Capstone Project

February 7, 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
        import random
        from scipy.io import loadmat
        from matplotlib.pyplot import figure, imshow, axis,subplots
        import numpy as np
```

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')
```

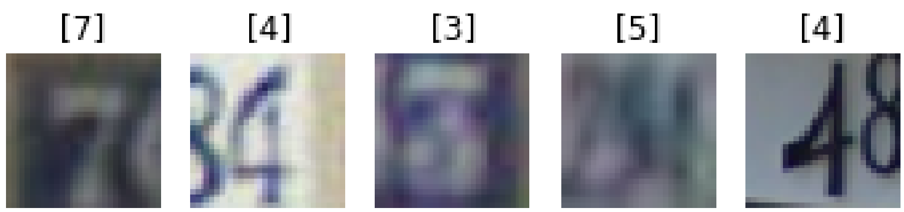
Both `train` and `test` are dictionaries with keys `X` and `y` for the input images and labels respectively.

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 [37]:
```
# extract data and labels
x_train =  train['X']
y_train =  train['y'] - 1
x_test =  test['X']
y_test =  test['y'] - 1      #convert to 0-indexed
```

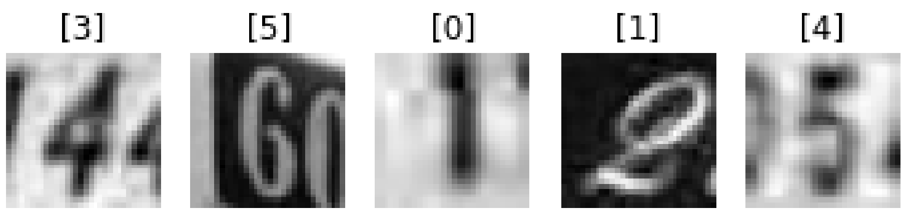
In [39]:
```
# display 5 random figures from traning set and 5 random figures from test set
train_sample_size = 5
test_sample_size = 5
train_size = x_train.shape[3]
test_size = x_test.shape[3]
random_train_index = random.sample(range(0, train_size), train_sample_size)
random_test_index = random.sample(range(0, test_size), test_sample_size)
_, train_axis = subplots(1,train_sample_size)
_, test_axis = subplots(1,test_sample_size)
# display
for i in range(0, train_sample_size):
    train_axis[i].imshow(x_train[:,:,:,random_train_index[i]])
    train_axis[i].axis('off')
    train_axis[i].set_title(str(y_train[random_train_index[i]]+1))
for i in range(0, test_sample_size):
    test_axis[i].imshow(x_test[:,:,:,random_test_index[i]])
    test_axis[i].axis('off')
    test_axis[i].set_title(str(y_test[random_test_index[i]]+1))
```

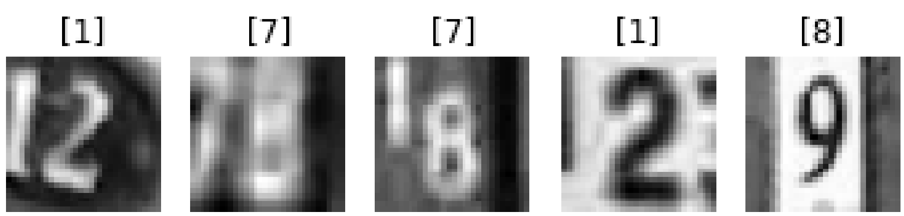


In [43]:
```
# compute average along color channel.
x_train = np.mean(x_train, axis=2, keepdims=True)
x_test = np.mean(x_test, axis=2, keepdims=True)
```

