Capstone Project

Image classifier for the SVHN dataset

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.

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.

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.

```
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.

```
# Run this cell to load the dataset
train_url = 'http://ufldl.stanford.edu/housenumbers/train_32x32.mat'
test_url =  'http://ufldl.stanford.edu/housenumbers/test_32x32.mat'
# train = loadmat('data/train_32x32.mat')
# test = loadmat('data/test_32x32.mat')
```

```
!wget http://ufldl.stanford.edu/housenumbers/train_32x32.mat
!wget http://ufldl.stanford.edu/housenumbers/test_32x32.mat
```

```
--2021-01-24 17:24:10--  http://ufldl.stanford.edu/housenumbers/train_32x32
Resolving ufldl.stanford.edu (ufldl.stanford.edu)... 171.64.68.10
Connecting to ufldl.stanford.edu (ufldl.stanford.edu)|171.64.68.10|:80... c
HTTP request sent, awaiting response... 200 OK
Length: 182040794 (174M) [text/plain]
Saving to: ‘train_32x32.mat’

train_32x32.mat     100%[===================>] 173.61M  76.9MB/s    in 2.3s

2021-01-24 17:24:12 (76.9 MB/s) - ‘train_32x32.mat’ saved [182040794/182040

--2021-01-24 17:24:12--  http://ufldl.stanford.edu/housenumbers/test_32x32.
Resolving ufldl.stanford.edu (ufldl.stanford.edu)... 171.64.68.10
Connecting to ufldl.stanford.edu (ufldl.stanford.edu)|171.64.68.10|:80... c
HTTP request sent, awaiting response... 200 OK
Length: 64275384 (61M) [text/plain]
Saving to: ‘test_32x32.mat’

test_32x32.mat      100%[===================>]  61.30M  52.8MB/s    in 1.2s

2021-01-24 17:24:13 (52.8 MB/s) - ‘test_32x32.mat’ saved [64275384/64275384
```

```
train = loadmat('train_32x32.mat')
test = loadmat('test_32x32.mat')
```

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

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.

```
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.layers import Dropout, BatchNormalization
from tensorflow.keras import regularizers
import random
```

```
train_images, train_labels = train['X'], train['y']
```

```
test_images, test_labels = test['X'], test['y']
```

```
train_labels[0]
```

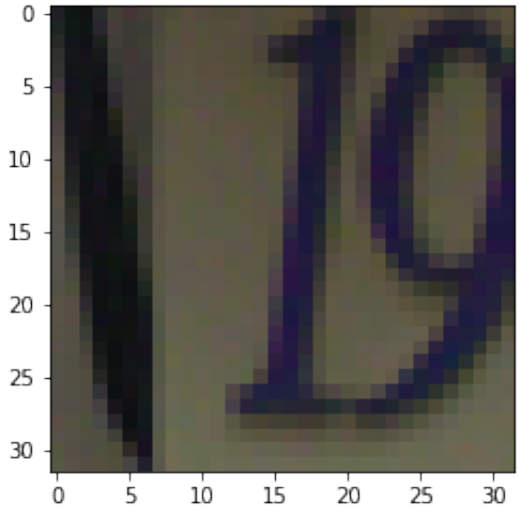
```
array([1], dtype=uint8)
```

```
train_images.shape,train_labels.shape
```

```
((32, 32, 3, 73257), (73257, 1))
```

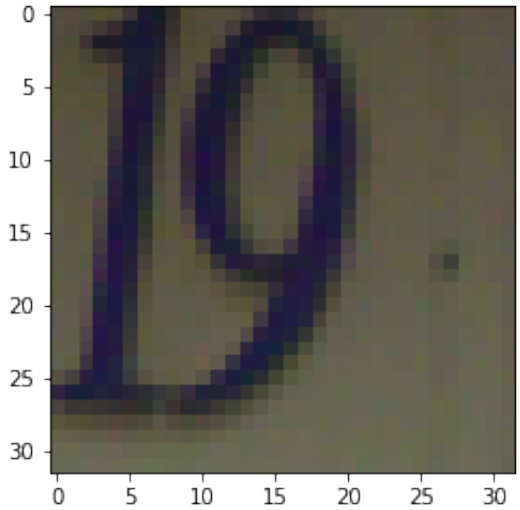
```
# Display one of the images

i=0
img=train_images[:,:,:,i]
plt.imshow(img)
print(f'label: {train_labels[i]}')
```

