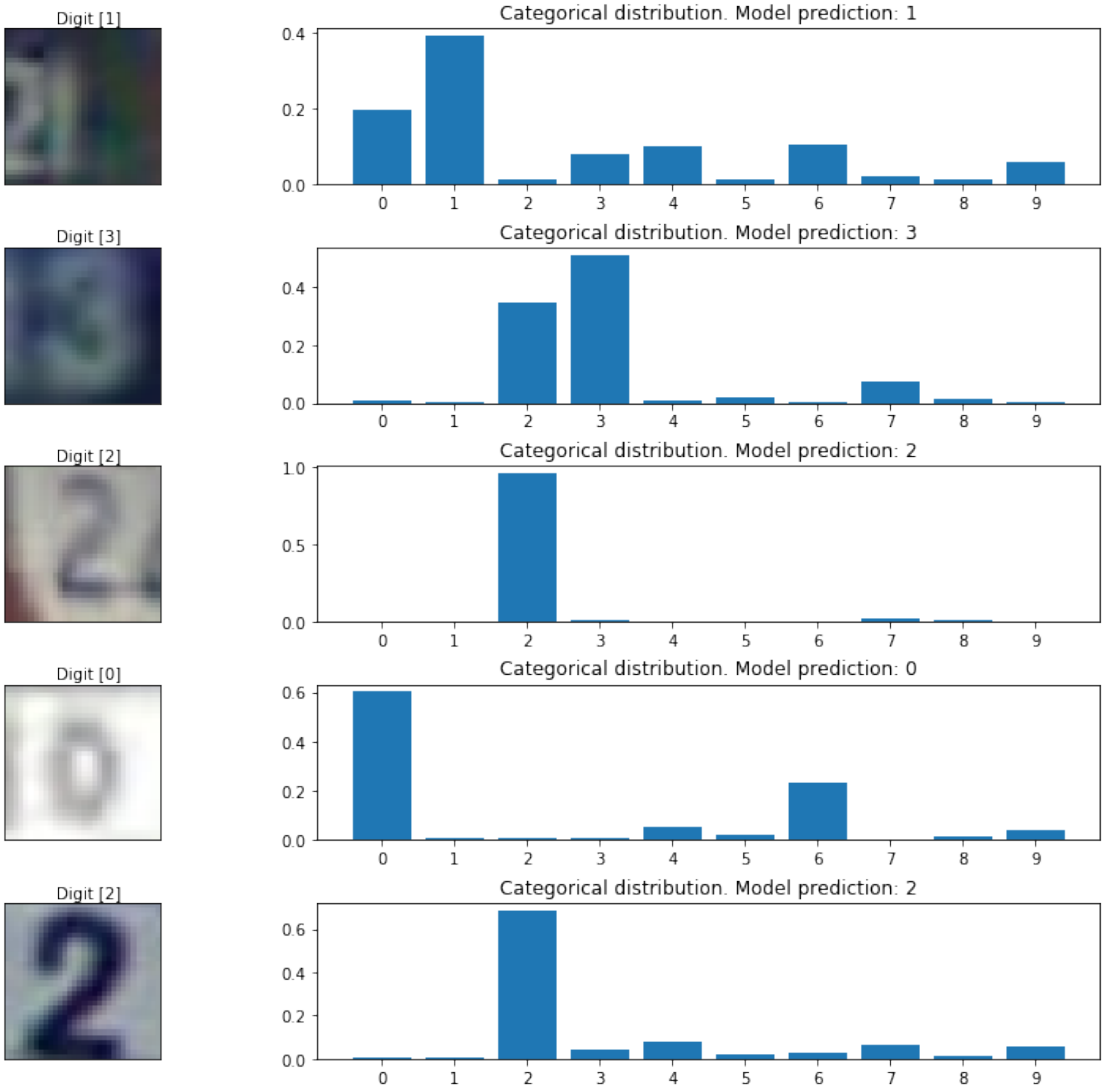
```
In [31]: best_mlp = get_new_model(train_X[0].shape)
         best_mlp.load_weights(checkpoint_path)
         best_cnn = get_new_cnn_model(train_X[0].shape)
         best_cnn.load_weights(cnn_checkpoint_path)
```

```
Out[31]: <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7fb2d87f3908>
```

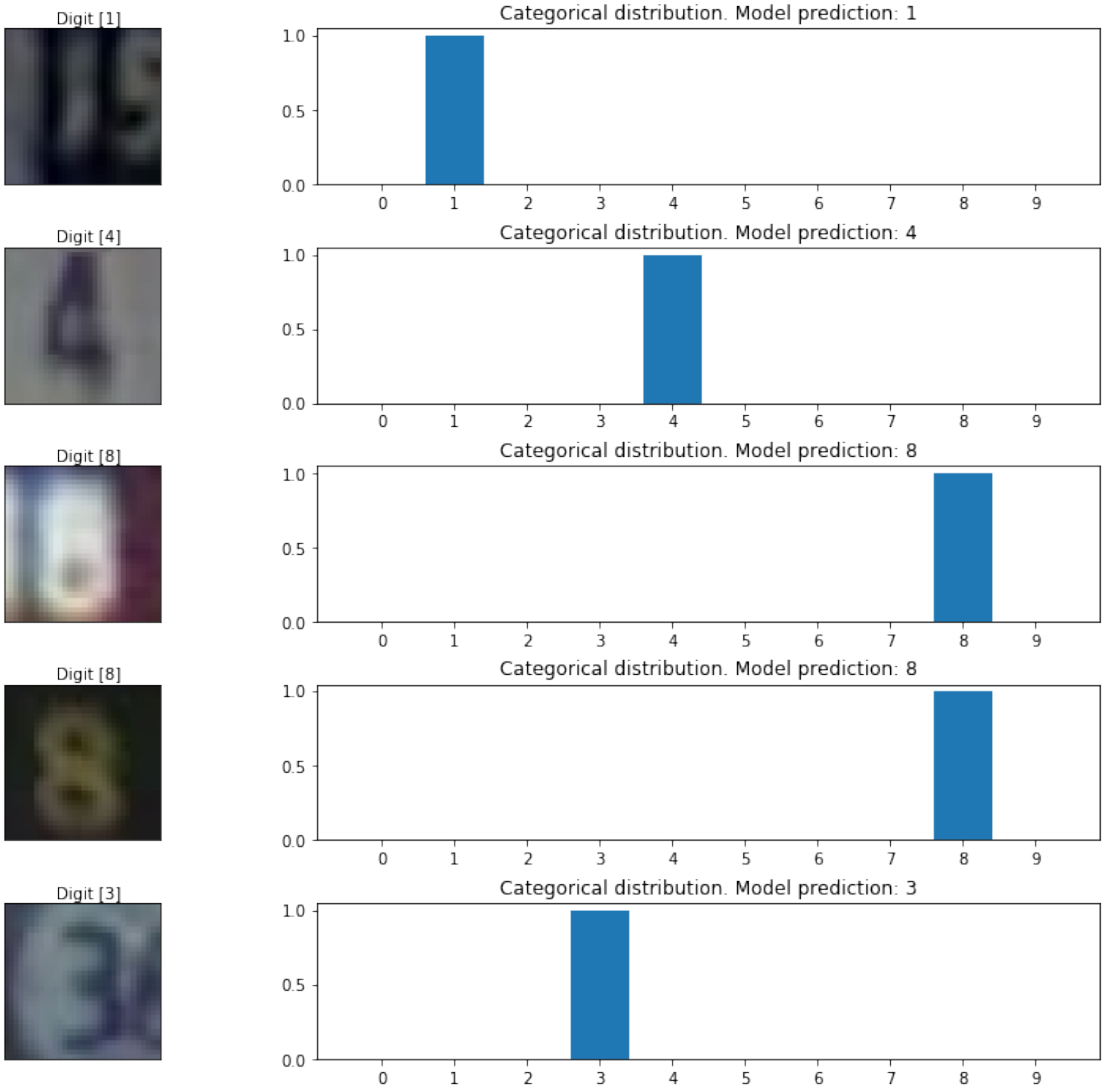
```
In [32]: def show_predictive_distribution(model):
```

```
    num_test_images = test_X.shape[0]

    random_inx = np.random.choice(num_test_images, 5)
    random_test_images = test_X[random_inx, ...]
    random_test_labels = test_y[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 [33]:
```
show_predictive_distribution(best_mlp)
```



In [34]: `show_predictive_distribution(best_cnn)`


Digit [1]
Categorical distribution. Model prediction: 1
Digit [4]
Categorical distribution. Model prediction: 4
Digit [8]
Categorical distribution. Model prediction: 8
Digit [8]
Categorical distribution. Model prediction: 8
Digit [3]
Categorical distribution. Model prediction: 3
1.0
0.5
0.0
0 1 2 3 4 5 6 7 8 9

In [ ]: