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 co 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 instr 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 nc 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 reviewer being able to view the outputs of your notebook. Save the notebook as a pdf (File -> Download as -> PDF via LaTeX). You shoul 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 notebo you wish.

```
import tensorflow as tf
from scipy.io import loadmat
```

SVHN overview image For the capstone project, you will use the [SVHN dataset](). This is an image dataset of over 600,000 digit images and is a harder dataset than MNIST as the numbers appear in the context of natural scene images. SVHN is obtained from house numb 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 Learni
  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 r
world image into one of ten classes.

```
# Run this cell to connect to your Drive folder

from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3

Enter your authorization code:
··········
Mounted at /content/gdrive
```

```
# Run this cell to load the dataset

# train = loadmat('data/train_32x32.mat')
# test = loadmat('data/test_32x32.mat')

# Load the dataset from your Drive folder

train = loadmat('/content/gdrive/My Drive/TensorFlow/train_32x32.mat')
test = loadmat('/content/gdrive/My Drive/TensorFlow/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 cl 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 fig

```
X_train , y_train = train['X'] , train['y']
X_test , y_test = test['X'] , test['y']
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(32, 32, 3, 73257)
(73257, 1)
(32, 32, 3, 26032)
(26032, 1)
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.io import loadmat
from skimage import color
from skimage import io
from sklearn.model_selection import train_test_split

%matplotlib inline
plt.rcParams['figure.figsize'] = (16.0, 4.0)
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is
  import pandas.util.testing as tm
```

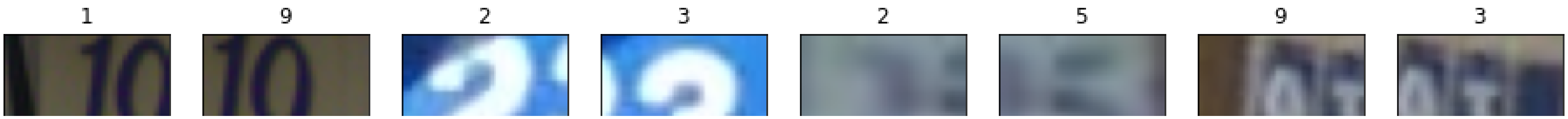
```
X_train, y_train = X_train.transpose((3,0,1,2)), y_train[:,0]
X_test, y_test = X_test.transpose((3,0,1,2)), y_test[:,0]

print("Training Set", X_train.shape)
print("Test Set", X_test.shape)
print('')
```

```
# Calculate the total number of images
num_images = X_train.shape[0] + X_test.shape[0]

print("Total Number of Images", num_images)
```

```
Training Set (73257, 32, 32, 3)
Test Set (26032, 32, 32, 3)

Total Number of Images 99289
```

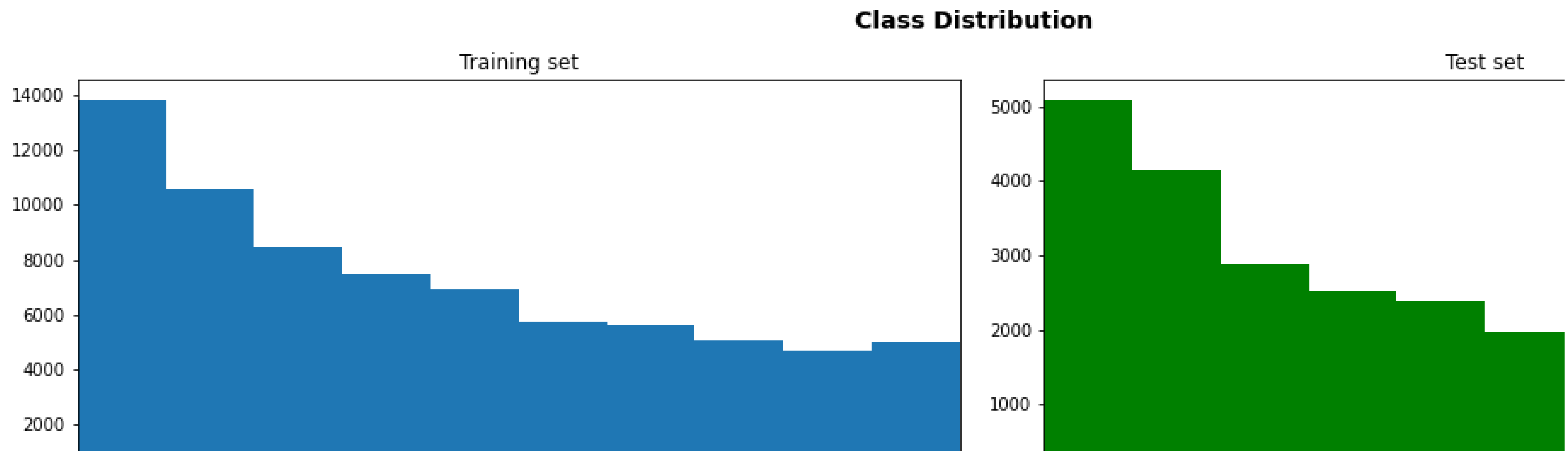
```
def plot_images(img, labels, nrows, ncols):
    """ Plot nrows x ncols images
    """
    fig, axes = plt.subplots(nrows, ncols)
    for i, ax in enumerate(axes.flat):
        if img[i].shape == (32, 32, 3):
            ax.imshow(img[i])
        else:
            ax.imshow(img[i,:,:,0])
        ax.set_xticks([]); ax.set_yticks([])
        ax.set_title(labels[i])
```

```
# Plot some training set images
plot_images(X_train, y_train, 2, 8)
```




```
# Plot some test set images
plot_images(X_test, y_test, 2, 8)
```


```
fig, (ax1, ax2) = plt.subplots(1, 2, sharex=True)

fig.suptitle('Class Distribution', fontsize=14, fontweight='bold', y=1.05)

ax1.hist(y_train, bins=10)
ax1.set_title("Training set")
ax1.set_xlim(1, 10)

ax2.hist(y_test, color='g', bins=10)
ax2.set_title("Test set")

fig.tight_layout()
```




```
# Converting Label 10 -> 0
y_train[y_train == 10] = 0
y_test[y_test == 10] = 0
```

Splitting the Training to Train+Validation Splitting to 13% in Val Set as it gives around 9500 data having min. of 800 instances of each cla

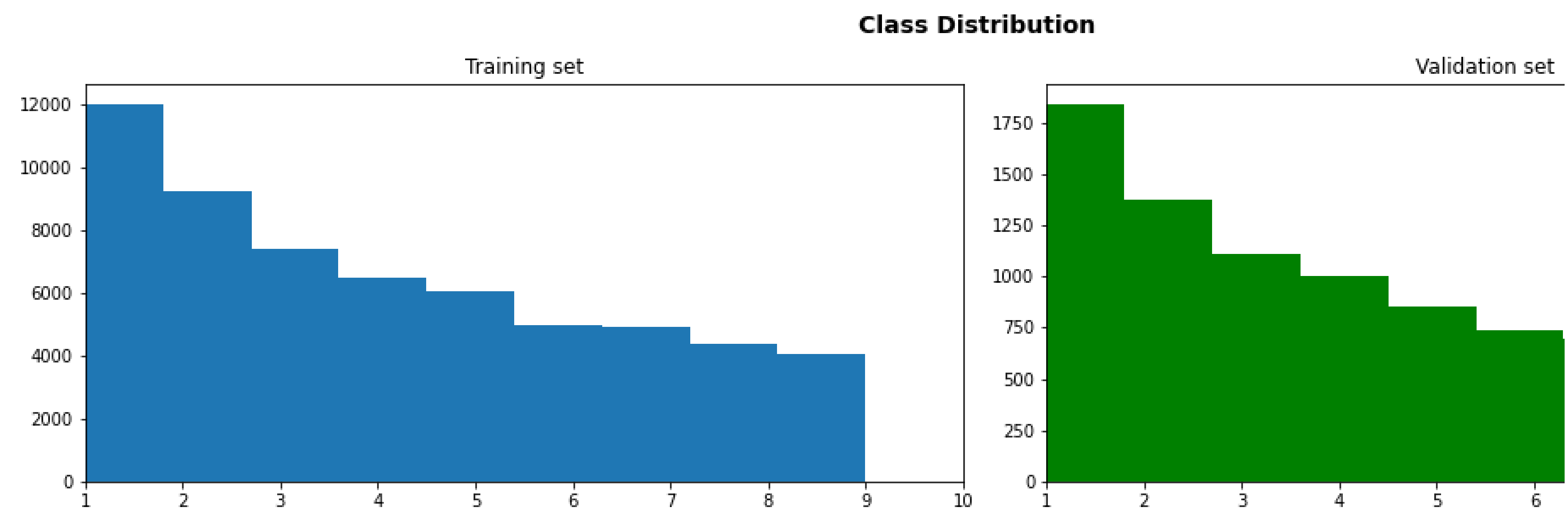
Using random state to regenrate the whole Dataset in re-run

```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.13, random_state=7)
```

Visualize New Distribution

```
fig, (ax1, ax2) = plt.subplots(1, 2, sharex=True)

fig.suptitle('Class Distribution', fontsize=14, fontweight='bold', y=1.05)

ax1.hist(y_train, bins=10)
ax1.set_title("Training set")
ax1.set_xlim(1, 10)
```

```
ax2.hist(y_val, color='g', bins=10)
ax2.set_title("Validation set")

fig.tight_layout()
```




```
y_train.shape, y_val.shape, y_test.shape
```

```
((63733,), (9524,), (26032,))
```

Grayscale Conversion

To speed up our experiments we will convert our images from RGB to Grayscale, which grately reduces the amount of data we will have process.

Y = 0.2990R + 0.5870G + 0.1140B

Here is a simple function that helps us print the size of a numpy array in a human readable format.

```
def rgb2gray(images):
    return np.expand_dims(np.dot(images, [0.2990, 0.5870, 0.1140]), axis=3)
```

```
    return np.expand_dims(np.dot(images, [0.2990, 0.5870, 0.1140]), axis=3)
```

Converting to fload for numpy computation

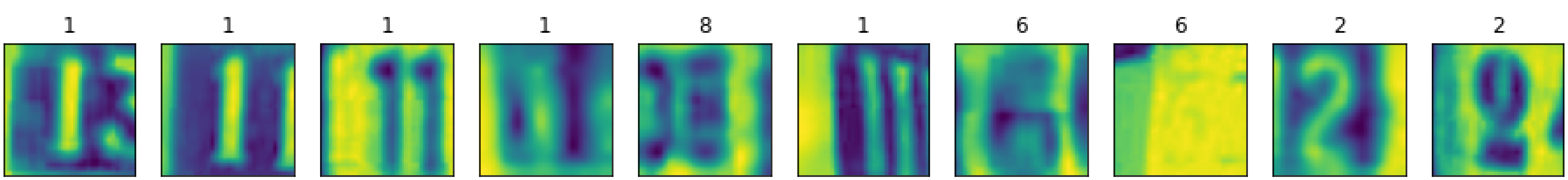
```
train_greyscale = rgb2gray(X_train).astype(np.float32)
test_greyscale = rgb2gray(X_test).astype(np.float32)
val_greyscale = rgb2gray(X_val).astype(np.float32)
```

```
print("Training Set", train_greyscale.shape)
print("Validation Set", val_greyscale.shape)
print("Test Set", test_greyscale.shape)
print('')
```

```
Training Set (63733, 32, 32, 1)
Validation Set (9524, 32, 32, 1)
Test Set (26032, 32, 32, 1)
```

```
del X_train, X_test, X_val
```

```
plot_images(train_greyscale, y_train, 1, 10)
```



Doing Normalization

```
# Calculate the mean on the training data
train_mean = np.mean(train_greyscale, axis=0)

# Calculate the std on the training data
train_std = np.std(train_greyscale, axis=0)

# Subtract it equally from all splits
train_greyscale_norm = (train_greyscale - train_mean) / train_std
test_greyscale_norm = (test_greyscale - train_mean)  / train_std
val_greyscale_norm = (val_greyscale - train_mean) / train_std
```

```
from sklearn.preprocessing import OneHotEncoder

# Fit the OneHotEncoder
enc = OneHotEncoder().fit(y_train.reshape(-1, 1))

# Transform the label values to a one-hot-encoding scheme
y_train = enc.transform(y_train.reshape(-1, 1)).toarray()
y_test = enc.transform(y_test.reshape(-1, 1)).toarray()
y_val = enc.transform(y_val.reshape(-1, 1)).toarray()

print("Training set", y_train.shape)
print("Validation set", y_val.shape)
print("Test set", y_test.shape)
```

```
Training set (63733, 10)
Validation set (9524, 10)
Test set (26032, 10)
```

Storing Data to Disk

Stored only the Grayscale Data not the RGB

```
import h5py

# Create file
```

```
h5f = h5py.File('SVHN_grey.h5', 'w')

# Store the datasets
h5f.create_dataset('X_train', data=train_greyscale_norm)
h5f.create_dataset('y_train', data=y_train)
h5f.create_dataset('X_test', data=test_greyscale_norm)
h5f.create_dataset('y_test', data=y_test)
h5f.create_dataset('X_val', data=val_greyscale_norm)
h5f.create_dataset('y_val', data=y_val)

# Close the file
h5f.close()
```

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 h 10-way softmax output.
- You should design and build the model yourself. Feel free to experiment with different MLP architectures. *Hint: to achieve a reason 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 th 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 h
- 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.

```
import os
import time
# from __future__ import absolute_import
# from __future__ import print_function
```

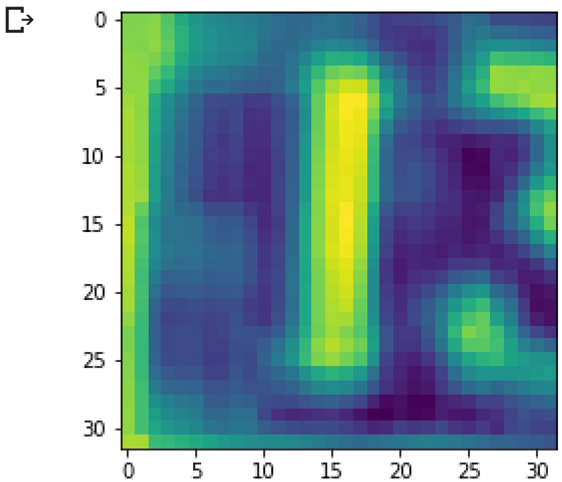
```
from datetime import timedelta
import h5py
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
plt.rcParams['figure.figsize'] = (16.0, 4.0) # Set default figure size
```

```
h5f = h5py.File('SVHN_grey.h5', 'r')

# Load the training, test and validation set
X_train = h5f['X_train'][:]
y_train = h5f['y_train'][:]
X_test = h5f['X_test'][:]
y_test = h5f['y_test'][:]
X_val = h5f['X_val'][:]
y_val = h5f['y_val'][:]

# Close this file
h5f.close()

print('Training set', X_train.shape, y_train.shape)
print('Validation set', X_val.shape, y_val.shape)
print('Test set', X_test.shape, y_test.shape)
```

```
Training set (63733, 32, 32, 1) (63733, 10)
Validation set (9524, 32, 32, 1) (9524, 10)
Test set (26032, 32, 32, 1) (26032, 10)
```

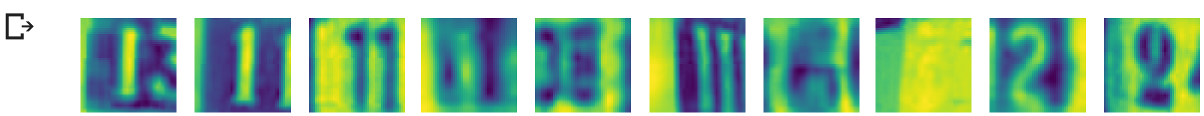
```
# Display one of the images
i = 0
labels = np.argmax(y_train[i])
img = X_train[i,:,:,0]
plt.imshow(img)
```

```
plt.show()
print(f"label: {labels}")
```



```
label: 1
```

```
X_train[0].shape
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 10, figsize=(10, 1))
for i in range(10):
    ax[i].set_axis_off()
    ax[i].imshow(X_train[i,:,:,0])
```



```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten


def get_model(input_shape):

    model = Sequential([
```

```
        Flatten(input_shape = input_shape),
        Dense(128, activation = 'relu'),
        Dense(128, activation = 'relu'),
        Dense(128, activation = 'relu'),
        Dense(128, activation = 'relu'),
        Dense(128, activation = 'relu'),
        Dense(10, activation = 'softmax')
    ])

    return model
```

```
model = get_model(X_train[0].shape)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 128)               131200
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_3 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_4 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290
=================================================================
Total params: 198,538
Trainable params: 198,538
Non-trainable params: 0
_________________________________________________________________
```

```
def compile_model(model):
```

```
    model.compile(
        optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001),
        loss = 'categorical_crossentropy',
        metrics = ['accuracy']
    )



compile_model(model)
print(model.optimizer)
print(model.loss)
print(model.metrics)
print(model.optimizer.lr)
```

```
<tensorflow.python.keras.optimizer_v2.adam.Adam object at 0x7f4e80342be0>
categorical_crossentropy
[]
<tf.Variable 'learning_rate:0' shape=() dtype=float32, numpy=1e-04>
```

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

class TrainingCallback(Callback):

    def on_train_begin(self, logs=None):
        print("Starting training....")

    def on_epoch_begin(self, epoch, logs=None):
        print(f"Starting epoch {epoch}")

    def on_epoch_end(self, epoch, logs=None):
        print(f"Finishing epoch {epoch}")

    def on_train_end(self, logs=None):
        print("Finished training:")



def get_checkpoint_best_only():
```

```
def get_checkpoint_best_only():

    checkpoint_best_path = 'checkpoints_best_only/checkpoint'
    checkpoint_best_only = ModelCheckpoint(filepath=checkpoint_best_path, save_freq='epoch',
                            save_weights_only=True, monitor = 'val_accuracy',
                            save_best_only=True,verbose = 1)

    return checkpoint_best_only



TrainingCallback = TrainingCallback()
checkpoint_best_only = get_checkpoint_best_only()



def train_model(model, train_data, train_targets, epochs):

    history = model.fit(train_data, train_targets, epochs=epochs,
                        batch_size=64, validation_data=(X_val,y_val),verbose=False)

    return history



callbacks = [TrainingCallback, checkpoint_best_only]
history = model.fit(X_train, y_train, epochs=30, batch_size=64, validation_data=(X_val,y_val),callbacks=callbacks)
```

```
Starting training....
Starting epoch 0
Epoch 1/30
985/996 [============================>.] - ETA: 0s - loss: 1.5683 - accuracy: 0.4728Finishing epoch 0

Epoch 00001: val_accuracy improved from -inf to 0.65981, saving model to checkpoints_best_only/checkpoint
996/996 [==============================] - 5s 5ms/step - loss: 1.5620 - accuracy: 0.4753 - val_loss: 1.0931 -
Starting epoch 1
Epoch 2/30
987/996 [============================>.] - ETA: 0s - loss: 0.9865 - accuracy: 0.6966Finishing epoch 1

Epoch 00002: val_accuracy improved from 0.65981 to 0.72764, saving model to checkpoints_best_only/checkpoint
996/996 [==============================] - 5s 5ms/step - loss: 0.9857 - accuracy: 0.6968 - val_loss: 0.9042 -
Starting epoch 2
Epoch 3/30
991/996 [============================>.] - ETA: 0s - loss: 0.8452 - accuracy: 0.7430Finishing epoch 2

Epoch 00003: val_accuracy improved from 0.72764 to 0.75430, saving model to checkpoints_best_only/checkpoint
996/996 [==============================] - 5s 5ms/step - loss: 0.8449 - accuracy: 0.7431 - val_loss: 0.8217 -
Starting epoch 3
Epoch 4/30
986/996 [============================>.] - ETA: 0s - loss: 0.7582 - accuracy: 0.7699Finishing epoch 3

Epoch 00004: val_accuracy improved from 0.75430 to 0.77572, saving model to checkpoints_best_only/checkpoint
996/996 [==============================] - 5s 5ms/step - loss: 0.7575 - accuracy: 0.7702 - val_loss: 0.7477 -
Starting epoch 4
Epoch 5/30
990/996 [============================>.] - ETA: 0s - loss: 0.6957 - accuracy: 0.7895Finishing epoch 4

Epoch 00005: val_accuracy improved from 0.77572 to 0.79074, saving model to checkpoints_best_only/checkpoint
996/996 [==============================] - 5s 5ms/step - loss: 0.6954 - accuracy: 0.7894 - val_loss: 0.7049 -
Starting epoch 5
Epoch 6/30
985/996 [============================>.] - ETA: 0s - loss: 0.6499 - accuracy: 0.8030Finishing epoch 5

Epoch 00006: val_accuracy improved from 0.79074 to 0.79473, saving model to checkpoints_best_only/checkpoint
996/996 [==============================] - 5s 5ms/step - loss: 0.6500 - accuracy: 0.8029 - val_loss: 0.6912 -
Starting epoch 6
Epoch 7/30
989/996 [============================>.] - ETA: 0s - loss: 0.6133 - accuracy: 0.8163Finishing epoch 6

Epoch 00007: val_accuracy improved from 0.79473 to 0.80292, saving model to checkpoints_best_only/checkpoint
996/996 [==============================] - 5s 5ms/step - loss: 0.6140 - accuracy: 0.8161 - val_loss: 0.6678 -
```

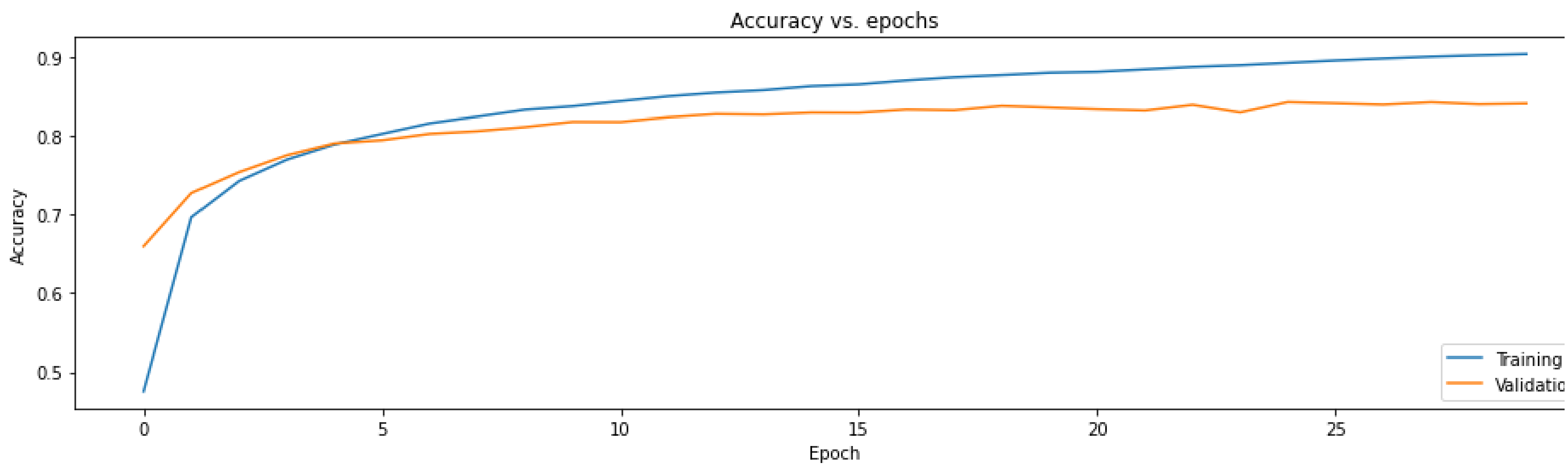
```
Starting epoch 7
Epoch 8/30
994/996 [============================>.] - ETA: 0s - loss: 0.5829 - accuracy: 0.8248Finishing epoch 7

Epoch 00008: val_accuracy improved from 0.80292 to 0.80617, saving model to checkpoints_best_only/checkpoint
996/996 [==============================] - 5s 5ms/step - loss: 0.5832 - accuracy: 0.8247 - val_loss: 0.6542 -
Starting epoch 8
Epoch 9/30
989/996 [============================>.] - ETA: 0s - loss: 0.5560 - accuracy: 0.8332Finishing epoch 8

Epoch 00009: val_accuracy improved from 0.80617 to 0.81142, saving model to checkpoints_best_only/checkpoint
996/996 [==============================] - 5s 5ms/step - loss: 0.5556 - accuracy: 0.8335 - val_loss: 0.6367 -
Starting epoch 9
Epoch 10/30
992/996 [============================>.] - ETA: 0s - loss: 0.5326 - accuracy: 0.8380Finishing epoch 9

Epoch 00010: val_accuracy improved from 0.81142 to 0.81804, saving model to checkpoints_best_only/checkpoint
996/996 [==============================] - 5s 5ms/step - loss: 0.5328 - accuracy: 0.8380 - val_loss: 0.6250 -
Starting epoch 10
Epoch 11/30
989/996 [============================>.] - ETA: 0s - loss: 0.5144 - accuracy: 0.8444Finishing epoch 10

Epoch 00011: val_accuracy did not improve from 0.81804
996/996 [==============================] - 5s 5ms/step - loss: 0.5141 - accuracy: 0.8445 - val_loss: 0.6171 -
Starting epoch 11
Epoch 12/30
992/996 [============================>.] - ETA: 0s - loss: 0.4970 - accuracy: 0.8506Finishing epoch 11

Epoch 00012: val_accuracy improved from 0.81804 to 0.82392, saving model to checkpoints_best_only/checkpoint
996/996 [==============================] - 5s 5ms/step - loss: 0.4969 - accuracy: 0.8506 - val_loss: 0.6131 -
Starting epoch 12
Epoch 13/30
986/996 [============================>.] - ETA: 0s - loss: 0.4788 - accuracy: 0.8552Finishing epoch 12

Epoch 00013: val_accuracy improved from 0.82392 to 0.82812, saving model to checkpoints_best_only/checkpoint
996/996 [==============================] - 5s 5ms/step - loss: 0.4789 - accuracy: 0.8552 - val_loss: 0.5965 -
Starting epoch 13
Epoch 14/30
991/996 [============================>.] - ETA: 0s - loss: 0.4663 - accuracy: 0.8584Finishing epoch 13

Epoch 00014: val_accuracy did not improve from 0.82812
996/996 [==============================] - 5s 5ms/step - loss: 0.4666 - accuracy: 0.8583 - val_loss: 0.5997 -
Starting epoch 14
Epoch 15/30
```

```
 -
991/996 [============================>.] - ETA: 0s - loss: 0.4492 - accuracy: 0.8634Finishing epoch 14

Epoch 00015: val_accuracy improved from 0.82812 to 0.82980, saving model to checkpoints_best_only/checkpoint
996/996 [==============================] - 5s 5ms/step - loss: 0.4494 - accuracy: 0.8633 - val_loss: 0.5943 -
Starting epoch 15
Epoch 16/30
993/996 [============================>.] - ETA: 0s - loss: 0.4393 - accuracy: 0.8656Finishing epoch 15

Epoch 00016: val_accuracy did not improve from 0.82980
996/996 [==============================] - 5s 5ms/step - loss: 0.4395 - accuracy: 0.8656 - val_loss: 0.5958 -
Starting epoch 16
Epoch 17/30
991/996 [============================>.] - ETA: 0s - loss: 0.4241 - accuracy: 0.8706Finishing epoch 16

Epoch 00017: val_accuracy improved from 0.82980 to 0.83358, saving model to checkpoints_best_only/checkpoint
996/996 [==============================] - 5s 5ms/step - loss: 0.4239 - accuracy: 0.8706 - val_loss: 0.5829 -
Starting epoch 17
Epoch 18/30
988/996 [============================>.] - ETA: 0s - loss: 0.4148 - accuracy: 0.8748Finishing epoch 17

Epoch 00018: val_accuracy did not improve from 0.83358
996/996 [==============================] - 5s 5ms/step - loss: 0.4150 - accuracy: 0.8747 - val_loss: 0.5880 -
Starting epoch 18
Epoch 19/30
986/996 [============================>.] - ETA: 0s - loss: 0.4028 - accuracy: 0.8775Finishing epoch 18

Epoch 00019: val_accuracy improved from 0.83358 to 0.83830, saving model to checkpoints_best_only/checkpoint
996/996 [==============================] - 5s 5ms/step - loss: 0.4027 - accuracy: 0.8775 - val_loss: 0.5709 -
Starting epoch 19
Epoch 20/30
992/996 [============================>.] - ETA: 0s - loss: 0.3928 - accuracy: 0.8805Finishing epoch 19

Epoch 00020: val_accuracy did not improve from 0.83830
996/996 [==============================] - 5s 5ms/step - loss: 0.3931 - accuracy: 0.8804 - val_loss: 0.5877 -
Starting epoch 20
Epoch 21/30
994/996 [============================>.] - ETA: 0s - loss: 0.3849 - accuracy: 0.8816Finishing epoch 20

Epoch 00021: val_accuracy did not improve from 0.83830
996/996 [==============================] - 5s 5ms/step - loss: 0.3848 - accuracy: 0.8816 - val_loss: 0.5958 -
Starting epoch 21
Epoch 22/30
992/996 [============================>.] - ETA: 0s - loss: 0.3758 - accuracy: 0.8846Finishing epoch 21
```

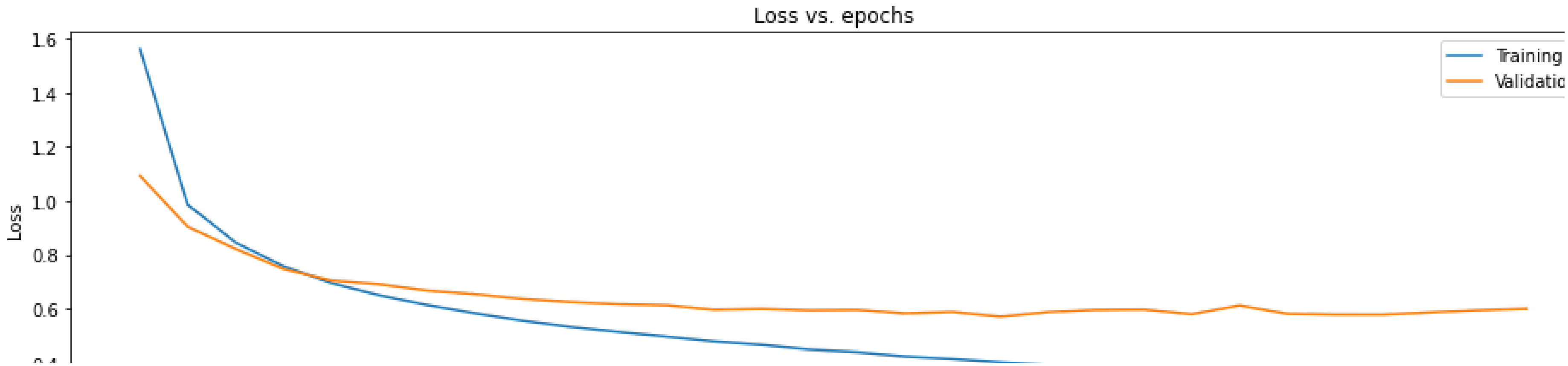
```
Epoch 00022: val_accuracy did not improve from 0.83830
996/996 [==============================] - 5s 5ms/step - loss: 0.3756 - accuracy: 0.8847 - val_loss: 0.5969 -
Starting epoch 22
Epoch 23/30
993/996 [============================>.] - ETA: 0s - loss: 0.3656 - accuracy: 0.8878Finishing epoch 22

Epoch 00023: val_accuracy improved from 0.83830 to 0.83956, saving model to checkpoints_best_only/checkpoint
996/996 [==============================] - 5s 5ms/step - loss: 0.3654 - accuracy: 0.8878 - val_loss: 0.5798 -
Starting epoch 23
Epoch 24/30
996/996 [==============================] - ETA: 0s - loss: 0.3579 - accuracy: 0.8900Finishing epoch 23

Epoch 00024: val_accuracy did not improve from 0.83956
996/996 [==============================] - 5s 5ms/step - loss: 0.3579 - accuracy: 0.8900 - val_loss: 0.6121 -
Starting epoch 24
Epoch 25/30
995/996 [============================>.] - ETA: 0s - loss: 0.3488 - accuracy: 0.8930Finishing epoch 24

Epoch 00025: val_accuracy improved from 0.83956 to 0.84313, saving model to checkpoints_best_only/checkpoint
996/996 [==============================] - 5s 5ms/step - loss: 0.3486 - accuracy: 0.8930 - val_loss: 0.5813 -
Starting epoch 25
Epoch 26/30
988/996 [============================>.] - ETA: 0s - loss: 0.3402 - accuracy: 0.8962Finishing epoch 25

Epoch 00026: val_accuracy did not improve from 0.84313
996/996 [==============================] - 5s 5ms/step - loss: 0.3407 - accuracy: 0.8960 - val_loss: 0.5783 -
Starting epoch 26
Epoch 27/30
991/996 [============================>.] - ETA: 0s - loss: 0.3314 - accuracy: 0.8987Finishing epoch 26

Epoch 00027: val_accuracy did not improve from 0.84313
996/996 [==============================] - 5s 5ms/step - loss: 0.3320 - accuracy: 0.8985 - val_loss: 0.5780 -
Starting epoch 27
Epoch 28/30
996/996 [==============================] - ETA: 0s - loss: 0.3241 - accuracy: 0.9009Finishing epoch 27

Epoch 00028: val_accuracy did not improve from 0.84313
996/996 [==============================] - 5s 5ms/step - loss: 0.3241 - accuracy: 0.9009 - val_loss: 0.5866 -
Starting epoch 28
```

```
try:
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
```

```
except KeyError:
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
plt.title('Accuracy vs. epochs')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='lower right')
plt.show()
```




```
#Run this cell to plot the epoch vs loss graph
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()
```




```
import pandas as pd
df = pd.DataFrame(history.history)
df.head(30)
```

- Build a CNN classifier model using the Sequential API. Your model should use the Conv2D, MaxPool2D, BatchNormalization, Flatte 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 reason 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 th 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.

```
h5f = h5py.File('SVHN_grey.h5', 'r')

# Load the training, test and validation set
X_train = h5f['X_train'][:]
y_train = h5f['y_train'][:]
X_test = h5f['X_test'][:]
y_test = h5f['y_test'][:]
X_val = h5f['X_val'][:]
y_val = h5f['y_val'][:]

# Close this file
h5f.close()

print('Training set', X_train.shape, y_train.shape)
print('Validation set', X_val.shape, y_val.shape)
print('Test set', X_test.shape, y_test.shape)
```

	loss	accuracy	val_loss	val_accuracy
0	1.561980	0.475327	1.093143	0.659807
1	0.985651	0.696813	0.904183	0.727635
2	0.844937	0.743053	0.821673	0.754305
3	0.757547	0.770166	0.747658	0.775724
4	0.695384	0.789434	0.704870	0.790739
5	0.649967	0.802881	0.691177	0.794729
6	0.613982	0.816139	0.667764	0.802919
7	0.583163	0.824738	0.654172	0.806174
8	0.555633	0.833477	0.636668	0.811424
9	0.532793	0.837965	0.625032	0.818039
10	0.514102	0.844460	0.617072	0.817829
11	0.496858	0.850595	0.613067	0.823919
12	0.478938	0.855161	0.596519	0.828118
13	0.466632	0.858315	0.599689	0.827383
14	0.449364	0.863336	0.594293	0.829798
15	0.439456	0.865627	0.595806	0.829588
16	0.423869	0.870601	0.582872	0.833578

```
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=2)
```

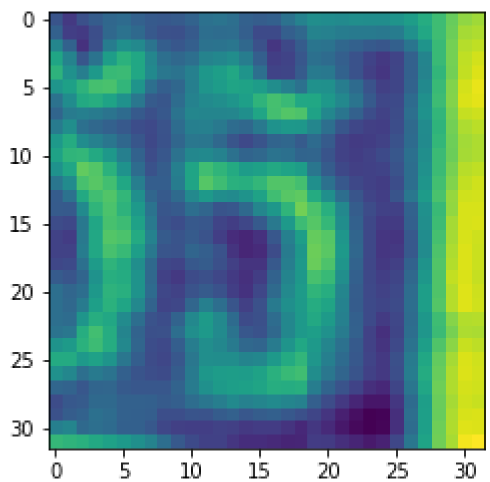
```
814/814 - 2s - loss: 0.7499 - accuracy: 0.8161
```

20 0.384706 0.881615 0.595773 0.833098

3. CNN neural network classifier

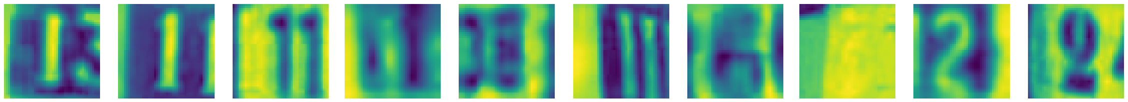
```
Training set (63733, 32, 32, 1) (63733, 10)
```

```
# Display one of the images
i = 30
labels = np.argmax(y_train[i])
img = X_train[i,:,:,0]
plt.imshow(img)
plt.show()
print(f"label: {labels}")
```



```
label: 5
```

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 10, figsize=(10, 1))
for i in range(10):
    ax[i].set_axis_off()
    ax[i].imshow(X_train[i,:,:,0])
```



```
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, BatchNormalization, Dropout
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
import os
import numpy as np
import pandas as pd



def get_new_model(input_shape):

    model = Sequential([
        Conv2D(16,(3,3),padding="SAME",activation='relu',name='conv_1', input_shape=(input_shape)),
        Dropout(0.5),
        Conv2D(8,(3,3),padding="SAME",activation='relu',name='conv_2'),
        BatchNormalization(),
        Dropout(0.5),
        MaxPooling2D((4,4),name='pool_1'),
        Dense(128,activation='relu',name='dense_1'),
        Flatten(name='flatten2'),
        Dense(10,activation='softmax',name='dense_2')
    ])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model



model = get_new_model(X_train[0].shape)
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv_1 (Conv2D)              (None, 32, 32, 16)        160       
_________________________________________________________________
dropout (Dropout)            (None, 32, 32, 16)        0         
_________________________________________________________________
conv_2 (Conv2D)              (None, 32, 32, 8)         1160      
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 8)         32        
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 8)         0         
_________________________________________________________________
pool_1 (MaxPooling2D)        (None, 8, 8, 8)           0         
_________________________________________________________________
dense_1 (Dense)              (None, 8, 8, 128)         1152      
_________________________________________________________________
flatten2 (Flatten)           (None, 8192)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                81930     
=================================================================
Total params: 84,434
Trainable params: 84,418
```

```
compile_model(model)
print(model.optimizer)
print(model.loss)
print(model.metrics)
print(model.optimizer.lr)
```

```
<tensorflow.python.keras.optimizer_v2.adam.Adam object at 0x7f4e8008f7f0>
categorical_crossentropy
[]
<tf.Variable 'learning_rate:0' shape=() dtype=float32, numpy=1e-04>
```

```
def get_early_stopping():

    early_stopping = EarlyStopping(monitor='val_accuracy', patience = 7)

    return early_stopping
```

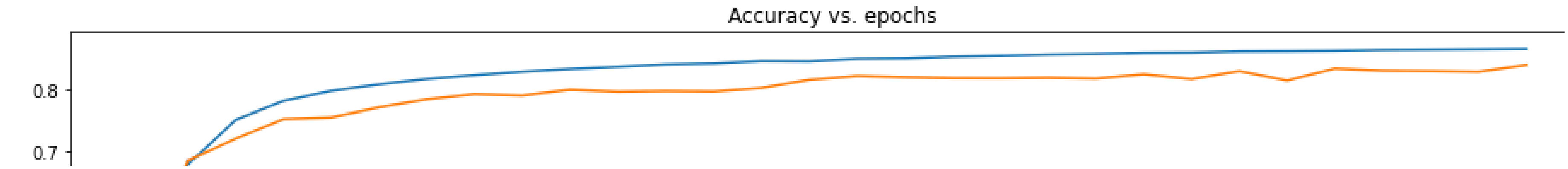
```
    return early_stopping


def get_checkpoint_best_CNN():

    checkpoint_best_path = 'checkpoints_best_only_CNN/checkpoint'
    checkpoint_best_only_CNN = ModelCheckpoint(filepath=checkpoint_best_path, save_freq='epoch',
                                save_weights_only=True, monitor = 'val_accuracy',
                                save_best_only=True,verbose = 1)

    return checkpoint_best_only_CNN


early_stopping = get_early_stopping()
checkpoint_best_only_CNN = get_checkpoint_best_CNN()



callbacks = [checkpoint_best_only_CNN, early_stopping]



history = model.fit(X_train, y_train, epochs=30, batch_size=64, validation_data=(X_val,y_val),callbacks=callbacks)
```

```
Epoch 1/30
996/996 [==============================] - ETA: 0s - loss: 1.9911 - accuracy: 0.3186
Epoch 00001: val_accuracy improved from -inf to 0.47848, saving model to checkpoints_best_only_CNN/checkpoint
996/996 [==============================] - 10s 10ms/step - loss: 1.9911 - accuracy: 0.3186 - val_loss: 1.5333
Epoch 2/30
991/996 [============================>.] - ETA: 0s - loss: 1.0562 - accuracy: 0.6781
Epoch 00002: val_accuracy improved from 0.47848 to 0.68490, saving model to checkpoints_best_only_CNN/checkpo
996/996 [==============================] - 9s 9ms/step - loss: 1.0554 - accuracy: 0.6784 - val_loss: 1.0023 -
Epoch 3/30
990/996 [============================>.] - ETA: 0s - loss: 0.8436 - accuracy: 0.7510
Epoch 00003: val_accuracy improved from 0.68490 to 0.72071, saving model to checkpoints_best_only_CNN/checkpo
996/996 [==============================] - 9s 9ms/step - loss: 0.8428 - accuracy: 0.7511 - val_loss: 0.8803 -
Epoch 4/30
995/996 [============================>.] - ETA: 0s - loss: 0.7528 - accuracy: 0.7814
Epoch 00004: val_accuracy improved from 0.72071 to 0.75231, saving model to checkpoints_best_only_CNN/checkpo
996/996 [==============================] - 9s 9ms/step - loss: 0.7527 - accuracy: 0.7814 - val_loss: 0.8121 -
Epoch 5/30
996/996 [==============================] - ETA: 0s - loss: 0.7008 - accuracy: 0.7978
Epoch 00005: val_accuracy improved from 0.75231 to 0.75493, saving model to checkpoints_best_only_CNN/checkpo
996/996 [==============================] - 9s 9ms/step - loss: 0.7008 - accuracy: 0.7978 - val_loss: 0.7873 -
Epoch 6/30
991/996 [============================>.] - ETA: 0s - loss: 0.6637 - accuracy: 0.8082
Epoch 00006: val_accuracy improved from 0.75493 to 0.77110, saving model to checkpoints_best_only_CNN/checkpo
996/996 [==============================] - 9s 9ms/step - loss: 0.6631 - accuracy: 0.8082 - val_loss: 0.7438 -
Epoch 7/30
993/996 [============================>.] - ETA: 0s - loss: 0.6322 - accuracy: 0.8170
Epoch 00007: val_accuracy improved from 0.77110 to 0.78402, saving model to checkpoints_best_only_CNN/checkpo
996/996 [==============================] - 9s 9ms/step - loss: 0.6322 - accuracy: 0.8170 - val_loss: 0.7134 -
Epoch 8/30
993/996 [============================>.] - ETA: 0s - loss: 0.6080 - accuracy: 0.8231
Epoch 00008: val_accuracy improved from 0.78402 to 0.79242, saving model to checkpoints_best_only_CNN/checkpo
996/996 [==============================] - 9s 9ms/step - loss: 0.6080 - accuracy: 0.8232 - val_loss: 0.6862 -
Epoch 9/30
991/996 [============================>.] - ETA: 0s - loss: 0.5915 - accuracy: 0.8286
Epoch 00009: val_accuracy did not improve from 0.79242
996/996 [==============================] - 9s 9ms/step - loss: 0.5910 - accuracy: 0.8288 - val_loss: 0.6878 -
Epoch 10/30
995/996 [============================>.] - ETA: 0s - loss: 0.5739 - accuracy: 0.8332
Epoch 00010: val_accuracy improved from 0.79242 to 0.79966, saving model to checkpoints_best_only_CNN/checkpo
996/996 [==============================] - 9s 9ms/step - loss: 0.5739 - accuracy: 0.8332 - val_loss: 0.6625 -
Epoch 11/30
995/996 [============================>.] - ETA: 0s - loss: 0.5585 - accuracy: 0.8368
Epoch 00011: val_accuracy did not improve from 0.79966
```

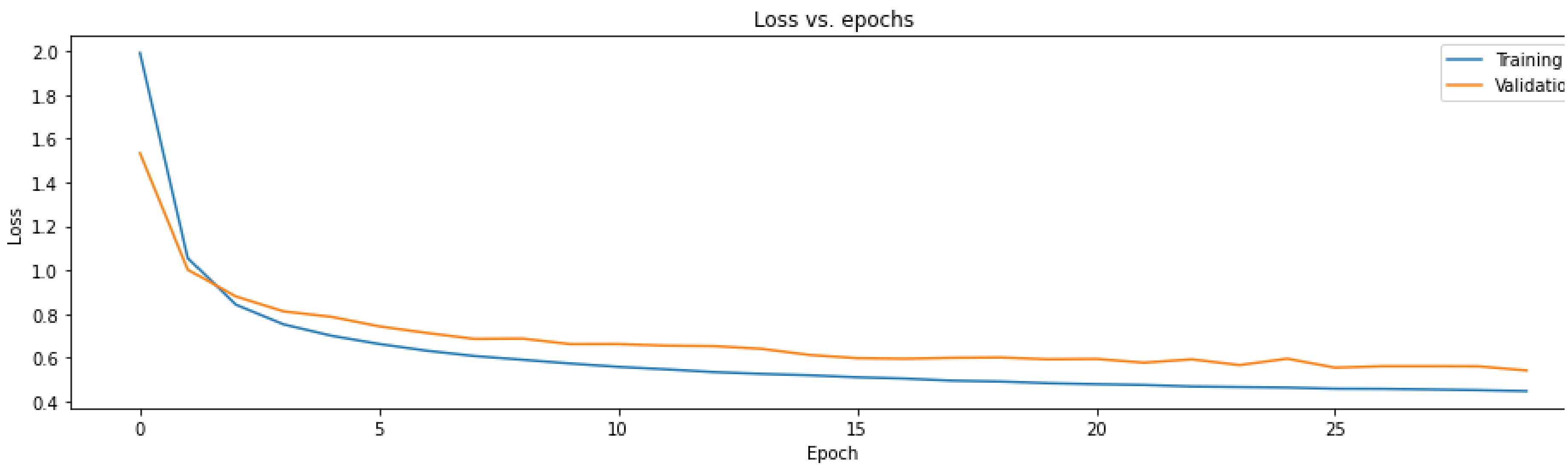
```
996/996 [==============================] - 9s 9ms/step - loss: 0.5585 - accuracy: 0.8368 - val_loss: 0.6626 -
Epoch 12/30
992/996 [============================>.] - ETA: 0s - loss: 0.5474 - accuracy: 0.8405
Epoch 00012: val_accuracy did not improve from 0.79966
996/996 [==============================] - 9s 9ms/step - loss: 0.5473 - accuracy: 0.8405 - val_loss: 0.6556 -
Epoch 13/30
992/996 [============================>.] - ETA: 0s - loss: 0.5348 - accuracy: 0.8422
Epoch 00013: val_accuracy did not improve from 0.79966
996/996 [==============================] - 9s 9ms/step - loss: 0.5348 - accuracy: 0.8422 - val_loss: 0.6534 -
Epoch 14/30
992/996 [============================>.] - ETA: 0s - loss: 0.5263 - accuracy: 0.8462
Epoch 00014: val_accuracy improved from 0.79966 to 0.80250, saving model to checkpoints_best_only_CNN/checkpo
996/996 [==============================] - 9s 9ms/step - loss: 0.5265 - accuracy: 0.8461 - val_loss: 0.6410 -
Epoch 15/30
992/996 [============================>.] - ETA: 0s - loss: 0.5202 - accuracy: 0.8457
Epoch 00015: val_accuracy improved from 0.80250 to 0.81562, saving model to checkpoints_best_only_CNN/checkpo
996/996 [==============================] - 9s 9ms/step - loss: 0.5201 - accuracy: 0.8457 - val_loss: 0.6129 -
Epoch 16/30
994/996 [============================>.] - ETA: 0s - loss: 0.5110 - accuracy: 0.8497
Epoch 00016: val_accuracy improved from 0.81562 to 0.82182, saving model to checkpoints_best_only_CNN/checkpo
996/996 [==============================] - 9s 9ms/step - loss: 0.5109 - accuracy: 0.8497 - val_loss: 0.5984 -
Epoch 17/30
993/996 [============================>.] - ETA: 0s - loss: 0.5052 - accuracy: 0.8504
Epoch 00017: val_accuracy did not improve from 0.82182
996/996 [==============================] - 9s 9ms/step - loss: 0.5051 - accuracy: 0.8504 - val_loss: 0.5956 -
Epoch 18/30
992/996 [============================>.] - ETA: 0s - loss: 0.4953 - accuracy: 0.8533
Epoch 00018: val_accuracy did not improve from 0.82182
996/996 [==============================] - 9s 9ms/step - loss: 0.4954 - accuracy: 0.8533 - val_loss: 0.5999 -
Epoch 19/30
994/996 [============================>.] - ETA: 0s - loss: 0.4914 - accuracy: 0.8548
Epoch 00019: val_accuracy did not improve from 0.82182
996/996 [==============================] - 9s 9ms/step - loss: 0.4915 - accuracy: 0.8548 - val_loss: 0.6015 -
Epoch 20/30
991/996 [============================>.] - ETA: 0s - loss: 0.4837 - accuracy: 0.8567
Epoch 00020: val_accuracy did not improve from 0.82182
996/996 [==============================] - 9s 9ms/step - loss: 0.4837 - accuracy: 0.8566 - val_loss: 0.5930 -
Epoch 21/30
991/996 [============================>.] - ETA: 0s - loss: 0.4794 - accuracy: 0.8577
Epoch 00021: val_accuracy did not improve from 0.82182
996/996 [==============================] - 9s 9ms/step - loss: 0.4794 - accuracy: 0.8577 - val_loss: 0.5948 -
Epoch 22/30
994/996 [============================>.] - ETA: 0s - loss: 0.4760 - accuracy: 0.8592
Epoch 00022: val_accuracy improved from 0.82182 to 0.82455, saving model to checkpoints_best_only_CNN/checkpo
```

```
996/996 [==============================] - 9s 9ms/step - loss: 0.4759 - accuracy: 0.8593 - val_loss: 0.5777 -
Epoch 23/30
993/996 [============================>.] - ETA: 0s - loss: 0.4690 - accuracy: 0.8598
Epoch 00023: val_accuracy did not improve from 0.82455
996/996 [==============================] - 9s 9ms/step - loss: 0.4690 - accuracy: 0.8598 - val_loss: 0.5930 -
Epoch 24/30
993/996 [============================>.] - ETA: 0s - loss: 0.4662 - accuracy: 0.8618
Epoch 00024: val_accuracy improved from 0.82455 to 0.82969, saving model to checkpoints_best_only_CNN/checkpo:
996/996 [==============================] - 9s 9ms/step - loss: 0.4662 - accuracy: 0.8618 - val_loss: 0.5667 -
Epoch 25/30
996/996 [==============================] - ETA: 0s - loss: 0.4638 - accuracy: 0.8622
Epoch 00025: val_accuracy did not improve from 0.82969
996/996 [==============================] - 10s 10ms/step - loss: 0.4638 - accuracy: 0.8622 - val_loss: 0.5959
Epoch 26/30
996/996 [==============================] - ETA: 0s - loss: 0.4587 - accuracy: 0.8628
Epoch 00026: val_accuracy improved from 0.82969 to 0.83379, saving model to checkpoints_best_only_CNN/checkpo:
996/996 [==============================] - 9s 9ms/step - loss: 0.4587 - accuracy: 0.8628 - val_loss: 0.5549 -
```

```
try:
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
except KeyError:
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
plt.title('Accuracy vs. epochs')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='lower right')
plt.show()
```




```
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()
```




```
import pandas as pd
df = pd.DataFrame(history.history)
df.head(30)
```

	loss	accuracy	val_loss	val_accuracy
0	1.991114	0.318563	1.533335	0.478475
1	1.055429	0.678361	1.002318	0.684901
2	0.842808	0.751134	0.880338	0.720706
3	0.752694	0.781432	0.812097	0.752310
4	0.700818	0.797781	0.787272	0.754935
5	0.663128	0.808247	0.743797	0.771105
6	0.632209	0.817002	0.713448	0.784019
7	0.607964	0.823153	0.686152	0.792419
8	0.591037	0.828754	0.687796	0.790109
9	0.573904	0.833179	0.662540	0.799664
10	0.558532	0.836788	0.662647	0.796304
11	0.547331	0.840522	0.655577	0.797459
12	0.534765	0.842170	0.653356	0.796724
13	0.526503	0.846124	0.641011	0.802499
14	0.520106	0.845731	0.612947	0.815624
15	0.510939	0.849717	0.598359	0.821819
16	0.505103	0.850423	0.595601	0.819929
17	0.495357	0.853294	0.599889	0.818564
18	0.491547	0.854753	0.601457	0.818354
19	0.483743	0.856589	0.593048	0.819089
20	0.479432	0.857672	0.594792	0.817829
21	0.475936	0.859272	0.577689	0.824548
22	0.468976	0.859774	0.592951	0.816884

23	0.466246	0.861845	0.566677	0.829693
24	0.463752	0.862191	0.595900	0.814574
25	0.458705	0.862771	0.554933	0.833788
26	0.458083	0.864074	0.561464	0.830428
27	0.455302	0.864623	0.561529	0.829798
28	0.452050	0.865203	0.560792	0.828748

```
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=2)
```

```
814/814 - 3s - loss: 0.5946 - accuracy: 0.8239
```

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 th with maximum probability.

```
import os
print(os.getcwd())
```

```
/content
```

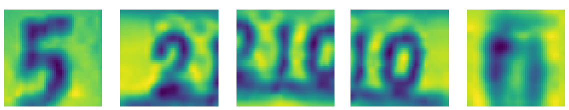
```
checkpoint_best_path = 'checkpoints_best_only/checkpoint'
model_MLP = get_model(X_train[0].shape)
model_MLP.load_weights(checkpoint_best_path)
```

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f4e8d326da0>
```

```
checkpoints_best_path = 'checkpoints_best_only_CNN/checkpoint'
model_CNN = get_new_model(X_train[0].shape)
model_CNN.load_weights(checkpoints_best_path)
```

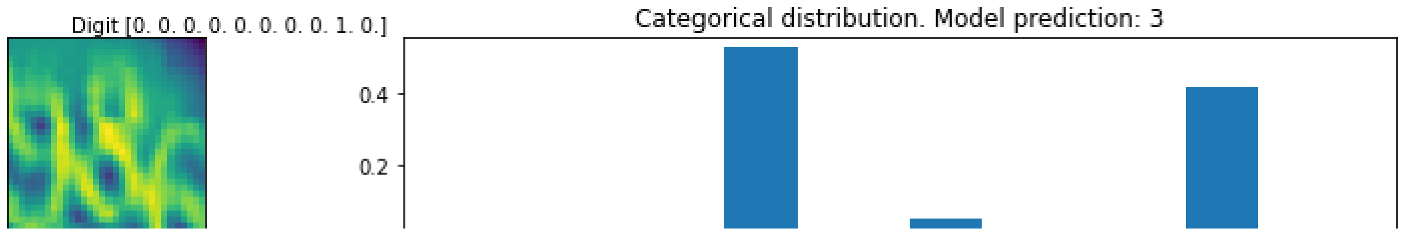
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f4e8d1de8d0>
```

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 5, figsize=(5, 1))
for i in range(5):
    ax[i].set_axis_off()
    ax[i].imshow(X_test[i,:,:,0])
```

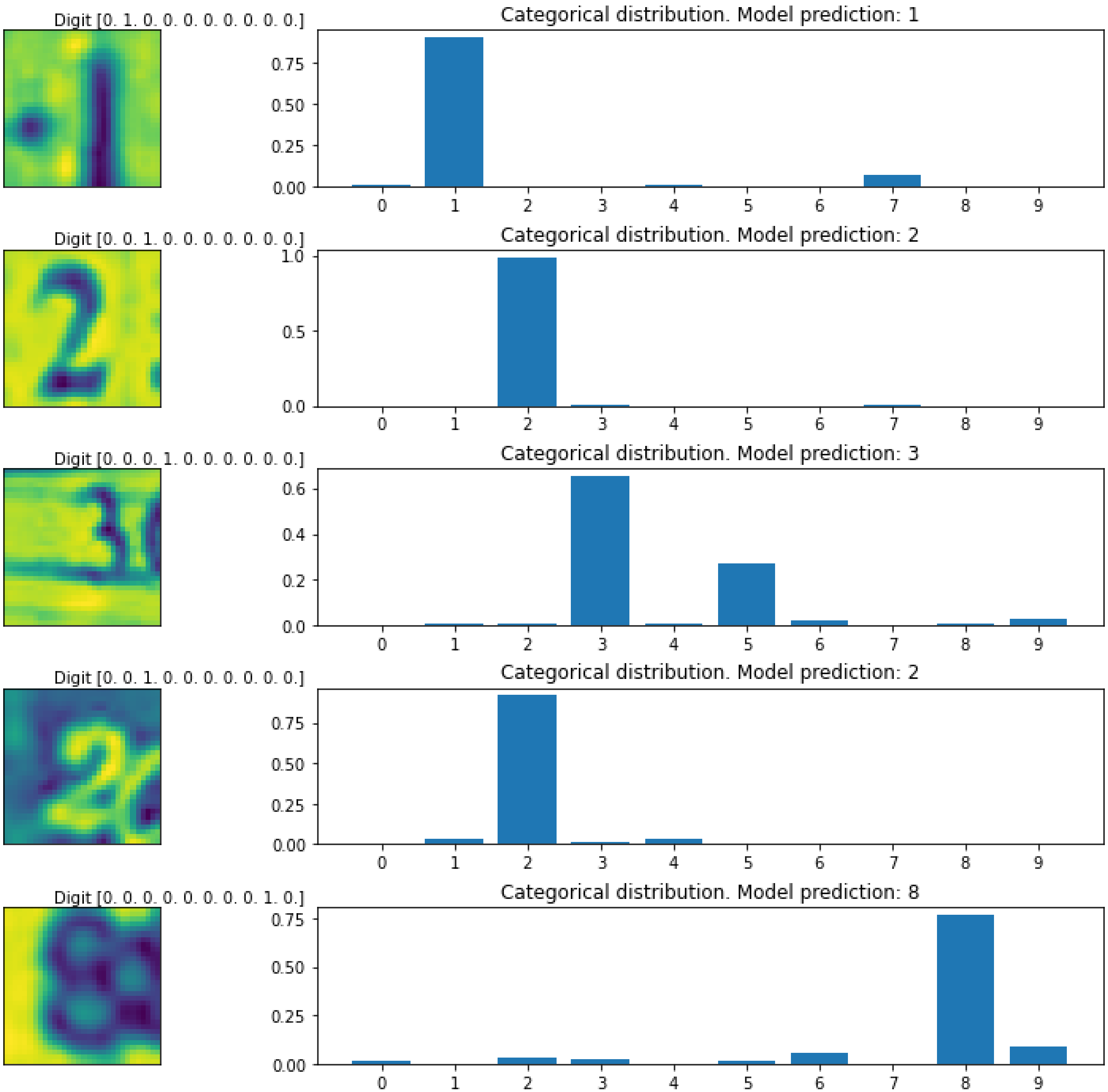


```
num_test_images = X_test.shape[0]

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_MLP.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, random_test_labels)):
    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(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[random_inx, ...]

predictions = model_CNN.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, random_test_labels)):
    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(prediction)}")

plt.show()
```


Digit [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
Categorical distribution. Model prediction: 1
Digit [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
Categorical distribution. Model prediction: 2
Digit [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
Categorical distribution. Model prediction: 3
Digit [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
Categorical distribution. Model prediction: 2
Digit [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
Categorical distribution. Model prediction: 8