Course_1_Capstone_Project COMPLETED

May 18, 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 (you could download the notebook with File -> Download .ipynb, open the notebook locally, and then File -> Download as -> PDF via LaTeX), and 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.

```python
[1]: import tensorflow as tf
     from scipy.io import loadmat
     import numpy as np
     import matplotlib.pyplot as plt
     from tensorflow.keras import Sequential
     from tensorflow.keras.layers import Dense, Softmax, Conv2D, MaxPooling2D,␣
      ↪Flatten
     from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
     import os
```

```python
[2]: plt.rcParams['figure.figsize'] = 16,12
```

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.

The train and test datasets required for this project can be downloaded from here and here. Once unzipped, you will have two files: `train_32x32.mat` and `test_32x32.mat`. You should store these files in Drive for use in this Colab notebook.

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.

```
[3]: # Run this cell to connect to your Drive folder

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

Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).

```
[6]: # Load the dataset from your Drive folder

     train = loadmat('/content/gdrive/MyDrive/Colab Notebooks/Course 1/data/
       ↪train_32x32.mat')
     test = loadmat('/content/gdrive/MyDrive/Colab Notebooks/Course 1/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.
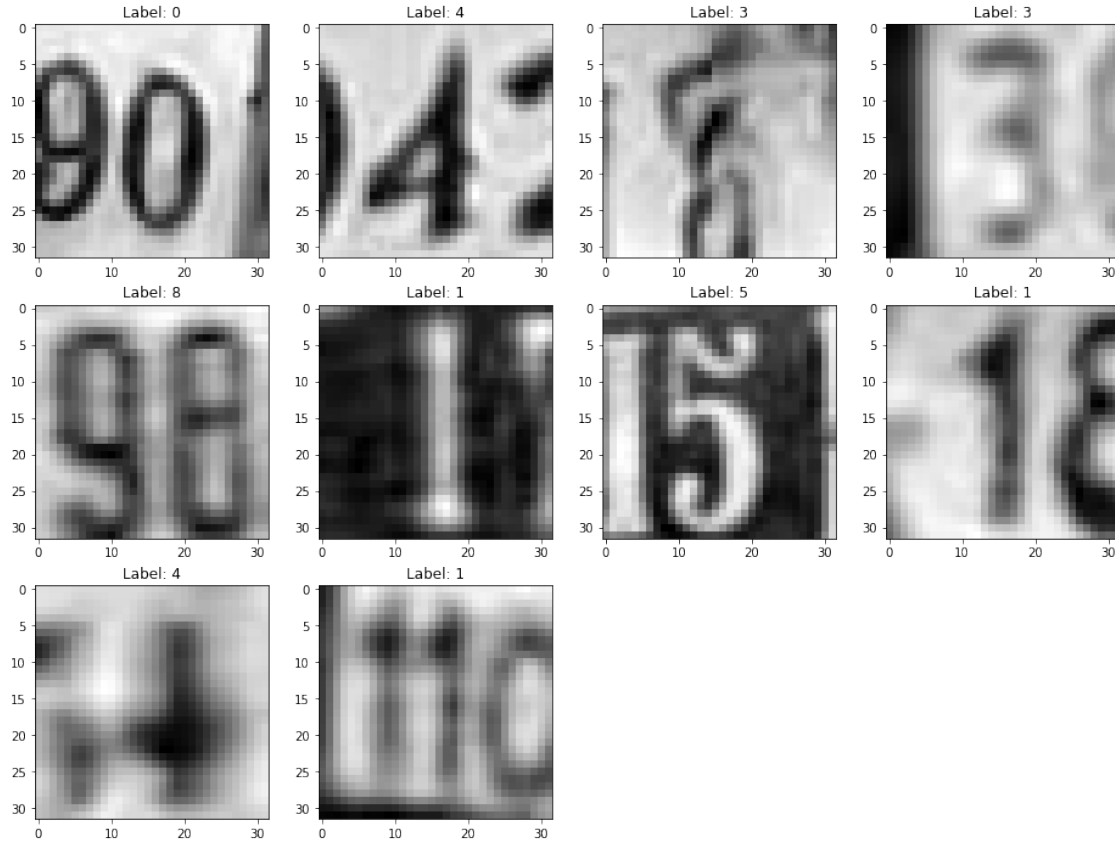
```
[7]:
```

```
[11]:
```

10 Images from Train Data

Label: 6   Label: 4   Label: 7   Label: 2

Label: 9   Label: 8   Label: 5   Label: 0

Label: 1   Label: 2

[9]:

[12]:

```
[ ]:
```

```
[ ]:
```

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.

[13]:

```
Model: "sequential"

_____
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 1024)              0
_____
dense (Dense)                (None, 100)               102500
_____
dense_1 (Dense)              (None, 200)               20200
_____
dense_2 (Dense)              (None, 300)               60300
_____
dense_3 (Dense)              (None, 200)               60200
_____
dense_4 (Dense)              (None, 100)               20100
_____
dense_5 (Dense)              (None, 200)               20200
_____
dense_6 (Dense)              (None, 10)                2010
=================================================================
Total params: 285,510
Trainable params: 285,510
Non-trainable params: 0
_____
```

[24]: `# Include the epoch in the file name (uses `str.format`)`

2   LEARNING POINTS

1) Cannot use relative path i.e. use ./ in front of the path as a parameter to pass into the argument filepath.

2) If want to create a new folder, say, 'checkpoint_best_only_CNN', must end the string with a /, slash, e.g. 'something/checkpoint_best_only_CNN/'.

[25]:

[26]:

```
Epoch 1/30
487/487 [==============================] - 5s 11ms/step - loss: 1.0454 -
accuracy: 0.6597 - val_loss: 1.0196 - val_accuracy: 0.6709
Epoch 2/30
487/487 [==============================] - 5s 10ms/step - loss: 1.0195 -
accuracy: 0.6694 - val_loss: 1.0459 - val_accuracy: 0.6588
Epoch 3/30
```

```
487/487 [==============================] - 5s 10ms/step - loss: 1.0002 -
accuracy: 0.6756 - val_loss: 1.0289 - val_accuracy: 0.6670
Epoch 4/30
487/487 [==============================] - 5s 10ms/step - loss: 0.9812 -
accuracy: 0.6831 - val_loss: 1.0150 - val_accuracy: 0.6699
Epoch 5/30
487/487 [==============================] - 5s 10ms/step - loss: 0.9813 -
accuracy: 0.6822 - val_loss: 1.0144 - val_accuracy: 0.6701
Epoch 6/30
487/487 [==============================] - 5s 10ms/step - loss: 0.9642 -
accuracy: 0.6889 - val_loss: 0.9888 - val_accuracy: 0.6802
Epoch 7/30
487/487 [==============================] - 5s 10ms/step - loss: 0.9575 -
accuracy: 0.6900 - val_loss: 0.9949 - val_accuracy: 0.6811
Epoch 8/30
487/487 [==============================] - 5s 10ms/step - loss: 0.9470 -
accuracy: 0.6932 - val_loss: 1.0355 - val_accuracy: 0.6612
Epoch 9/30
487/487 [==============================] - 5s 10ms/step - loss: 0.9314 -
accuracy: 0.6993 - val_loss: 1.0068 - val_accuracy: 0.6750
```

[27]:

