Capstone Project

May 25, 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
```

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]: import tensorflow as tf
        from tensorflow.keras.preprocessing.image import load_img, img_to_array
        from tensorflow.keras.models import Sequential, load_model
        from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
        from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
        import os
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline

In [4]: def load_SVHN_data():
            x_train = train['X'].transpose(3,0,1,2)
            y_train = train['y']
            x_test = test['X'].transpose(3,0,1,2)
            y_test = test['y']
            y_train = np.where(y_train==10, 0, y_train)
            y_test = np.where(y_test==10, 0, y_test)


            return (x_train,y_train), (x_test,y_test)
        (x_train, y_train), (x_test, y_test) = load_SVHN_data()
```

```
        # x_train.shape
        # x_test.shape
```

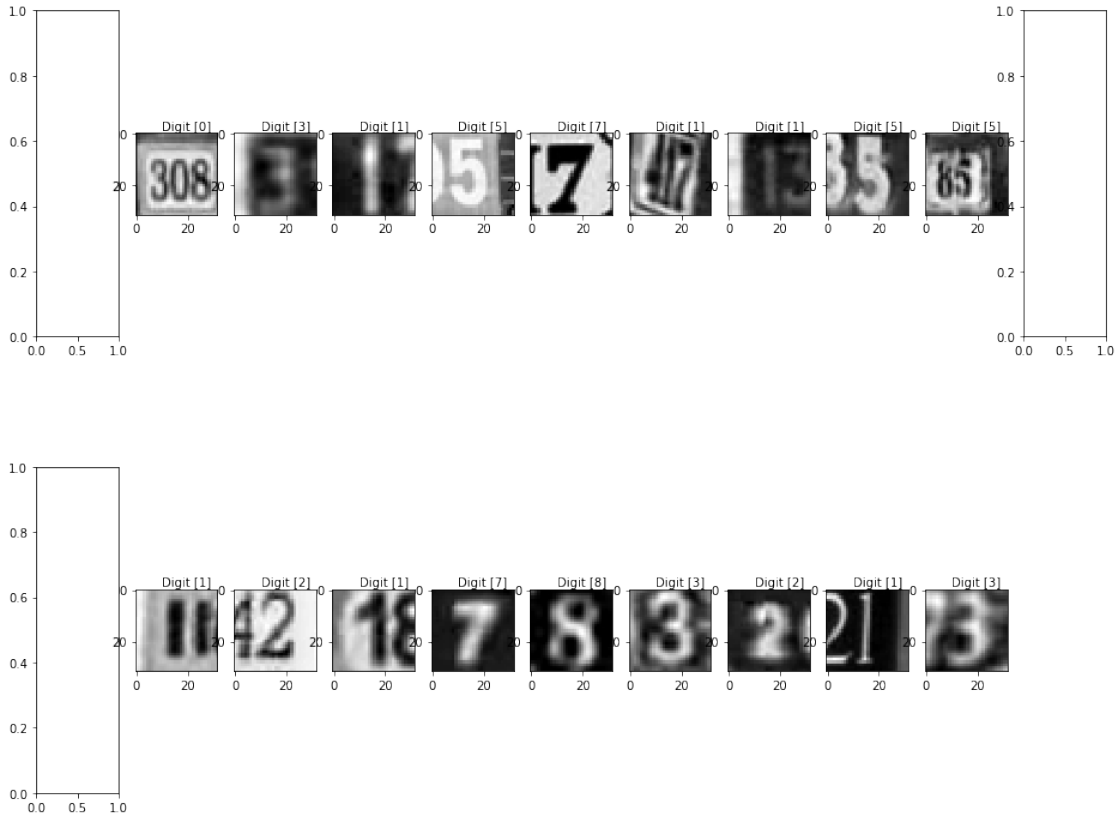
In [23]:
```
fig, axes = plt.subplots(2, 11,figsize=(16, 12))
fig.subplots_adjust(hspace=0.4, wspace=0.2)
plt.axis('off')
for i in range(1,10):
    random_inx = np.random.choice(x_train.shape[0])
#     plt.imshow(x_train[random_inx])
#     plt.show()
    axes[0,i].imshow(x_train[random_inx])

    axes[0, i].text(10., -1.5, f'Digit {y_train[random_inx]}')

    random_inx = np.random.choice(x_test.shape[0])
    axes[1,i].imshow(x_test[random_inx])

    axes[1, i].text(10., -1.5, f'Digit {y_test[random_inx]}')
```

In [5]:
```
x_train_gray = np.dot(x_train, [0.2989, 0.5870, 0.1140])
```

```
        x_test_gray = np.dot(x_test, [0.2989, 0.5870, 0.1140])
        x_train_gray.shape
```

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

```
In [50]: fig, axes = plt.subplots(2, 11,figsize=(16, 12))
         fig.subplots_adjust(hspace=0.4, wspace=0.2)
         plt.axis('off')
         for i in range(1,10):
             random_inx = np.random.choice(x_train.shape[0])

             axes[0,i].imshow(x_train_gray[random_inx], cmap=plt.cm.binary)

             axes[0, i].text(10., -1.5, f'Digit {y_train[random_inx]}')

             random_inx = np.random.choice(x_test.shape[0])
             axes[1,i].imshow(x_test_gray[random_inx], cmap=plt.cm.binary)

             axes[1, i].text(10., -1.5, f'Digit {y_test[random_inx]}')
```




```
In [6]: x_train_gray = x_train_gray[...,np.newaxis]/255.0
        x_test_gray = x_test_gray[...,np.newaxis]/255.0
```

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 [25]: from tensorflow.keras import regularizers
```

```
In [48]: batch_normalization = tf.keras.layers.BatchNormalization(
             momentum=0.95,
             epsilon=0.005,
             axis = -1,
             beta_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.05),
             gamma_initializer=tf.keras.initializers.Constant(value=0.9)
         )

         def get_model():
         #     model = Sequential([
         #         Dense(64, activation = 'relu',kernel_initializer='random_normal',bias_initial
         #         Flatten(),
         #         Dense(64, activation = 'relu'),
         #         Dense(64, activation = 'relu'),
         #         Dense(64, activation = 'relu'),
         # #         Dense(128, activation = 'relu'),
         #         Dense(64, activation = 'relu'),
         #        Dense(10,activation = 'softmax')

         #     ])
             model = Sequential([
                 Dense(64, activation = 'relu',
                 kernel_regularizer = regularizers.l2(1e-5),
                 kernel_initializer='random_normal',
                 bias_initializer='ones',
                 input_shape = (32,32,1)),
                 batch_normalization,
                 Flatten(),
                 Dense(64, activation = 'relu'),
                 Dense(64, activation = 'relu'),
```

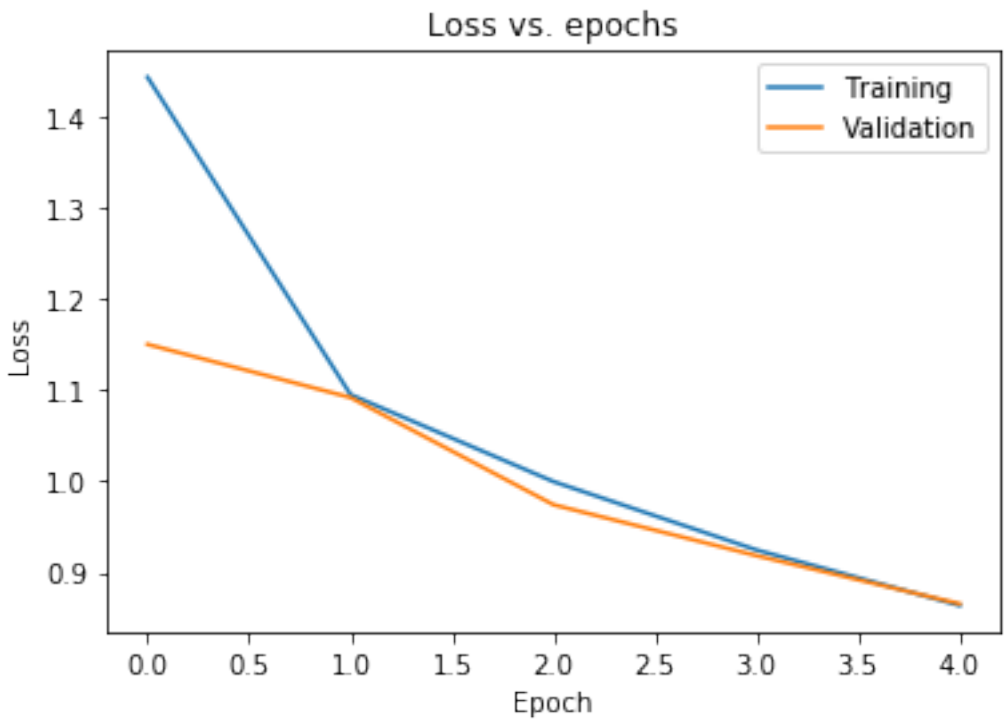
```
        #         Flatten(),
                Dense(64, activation = 'relu'),
        #         Flatten(),
                Dense(10,activation = 'softmax')
            ])
            model.compile(
                optimizer = 'Adam',
                loss = 'sparse_categorical_crossentropy',
                metrics= ['accuracy']
            )
            return model
```

In [27]:
```
model = get_model()
model.summary()
```

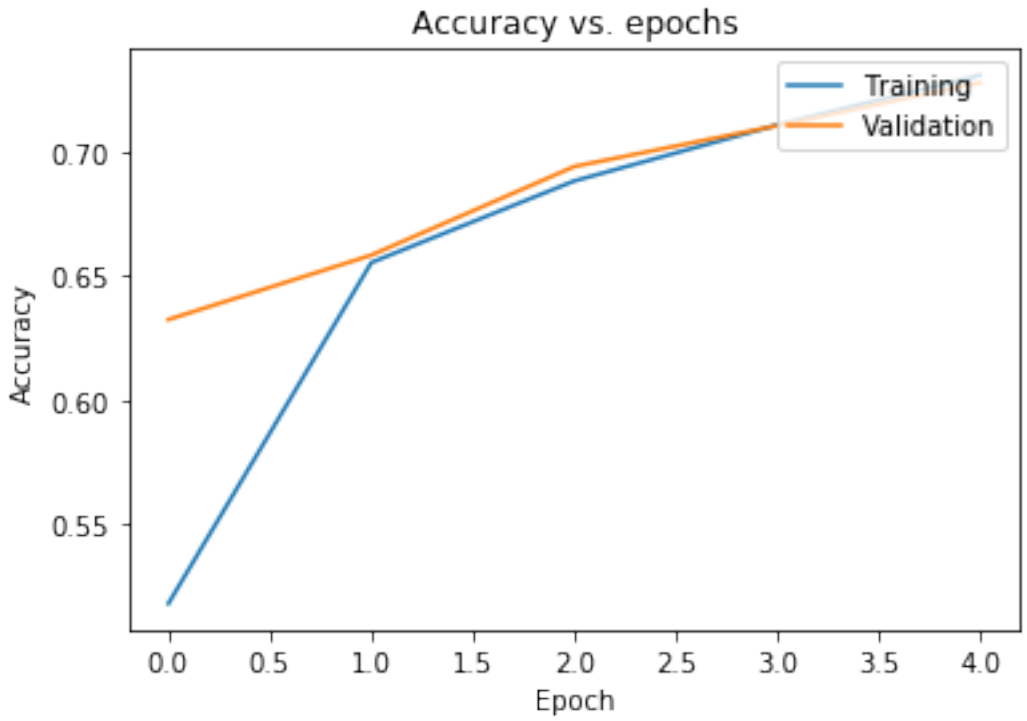
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32, 32, 64)        128
_________________________________________________________________
batch_normalization_3 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
flatten (Flatten)            (None, 65536)             0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4194368
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_3 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_4 (Dense)              (None, 10)                650
=================================================================
Total params: 4,203,722
Trainable params: 4,203,594
Non-trainable params: 128
_________________________________________________________________
```

In [16]:
```
from tensorflow.keras.callbacks import ModelCheckpoint
learning_rate_reduction=tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss',factor=
# Create Tensorflow checkpoint object
model_checkpoints = "model_checkpoints/checkpoint"
checkpoint = ModelCheckpoint(
                filepath = model_checkpoints,
                frequency = 'epoch',
                save_weights_only = True,
                verbose = 1
            )
```

```
In [17]: history = model.fit(x_train_gray, y_train, epochs =5, batch_size = 32, validation_split
                   verbose =2, callbacks = [learning_rate_reduction, checkpoint])

Train on 62268 samples, validate on 10989 samples
Epoch 1/5

Epoch 00001: saving model to model_checkpoints/checkpoint
62268/62268 - 837s - loss: 1.4425 - accuracy: 0.5182 - val_loss: 1.1499 - val_accuracy: 0.6325
Epoch 2/5

Epoch 00002: saving model to model_checkpoints/checkpoint
62268/62268 - 820s - loss: 1.0949 - accuracy: 0.6555 - val_loss: 1.0917 - val_accuracy: 0.6585
Epoch 3/5

Epoch 00003: saving model to model_checkpoints/checkpoint
62268/62268 - 823s - loss: 0.9996 - accuracy: 0.6883 - val_loss: 0.9742 - val_accuracy: 0.6941
Epoch 4/5

Epoch 00004: saving model to model_checkpoints/checkpoint
62268/62268 - 808s - loss: 0.9247 - accuracy: 0.7109 - val_loss: 0.9185 - val_accuracy: 0.7106
Epoch 5/5

Epoch 00005: saving model to model_checkpoints/checkpoint
62268/62268 - 808s - loss: 0.8639 - accuracy: 0.7309 - val_loss: 0.8658 - val_accuracy: 0.7279


In [18]: import matplotlib.pyplot as plt
         %matplotlib inline

In [19]: # Plot the training and validation loss

         plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
         plt.title('Loss vs. epochs')
         plt.ylabel('Loss')
         plt.xlabel('Epoch')
         plt.legend(['Training', 'Validation'], loc='upper right')
         plt.show()
```



In [20]:
```
# Plot the training and validation loss
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Accuracy vs. epochs')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```




```
In [21]: model.load_weights(model_checkpoints)
         model.evaluate(x_test_gray,y_test, verbose=2)
```

```
26032/1 - 73s - loss: 0.9161 - accuracy: 0.7050
```

```
Out[21]: [0.967449000186545, 0.7050169]
```

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 [7]: # Introduce function that creates a new instance of a simple CNN

        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout
        from tensorflow.keras import regularizers
        batch_normalization = tf.keras.layers.BatchNormalization(
            momentum=0.95,
            epsilon=0.005,
            axis = -1,
            beta_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.05),
            gamma_initializer=tf.keras.initializers.Constant(value=0.9)
        )
        def get_new_model():
            model = Sequential([
                Conv2D(filters=16, input_shape=(32, 32, 1), kernel_size=(3, 3),
                       activation='relu', name='conv_1'),
                Conv2D(filters=8, kernel_size=(3, 3), activation='relu', name='conv_2'),
                MaxPooling2D(pool_size=(4, 4), name='pool_1'),
                batch_normalization,
                Flatten(name='flatten'),
                Dense(units=32, kernel_regularizer = regularizers.l2(1e-5), activation='relu', n
                Dropout(0.2),
                Dense(units=10, activation='softmax', name='dense_2')
            ])
            model.compile(optimizer='adam',
                          loss='sparse_categorical_crossentropy',
                          metrics=['accuracy'])
            return model

In [8]: model = get_new_model()
        model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_1 (Conv2D)              (None, 30, 30, 16)        160
_________________________________________________________________
conv_2 (Conv2D)              (None, 28, 28, 8)         1160
_________________________________________________________________
pool_1 (MaxPooling2D)        (None, 7, 7, 8)           0
_________________________________________________________________
batch_normalization (BatchNo (None, 7, 7, 8)           32
_________________________________________________________________
flatten (Flatten)            (None, 392)               0
_________________________________________________________________
```

```
dense_1 (Dense)              (None, 32)                12576
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                330
=================================================================
Total params: 14,258
Trainable params: 14,242
Non-trainable params: 16
_________________________________________________________________
```