```
label: [1]
```




```
# Display one of the images

i=1
img=train_images[:,:,:,i]
plt.imshow(img)
print(f'label: {train_labels[i]}')
```

```
label: [9]
```

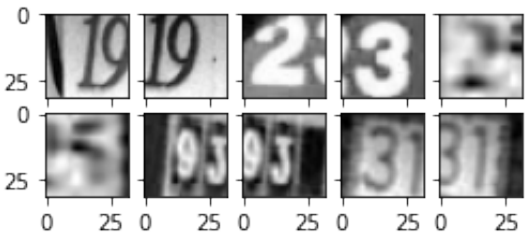


```
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid
import numpy as np

fig = plt.figure(figsize=(4., 4.))
grid = ImageGrid(fig, 111,  # similar to subplot(111)
                 nrows_ncols=(2, 5),  # creates 2x2 grid of axes
                 axes_pad=0.1,  # pad between axes in inch.
                 )

for ax, im_num in zip(grid, range(10)):
    # Iterating over the grid returns the Axes.
    ax.imshow(train_images[:,:,:,im_num])

plt.show()
print(f'label: \n {train_labels[0:10]}')
```

```
label:
 [[1]
 [9]
 [2]
 [3]
 [2]
 [5]
 [9]
 [3]
 [3]
 [1]]
```

```
X_train = np.moveaxis(train_images, -1, 0)
X_test = np.moveaxis(test_images, -1 , 0)


X_train_gs = np.mean(X_train, 3).reshape(73257, 32, 32, 1)/255
X_test_gs = np.mean(X_test,3).reshape(26032, 32,32 ,1)/255
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid
import numpy as np

fig = plt.figure(figsize=(4., 4.))
grid = ImageGrid(fig, 111,  # similar to subplot(111)
                 nrows_ncols=(2, 5),  # creates 2x2 grid of axes
                 axes_pad=0.1,  # pad between axes in inch.
                 )

for ax, im_num in zip(grid, range(10)):
    # Iterating over the grid returns the Axes.
    ax.imshow(X_train_gs[im_num,:,:,0],cmap=plt.get_cmap('gray'))

plt.show()
print(f'label: \n {train_labels[0:10]}')
```



```
label:
 [[1]
 [9]
 [2]
 [3]
 [2]
 [5]
 [9]
 [3]
 [3]
 [1]]
```

```
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder().fit(train_labels)
y_train_oh = enc.transform(train_labels).toarray()
y_test_oh = enc.transform(test_labels).toarray()
```

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.

```
def mlp_classifier(input_shape, wd, rate):
    """
    input_shape: input shape
    wd: weightdecay
    rate : regulizer rate
    """
    model = Sequential([
        Dense(round(32*32*0.1),activation='relu',kernel_regularizer=regularizers
        Flatten(),
        Dropout(rate),
        Dense(30,activation='relu',kernel_regularizer=regularizers.l2(wd)),
        BatchNormalization(),
        Dropout(rate),
        Dense(10,activation='softmax')
    ])
    return model


model = mlp_classifier(X_train_gs[0].shape,1e-4,0.3)
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense (Dense)                (None, 32, 32, 102)       204       
_________________________________________________________________
flatten (Flatten)            (None, 104448)            0         
_________________________________________________________________
dropout (Dropout)            (None, 104448)            0         
_________________________________________________________________
dense_1 (Dense)              (None, 30)                3133470   
_________________________________________________________________
batch_normalization (BatchNo (None, 30)                120       
_________________________________________________________________
dropout_1 (Dropout)          (None, 30)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                310       
=================================================================
Total params: 3,134,104
Trainable params: 3,134,044
Non-trainable params: 60
_________________________________________________________________
```

```
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accurac
```

```
# history = model.fit(x=X_train_gs, y=y_train_oh, epochs=3, validation_split=0.3
```

```
Epoch 1/3
401/401 - 8s - loss: 2.3189 - accuracy: 0.1654 - val_loss: 2.2877 - val_acc
Epoch 2/3
401/401 - 5s - loss: 2.1971 - accuracy: 0.2237 - val_loss: 2.0608 - val_acc
Epoch 3/3
401/401 - 5s - loss: 1.9760 - accuracy: 0.3122 - val_loss: 1.7612 - val_acc
```

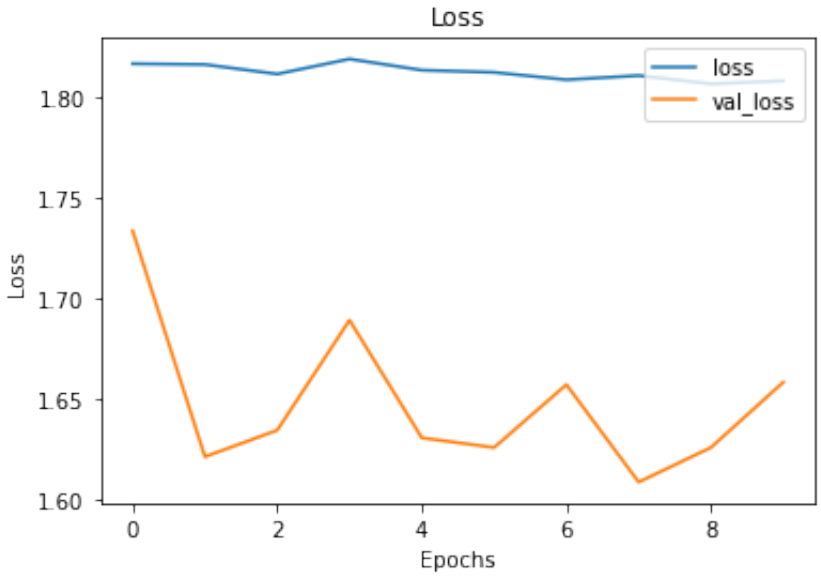
```
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

checkpoint = ModelCheckpoint(filepath = 'firstmodel/sequential', save_best_only=
earlystop = EarlyStopping(patience=5, monitor='loss')
```

```
history = model.fit(x=X_train_gs, y=y_train_oh, epochs=30, validation_split=0.15
```

```
Epoch 1/10
487/487 - 6s - loss: 1.8167 - accuracy: 0.3707 - val_loss: 1.7337 - val_acc

Epoch 00001: val_loss did not improve from 1.63698
Epoch 2/10
487/487 - 6s - loss: 1.8163 - accuracy: 0.3707 - val_loss: 1.6214 - val_acc

Epoch 00002: val_loss improved from 1.63698 to 1.62138, saving model to fir
Epoch 3/10
487/487 - 6s - loss: 1.8116 - accuracy: 0.3716 - val_loss: 1.6345 - val_acc

Epoch 00003: val_loss did not improve from 1.62138
Epoch 4/10
487/487 - 6s - loss: 1.8191 - accuracy: 0.3712 - val_loss: 1.6893 - val_acc

Epoch 00004: val_loss did not improve from 1.62138
Epoch 5/10
487/487 - 6s - loss: 1.8135 - accuracy: 0.3699 - val_loss: 1.6307 - val_acc

Epoch 00005: val_loss did not improve from 1.62138
Epoch 6/10
487/487 - 6s - loss: 1.8124 - accuracy: 0.3716 - val_loss: 1.6260 - val_acc

Epoch 00006: val_loss did not improve from 1.62138
Epoch 7/10
487/487 - 6s - loss: 1.8088 - accuracy: 0.3721 - val_loss: 1.6572 - val_acc

Epoch 00007: val_loss did not improve from 1.62138
Epoch 8/10
487/487 - 6s - loss: 1.8108 - accuracy: 0.3703 - val_loss: 1.6088 - val_acc

Epoch 00008: val_loss improved from 1.62138 to 1.60878, saving model to fir
Epoch 9/10
487/487 - 6s - loss: 1.8067 - accuracy: 0.3735 - val_loss: 1.6260 - val_acc

Epoch 00009: val_loss did not improve from 1.60878
Epoch 10/10
487/487 - 6s - loss: 1.8082 - accuracy: 0.3736 - val_loss: 1.6584 - val_acc

Epoch 00010: val_loss did not improve from 1.60878
```

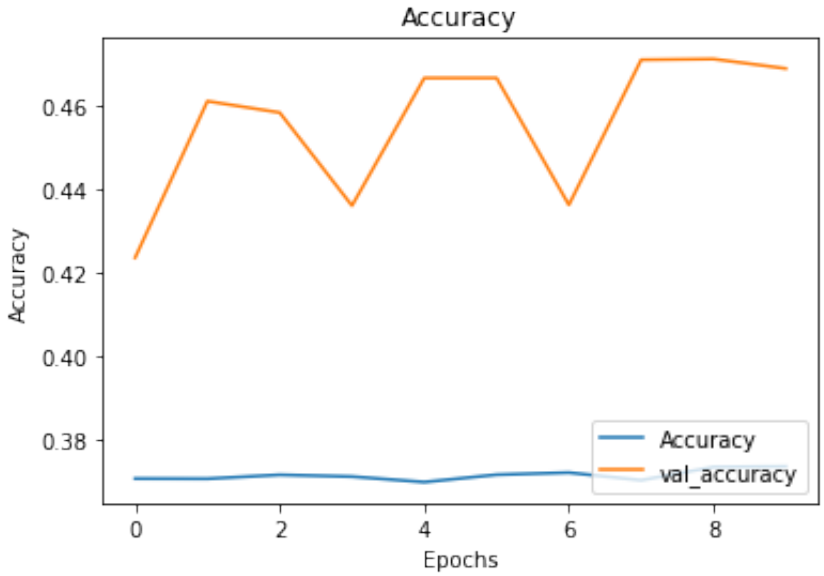
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['loss','val_loss'], loc='upper right')
plt.title("Loss")
```

```
Text(0.5, 1.0, 'Loss')
```



```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['Accuracy','val_accuracy'], loc='lower right')
plt.title("Accuracy")
```

```
Text(0.5, 1.0, 'Accuracy')
```



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.

```
def get_conv_model():
    model_conv = Sequential([
        Conv2D(filters= 16, kernel_size= 3, activation='relu', input_shape=X_tra
        MaxPooling2D(pool_size= (2,2)),
        Conv2D(filters= 32, kernel_size = 3, padding='valid', activation='relu')
        MaxPooling2D(pool_size = (3,3)),
        BatchNormalization(),
        Conv2D(filters= 32, kernel_size = 3, padding='valid', activation='relu')
        Dropout(0.5),
        Flatten(),
        Dense(64, activation='relu'),
        Dense(32, activation='relu'),
        tf.keras.layers.Dropout(0.3),
        Dense(10, activation='softmax')
    ])
    return model_conv
```

```
model_conv = get_conv_model()
model_conv.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 30, 30, 16)        448       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 16)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 13, 13, 32)        4640      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 32)          0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 4, 4, 32)          128       
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 2, 2, 32)          9248      
_________________________________________________________________
dropout_2 (Dropout)          (None, 2, 2, 32)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 128)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256      
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2080      
_________________________________________________________________
dropout_3 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_5 (Dense)              (None, 10)                330       
=================================================================
Total params: 25,130
Trainable params: 25,066
Non-trainable params: 64
_________________________________________________________________
```

```
checkpoint2 = ModelCheckpoint(filepath = 'secondmodel/cnnweights', save_best_onl
earlystop2 = EarlyStopping(monitor='loss',patience=7, verbose=1)
```

```
model_conv.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['ac
```

```
history = model_conv.fit(X_train, y_train_oh,  batch_size=256, validation_data=(
```

```
Epoch 16/30
287/287 [==============================] - 2s 6ms/step - loss: 0.4034 - acc

Epoch 00016: val_loss improved from 0.33072 to 0.32952, saving model to sec
Epoch 17/30
287/287 [==============================] - 2s 6ms/step - loss: 0.4053 - acc

Epoch 00017: val_loss did not improve from 0.32952
```

```
Epoch 18/30
287/287 [==============================] - 2s 6ms/step - loss: 0.4060 - acc

Epoch 00018: val_loss did not improve from 0.32952
Epoch 19/30
287/287 [==============================] - 2s 6ms/step - loss: 0.4069 - acc

Epoch 00019: val_loss did not improve from 0.32952
Epoch 20/30
287/287 [==============================] - 2s 6ms/step - loss: 0.4059 - acc

Epoch 00020: val_loss did not improve from 0.32952
Epoch 21/30
287/287 [==============================] - 2s 6ms/step - loss: 0.4025 - acc

Epoch 00021: val_loss did not improve from 0.32952
Epoch 22/30
287/287 [==============================] - 2s 6ms/step - loss: 0.4045 - acc

Epoch 00022: val_loss did not improve from 0.32952
Epoch 23/30
287/287 [==============================] - 2s 6ms/step - loss: 0.4078 - acc

Epoch 00023: val_loss did not improve from 0.32952
Epoch 24/30
287/287 [==============================] - 2s 6ms/step - loss: 0.4046 - acc

Epoch 00024: val_loss did not improve from 0.32952
Epoch 25/30
287/287 [==============================] - 2s 6ms/step - loss: 0.4038 - acc

Epoch 00025: val_loss did not improve from 0.32952
Epoch 26/30
287/287 [==============================] - 2s 6ms/step - loss: 0.4032 - acc

Epoch 00026: val_loss did not improve from 0.32952
Epoch 27/30
287/287 [==============================] - 2s 6ms/step - loss: 0.4002 - acc

Epoch 00027: val_loss improved from 0.32952 to 0.32920, saving model to sec
Epoch 28/30
287/287 [==============================] - 2s 6ms/step - loss: 0.4008 - acc

Epoch 00028: val_loss did not improve from 0.32920
Epoch 29/30
287/287 [==============================] - 2s 6ms/step - loss: 0.4011 - acc

Epoch 00029: val_loss did not improve from 0.32920
Epoch 30/30
287/287 [==============================] - 2s 6ms/step - loss: 0.4023 - acc

Epoch 00030: val_loss did not improve from 0.32920
```

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.

```
model_new = get_conv_model()
model_new.load_weights('secondmodel/cnnweights')
```

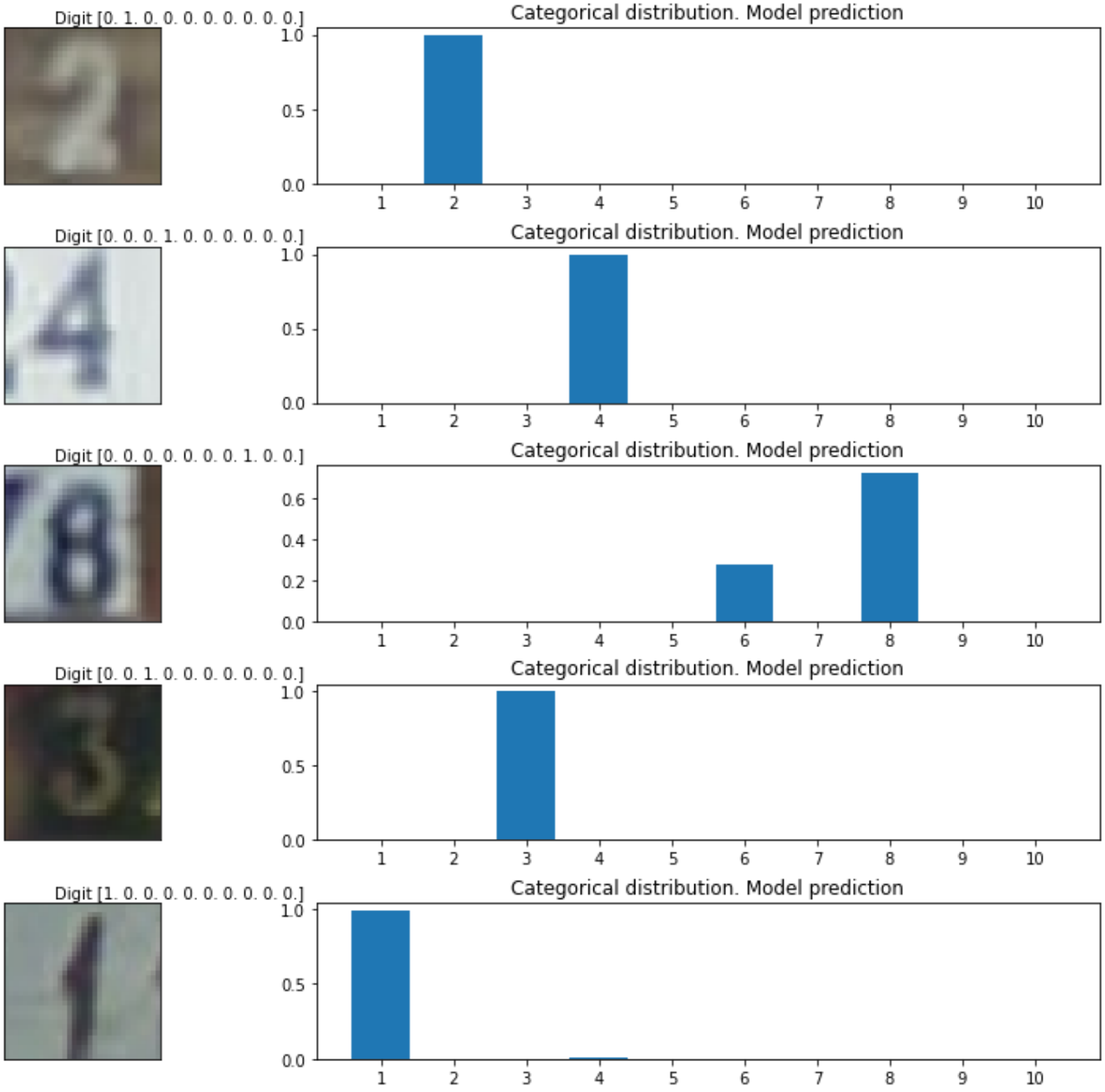
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f071e2
```

```
!ls
```

```
firstmodel  sample_data  secondmodel  test_32x32.mat  train_32x32.mat
```

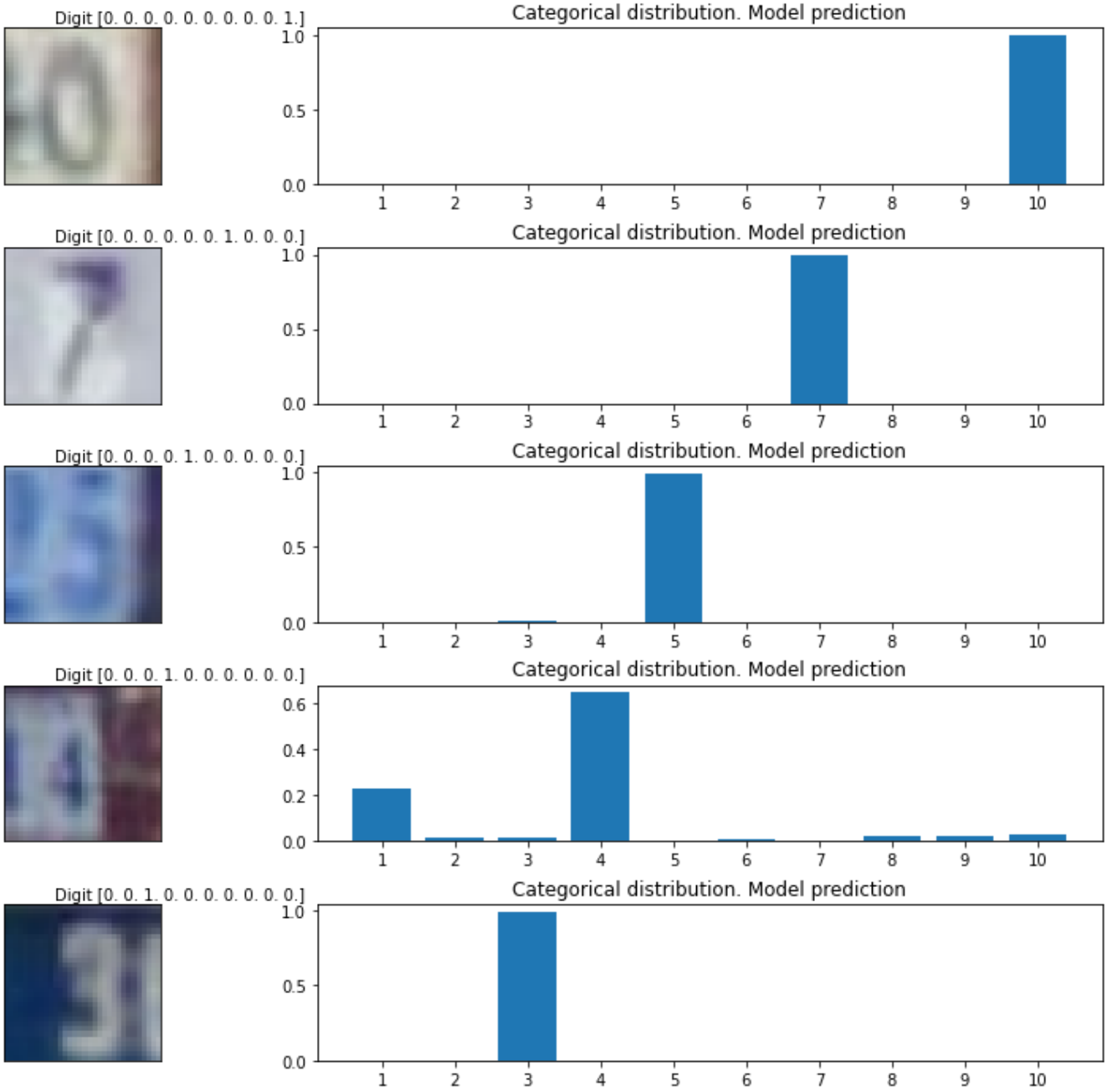
```
num_test_images = X_test.shape[0]

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

predictions = model_new.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_imag
    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(1,11), prediction)
    axes[i, 1].set_xticks(np.arange(1,11))
    axes[i, 1].set_title("Categorical distribution. Model prediction")
```

```
plt.show()
```




```
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_oh[random_inx, ...]

predictions = model_new.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_imag
    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(1,11), prediction)
    axes[i, 1].set_xticks(np.arange(1,11))
    axes[i, 1].set_title("Categorical distribution. Model prediction")

plt.show()
```


Digit [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
Categorical distribution. Model prediction
Digit [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
Categorical distribution. Model prediction
Digit [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
Categorical distribution. Model prediction
Digit [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
Categorical distribution. Model prediction
Digit [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
Categorical distribution. Model prediction