In [44]:
```
# display 5 random figures from traning set and 5 random figures from test set
train_sample_size = 5
```

```
test_sample_size = 5
train_size = x_train.shape[3]
test_size = x_test.shape[3]
random_train_index = random.sample(range(0, train_size), train_sample_size)
random_test_index = random.sample(range(0, test_size), test_sample_size)
_, train_axis = subplots(1,train_sample_size)
_, test_axis = subplots(1,test_sample_size)
train_axis[0].set_axis_off()
# display
for i in range(0, train_sample_size):
    train_axis[i].imshow(x_train[:,:,0,random_train_index[i]],cmap="gray")
    train_axis[i].axis('off')
    train_axis[i].set_title(str(y_train[random_train_index[i]]))
for i in range(0, test_sample_size):
    test_axis[i].imshow(x_test[:,:,0,random_test_index[i]],cmap="gray")
    test_axis[i].axis('off')
    test_axis[i].set_title(str(y_test[random_test_index[i]]))
```





In [45]:
```
# move the input data axis to the first place
x_train = np.moveaxis(x_train, 3, 0)
x_test = np.moveaxis(x_test, 3, 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 [8]:
```
# tf imports
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

In [9]:
```
def build_model(input_shape):
    model = Sequential([
        Flatten(input_shape=input_shape,name='flatten_1'),
        Dense(512, activation='relu',name='dense_1'),
        Dense(256, activation='relu',name='dense_2'),
        Dense(10, activation='softmax',name='output')
    ])
    #compile
    model.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy
    return model
```

In [10]:
```
#get input shape from one image
input_shape = x_train.shape[1:]
model = build_model(input_shape)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
output (Dense)               (None, 10)                2570
=================================================================
Total params: 658,698
```

```
Trainable params: 658,698
Non-trainable params: 0
_________________________________________________________________
```

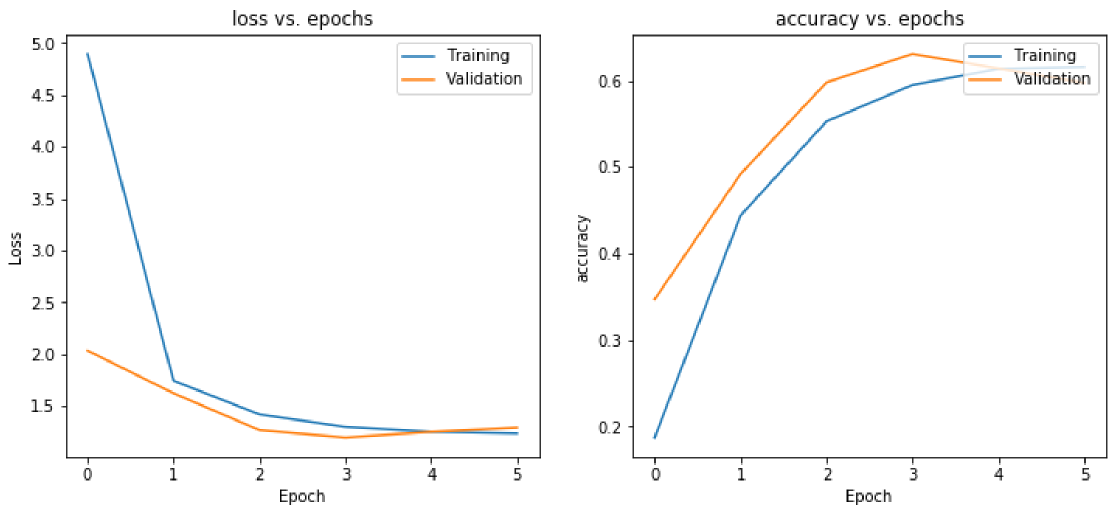
In [11]:
```
# save callbacks
best_ckpt_path = 'model_checkpoint_best/checkpoint'
best_ckpt = ModelCheckpoint(filepath= best_ckpt_path,
                            save_weights_only=True,
                            monitor='val_accuracy',
                            save_best_only=True,
                            verbose=0)

# stopping callbacks
early_stop = EarlyStopping(patience=2)
```

In [13]:
```
# training phase
history= model.fit(x_train,y_train, epochs=30,validation_split=0.15, batch_size=64,ve
                  callbacks=[best_ckpt,early_stop])
```

```
Train on 62268 samples, validate on 10989 samples
Epoch 1/30
62268/62268 - 33s - loss: 4.8971 - accuracy: 0.1873 - val_loss: 2.0356 - val_accuracy: 0.3471
Epoch 2/30
62268/62268 - 32s - loss: 1.7459 - accuracy: 0.4438 - val_loss: 1.6254 - val_accuracy: 0.4919
Epoch 3/30
62268/62268 - 32s - loss: 1.4212 - accuracy: 0.5527 - val_loss: 1.2697 - val_accuracy: 0.5978
Epoch 4/30
62268/62268 - 32s - loss: 1.3007 - accuracy: 0.5949 - val_loss: 1.1984 - val_accuracy: 0.6308
Epoch 5/30
62268/62268 - 32s - loss: 1.2537 - accuracy: 0.6135 - val_loss: 1.2533 - val_accuracy: 0.6141
Epoch 6/30
62268/62268 - 32s - loss: 1.2381 - accuracy: 0.6158 - val_loss: 1.2935 - val_accuracy: 0.5973
```

In [14]:
```
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(12, 5))

fig.add_subplot(121)

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')

fig.add_subplot(122)
```

```
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 [28]: # evaluation on test set
         loss,accuracy = model.evaluate(x_test,y_test, verbose=2)
         print("MLP loss on test set is {}".format(loss))
         print("MLP accuracy on test set is {}".format(accuracy))
```

```
26032/1 - 6s - loss: 1.1501 - accuracy: 0.5900
MLP loss on test set is 1.4010111775292746
MLP accuracy on test set is 0.5899661779403687
```

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 [19]:
```
from tensorflow.keras.layers import Conv2D, BatchNormalization, MaxPooling2D
def build_CNN_model(input_shape):
    model = Sequential([
        Conv2D(filters=4, kernel_size=(3,3),activation='relu', padding='same', kernel_
        BatchNormalization(),          #channel last
        MaxPooling2D(pool_size=(2,2)),
        Conv2D(filters=8, kernel_size=(3,3),activation='relu', padding='same', kernel_
        BatchNormalization(),          #channel last
        MaxPooling2D(pool_size=(2,2)),
        Flatten(),
        Dense(100, activation='relu',name='dense_1'),
        Dense(10, activation='softmax',name='output')
    ])
    model.compile(optimizer='adam', loss = tf.keras.losses.SparseCategoricalCrossentr
    return model
```

In [20]:
```
#get input shape from one image
input_shape = x_train.shape[1:]
cnn_model = build_CNN_model(input_shape)
cnn_model.summary()
```

```
Model: "sequential_3"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 4)         40

_________________________________________________________________
batch_normalization_4 (Batch (None, 32, 32, 4)         16

_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 16, 16, 4)         0

_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 8)         296

_________________________________________________________________
batch_normalization_5 (Batch (None, 16, 16, 8)         32

_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 8, 8, 8)           0

_________________________________________________________________
flatten_2 (Flatten)          (None, 512)               0

_________________________________________________________________
dense_1 (Dense)              (None, 100)               51300

_________________________________________________________________
```

```
output (Dense)               (None, 10)                1010
=================================================================
Total params: 52,694
Trainable params: 52,670
Non-trainable params: 24
_________________________________________________________________
```

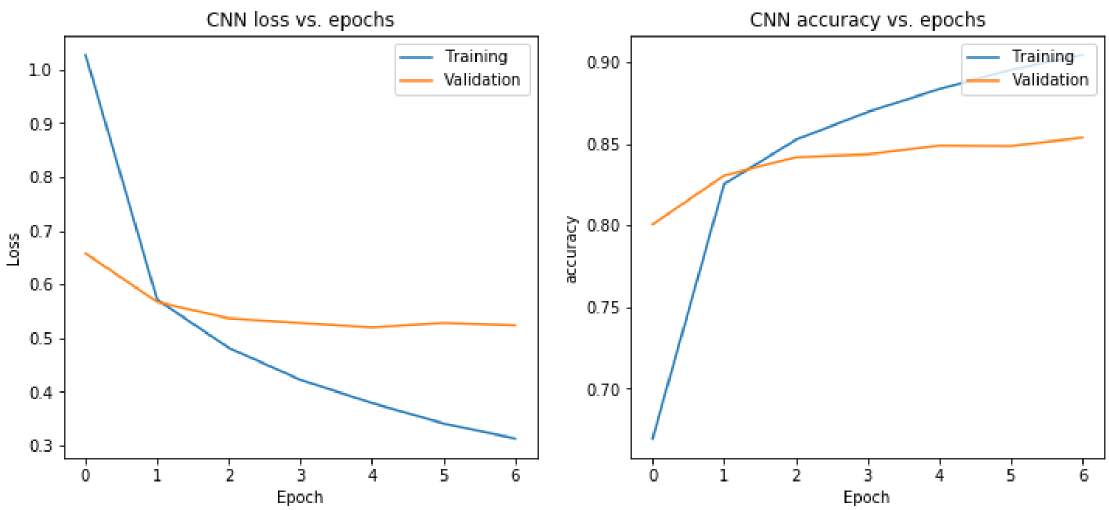
```
In [21]: # save callbacks
         cnn_best_ckpt_path = 'cnn_model_checkpoint_best/checkpoint'
         cnn_best_ckpt = ModelCheckpoint(filepath= cnn_best_ckpt_path,
                                         save_weights_only=True,
                                         monitor='val_accuracy',
                                         save_best_only=True,
                                         verbose=0)

         # stopping callbacks
         early_stop = EarlyStopping(patience=2)
```

```
In [22]: # training phase
         cnn_history= cnn_model.fit(x_train,y_train, epochs=30, validation_split=0.15, batch_s
                          callbacks=[cnn_best_ckpt,early_stop])
```

```
Train on 62268 samples, validate on 10989 samples
Epoch 1/30
62268/62268 - 178s - loss: 1.0272 - accuracy: 0.6696 - val_loss: 0.6577 - val_accuracy: 0.8006
Epoch 2/30
62268/62268 - 181s - loss: 0.5717 - accuracy: 0.8253 - val_loss: 0.5674 - val_accuracy: 0.8306
Epoch 3/30
62268/62268 - 178s - loss: 0.4819 - accuracy: 0.8524 - val_loss: 0.5369 - val_accuracy: 0.8418
Epoch 4/30
62268/62268 - 169s - loss: 0.4226 - accuracy: 0.8694 - val_loss: 0.5283 - val_accuracy: 0.8435
Epoch 5/30
62268/62268 - 168s - loss: 0.3793 - accuracy: 0.8834 - val_loss: 0.5201 - val_accuracy: 0.8488
Epoch 6/30
62268/62268 - 169s - loss: 0.3413 - accuracy: 0.8952 - val_loss: 0.5284 - val_accuracy: 0.8486
Epoch 7/30
62268/62268 - 168s - loss: 0.3133 - accuracy: 0.9043 - val_loss: 0.5239 - val_accuracy: 0.8537
```

```
In [23]: 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 loss vs. epochs')
         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 accuracy vs. epochs')
plt.ylabel('accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')

plt.show()
```




```
In [29]: # evaluation on test set
         loss,accuracy = cnn_model.evaluate(x_test,y_test, verbose=2)
         print("CNN loss on test set is {}".format(loss))
         print("CNN accuracy on test set is {}".format(accuracy))
```

```
26032/1 - 25s - loss: 0.5315 - accuracy: 0.8374
CNN loss on test set is 0.6039436107436132
CNN accuracy on test set is 0.8373540043830872
```

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]: #load best MLP
         load_MLP_model = build_model(input_shape)
         load_MLP_model.load_weights(best_ckpt_path)
         load_MLP_model.evaluate(x_test,y_test, verbose=2)
```

```
26032/1 - 6s - loss: 1.0726 - accuracy: 0.5943
```

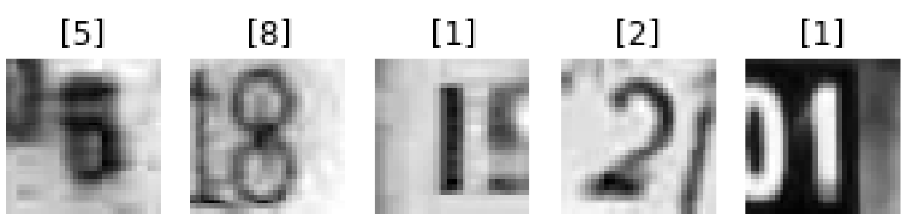
```
Out[31]: [1.376613517370265, 0.594307]
```

```
In [33]: #load best CNN
         load_CNN_model = build_CNN_model(input_shape)
         load_CNN_model.load_weights(cnn_best_ckpt_path)
         load_CNN_model.evaluate(x_test,y_test, verbose=2)
```

```
26032/1 - 23s - loss: 0.5315 - accuracy: 0.8374
```

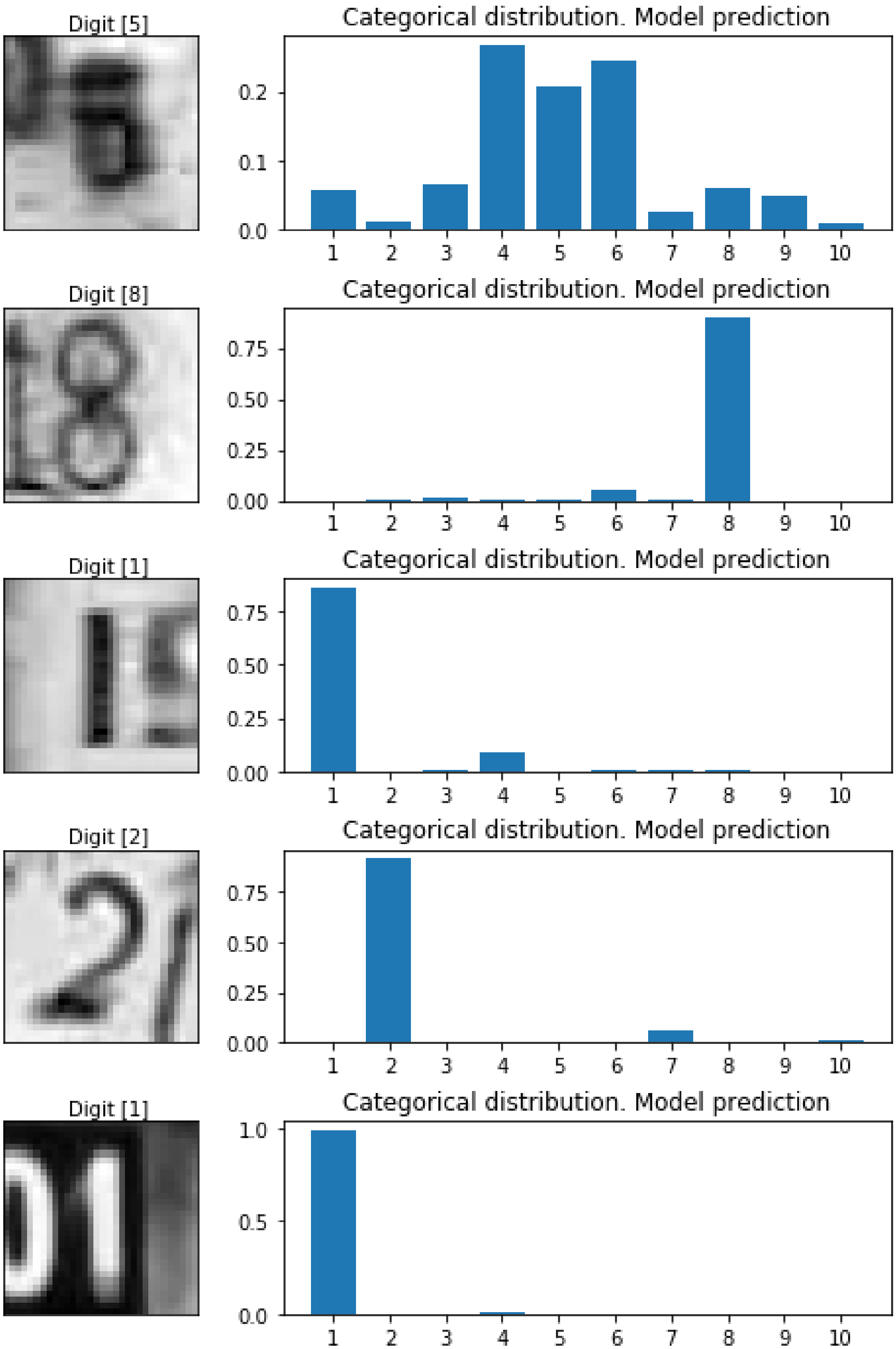
```
Out[33]: [0.6039436107436132, 0.837354]
```

```
In [54]: #random selection
         sample_size =5
         random_test_index_pred = random.sample(range(0, test_size), sample_size)
         _, pred_axis = subplots(1,sample_size)

         for i in range(0, sample_size):
             pred_axis[i].imshow(x_test[random_test_index_pred[i],:,:,0],cmap="gray")
             pred_axis[i].axis('off')
             pred_axis[i].set_title(str(y_test[random_test_index_pred[i]]+1))
```




```
In [79]: #display MLP rediction result
         pred_image_set = x_test[random_test_index_pred,:,:,:]
         pred_label_set = y_test[random_test_index_pred,:]
         result = load_MLP_model.predict(pred_image_set)
         predictions = np.argmax(result,axis=1)+1
         fig, axes = plt.subplots(5, 2, figsize=(10, 12))
         fig.subplots_adjust(hspace=0.4, wspace=-0.2)
```

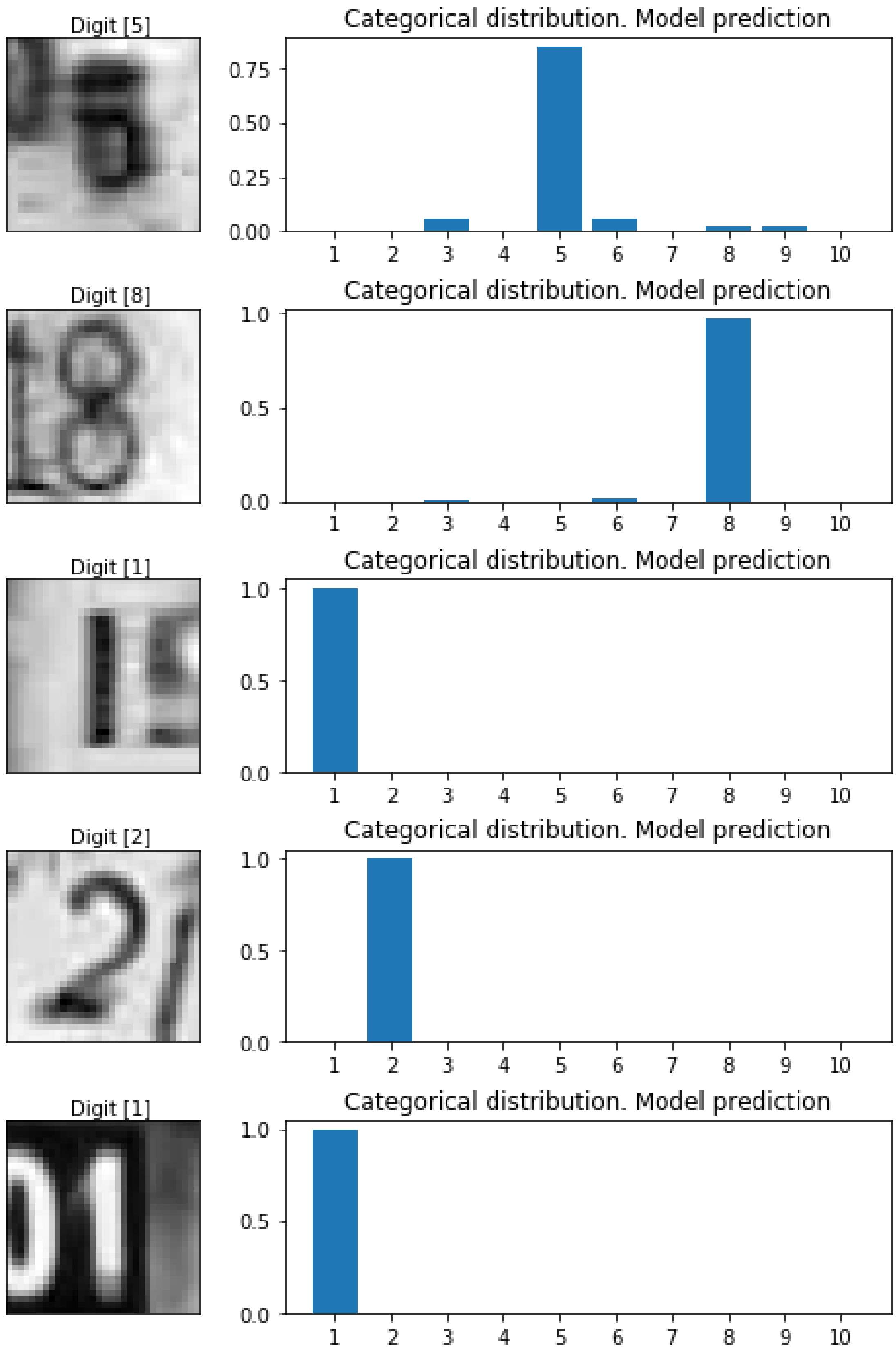
```
for i, (prediction, image, label) in enumerate(zip(result, pred_image_set, pred_label_
    axes[i, 0].imshow(np.squeeze(image),cmap="gray")
    axes[i, 0].get_xaxis().set_visible(False)
    axes[i, 0].get_yaxis().set_visible(False)
    axes[i, 0].text(10., -1.5, f'Digit {label+1}')
    axes[i, 1].bar(np.arange(1,11), prediction)
    axes[i, 1].set_xticks(np.arange(1,11))
    axes[i, 1].set_title("Categorical distribution. Model prediction")

plt.show()
print("predictions:{}".format(predictions))
```



Digit [5]
Categorical distribution. Model prediction
0.0
0.1
0.2
1
2
3
4
5
6
7
8
9
10
Digit [8]
Categorical distribution. Model prediction
0.00
0.25
0.50
0.75
1
2
3
4
5
6
7
8
9
10
Digit [1]
Categorical distribution. Model prediction
0.00
0.25
0.50
0.75
1
2
3
4
5
6
7
8
9
10
Digit [2]
Categorical distribution. Model prediction
0.00
0.25
0.50
0.75
1
2
3
4
5
6
7
8
9
10
Digit [1]
Categorical distribution. Model prediction
0.0
0.5
1.0
1
2
3
4
5
6
7
8
9
10

```
predictions:[4 8 1 2 1]
```

```
In [80]: #display CNN rediction result
         cnn_result = load_CNN_model.predict(pred_image_set)
         cnn_prediction = np.argmax(cnn_result,axis=1)+1
         fig, axes = plt.subplots(5, 2, figsize=(10, 12))
         fig.subplots_adjust(hspace=0.4, wspace=-0.2)

         for i, (prediction, image, label) in enumerate(zip(cnn_result, pred_image_set, pred_la
             axes[i, 0].imshow(np.squeeze(image),cmap="gray")
             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+1}')
             axes[i, 1].bar(np.arange(1,11), prediction)
             axes[i, 1].set_xticks(np.arange(1,11))
             axes[i, 1].set_title("Categorical distribution. Model prediction")

         plt.show()
         print("predictions:{}".format(cnn_prediction))
```


Digit [5]
Categorical distribution. Model prediction
Digit [8]
Categorical distribution. Model prediction
Digit [1]
Categorical distribution. Model prediction
Digit [2]
Categorical distribution. Model prediction
Digit [1]
Categorical distribution. Model prediction

```
predictions:[5 8 1 2 1]
```