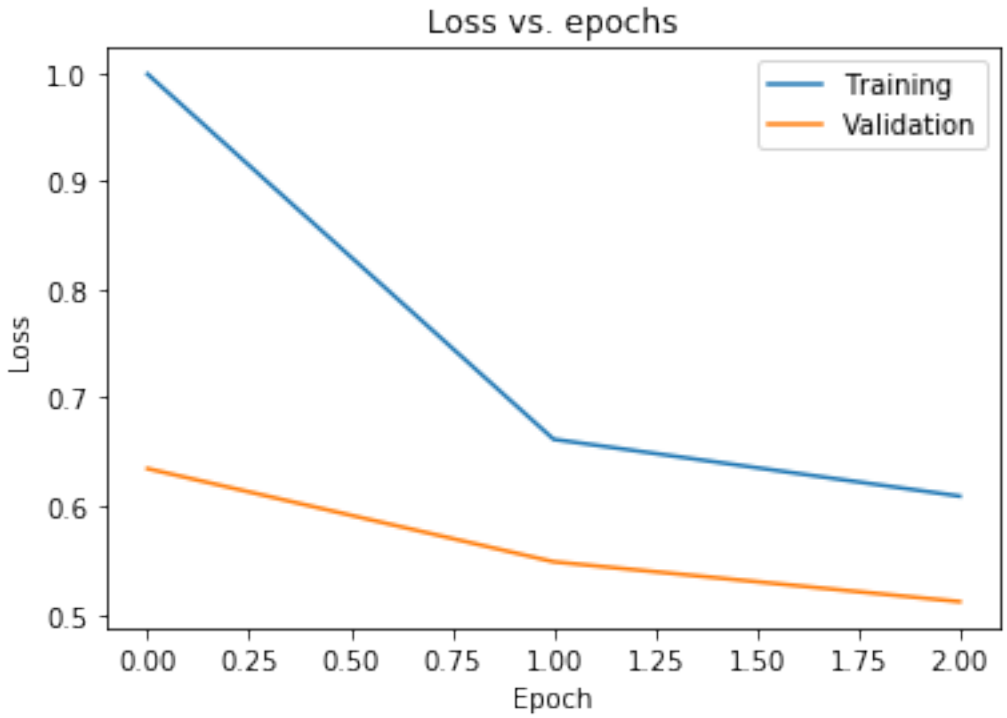
```
In [10]: from tensorflow.keras.callbacks import ModelCheckpoint
         learning_rate_reduction=tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss',factor=
         # Create Tensorflow checkpoint object
         model_checkpoints = "model_checkpoints/checkpoint_cnn"
         checkpoint = ModelCheckpoint(
                         filepath = model_checkpoints,
                         frequency = 'epoch',
                         save_weights_only = True,
                         verbose = 1
                     )
```

```
In [31]: history = model.fit(x_train_gray,y_train, epochs=3,validation_split=0.15, callbacks = [

Train on 62268 samples, validate on 10989 samples
Epoch 1/3
62240/62268 [============================>.] - ETA: 0s - loss: 0.9985 - accuracy: 0.6763
Epoch 00001: saving model to model_checkpoints/checkpoint_cnn
62268/62268 [==============================] - 294s 5ms/sample - loss: 0.9986 - accuracy: 0.6763
Epoch 2/3
62240/62268 [============================>.] - ETA: 0s - loss: 0.6615 - accuracy: 0.7972
Epoch 00002: saving model to model_checkpoints/checkpoint_cnn
62268/62268 [==============================] - 290s 5ms/sample - loss: 0.6616 - accuracy: 0.7972
Epoch 3/3
62240/62268 [============================>.] - ETA: 0s - loss: 0.6089 - accuracy: 0.8121
Epoch 00003: saving model to model_checkpoints/checkpoint_cnn
62268/62268 [==============================] - 289s 5ms/sample - loss: 0.6091 - accuracy: 0.8121
```

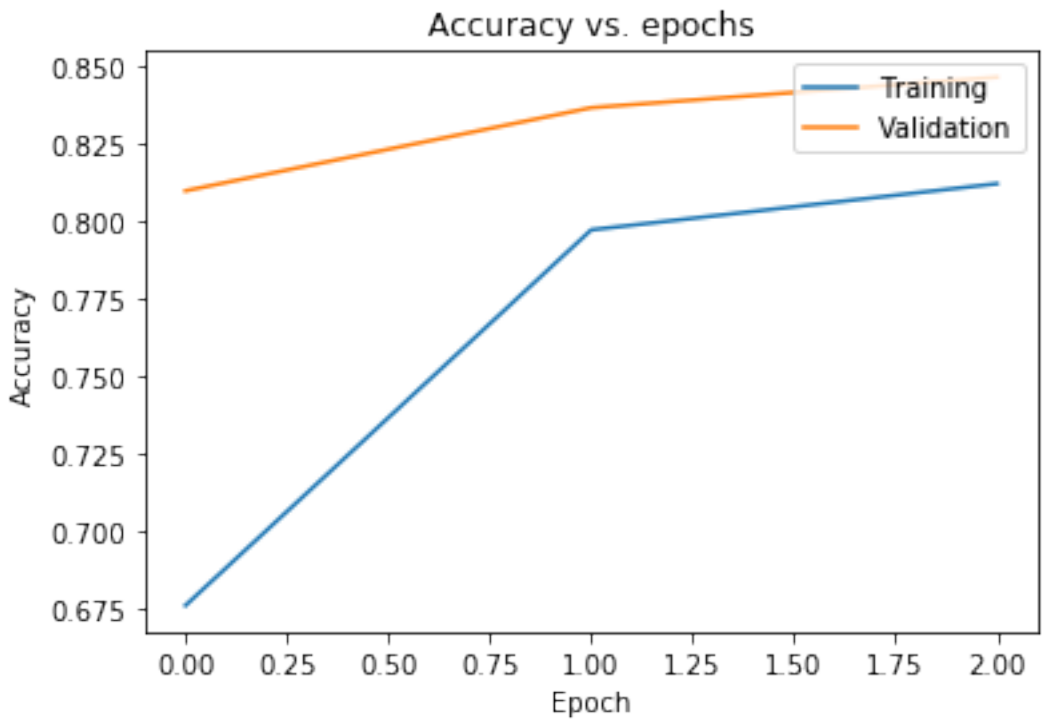
```
In [32]: import matplotlib.pyplot as plt
         %matplotlib inline
```

```
In [33]: # Plot the training and validation loss

         plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
         plt.title('Loss vs. epochs')
         plt.ylabel('Loss')
```

```
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```



In [34]:
```
# Plot the training and validation loss
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Accuracy vs. epochs')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.show()
```




```
In [11]: model.load_weights(model_checkpoints)
         model.evaluate(x_test_gray,y_test, verbose = 2)
```

```
26032/1 - 35s - loss: 0.4831 - accuracy: 0.8283
```

```
Out[11]: [0.5879021384321111, 0.82828826]
```

1.5 4. Get model predictions

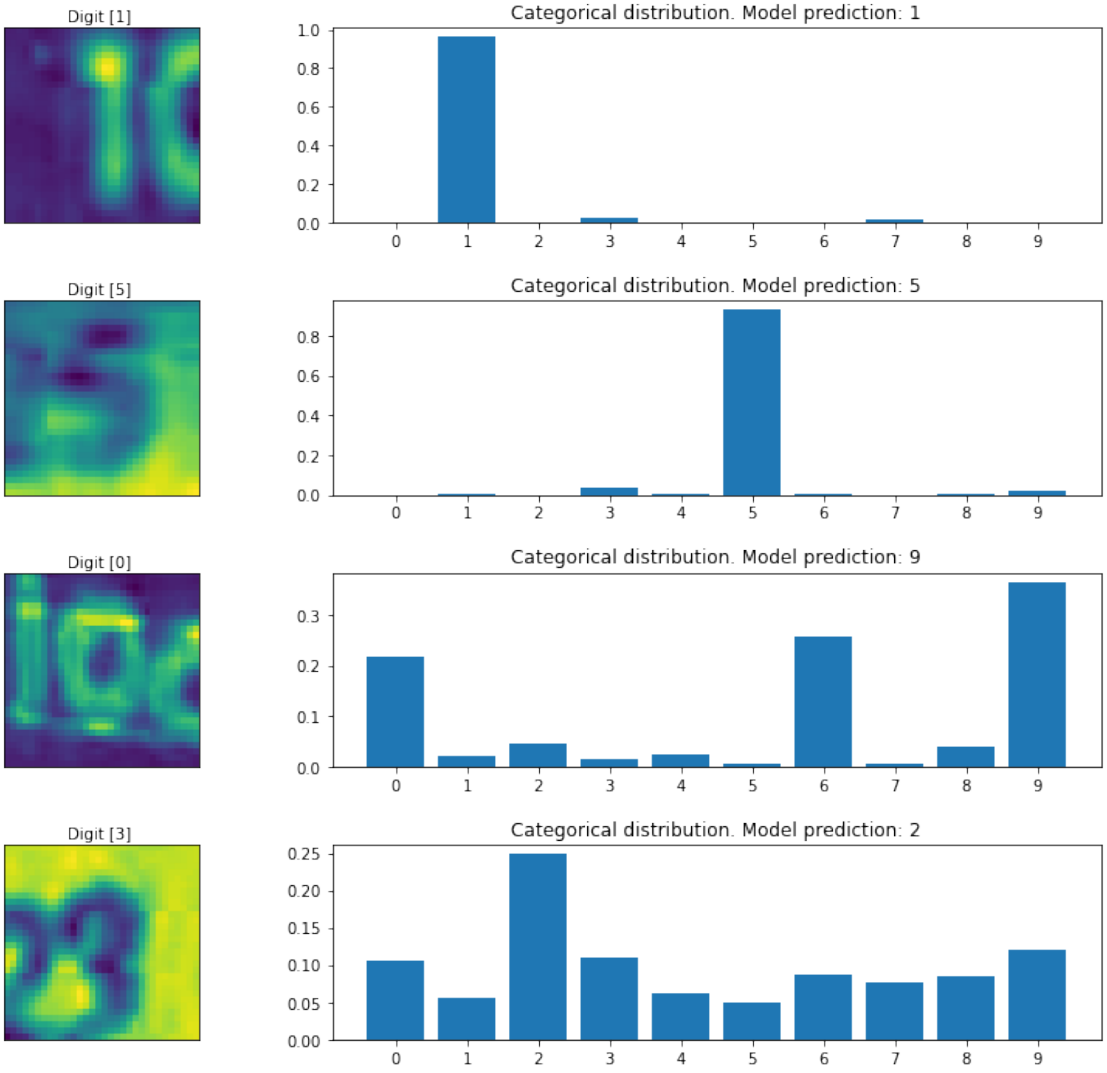
- Load the best weights for the MLP and CNN models that you saved during the training run.
- Randomly select 5 images and corresponding labels from the test set and display the images with their labels.
- Alongside the image and label, show each model's predictive distribution as a bar chart, and the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

```
In [41]: model = get_new_model()
         model.load_weights(model_checkpoints)
```

```
Out[41]: <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f4840779f98>
```

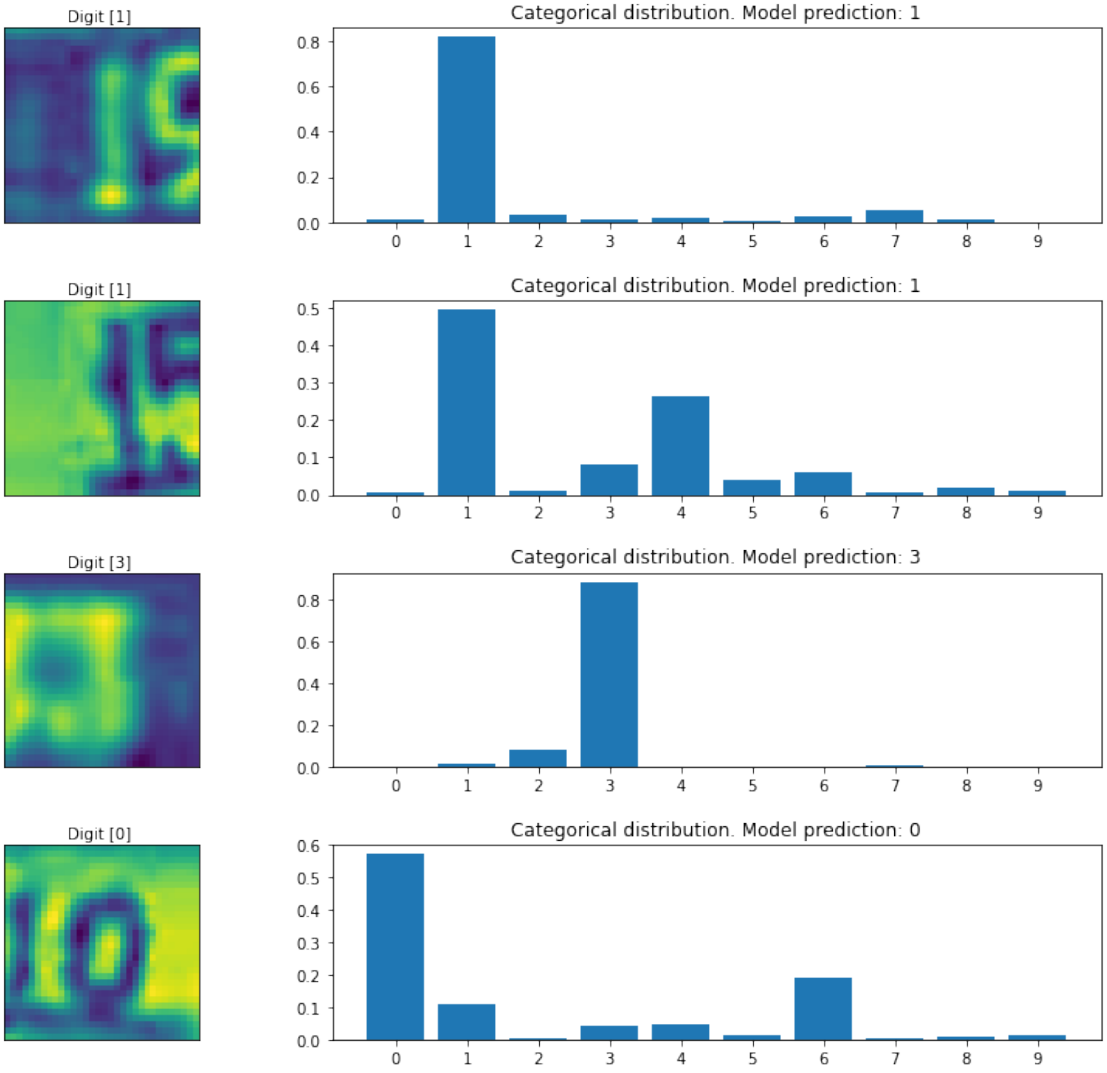
```
In [40]: # Run this cell to get model predictions on randomly selected test images

         num_test_images = x_test_gray.shape[0]
```

```
random_inx = np.random.choice(num_test_images, 4)
random_test_images = x_test_gray[random_inx, ...]
random_test_labels = y_test[random_inx, ...]

predictions = model.predict(random_test_images)

fig, axes = plt.subplots(4, 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, ran
    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(predi

plt.show()
```




```
In [54]: model_checkpoints_mlp = "model_checkpoints/checkpoint"

In [55]: model=get_model()
         model.load_weights(model_checkpoints_mlp)

Out[55]: <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f48dd2ec978>

In [56]: # Run this cell to get model predictions on randomly selected test images

         num_test_images = x_test_gray.shape[0]

         random_inx = np.random.choice(num_test_images, 4)
         random_test_images = x_test_gray[random_inx, ...]
         random_test_labels = y_test[random_inx, ...]
```

```
predictions = model.predict(random_test_images)

fig, axes = plt.subplots(4, 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, ran
    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(predi

plt.show()
```



In [ ]: