

# Capstone Project

January 1, 2022

## 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 pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
from scipy.io import loadmat

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout, Conv2D, MaxPooling2D, Bat
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.optimizers import Adam
```



For the capstone project, you will use the [SVHN dataset](#). This is an image dataset of over 600,000 digit images in all, and is a harder dataset than MNIST as the numbers appear in the context of natural scene images. SVHN is obtained from house numbers in Google Street View images.

- Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu and A. Y. Ng. “Reading Digits in Natural Images with Unsupervised Feature Learning”. NIPS Workshop on Deep Learning and Unsupervised Feature Learning, 2011.

Your goal is to develop an end-to-end workflow for building, training, validating, evaluating and saving a neural network that classifies a real-world image into one of ten classes.

In [2]: # Run this cell to load the dataset

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

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

## 1.2 1. Inspect and preprocess the dataset

- Extract the training and testing images and labels separately from the train and test dictionaries loaded for you.
- Select a random sample of images and corresponding labels from the dataset (at least 10), and display them in a figure.
- Convert the training and test images to grayscale by taking the average across all colour channels for each pixel. *Hint: retain the channel dimension, which will now have size 1.*
- Select a random sample of the grayscale images and corresponding labels from the dataset (at least 10), and display them in a figure.

In [3]: # print(train.keys())

```
train_images, train_labels = train['X'], train['y']
test_images, test_labels = test['X'], test['y']
```

```

# # plt.imshow(train_images)
# print(train)
print(train_images.shape)
print(test_images.ndim)

```

(32, 32, 3, 73257)

4

```

In [4]: train_images = np.moveaxis(train_images, -1, 0)
        test_images = np.moveaxis(test_images, -1, 0)

```

```

In [5]: print(train_images.shape)

```

(73257, 32, 32, 3)

```

In [6]: train_labels = np.where(train_labels==10, 0, train_labels)
        test_labels = np.where(test_labels==10, 0, test_labels)

```

```

In [7]: random_indices = random.sample(range(0,train_images.shape[0]), 10)
        fig, ax = plt.subplots(1,10, figsize = (10,10))

```

```

for i in range(10):
    ax[i].set_axis_off()
    ax[i].imshow(train_images[random_indices[i]])
    ax[i].set_title(train_labels[random_indices[i]])

```



```

In [8]: def convert2gray(train_data, test_data):

```

```

    train_data = np.mean(train_data, axis = -1, keepdims = True)
    test_data = np.mean(test_data, axis = -1, keepdims = True)

    return (train_data, test_data)

```

```

In [9]: gray_train_img, gray_test_img = convert2gray(train_images, test_images)

```

```

In [10]: random_indices = random.sample(range(0,gray_train_img.shape[0]), 10)
         fig, ax = plt.subplots(1,10, figsize = (10,10))

```

```

for i in range(10):
    ax[i].set_axis_off()
    ax[i].imshow(gray_train_img[random_indices[i], :, :, 0], cmap='gray')
    ax[i].set_title(train_labels[random_indices[i]])

```



### 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 [11]: `def get_new_MLP_model():`

```

    model = Sequential([
        Flatten(input_shape= train_images[0].shape),
        Dense(512, activation='relu'),
        Dense(512, activation='relu'),
        Dense(256, activation='relu'),
        Dense(128, activation='relu'),
        Dense(10, activation='softmax')
    ])

    return model

```

```

model = get_new_MLP_model()
model.summary()

```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 3072)	0

dense (Dense)	(None, 512)	1573376
-----		
dense_1 (Dense)	(None, 512)	262656
-----		
dense_2 (Dense)	(None, 256)	131328
-----		
dense_3 (Dense)	(None, 128)	32896
-----		
dense_4 (Dense)	(None, 10)	1290
=====		
Total params: 2,001,546		
Trainable params: 2,001,546		
Non-trainable params: 0		
-----		

```
In [12]: cp_path = 'Checkpoint/best_model'
         cp = ModelCheckpoint(cp_path,
                             save_best_only=True,
                             save_weights_only=True,
                             verbose=False,
                             save_freq='epoch',
                             monitor='val_loss',
                             mode='min')

In [13]: early_stop = EarlyStopping(monitor='val_loss', mode='min', patience=3)

In [14]: model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['acc'])

In [15]: history = model.fit(train_images, train_labels, epochs = 30, batch_size=64, verbose=1)

Train on 65931 samples, validate on 7326 samples
Epoch 1/30
65931/65931 [=====] - 102s 2ms/sample - loss: 12.0728 - accuracy: 0.21
Epoch 2/30
65931/65931 [=====] - 103s 2ms/sample - loss: 1.4894 - accuracy: 0.51
Epoch 3/30
65931/65931 [=====] - 99s 2ms/sample - loss: 1.3211 - accuracy: 0.583
Epoch 4/30
65931/65931 [=====] - 100s 2ms/sample - loss: 1.2268 - accuracy: 0.61
Epoch 5/30
65931/65931 [=====] - 100s 2ms/sample - loss: 1.1679 - accuracy: 0.63
Epoch 6/30
65931/65931 [=====] - 100s 2ms/sample - loss: 1.1252 - accuracy: 0.65
Epoch 7/30
65931/65931 [=====] - 100s 2ms/sample - loss: 1.0654 - accuracy: 0.67
Epoch 8/30
65931/65931 [=====] - 100s 2ms/sample - loss: 1.0283 - accuracy: 0.68
Epoch 9/30
```

```

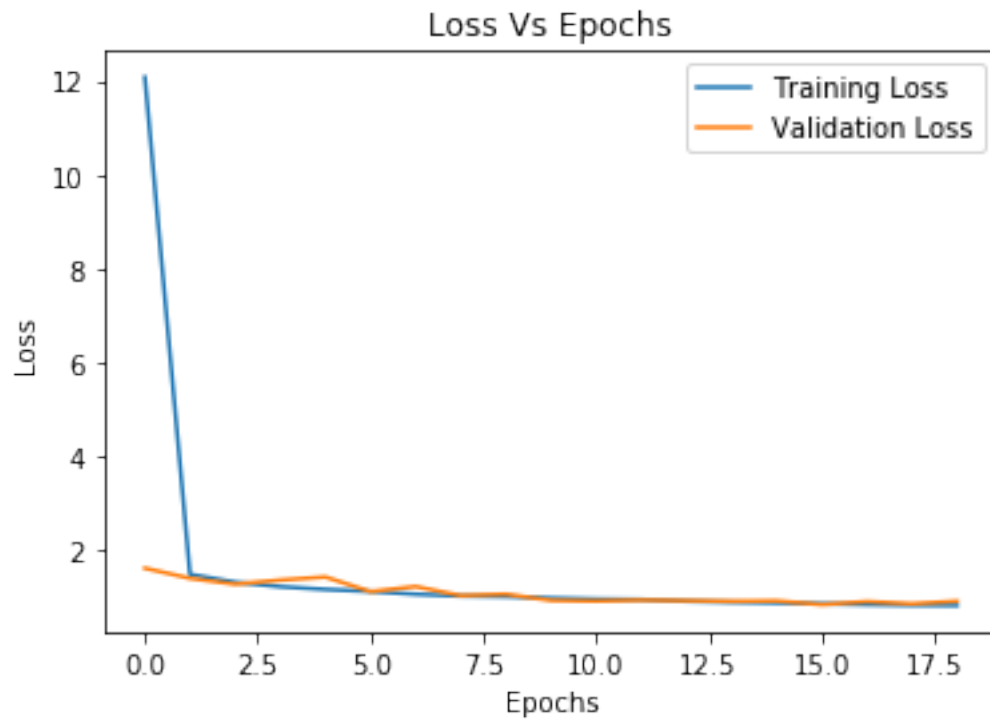
65931/65931 [=====] - 99s 2ms/sample - loss: 1.0159 - accuracy: 0.688
Epoch 10/30
65931/65931 [=====] - 100s 2ms/sample - loss: 0.9835 - accuracy: 0.69
Epoch 11/30
65931/65931 [=====] - 100s 2ms/sample - loss: 0.9622 - accuracy: 0.70
Epoch 12/30
65931/65931 [=====] - 100s 2ms/sample - loss: 0.9534 - accuracy: 0.70
Epoch 13/30
65931/65931 [=====] - 101s 2ms/sample - loss: 0.9173 - accuracy: 0.71
Epoch 14/30
65931/65931 [=====] - 101s 2ms/sample - loss: 0.8975 - accuracy: 0.72
Epoch 15/30
65931/65931 [=====] - 102s 2ms/sample - loss: 0.8813 - accuracy: 0.72
Epoch 16/30
65931/65931 [=====] - 100s 2ms/sample - loss: 0.8630 - accuracy: 0.73
Epoch 17/30
65931/65931 [=====] - 103s 2ms/sample - loss: 0.8452 - accuracy: 0.74
Epoch 18/30
65931/65931 [=====] - 103s 2ms/sample - loss: 0.8324 - accuracy: 0.74
Epoch 19/30
65931/65931 [=====] - 103s 2ms/sample - loss: 0.8260 - accuracy: 0.74

```

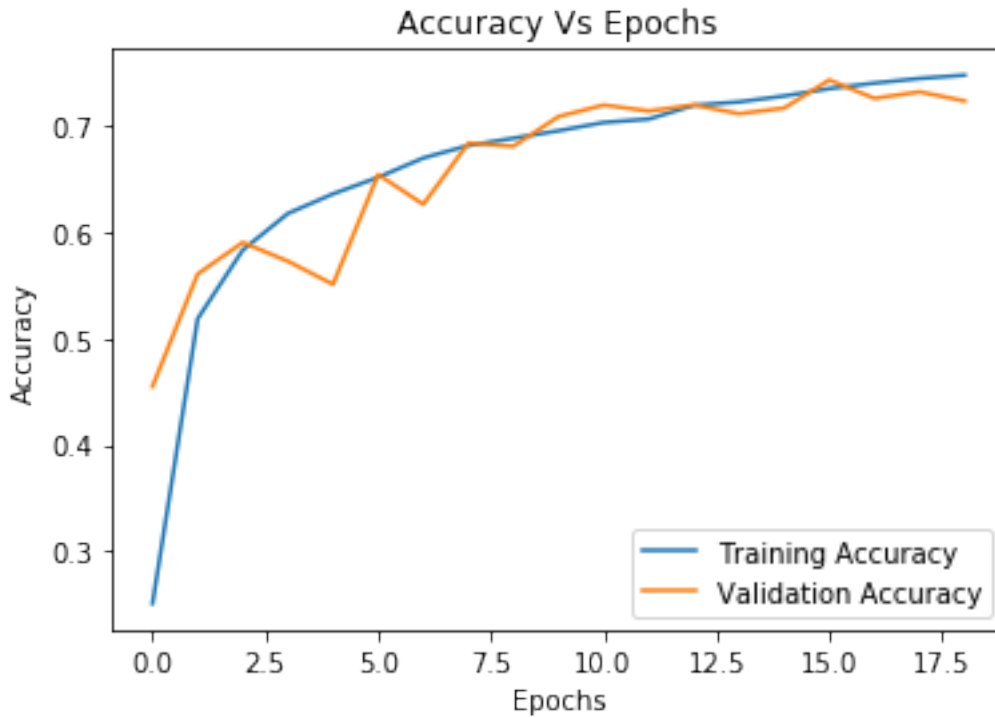
```

In [16]: plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
         plt.title("Loss Vs Epochs")
         plt.ylabel('Loss')
         plt.xlabel('Epochs')
         plt.legend(['Training Loss', 'Validation Loss'], loc='upper right')
         plt.show()

```



```
In [17]: plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title("Accuracy Vs Epochs")
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['Training Accuracy', 'Validation Accuracy'], loc='lower right')
plt.show()
```



```
In [18]: test_loss, test_accuracy = model.evaluate(test_images, test_labels, verbose=False)
         print(f"Test Loss is {test_loss}")
         print(f"Test Accuracy is {test_accuracy}")
```

Test Loss is 1.065633964450651  
 Test Accuracy is 0.6899201273918152

### 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]: def get_new_CNN_model():
```

```
    model = Sequential([
        Conv2D(16, (3,3), padding='same', activation='relu', input_shape= train_image_shape),
        BatchNormalization(),
        MaxPooling2D((2,2)),
        Conv2D(16, (3,3), padding='same', activation='relu'),
        BatchNormalization(),
        MaxPooling2D((2,2)),
        Conv2D(16, (3,3), padding='same', activation='relu'),
        BatchNormalization(),
        MaxPooling2D((2,2)),
        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.2),
        Dense(128, activation='relu'),
        Dropout(0.2),
        Dense(10, activation='softmax')
    ])

    return model
```

```
cnn_model = get_new_CNN_model()
cnn_model.summary()
```

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 16)	448
batch_normalization (Batch Normalization)	(None, 32, 32, 16)	64
max_pooling2d (MaxPooling2D)	(None, 16, 16, 16)	0
conv2d_1 (Conv2D)	(None, 16, 16, 16)	2320
batch_normalization_1 (Batch Normalization)	(None, 16, 16, 16)	64
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 16)	0
conv2d_2 (Conv2D)	(None, 8, 8, 16)	2320
batch_normalization_2 (Batch Normalization)	(None, 8, 8, 16)	64
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 16)	0

```

-----
flatten_1 (Flatten)          (None, 256)          0
-----
dense_5 (Dense)              (None, 256)         65792
-----
dropout (Dropout)           (None, 256)          0
-----
dense_6 (Dense)              (None, 128)         32896
-----
dropout_1 (Dropout)          (None, 128)          0
-----
dense_7 (Dense)              (None, 10)          1290
=====
Total params: 105,258
Trainable params: 105,162
Non-trainable params: 96
-----

```

```

In [20]: checkpoint_path = 'CNN_Checkpoint/best_model'
         cp = ModelCheckpoint(checkpoint_path,
                             save_best_only=True,
                             save_weights_only=True,
                             verbose=False,
                             save_freq='epoch',
                             monitor='val_loss',
                             mode='min')

```

```

In [21]: early_stop = EarlyStopping(monitor='val_loss', mode='min', patience=3)

```

```

In [22]: cnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=[

```

```

In [23]: history = cnn_model.fit(train_images, train_labels, epochs=30, batch_size=64,
                                verbose=1, validation_split=0.1, callbacks=[cp, early_stop])

```

Train on 65931 samples, validate on 7326 samples

Epoch 1/30

65931/65931 [=====] - 366s 6ms/sample - loss: 0.9533 - accuracy: 0.688

Epoch 2/30

65931/65931 [=====] - 364s 6ms/sample - loss: 0.4912 - accuracy: 0.84

Epoch 3/30

65931/65931 [=====] - 362s 5ms/sample - loss: 0.4120 - accuracy: 0.87

Epoch 4/30

65931/65931 [=====] - 365s 6ms/sample - loss: 0.3660 - accuracy: 0.88

Epoch 5/30

65931/65931 [=====] - 362s 5ms/sample - loss: 0.3310 - accuracy: 0.89

Epoch 6/30

65931/65931 [=====] - 363s 6ms/sample - loss: 0.3069 - accuracy: 0.90

Epoch 7/30

```

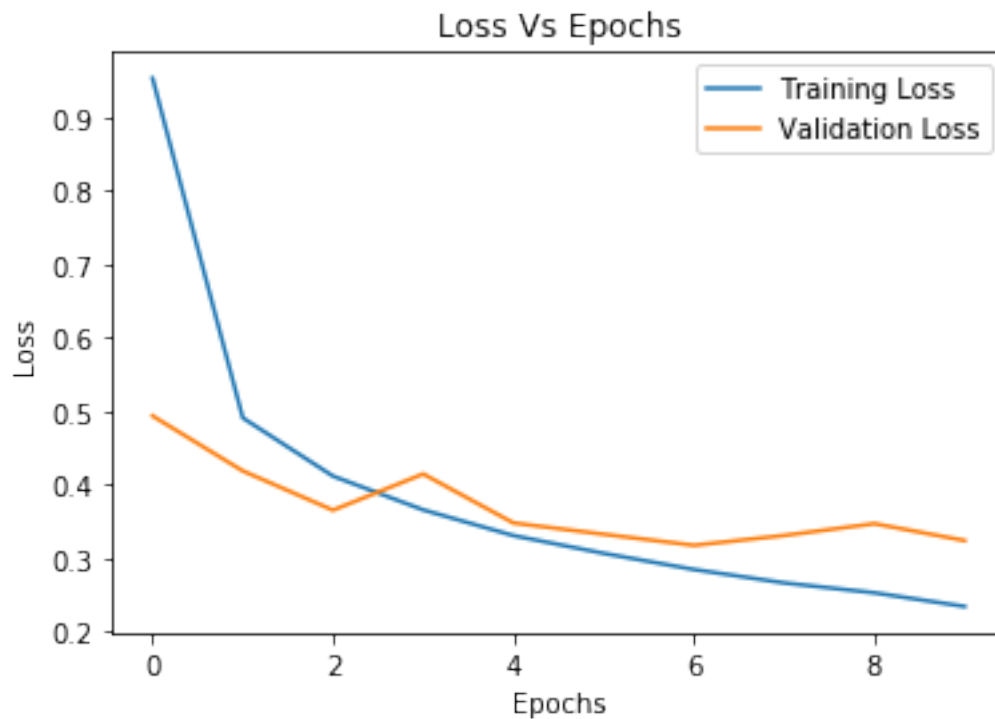
65931/65931 [=====] - 363s 6ms/sample - loss: 0.2849 - accuracy: 0.91
Epoch 8/30
65931/65931 [=====] - 361s 5ms/sample - loss: 0.2667 - accuracy: 0.91
Epoch 9/30
65931/65931 [=====] - 361s 5ms/sample - loss: 0.2532 - accuracy: 0.92
Epoch 10/30
65931/65931 [=====] - 360s 5ms/sample - loss: 0.2346 - accuracy: 0.92

```

```

In [24]: plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("Loss Vs Epochs")
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['Training Loss', 'Validation Loss'], loc='upper right')
plt.show()

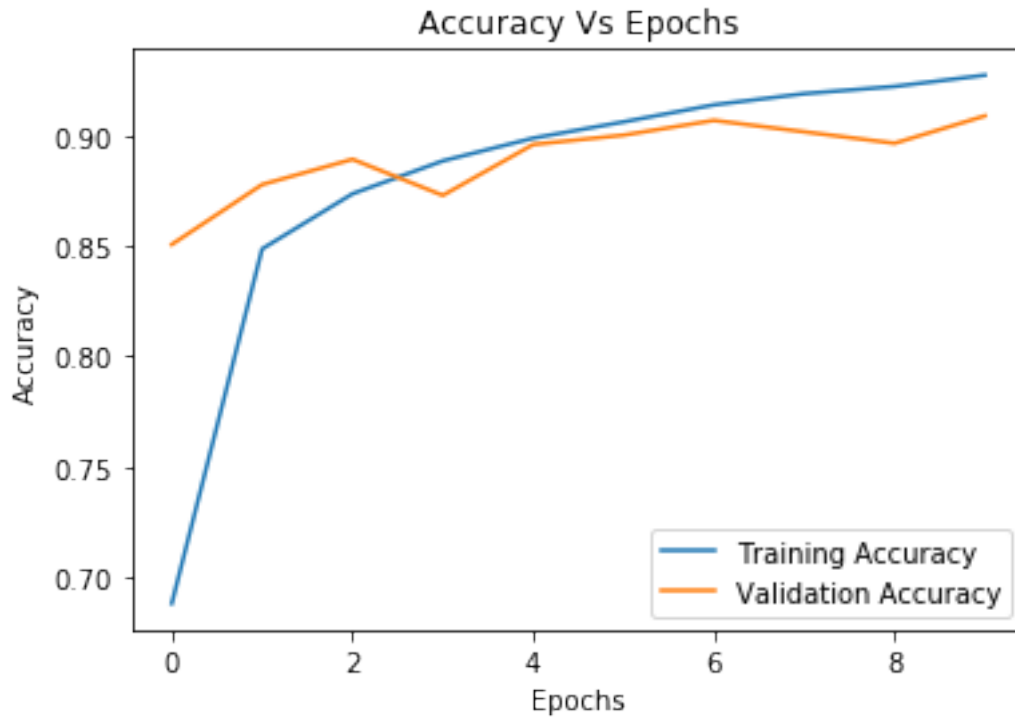
```



```

In [25]: plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title("Accuracy Vs Epochs")
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['Training Accuracy', 'Validation Accuracy'], loc='lower right')
plt.show()

```



## 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 [27]: Best_MLP = get_new_MLP_model()
        Best_MLP.load_weights('Checkpoint/best_model')
```

```
Out[27]: <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f9490322cc0>
```

```
In [28]: Best_CNN = get_new_CNN_model()
        Best_CNN.load_weights('CNN_Checkpoint/best_model')
```

```
Out[28]: <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f9490057a90>
```

```
In [29]: def show_predictive_distribution(model):

        num_test_images = test_images.shape[0]

        random_inx = np.random.choice(num_test_images, 5)
        random_test_images = test_images[random_inx, ...]
```

```

random_test_labels = test_labels[random_inx, ...]

predictions = model.predict(random_test_images)

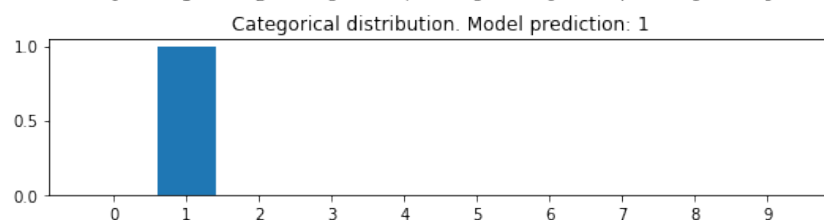
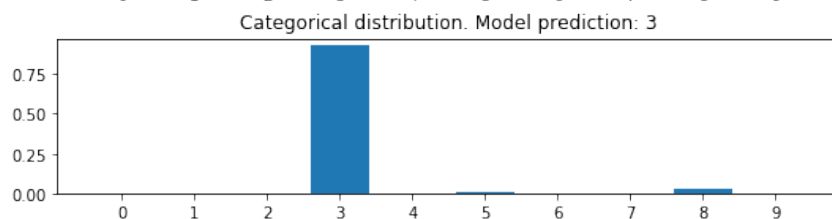
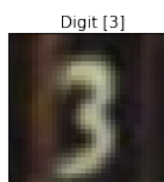
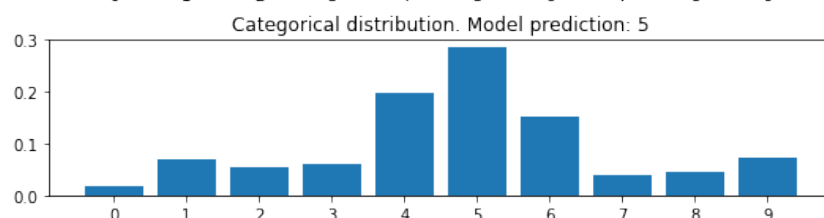
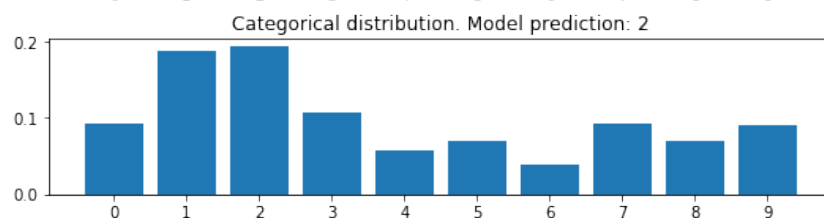
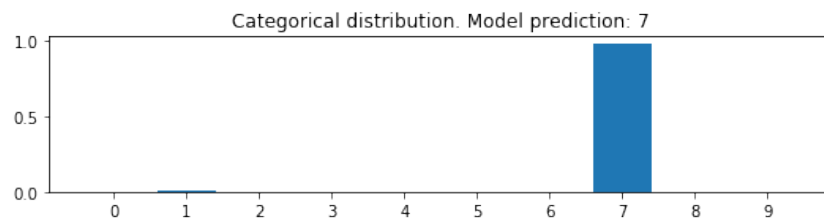
fig, axes = plt.subplots(5, 2, figsize=(16, 12))
fig.subplots_adjust(hspace=0.4, wspace=-0.2)

for i, (prediction, image, label) in enumerate(zip(predictions, random_test_images)):
    axes[i, 0].imshow(np.squeeze(image))
    axes[i, 0].get_xaxis().set_visible(False)
    axes[i, 0].get_yaxis().set_visible(False)
    axes[i, 0].text(10., -1.5, f'Digit {label}')
    axes[i, 1].bar(np.arange(len(prediction)), prediction)
    axes[i, 1].set_xticks(np.arange(len(prediction)))
    axes[i, 1].set_title(f"Categorical distribution. Model prediction: {np.argmax(prediction)}")

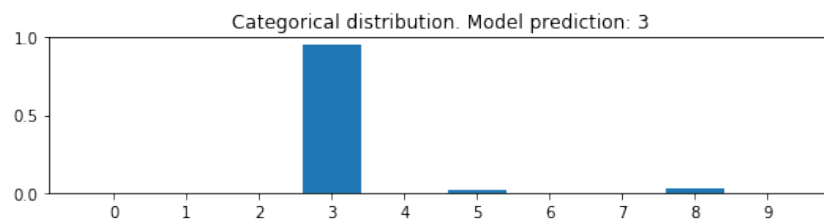
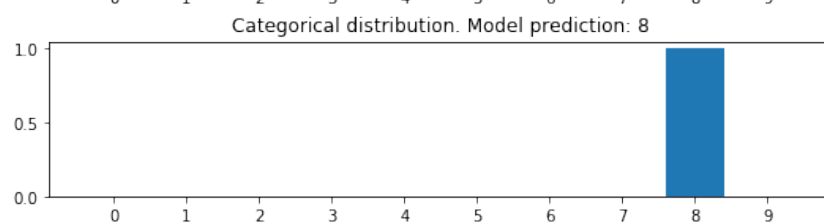
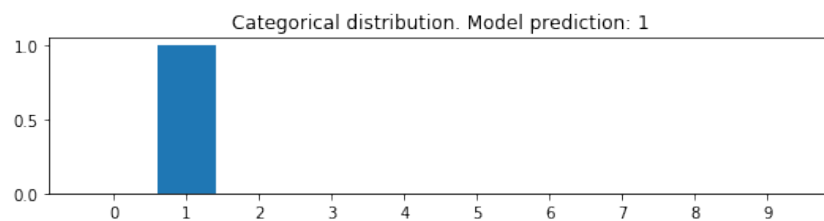
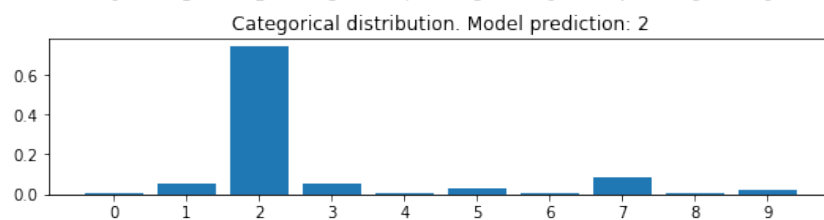
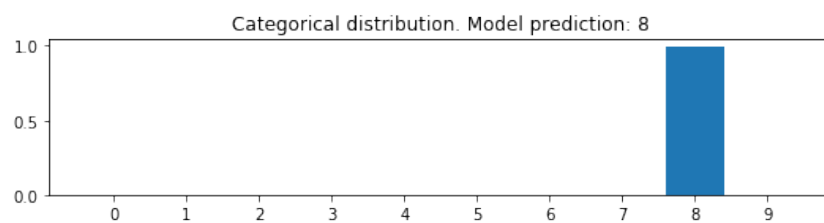
plt.show()

```

In [30]: show\_predictive\_distribution(Best\_MLP)



In [31]: `show_predictive_distribution(Best_CNN)`



In [ ]: