Capstone_Project(2)

September 7, 2020

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.

[82]:
```
import tensorflow as tf
from scipy.io import loadmat
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,␣
↪Dropout, BatchNormalization
import random
```

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.

[5]:
```
# 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
=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redire
ct_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&scope=email%20https%3a%2f%2fwww.googl
eapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%2
0https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2
fwww.googleapis.com%2fauth%2fpeopleapi.readonly&response_type=code

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

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

train = loadmat('gdrive/My Drive/train_32x32.mat')
test = loadmat('gdrive/My Drive/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]:
```python
# Loading the dataset

x_train = train['X']
x_test = test['X']
y_train = train['y']
y_test = test['y']
```

[8]:
```python
# analysing the dimensions of the input
x_train.shape, x_test.shape
```

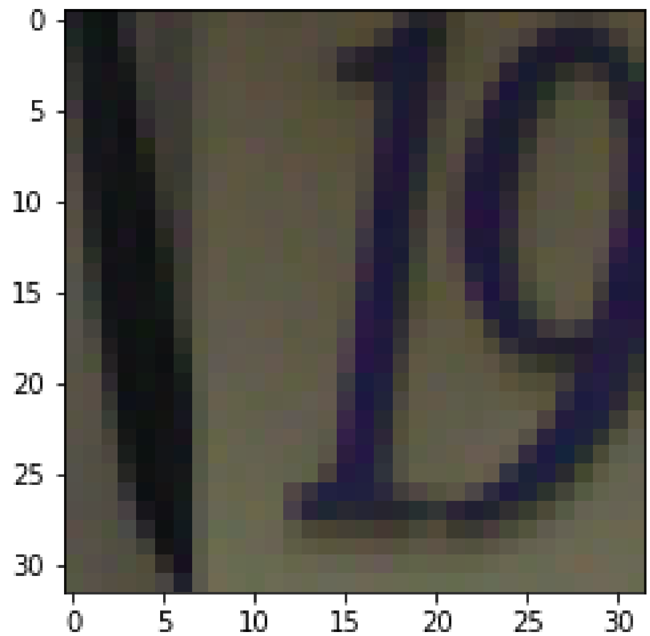
[8]: ((32, 32, 3, 73257), (32, 32, 3, 26032))

[9]:
```python
# changing dimensions from (a, b, c, num_examples) to (num_examples, a, b, c)

x_train = np.moveaxis(x_train, -1, 0)
x_test = np.moveaxis(x_test, -1, 0)
```

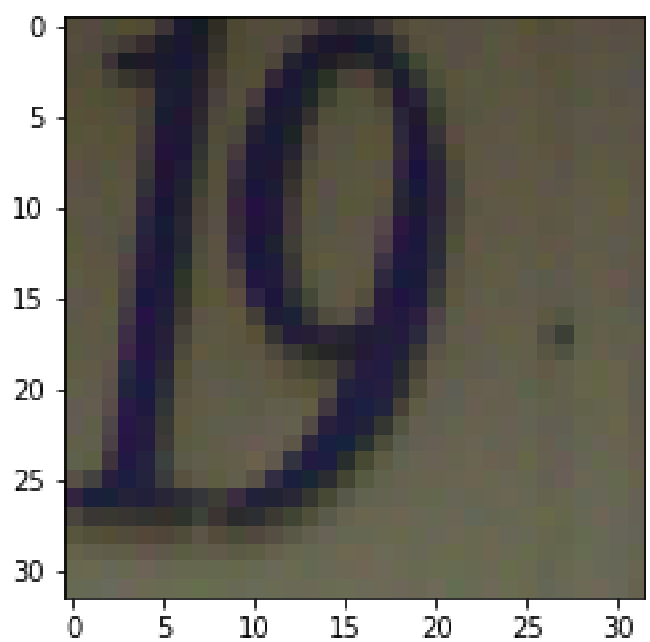
[10]:
```python
# checking if dimensions have actually changed

x_train.shape, x_test.shape
```

[10]: ((73257, 32, 32, 3), (26032, 32, 32, 3))

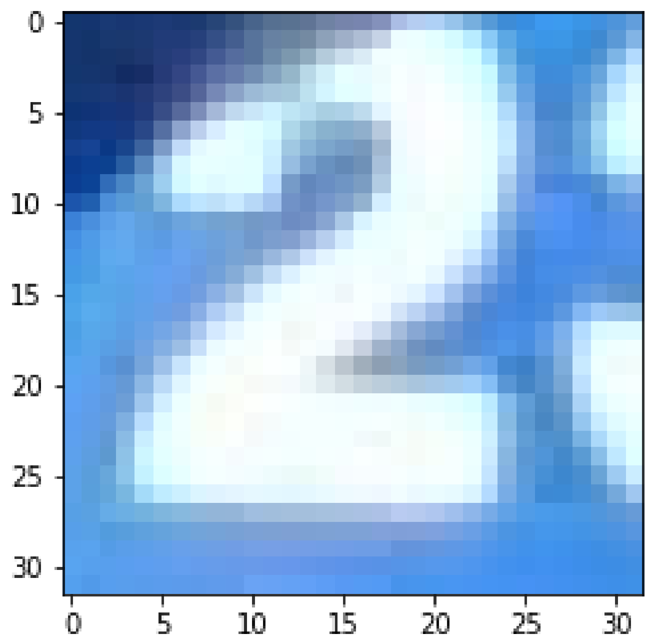
[11]:
```python
# plotting some sample images

for i in range(10):
    plt.imshow(x_train[i, :, :, :])
    plt.show()
    print(y_train[i])
```



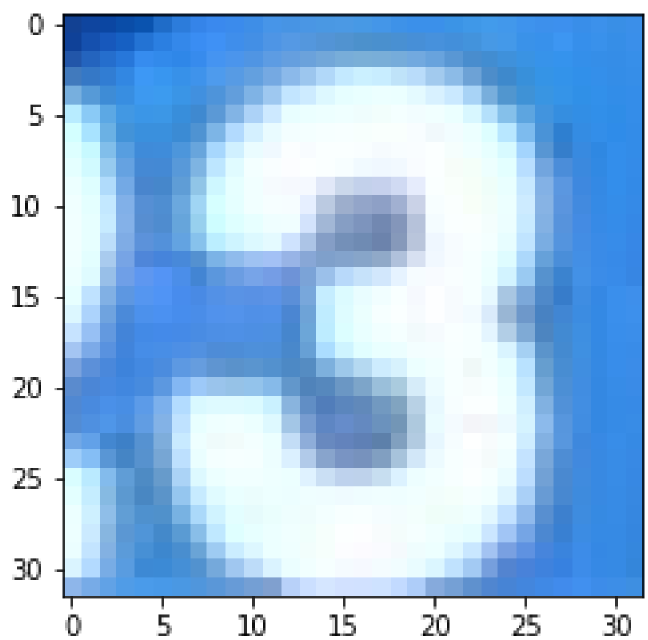
[1]



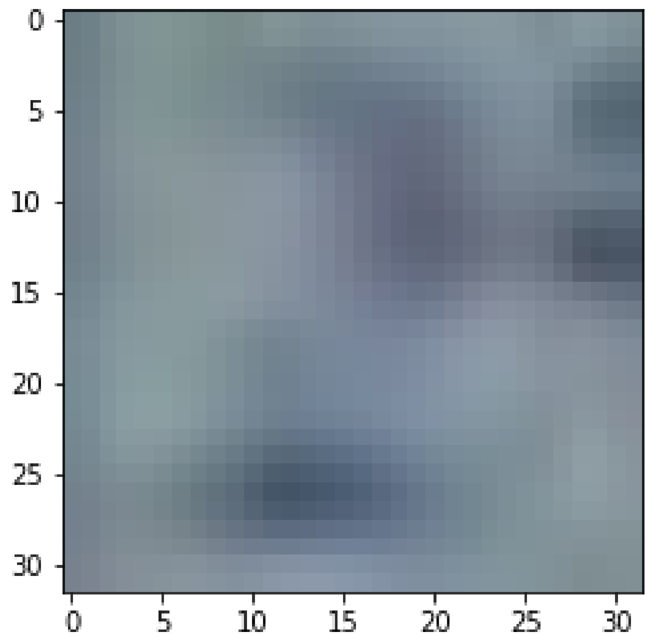
[9]



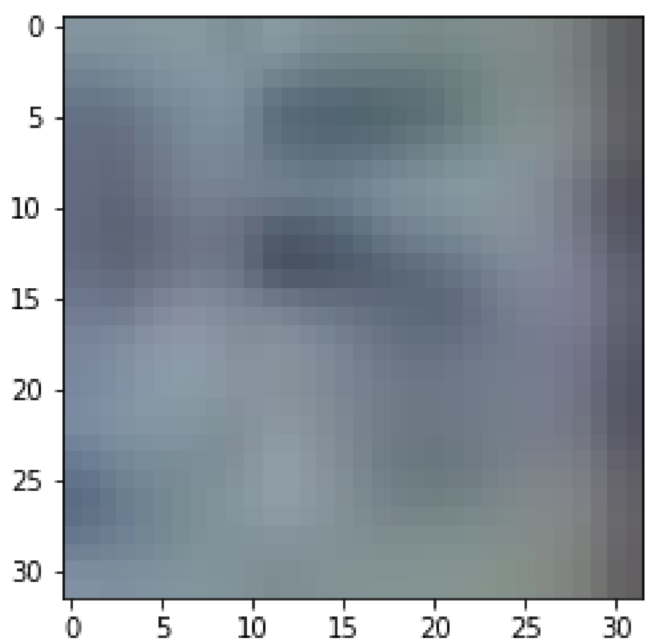
[2]



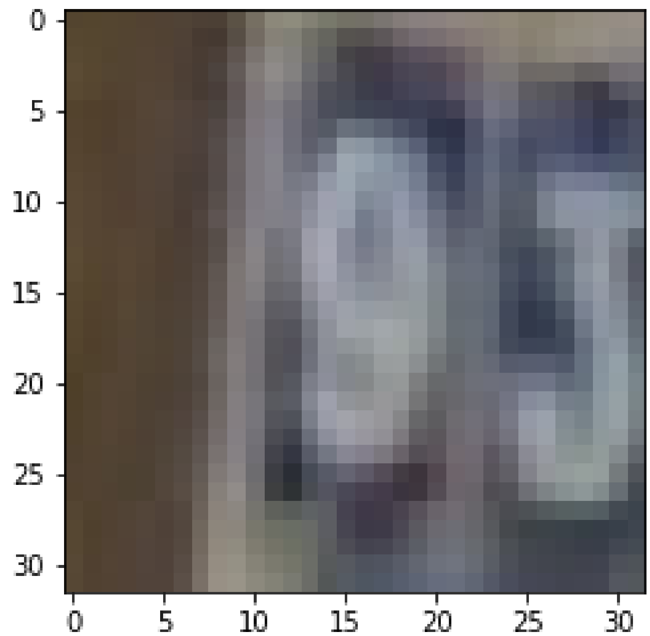
[3]



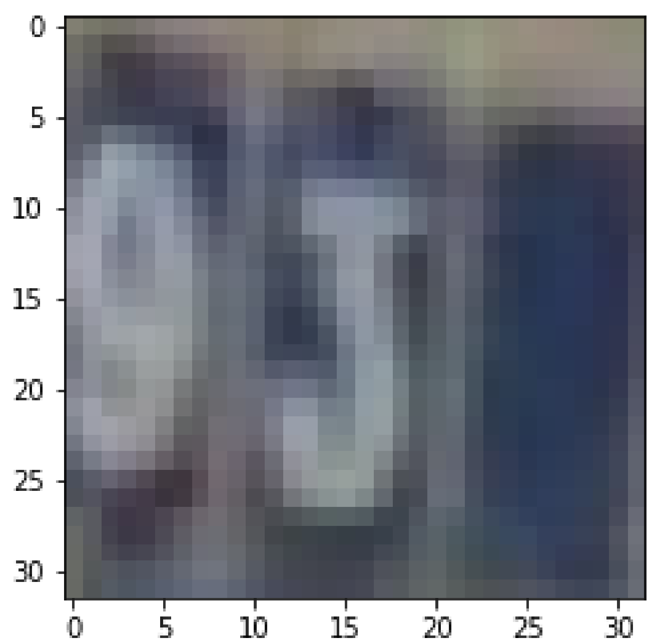
[2]



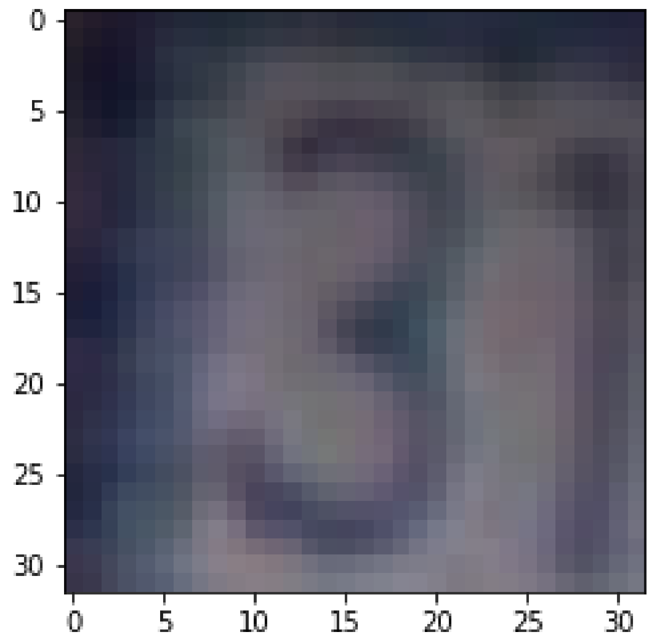
[5]



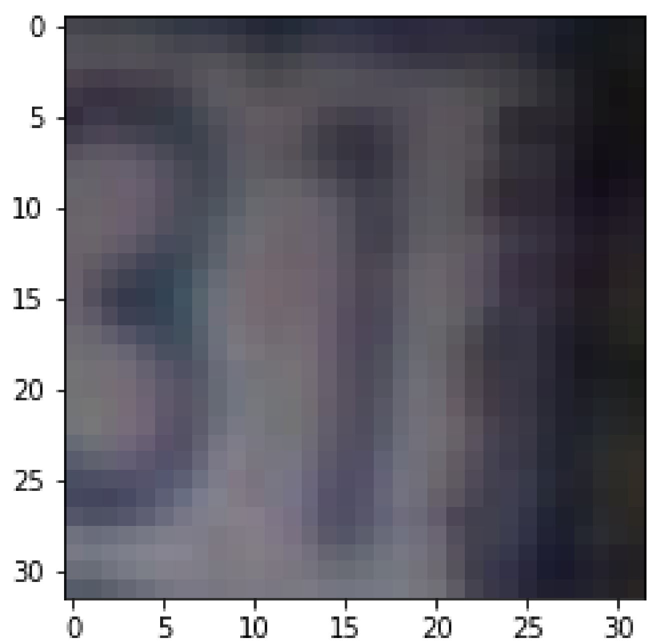
[9]



[3]



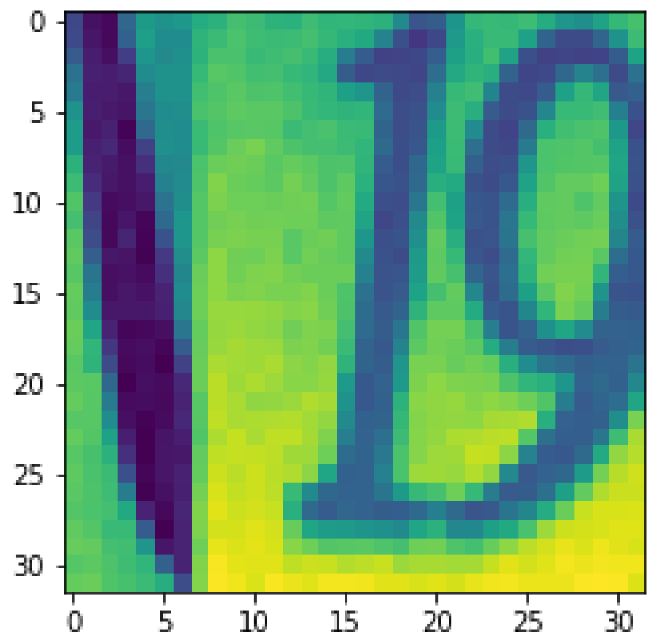
[3]



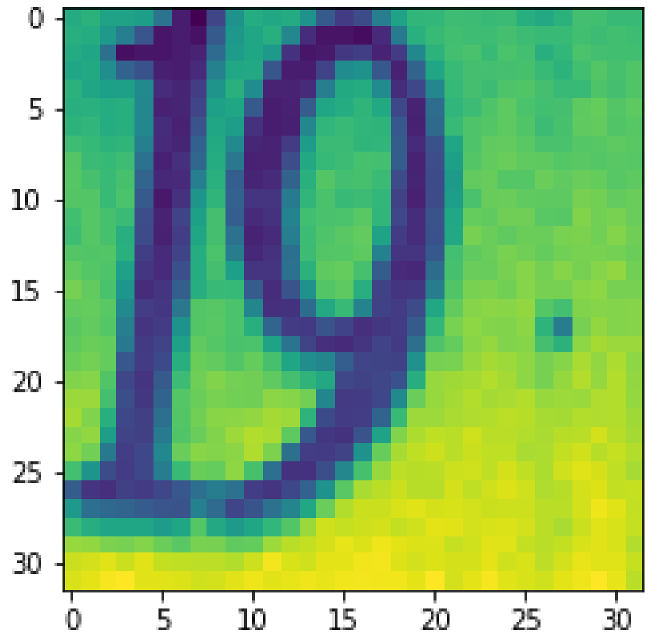
[1]

[12]:
```
# making changes to the images

x_train_gray = np.mean(x_train, 3).reshape(73257, 32, 32, 1) / 255.
x_test_gray = np.mean(x_test, 3).reshape(26032, 32, 32, 1) / 255.
x_train_plot = np.mean(x_train, 3)
```

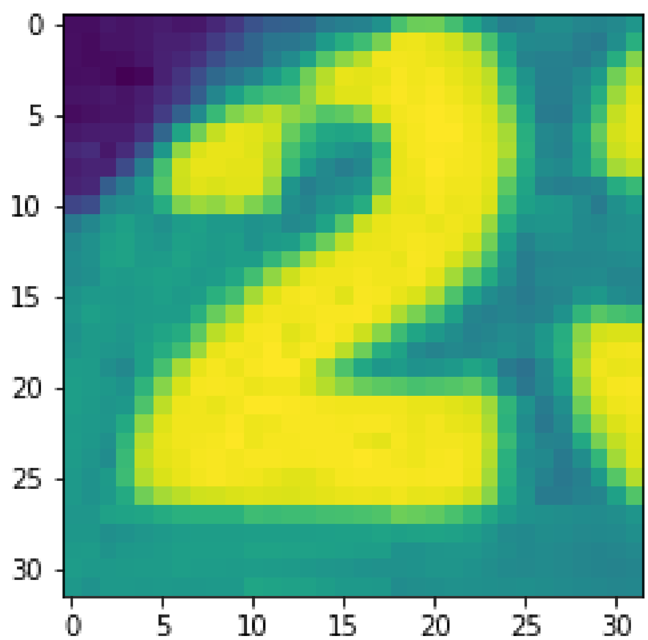
[13]:
```
# plotting the training images

for i in range(10):
    plt.imshow(x_train_plot[i, :, :,])
    plt.show()
    print(y_train[i])
```



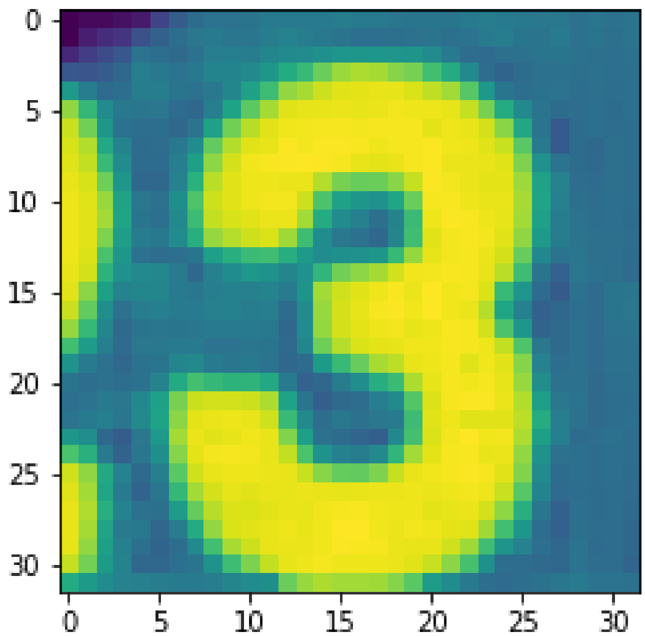
```
[1]
```



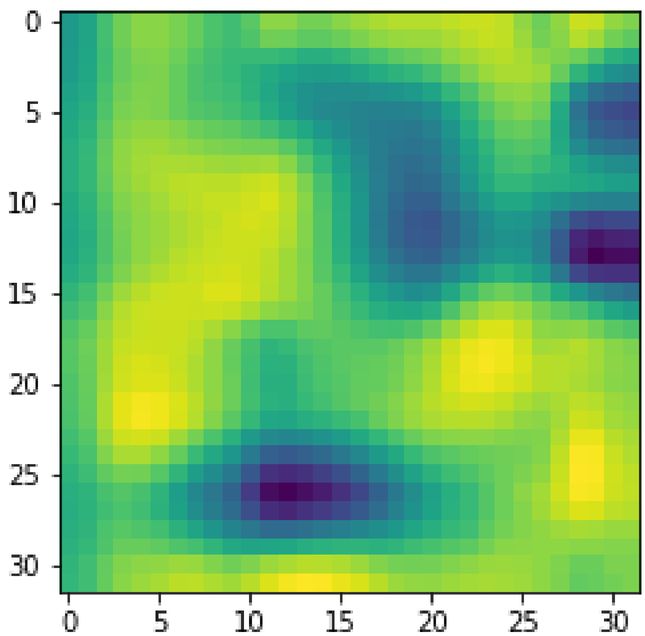
[9]



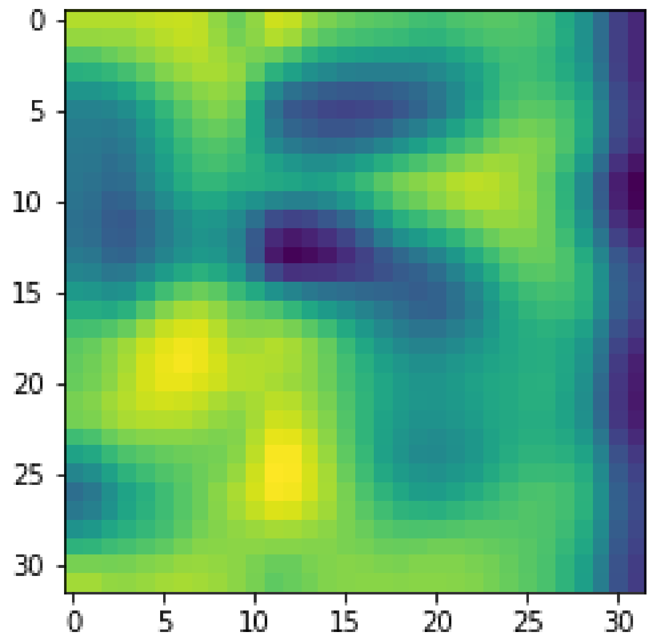
[2]



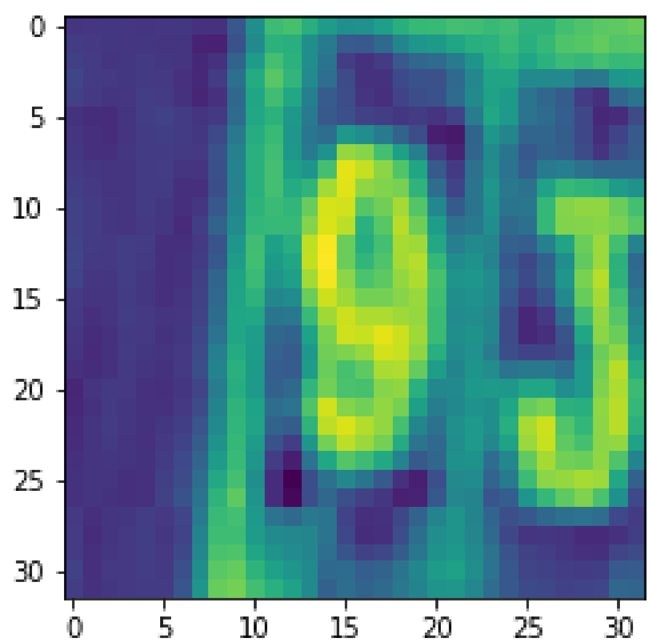
[3]



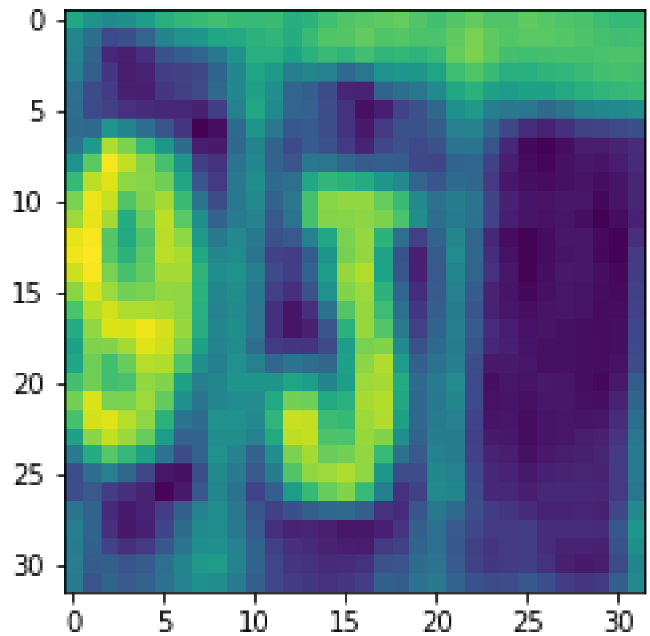
[2]



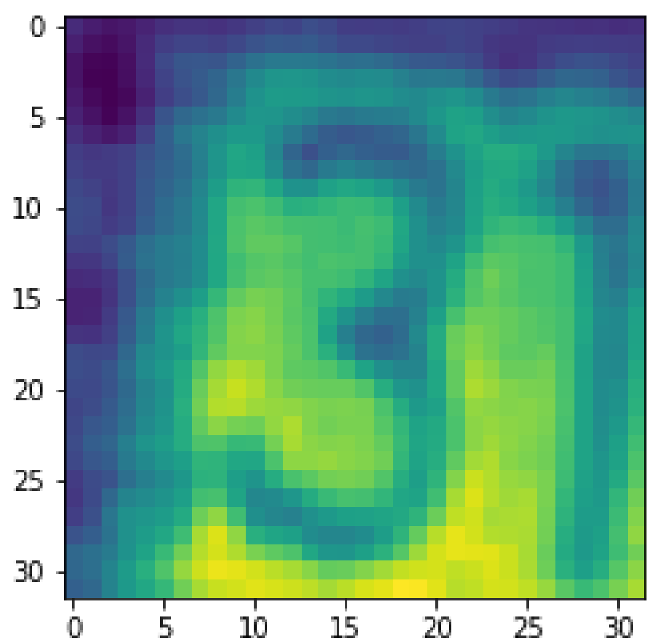
[5]



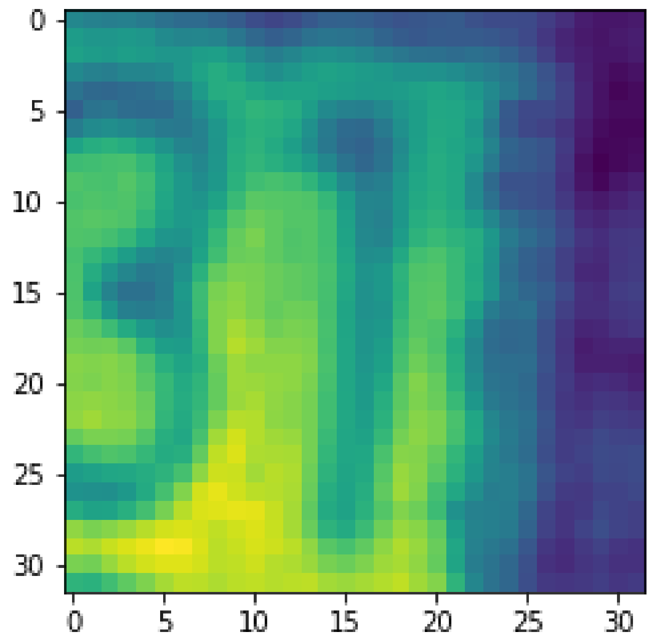
[9]



[3]



[3]



[1]

[62]:
```
# was having trouble in MLP NN classifier, so converting to one-hot labels

x_train[0].shape
```

[62]: (32, 32, 3)

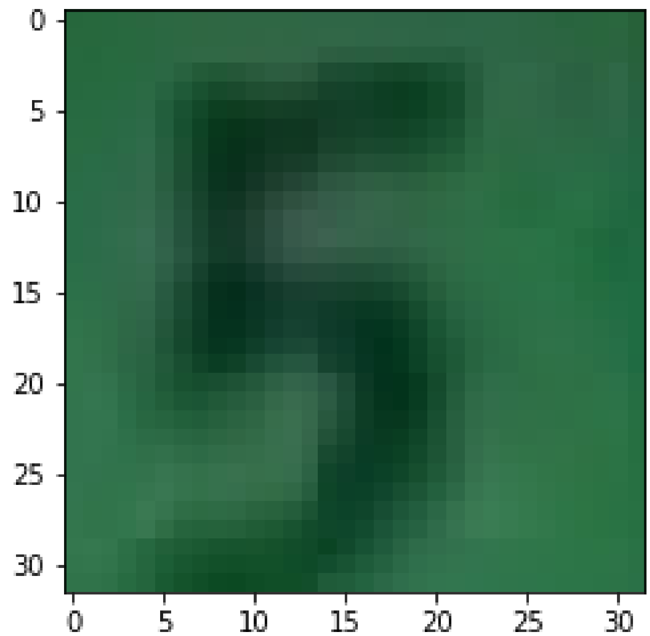
[63]:
```
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder().fit(y_train)
y_train_oh = enc.transform(y_train).toarray()
y_test_oh = enc.transform(y_test).toarray()
```

[64]:
```
y_test_oh[0]
```

[64]: array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0.])

[65]:
```
plt.imshow(x_test[0])
```

[65]: <matplotlib.image.AxesImage at 0x7fd52a5facf8>



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.

[18]:
```
model_seq = Sequential([
                        Flatten(input_shape=x_train[0].shape),
                        Dense(128, activation='relu'),
                        Dense(256, activation='relu'),
                        BatchNormalization(),
                        Dense(256, activation='relu'),
                        Dropout(0.5),
```

```
                        Dense(512, activation='relu'),
                        Dense(10, activation='softmax')

])

model_seq.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 3072)              0
_________________________________________________________________
dense (Dense)                (None, 128)               393344
_________________________________________________________________
dense_1 (Dense)              (None, 256)               33024
_________________________________________________________________
batch_normalization (BatchNo (None, 256)               1024
_________________________________________________________________
dense_2 (Dense)              (None, 256)               65792
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               131584
_________________________________________________________________
dense_4 (Dense)              (None, 10)                5130
=================================================================
Total params: 629,898
Trainable params: 629,386
Non-trainable params: 512
_________________________________________________________________
```

[19]:
```
model_seq.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

[20]:
```
checkpoint = ModelCheckpoint(filepath='sequential',
                             save_best_only=True,
                             save_weights_only=True,
                             monitor='val_loss',
                             mode='min',
                             verbose=1)

early_stop = EarlyStopping(patience=5, monitor='loss')
```

[21]:
```
history = model_seq.fit(x_train, y_train_oh, epochs=30,
                        validation_data=(x_test, y_test_oh),
```

```
                        batch_size=128,
                        callbacks=[checkpoint, early_stop])
```

```
Epoch 1/30
564/573 [============================>.] - ETA: 0s - loss: 1.7239 - accuracy:
0.3994
Epoch 00001: val_loss improved from inf to 2.01975, saving model to sequential
573/573 [==============================] - 3s 4ms/step - loss: 1.7188 -
accuracy: 0.4018 - val_loss: 2.0197 - val_accuracy: 0.3412
Epoch 2/30
559/573 [============================>.] - ETA: 0s - loss: 1.2283 - accuracy:
0.6102
Epoch 00002: val_loss did not improve from 2.01975
573/573 [==============================] - 2s 4ms/step - loss: 1.2277 -
accuracy: 0.6104 - val_loss: 2.3677 - val_accuracy: 0.3506
Epoch 3/30
563/573 [============================>.] - ETA: 0s - loss: 1.1261 - accuracy:
0.6475
Epoch 00003: val_loss improved from 2.01975 to 1.91625, saving model to
sequential
573/573 [==============================] - 2s 4ms/step - loss: 1.1250 -
accuracy: 0.6476 - val_loss: 1.9162 - val_accuracy: 0.4145
Epoch 4/30
566/573 [============================>.] - ETA: 0s - loss: 1.0364 - accuracy:
0.6775
Epoch 00004: val_loss improved from 1.91625 to 1.54374, saving model to
sequential
573/573 [==============================] - 2s 4ms/step - loss: 1.0353 -
accuracy: 0.6778 - val_loss: 1.5437 - val_accuracy: 0.5225
Epoch 5/30
571/573 [============================>.] - ETA: 0s - loss: 0.9973 - accuracy:
0.6885
Epoch 00005: val_loss did not improve from 1.54374
573/573 [==============================] - 2s 4ms/step - loss: 0.9970 -
accuracy: 0.6886 - val_loss: 1.8820 - val_accuracy: 0.4585
Epoch 6/30
572/573 [============================>.] - ETA: 0s - loss: 0.9263 - accuracy:
0.7130
Epoch 00006: val_loss improved from 1.54374 to 1.14499, saving model to
sequential
573/573 [==============================] - 2s 4ms/step - loss: 0.9263 -
accuracy: 0.7130 - val_loss: 1.1450 - val_accuracy: 0.6452
Epoch 7/30
566/573 [============================>.] - ETA: 0s - loss: 0.8867 - accuracy:
0.7239
Epoch 00007: val_loss improved from 1.14499 to 1.03599, saving model to
sequential
```

```
573/573 [==============================] - 2s 4ms/step - loss: 0.8859 -
accuracy: 0.7240 - val_loss: 1.0360 - val_accuracy: 0.6787
Epoch 8/30
556/573 [============================>.] - ETA: 0s - loss: 0.8566 - accuracy:
0.7325
Epoch 00008: val_loss improved from 1.03599 to 0.93794, saving model to
sequential
573/573 [==============================] - 2s 4ms/step - loss: 0.8560 -
accuracy: 0.7329 - val_loss: 0.9379 - val_accuracy: 0.7068
Epoch 9/30
567/573 [============================>.] - ETA: 0s - loss: 0.8503 - accuracy:
0.7349
Epoch 00009: val_loss did not improve from 0.93794
573/573 [==============================] - 2s 4ms/step - loss: 0.8508 -
accuracy: 0.7350 - val_loss: 1.1038 - val_accuracy: 0.6475
Epoch 10/30
573/573 [==============================] - ETA: 0s - loss: 0.8215 - accuracy:
0.7462
Epoch 00010: val_loss did not improve from 0.93794
573/573 [==============================] - 2s 4ms/step - loss: 0.8215 -
accuracy: 0.7462 - val_loss: 1.0235 - val_accuracy: 0.6829
Epoch 11/30
573/573 [==============================] - ETA: 0s - loss: 0.7973 - accuracy:
0.7530
Epoch 00011: val_loss did not improve from 0.93794
573/573 [==============================] - 2s 4ms/step - loss: 0.7973 -
accuracy: 0.7530 - val_loss: 1.1812 - val_accuracy: 0.6357
Epoch 12/30
568/573 [============================>.] - ETA: 0s - loss: 0.7838 - accuracy:
0.7578
Epoch 00012: val_loss did not improve from 0.93794
573/573 [==============================] - 2s 4ms/step - loss: 0.7839 -
accuracy: 0.7578 - val_loss: 1.0245 - val_accuracy: 0.6788
Epoch 13/30
568/573 [============================>.] - ETA: 0s - loss: 0.7779 - accuracy:
0.7590
Epoch 00013: val_loss did not improve from 0.93794
573/573 [==============================] - 2s 4ms/step - loss: 0.7782 -
accuracy: 0.7588 - val_loss: 1.1422 - val_accuracy: 0.6450
Epoch 14/30
563/573 [============================>.] - ETA: 0s - loss: 0.7591 - accuracy:
0.7627
Epoch 00014: val_loss did not improve from 0.93794
573/573 [==============================] - 2s 4ms/step - loss: 0.7595 -
accuracy: 0.7625 - val_loss: 0.9860 - val_accuracy: 0.6989
Epoch 15/30
557/573 [============================>.] - ETA: 0s - loss: 0.7573 - accuracy:
0.7651
```

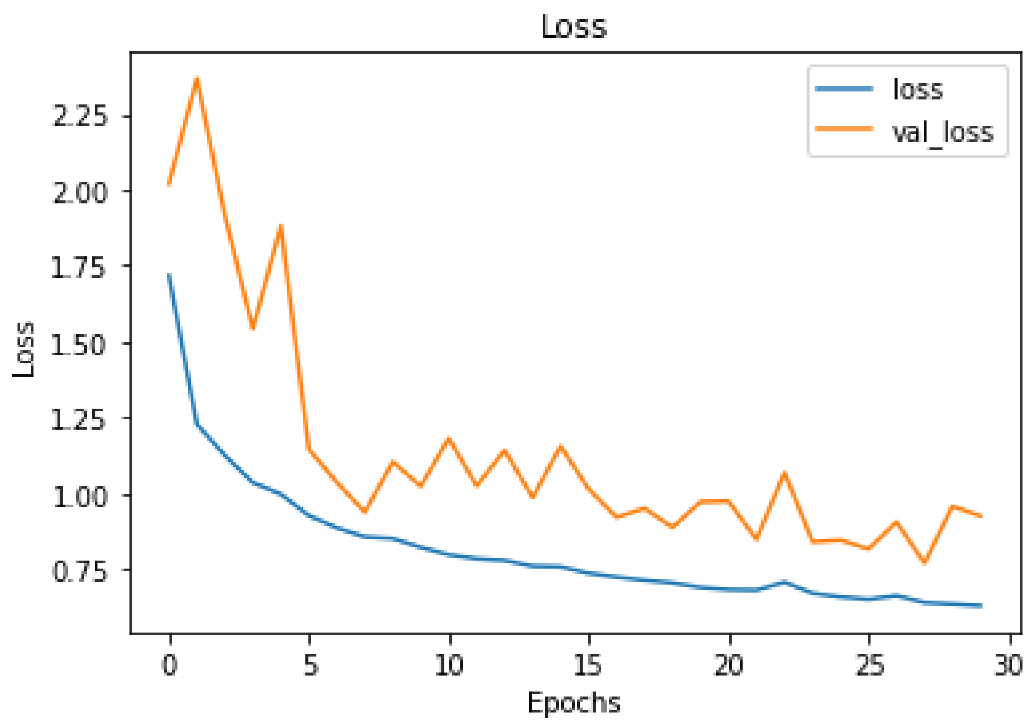
```
Epoch 00015: val_loss did not improve from 0.93794
573/573 [==============================] - 2s 4ms/step - loss: 0.7574 -
accuracy: 0.7652 - val_loss: 1.1555 - val_accuracy: 0.6411
Epoch 16/30
564/573 [============================>.] - ETA: 0s - loss: 0.7371 - accuracy:
0.7715
Epoch 00016: val_loss did not improve from 0.93794
573/573 [==============================] - 2s 4ms/step - loss: 0.7361 -
accuracy: 0.7719 - val_loss: 1.0144 - val_accuracy: 0.6901
Epoch 17/30
572/573 [============================>.] - ETA: 0s - loss: 0.7236 - accuracy:
0.7757
Epoch 00017: val_loss improved from 0.93794 to 0.92079, saving model to
sequential
573/573 [==============================] - 2s 4ms/step - loss: 0.7236 -
accuracy: 0.7757 - val_loss: 0.9208 - val_accuracy: 0.7214
Epoch 18/30
571/573 [============================>.] - ETA: 0s - loss: 0.7132 - accuracy:
0.7778
Epoch 00018: val_loss did not improve from 0.92079
573/573 [==============================] - 2s 4ms/step - loss: 0.7131 -
accuracy: 0.7779 - val_loss: 0.9502 - val_accuracy: 0.7121
Epoch 19/30
564/573 [============================>.] - ETA: 0s - loss: 0.7043 - accuracy:
0.7798
Epoch 00019: val_loss improved from 0.92079 to 0.88748, saving model to
sequential
573/573 [==============================] - 2s 4ms/step - loss: 0.7043 -
accuracy: 0.7798 - val_loss: 0.8875 - val_accuracy: 0.7436
Epoch 20/30
561/573 [============================>.] - ETA: 0s - loss: 0.6890 - accuracy:
0.7860
Epoch 00020: val_loss did not improve from 0.88748
573/573 [==============================] - 2s 4ms/step - loss: 0.6892 -
accuracy: 0.7859 - val_loss: 0.9715 - val_accuracy: 0.7039
Epoch 21/30
564/573 [============================>.] - ETA: 0s - loss: 0.6830 - accuracy:
0.7866
Epoch 00021: val_loss did not improve from 0.88748
573/573 [==============================] - 2s 4ms/step - loss: 0.6820 -
accuracy: 0.7868 - val_loss: 0.9741 - val_accuracy: 0.6929
Epoch 22/30
567/573 [============================>.] - ETA: 0s - loss: 0.6806 - accuracy:
0.7867
Epoch 00022: val_loss improved from 0.88748 to 0.84766, saving model to
sequential
573/573 [==============================] - 2s 4ms/step - loss: 0.6804 -
accuracy: 0.7868 - val_loss: 0.8477 - val_accuracy: 0.7499
```

```
Epoch 23/30
571/573 [============================>.] - ETA: 0s - loss: 0.7068 - accuracy:
0.7807
Epoch 00023: val_loss did not improve from 0.84766
573/573 [==============================] - 2s 4ms/step - loss: 0.7064 -
accuracy: 0.7808 - val_loss: 1.0679 - val_accuracy: 0.6523
Epoch 24/30
569/573 [============================>.] - ETA: 0s - loss: 0.6704 - accuracy:
0.7915
Epoch 00024: val_loss improved from 0.84766 to 0.83966, saving model to
sequential
573/573 [==============================] - 2s 4ms/step - loss: 0.6699 -
accuracy: 0.7916 - val_loss: 0.8397 - val_accuracy: 0.7399
Epoch 25/30
556/573 [============================>.] - ETA: 0s - loss: 0.6583 - accuracy:
0.7944
Epoch 00025: val_loss did not improve from 0.83966
573/573 [==============================] - 2s 4ms/step - loss: 0.6580 -
accuracy: 0.7946 - val_loss: 0.8457 - val_accuracy: 0.7429
Epoch 26/30
572/573 [============================>.] - ETA: 0s - loss: 0.6501 - accuracy:
0.7959
Epoch 00026: val_loss improved from 0.83966 to 0.81609, saving model to
sequential
573/573 [==============================] - 2s 4ms/step - loss: 0.6501 -
accuracy: 0.7959 - val_loss: 0.8161 - val_accuracy: 0.7528
Epoch 27/30
572/573 [============================>.] - ETA: 0s - loss: 0.6621 - accuracy:
0.7928
Epoch 00027: val_loss did not improve from 0.81609
573/573 [==============================] - 2s 4ms/step - loss: 0.6620 -
accuracy: 0.7928 - val_loss: 0.9051 - val_accuracy: 0.7197
Epoch 28/30
564/573 [============================>.] - ETA: 0s - loss: 0.6395 - accuracy:
0.7999
Epoch 00028: val_loss improved from 0.81609 to 0.76925, saving model to
sequential
573/573 [==============================] - 2s 4ms/step - loss: 0.6393 -
accuracy: 0.8000 - val_loss: 0.7692 - val_accuracy: 0.7642
Epoch 29/30
573/573 [==============================] - ETA: 0s - loss: 0.6346 - accuracy:
0.8019
Epoch 00029: val_loss did not improve from 0.76925
573/573 [==============================] - 2s 4ms/step - loss: 0.6346 -
accuracy: 0.8019 - val_loss: 0.9571 - val_accuracy: 0.7036
Epoch 30/30
565/573 [============================>.] - ETA: 0s - loss: 0.6291 - accuracy:
0.8034
```

```
Epoch 00030: val_loss did not improve from 0.76925
573/573 [==============================] - 2s 4ms/step - loss: 0.6291 -
accuracy: 0.8033 - val_loss: 0.9251 - val_accuracy: 0.7048
```

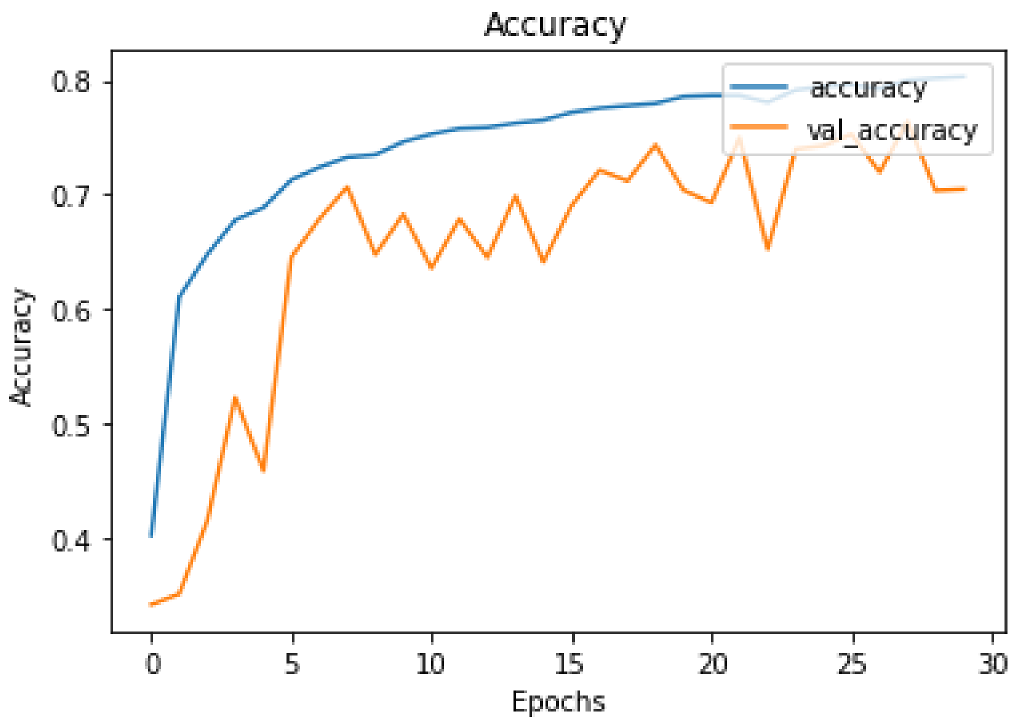
[24]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['loss', 'val_loss'], loc='upper right')
plt.title('Loss')
```

[24]:
```
Text(0.5, 1.0, 'Loss')
```



[25]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['accuracy', 'val_accuracy'], loc='upper right')
plt.title('Accuracy')
```

[25]:
```
Text(0.5, 1.0, 'Accuracy')
```



[26]:
```
model_seq.evaluate(x_test, y_test_oh, verbose=2)
```

```
814/814 - 1s - loss: 0.9251 - accuracy: 0.7048
```

[26]:
```
[0.9251210689544678, 0.7048248052597046]
```

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.

[73]:
```
model_cnn = Sequential([
                        Conv2D(16, (3, 3), padding='same', activation='relu',␣
 ↪input_shape=x_train[0].shape),
                        MaxPooling2D((3, 3,)),
                        Conv2D(32, (3, 3), padding='same', activation='relu'),
                        MaxPooling2D((3, 3,)),
                        BatchNormalization(),
                        Conv2D(64, (3, 3), padding='same', activation='relu'),
                        MaxPooling2D((3, 3,)),
                        Dropout(0.5),
                        Flatten(),
                        Dense(64, activation='relu'),
                        Dropout(0.5),
                        Dense(10, activation='softmax')
])

model_cnn.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_33 (Conv2D)           (None, 32, 32, 16)        448
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 10, 10, 16)        0
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 10, 10, 32)        4640
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 3, 3, 32)          0
_________________________________________________________________
batch_normalization_12 (Batc (None, 3, 3, 32)          128
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 3, 3, 64)          18496
_________________________________________________________________
max_pooling2d_35 (MaxPooling (None, 1, 1, 64)          0
_________________________________________________________________
dropout_17 (Dropout)         (None, 1, 1, 64)          0
_________________________________________________________________
flatten_12 (Flatten)         (None, 64)                0
_________________________________________________________________
dense_30 (Dense)             (None, 64)                4160
_________________________________________________________________
dropout_18 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_31 (Dense)             (None, 10)                650
=================================================================
Total params: 28,522
```

```
Trainable params: 28,458
Non-trainable params: 64
_________________________________________________________________
```

[74]:
```
model_cnn.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

[75]:
```
checkpoint_cnn = ModelCheckpoint(filepath='CNN', save_best_only=True,
                                 save_weights_only=True,
                                 save_freq=5000,
                                 monitor='val_acc',
                                 mode='max')
early_stop_cnn = EarlyStopping(monitor='loss', patience=7, verbose=1)
```

[76]:
```
history = model_cnn.fit(x_train,  y_train_oh,
                        callbacks=[checkpoint_cnn, early_stop_cnn],
                        batch_size=128, validation_data=(x_test, y_test_oh),
                        epochs=30)

-
```

```
Epoch 1/30
573/573 [==============================] - 3s 5ms/step - loss: 1.9691 -
accuracy: 0.3054 - val_loss: 1.2979 - val_accuracy: 0.5999
Epoch 2/30
573/573 [==============================] - 2s 4ms/step - loss: 1.1966 -
accuracy: 0.6077 - val_loss: 0.9446 - val_accuracy: 0.7121
Epoch 3/30
573/573 [==============================] - 3s 5ms/step - loss: 0.9402 -
accuracy: 0.7050 - val_loss: 0.7039 - val_accuracy: 0.7875
Epoch 4/30
573/573 [==============================] - 3s 5ms/step - loss: 0.8278 -
accuracy: 0.7458 - val_loss: 0.6627 - val_accuracy: 0.8019
Epoch 5/30
573/573 [==============================] - 3s 5ms/step - loss: 0.7609 -
accuracy: 0.7723 - val_loss: 0.6224 - val_accuracy: 0.8142
Epoch 6/30
573/573 [==============================] - 3s 5ms/step - loss: 0.7127 -
accuracy: 0.7867 - val_loss: 0.5515 - val_accuracy: 0.8375
Epoch 7/30
573/573 [==============================] - 2s 4ms/step - loss: 0.6852 -
accuracy: 0.7956 - val_loss: 0.5291 - val_accuracy: 0.8436
Epoch 8/30
573/573 [==============================] - 2s 4ms/step - loss: 0.6629 -
accuracy: 0.8041 - val_loss: 0.6309 - val_accuracy: 0.8085
Epoch 9/30
403/573 [===================>…] - ETA: 0s - loss: 0.6416 - accuracy:
0.8102WARNING:tensorflow:Can save best model only with val_acc available,
```

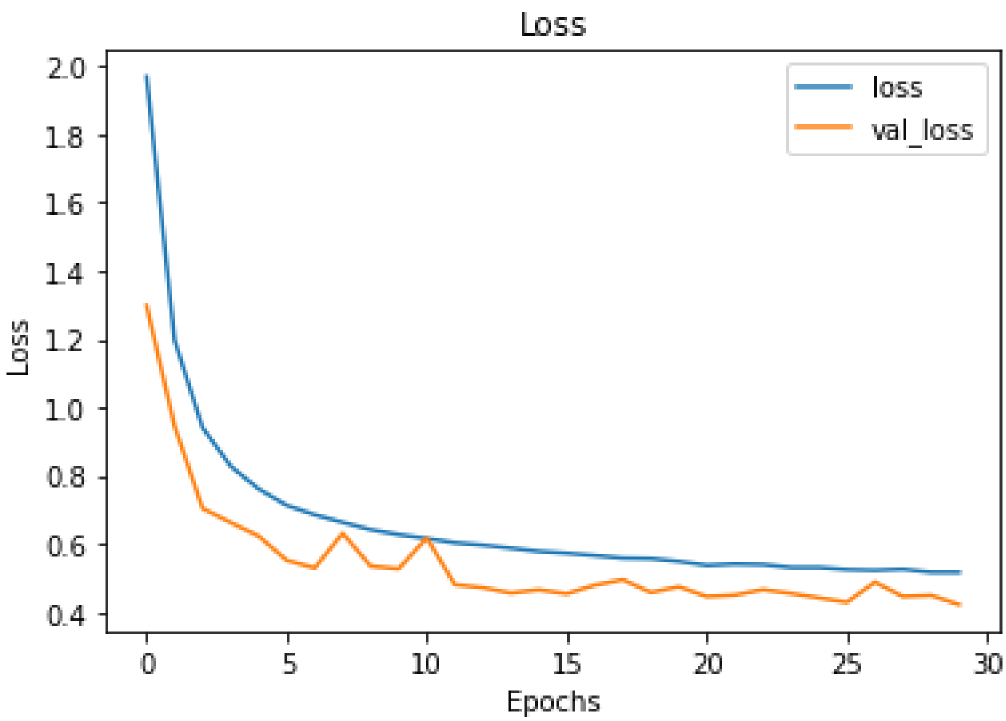
```
skipping.
573/573 [==============================] - 2s 4ms/step - loss: 0.6414 -
accuracy: 0.8103 - val_loss: 0.5345 - val_accuracy: 0.8418
Epoch 10/30
573/573 [==============================] - 2s 4ms/step - loss: 0.6269 -
accuracy: 0.8154 - val_loss: 0.5266 - val_accuracy: 0.8442
Epoch 11/30
573/573 [==============================] - 2s 4ms/step - loss: 0.6155 -
accuracy: 0.8200 - val_loss: 0.6170 - val_accuracy: 0.8127
Epoch 12/30
573/573 [==============================] - 2s 4ms/step - loss: 0.6030 -
accuracy: 0.8223 - val_loss: 0.4806 - val_accuracy: 0.8590
Epoch 13/30
573/573 [==============================] - 2s 4ms/step - loss: 0.5960 -
accuracy: 0.8255 - val_loss: 0.4718 - val_accuracy: 0.8628
Epoch 14/30
573/573 [==============================] - 2s 4ms/step - loss: 0.5871 -
accuracy: 0.8270 - val_loss: 0.4560 - val_accuracy: 0.8651
Epoch 15/30
573/573 [==============================] - 2s 4ms/step - loss: 0.5777 -
accuracy: 0.8311 - val_loss: 0.4655 - val_accuracy: 0.8646
Epoch 16/30
573/573 [==============================] - 2s 4ms/step - loss: 0.5722 -
accuracy: 0.8321 - val_loss: 0.4537 - val_accuracy: 0.8672
Epoch 17/30
573/573 [==============================] - 2s 4ms/step - loss: 0.5654 -
accuracy: 0.8350 - val_loss: 0.4792 - val_accuracy: 0.8594
Epoch 18/30
249/573 [============>…] - ETA: 1s - loss: 0.5435 - accuracy:
0.8398WARNING:tensorflow:Can save best model only with val_acc available,
skipping.
573/573 [==============================] - 2s 4ms/step - loss: 0.5580 -
accuracy: 0.8372 - val_loss: 0.4949 - val_accuracy: 0.8526
Epoch 19/30
573/573 [==============================] - 2s 4ms/step - loss: 0.5567 -
accuracy: 0.8385 - val_loss: 0.4576 - val_accuracy: 0.8679
Epoch 20/30
573/573 [==============================] - 2s 4ms/step - loss: 0.5483 -
accuracy: 0.8399 - val_loss: 0.4743 - val_accuracy: 0.8601
Epoch 21/30
573/573 [==============================] - 2s 4ms/step - loss: 0.5369 -
accuracy: 0.8440 - val_loss: 0.4456 - val_accuracy: 0.8690
Epoch 22/30
573/573 [==============================] - 2s 4ms/step - loss: 0.5407 -
accuracy: 0.8429 - val_loss: 0.4500 - val_accuracy: 0.8705
Epoch 23/30
573/573 [==============================] - 2s 4ms/step - loss: 0.5390 -
accuracy: 0.8436 - val_loss: 0.4659 - val_accuracy: 0.8636
```

```
Epoch 24/30
573/573 [==============================] - 2s 4ms/step - loss: 0.5305 -
accuracy: 0.8463 - val_loss: 0.4548 - val_accuracy: 0.8666
Epoch 25/30
573/573 [==============================] - 2s 4ms/step - loss: 0.5304 -
accuracy: 0.8452 - val_loss: 0.4417 - val_accuracy: 0.8705
Epoch 26/30
573/573 [==============================] - 2s 4ms/step - loss: 0.5243 -
accuracy: 0.8473 - val_loss: 0.4289 - val_accuracy: 0.8767
Epoch 27/30
 94/573 [===>…] - ETA: 1s - loss: 0.5066 - accuracy:
0.8492WARNING:tensorflow:Can save best model only with val_acc available,
skipping.
573/573 [==============================] - 2s 4ms/step - loss: 0.5223 -
accuracy: 0.8465 - val_loss: 0.4881 - val_accuracy: 0.8548
Epoch 28/30
573/573 [==============================] - 2s 4ms/step - loss: 0.5253 -
accuracy: 0.8470 - val_loss: 0.4457 - val_accuracy: 0.8685
Epoch 29/30
573/573 [==============================] - 2s 4ms/step - loss: 0.5166 -
accuracy: 0.8496 - val_loss: 0.4490 - val_accuracy: 0.8693
Epoch 30/30
573/573 [==============================] - 2s 4ms/step - loss: 0.5164 -
accuracy: 0.8488 - val_loss: 0.4219 - val_accuracy: 0.8770
```

[76]: (73257, 10)

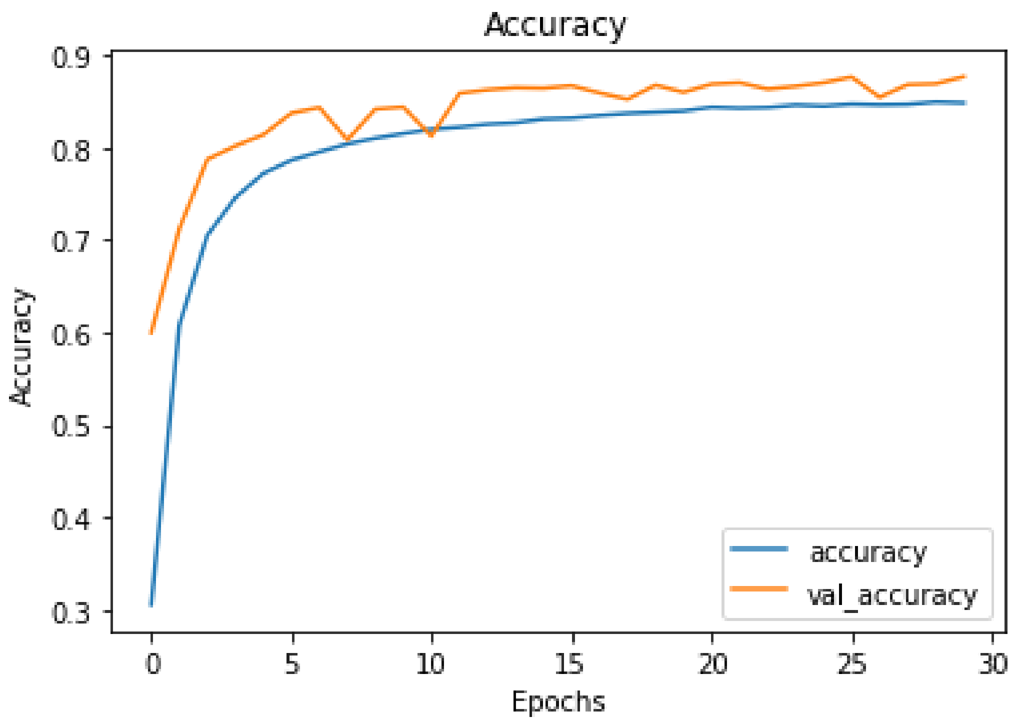
[78]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['loss', 'val_loss'], loc='upper right')
plt.title('Loss')
```

[78]: Text(0.5, 1.0, 'Loss')



[80]:
```
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')
```

[80]:
```
Text(0.5, 1.0, 'Accuracy')
```



[87]:
```
model_cnn.evaluate(x_test, y_test_oh, verbose=2)
```

```
814/814 - 2s - loss: 0.4219 - accuracy: 0.8770
```

[87]:
```
[0.4219440221786499, 0.8770359754562378]
```

1.5 4. Get model predictions

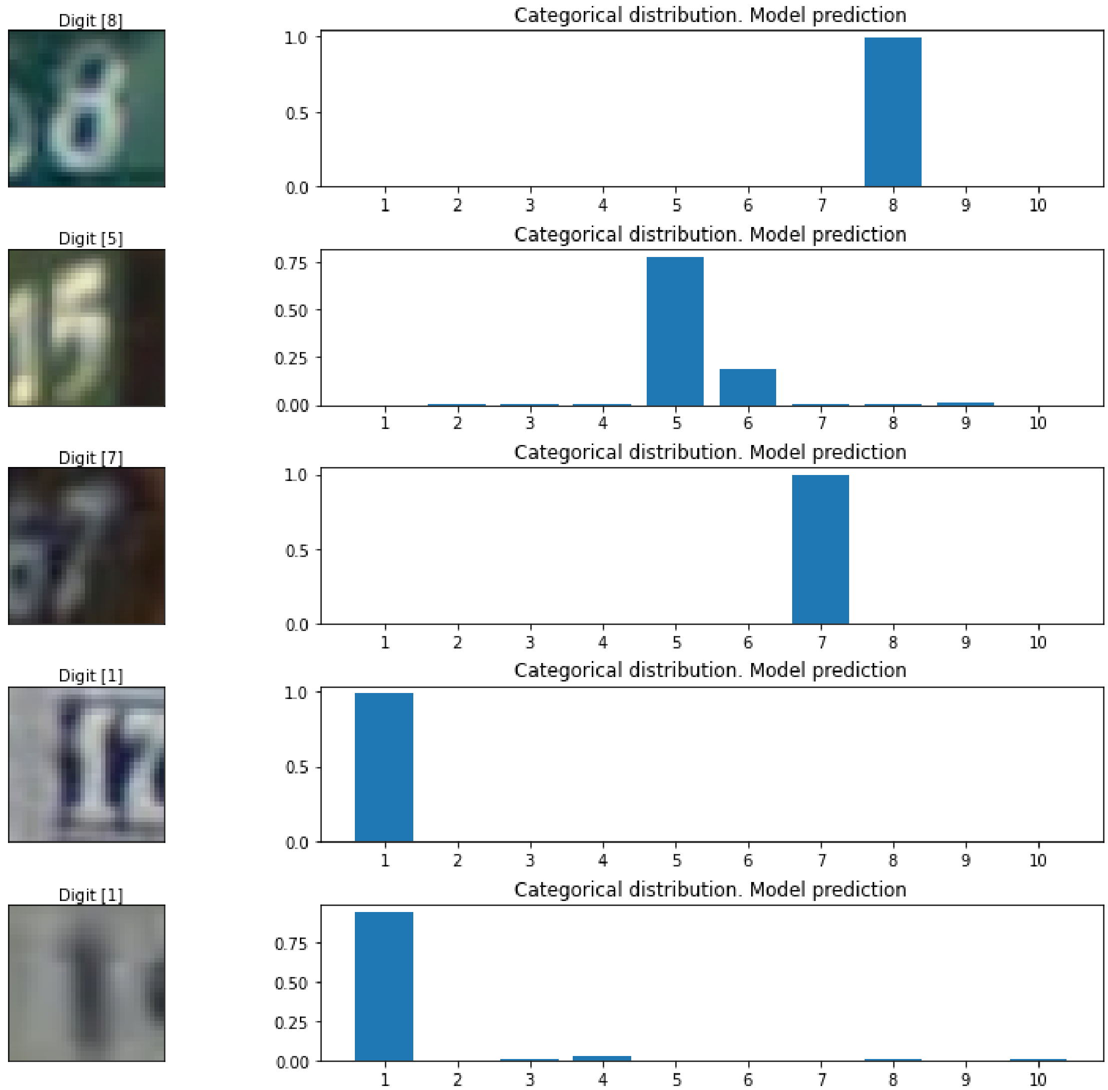
- Load the best weights for the MLP and CNN models that you saved during the training run.
- Randomly select 5 images and corresponding labels from the test set and display the images with their labels.
- Alongside the image and label, show each model's predictive distribution as a bar chart, and the final model prediction given by the label with maximum probability.

[81]:
```
model_seq.load_weights('sequential')
```

[81]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at
0x7fd5dfe8f208>
```

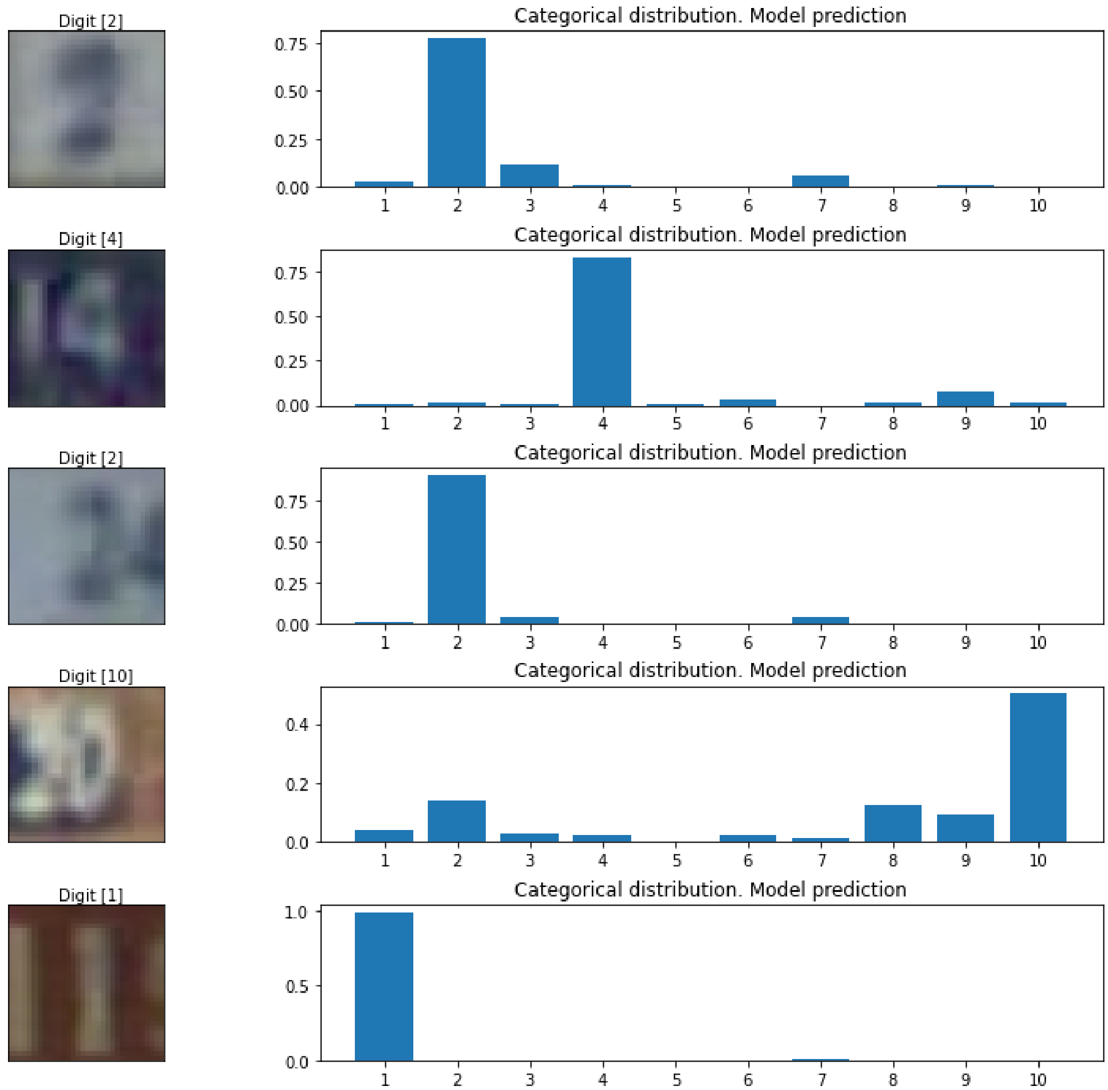
[85]:
```
num_test_images = x_test.shape[0]

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_seq.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(1,11), prediction)
    axes[i, 1].set_xticks(np.arange(1,11))
    axes[i, 1].set_title("Categorical distribution. Model prediction")

plt.show()
```



[86]:
```
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(1,11), prediction)
    axes[i, 1].set_xticks(np.arange(1,11))
    axes[i, 1].set_title("Categorical distribution. Model prediction")

plt.show()
```



[ ]:

[ ]:

```
[ ]:
```