Accuracy vs. epochs
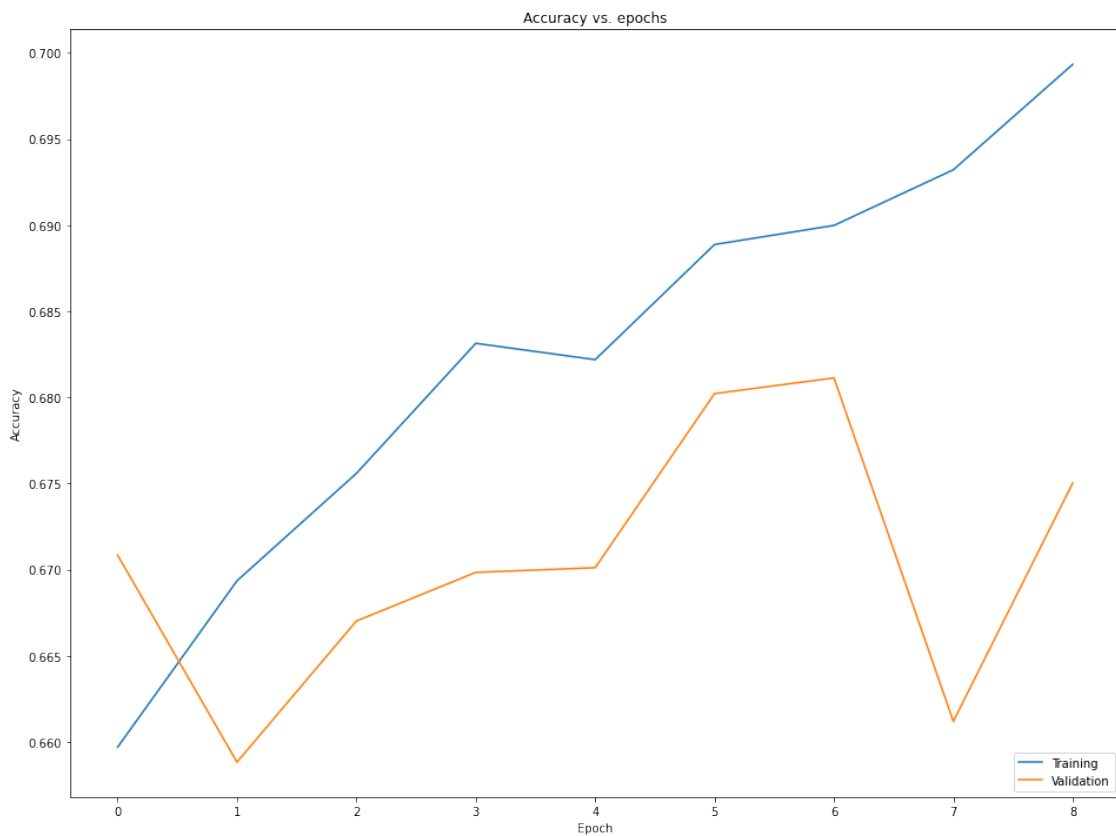
[28]:
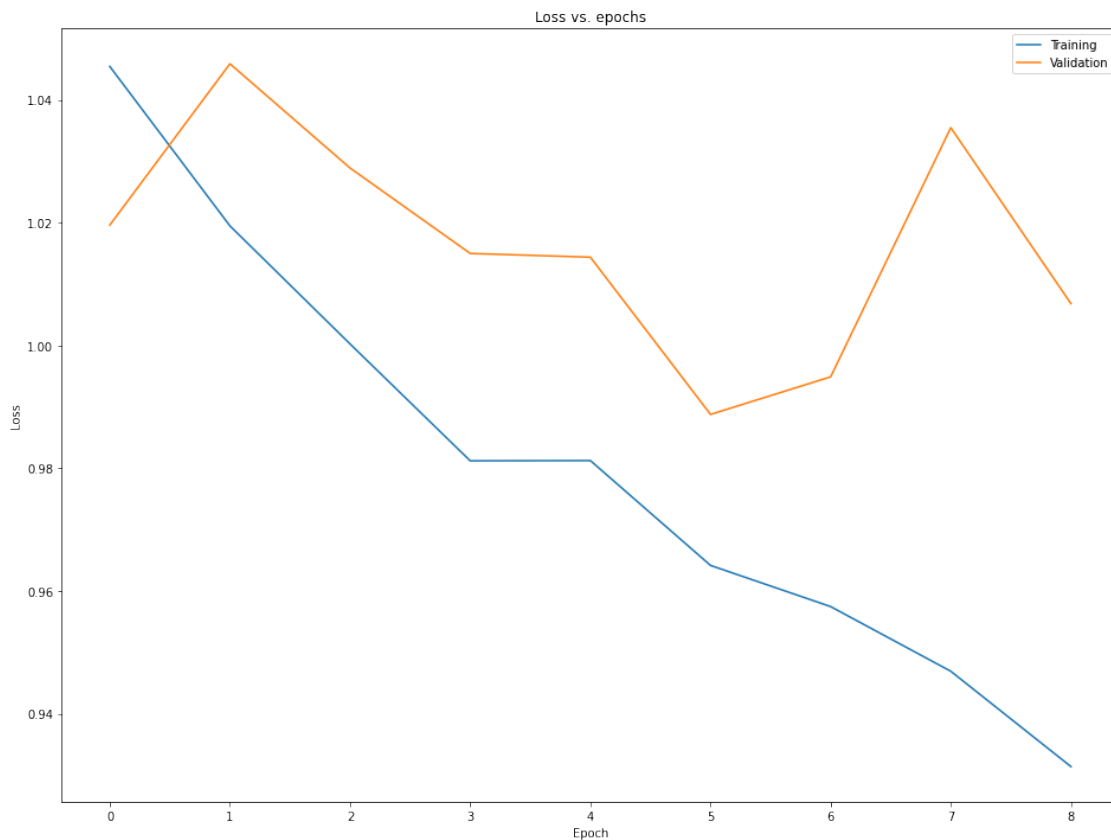
[29]:

[29]: [1.1166274547576904, 0.6447833180427551]

2.1 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.

[30]:

```
Model: "sequential_1"
_____
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 16)        416

max_pooling2d (MaxPooling2D) (None, 16, 16, 16)        0

conv2d_1 (Conv2D)            (None, 12, 12, 32)        12832

max_pooling2d_1 (MaxPooling2 (None, 6, 6, 32)          0

conv2d_2 (Conv2D)            (None, 4, 4, 64)          18496

max_pooling2d_2 (MaxPooling2 (None, 2, 2, 64)          0

flatten_1 (Flatten)          (None, 256)               0

dense_7 (Dense)              (None, 40)                10280

dense_8 (Dense)              (None, 10)                410
=================================================================
Total params: 42,434
Trainable params: 42,434
Non-trainable params: 0
_____
```

[31]: `# Include the epoch in the file name (uses `str.format`)`

[32]:

[33]:

```
Epoch 1/30
487/487 [==============================] - 63s 129ms/step - loss: 1.8387 -
accuracy: 0.3591 - val_loss: 0.7254 - val_accuracy: 0.7877
Epoch 2/30
487/487 [==============================] - 63s 130ms/step - loss: 0.6583 -
accuracy: 0.8088 - val_loss: 0.5578 - val_accuracy: 0.8369
Epoch 3/30
487/487 [==============================] - 63s 129ms/step - loss: 0.5172 -
accuracy: 0.8481 - val_loss: 0.4992 - val_accuracy: 0.8615
```

```
Epoch 4/30
487/487 [==============================] - 63s 130ms/step - loss: 0.4570 -
accuracy: 0.8680 - val_loss: 0.4439 - val_accuracy: 0.8719
Epoch 5/30
487/487 [==============================] - 63s 130ms/step - loss: 0.4030 -
accuracy: 0.8835 - val_loss: 0.4374 - val_accuracy: 0.8719
Epoch 6/30
487/487 [==============================] - 63s 129ms/step - loss: 0.3866 -
accuracy: 0.8882 - val_loss: 0.3931 - val_accuracy: 0.8850
Epoch 7/30
487/487 [==============================] - 63s 129ms/step - loss: 0.3592 -
accuracy: 0.8985 - val_loss: 0.3752 - val_accuracy: 0.8922
Epoch 8/30
487/487 [==============================] - 63s 129ms/step - loss: 0.3370 -
accuracy: 0.9021 - val_loss: 0.3681 - val_accuracy: 0.8916
Epoch 9/30
487/487 [==============================] - 63s 128ms/step - loss: 0.3212 -
accuracy: 0.9069 - val_loss: 0.3826 - val_accuracy: 0.8884
Epoch 10/30
487/487 [==============================] - 63s 129ms/step - loss: 0.3100 -
accuracy: 0.9094 - val_loss: 0.3597 - val_accuracy: 0.8967
Epoch 11/30
487/487 [==============================] - 62s 128ms/step - loss: 0.2920 -
accuracy: 0.9165 - val_loss: 0.3507 - val_accuracy: 0.9008
Epoch 12/30
487/487 [==============================] - 64s 131ms/step - loss: 0.2776 -
accuracy: 0.9195 - val_loss: 0.3502 - val_accuracy: 0.8999
Epoch 13/30
487/487 [==============================] - 62s 128ms/step - loss: 0.2703 -
accuracy: 0.9214 - val_loss: 0.3550 - val_accuracy: 0.8982
Epoch 14/30
487/487 [==============================] - 62s 128ms/step - loss: 0.2685 -
accuracy: 0.9211 - val_loss: 0.3531 - val_accuracy: 0.8994
Epoch 15/30
487/487 [==============================] - 62s 128ms/step - loss: 0.2554 -
accuracy: 0.9248 - val_loss: 0.3514 - val_accuracy: 0.9004
```

[34]:

Accuracy vs. epochs

[35]:

Loss vs. epochs
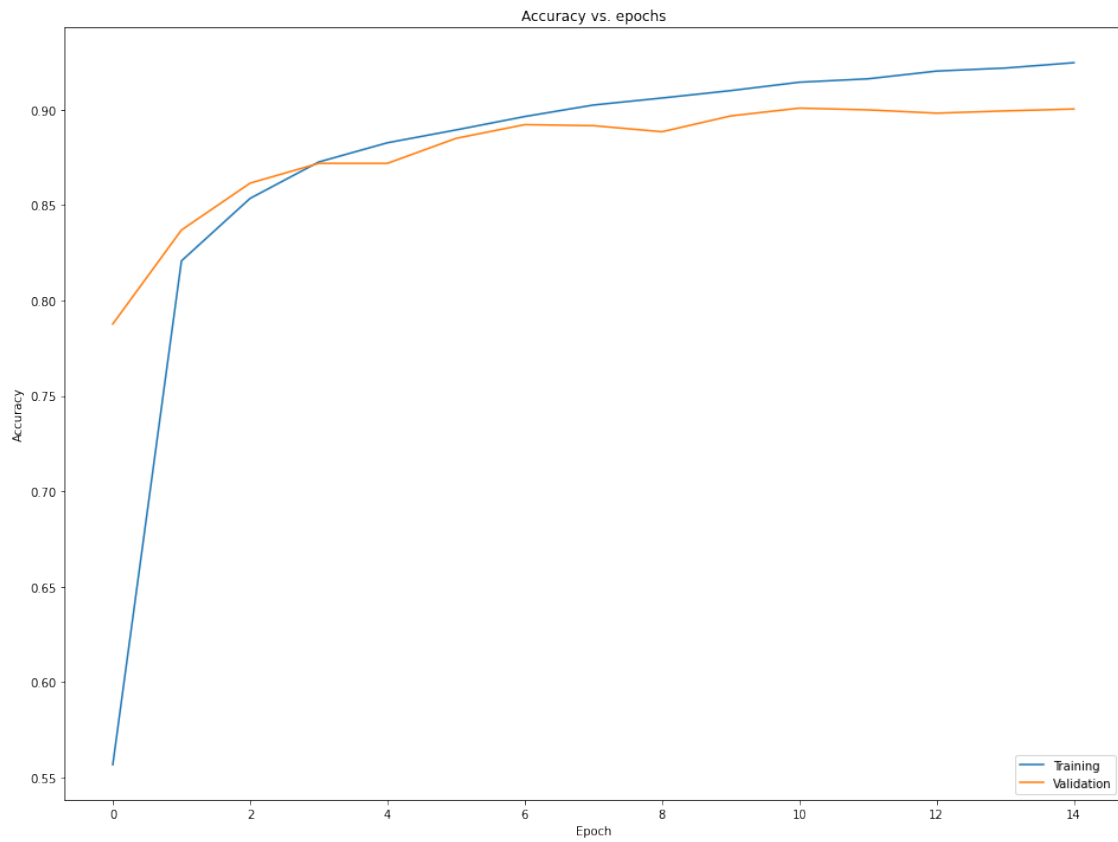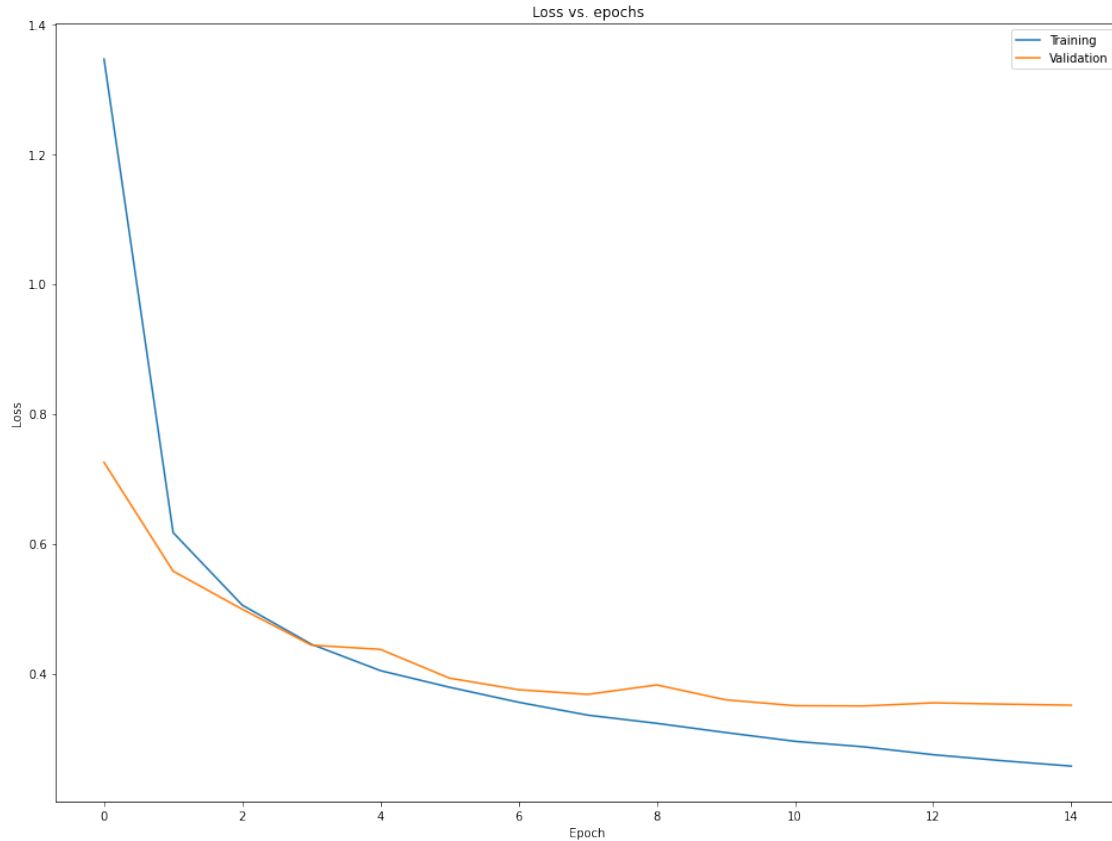
[36]: [0.3826526403427124, 0.8985095024108887]

2.2 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.

[42]:

```
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.iter
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.beta_1
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.beta_2
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.decay
WARNING:tensorflow:Unresolved object in checkpoint:
(root).optimizer.learning_rate
```

```
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'
for (root).layer_with_weights-0.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'
for (root).layer_with_weights-0.bias
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'
for (root).layer_with_weights-1.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'
for (root).layer_with_weights-1.bias
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'
for (root).layer_with_weights-2.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'
for (root).layer_with_weights-2.bias
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'
for (root).layer_with_weights-3.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'
for (root).layer_with_weights-3.bias
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'
for (root).layer_with_weights-4.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'
for (root).layer_with_weights-4.bias
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'
for (root).layer_with_weights-5.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'
for (root).layer_with_weights-5.bias
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'
for (root).layer_with_weights-6.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'm'
for (root).layer_with_weights-6.bias
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-0.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-0.bias
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-1.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-1.bias
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-2.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-2.bias
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-3.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-3.bias
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-4.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-4.bias
```

```
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-5.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-5.bias
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-6.kernel
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer's state 'v'
for (root).layer_with_weights-6.bias
WARNING:tensorflow:A checkpoint was restored (e.g. tf.train.Checkpoint.restore
or tf.keras.Model.load_weights) but not all checkpointed values were used. See
above for specific issues. Use expect_partial() on the load status object, e.g.
tf.train.Checkpoint.restore(…).expect_partial(), to silence these warnings, or
use assert_consumed() to make the check explicit. See
https://www.tensorflow.org/guide/checkpoint#loading_mechanics for details.
```

[42]: `<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f31e5390d10>`

3 Learning Point

load_weights() returns a 'CheckpointLoadStatus' object and NOT a model, because the weights are loaded IN-PLACE! Therefore, do not do this:

```
model = blabla
model2 = model.load_weights()
```

should instead do:

```
model = blabla
model.load_weights()
```

[43]:

```
[[2.87126862e-02 1.76016055e-03 6.42592553e-03 2.71607051e-03
  8.75100493e-03 3.27603668e-02 7.88829327e-01 2.43691844e-03
  1.18041404e-01 9.56623256e-03]
 [1.20029246e-04 1.20691108e-02 1.88511312e-01 6.56933757e-03
  2.85679766e-04 7.83334603e-04 9.40611819e-04 7.89610207e-01
  1.56597831e-04 9.53774608e-04]
 [2.88867410e-02 3.24640833e-02 2.39320528e-02 3.44878346e-01
  4.27006707e-02 3.22094858e-01 9.77263227e-02 7.08594173e-03
  4.61586230e-02 5.40722497e-02]
 [5.09630283e-03 1.57797299e-02 7.82472312e-01 3.44943143e-02
  4.04895190e-03 3.62685770e-02 1.00492798e-02 1.23272268e-02
  3.11413798e-02 6.83219358e-02]
 [2.00383201e-05 6.40614948e-04 5.44675708e-01 2.59770197e-04
  1.58106395e-05 3.03087108e-05 1.53798115e-04 4.54153419e-01
  7.13381860e-06 4.34253270e-05]]
```
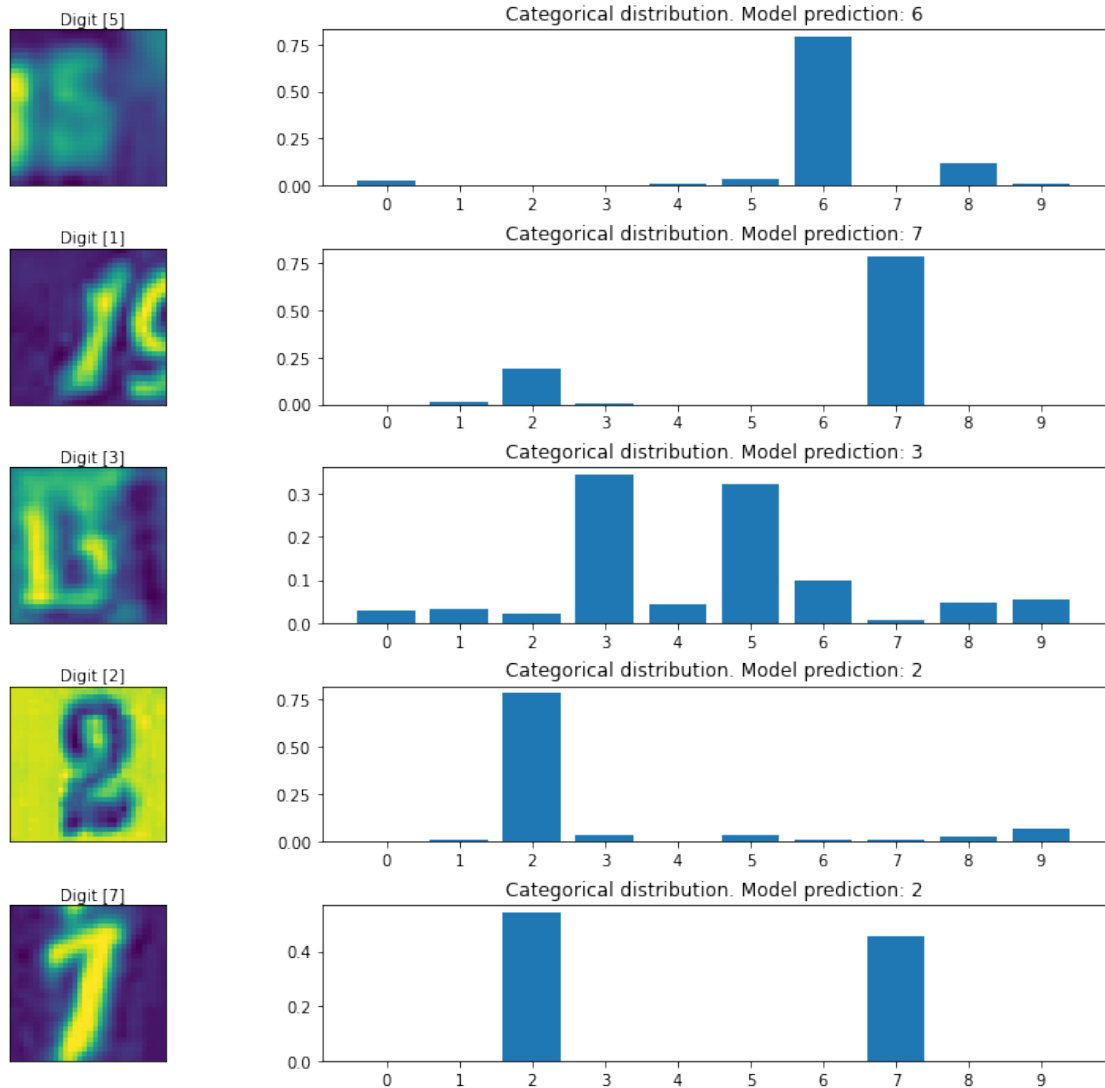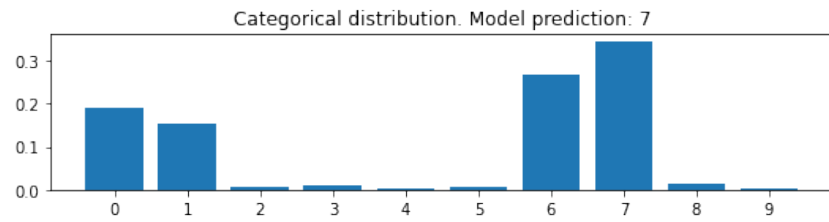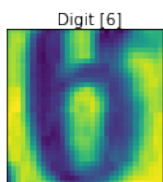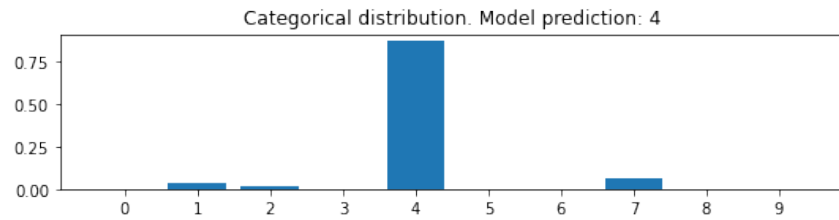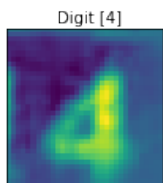
```
[44]:
```
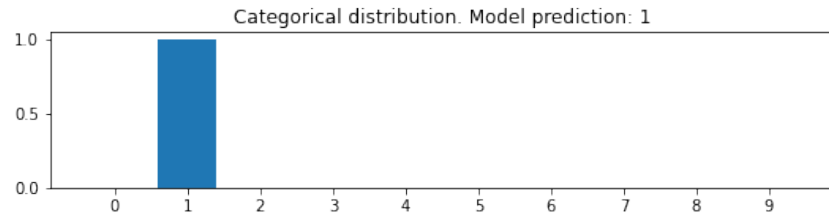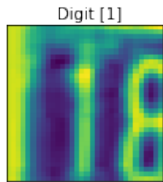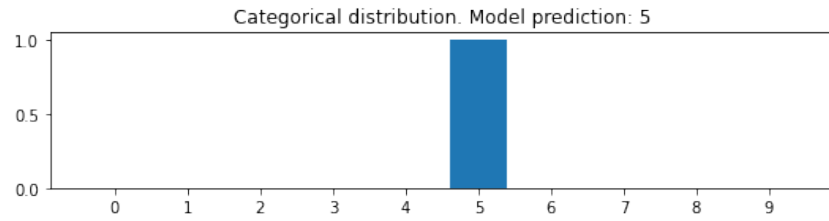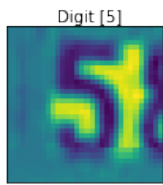
```
[[3.00557490e-09 5.42549697e-06 1.47472633e-06 2.00648938e-05
  9.06039350e-06 9.99883652e-01 3.95246498e-05 5.88889968e-08
  3.81590326e-05 2.53063786e-06]
 [2.19913616e-08 9.99997973e-01 1.51809374e-08 1.56325672e-08
  3.61447827e-09 7.09527714e-10 2.90430235e-10 1.99514830e-06
  4.17351975e-09 7.27744531e-10]
 [9.49301011e-07 5.95623078e-07 9.99758184e-01 1.02585494e-04
  2.46748385e-07 1.11473740e-10 1.20002253e-09 1.37316718e-04
  9.81143486e-08 1.25567432e-08]
 [4.57638787e-04 4.15605567e-02 2.42771506e-02 1.18775782e-03
  8.65850389e-01 2.83534901e-05 3.18366947e-05 6.59402460e-02
  6.50989357e-04 1.50606065e-05]
```

```
[1.90471619e-01 1.54100329e-01 7.36212591e-03 8.74573179e-03
 3.25431046e-03 5.58026880e-03 2.68594146e-01 3.45767349e-01
 1.45934811e-02 1.53064216e-03]]
```


[ ]:

[ ]: