Capstone Project

October 13, 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 (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
        from scipy.io import loadmat
```

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

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

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

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

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

```
In [3]: train.keys()
```

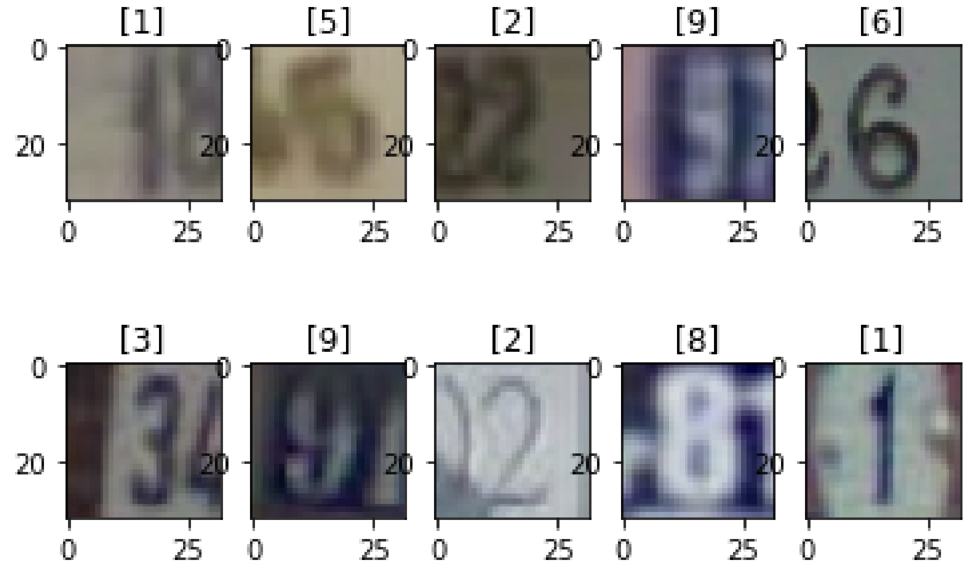
```
Out[3]: dict_keys(['__header__', '__version__', '__globals__', 'X', 'y'])
```

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 [4]: x_train = train['X'] / 255
        y_train = train['y']
        x_test = test['X'] / 255
        y_test = test['y']

In [5]: 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)


In [6]: import numpy as np

        x_train = np.transpose(x_train, (3,0,1,2))
        x_test = np.transpose(x_test, (3,0,1,2))

In [9]: # Converting number 10 to 0
        y_train[y_train == 10] = 0
        y_test[y_test == 10] = 0

        print(np.unique(y_train))

[0 1 2 3 4 5 6 7 8 9]


In [18]: import matplotlib.pyplot as plt
         import random
         plt.figure(figsize=(5,5))

         %matplotlib inline


         num_test_images = x_train.shape[0]

         random_inx = np.random.choice(num_test_images, 10)
         random_images = x_train[random_inx, ...]
         random_labels = y_train[random_inx, ...]

         for i in range(10):
             plt.subplot(2, 5, i + 1)
             plt.imshow(random_images[i])
             plt.title(random_labels[i])
```



In [12]:
```
def rgb2gray(images):
    return np.expand_dims(np.dot(images, [0.2990, 0.5870, 0.1140]), axis=3)
```

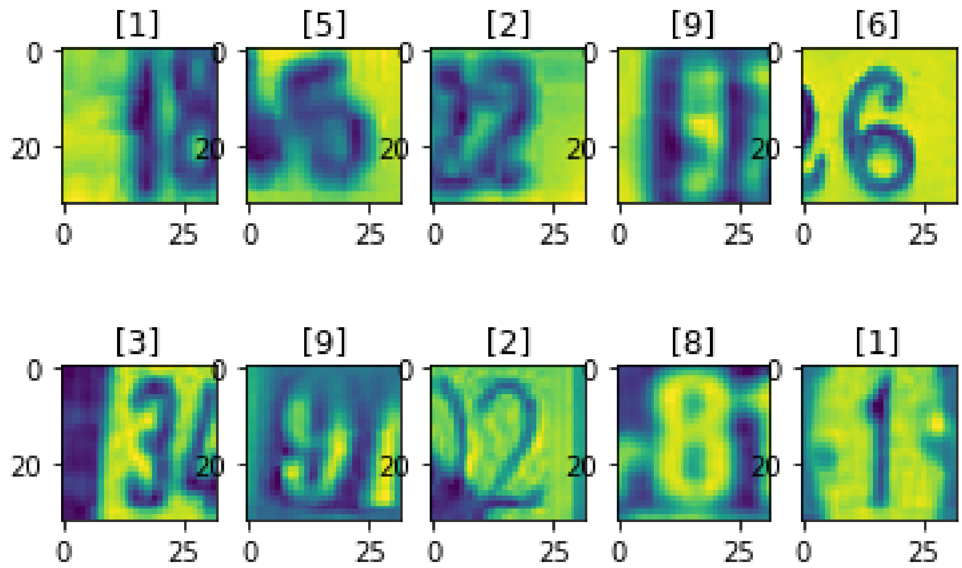
In [20]:
```
train_gs = rgb2gray(x_train).astype(np.float32)
test_gs = rgb2gray(x_test).astype(np.float32)
train_gs.shape
```

Out[20]: (73257, 32, 32, 1)

In [23]:
```
random_images = train_gs[random_inx, ...]
random_images.shape
```

Out[23]: (10, 32, 32, 1)

In [30]:
```
random_images = train_gs[random_inx, ...]
random_labels = y_train[random_inx, ...]

for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(random_images[i,:,:,0])
    plt.title(random_labels[i])
```




```
In [31]: 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()

In [32]: y_test_oh[0]

Out[32]: array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.])

In [78]: y_test

Out[78]: array([[5],
                [2],
                [1],
                ...,
                [7],
                [6],
                [7]], dtype=uint8)
```

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 [60]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Softmax, Conv2D, MaxPooling2D, Ba
```

In [61]:
```
model = Sequential([
    Flatten(input_shape=train_gs[0].shape),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(16, activation='relu'),
    Dense(10, activation='softmax')
])
```

In [62]:
```
model.summary()
```

```
Model: "sequential_2"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 1024)              0

_________________________________________________________________
dense_7 (Dense)              (None, 128)               131200

_________________________________________________________________
dense_8 (Dense)              (None, 64)                8256

_________________________________________________________________
dense_9 (Dense)              (None, 16)                1040

_________________________________________________________________
dense_10 (Dense)             (None, 10)                170
=================================================================
Total params: 140,666
Trainable params: 140,666
Non-trainable params: 0

_________________________________________________________________
```

In [63]:
```
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

checkpoint = ModelCheckpoint(filepath = 'mlp_checkpoints', save_best_only=True, save_w
earlystop = EarlyStopping(patience=5, monitor='loss')
```

In [64]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
```

In [65]:
```
history = model.fit(train_gs, y_train_oh, callbacks=[checkpoint, earlystop], batch_siz
```

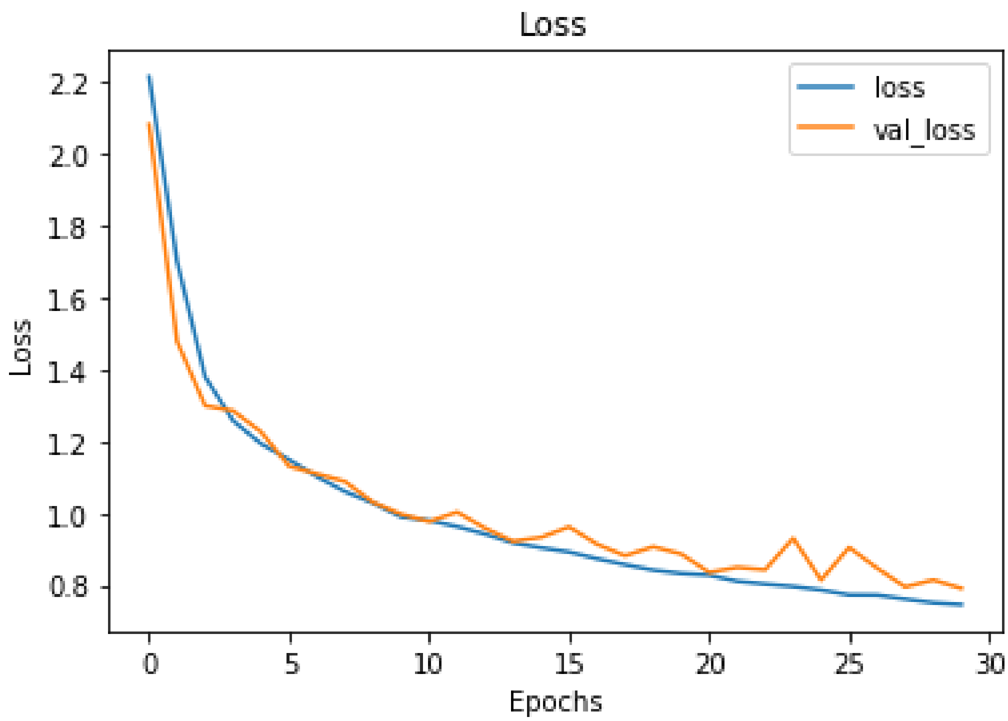
```
Train on 62268 samples, validate on 10989 samples
Epoch 1/30
61824/62268 [============================>.] - ETA: 0s - loss: 2.2164 - acc: 0.1930
Epoch 00001: val_loss improved from inf to 2.08266, saving model to mlp_checkpoints
62268/62268 [==============================] - 13s 206us/sample - loss: 2.2158 - acc: 0.1933 -
Epoch 2/30
61824/62268 [============================>.] - ETA: 0s - loss: 1.7028 - acc: 0.4056
Epoch 00002: val_loss improved from 2.08266 to 1.47808, saving model to mlp_checkpoints
62268/62268 [==============================] - 12s 191us/sample - loss: 1.7013 - acc: 0.4064 -
Epoch 3/30
62208/62268 [============================>.] - ETA: 0s - loss: 1.3802 - acc: 0.5408
Epoch 00003: val_loss improved from 1.47808 to 1.29950, saving model to mlp_checkpoints
62268/62268 [==============================] - 12s 191us/sample - loss: 1.3802 - acc: 0.5408 -
Epoch 4/30
62208/62268 [============================>.] - ETA: 0s - loss: 1.2575 - acc: 0.5977
Epoch 00004: val_loss improved from 1.29950 to 1.28550, saving model to mlp_checkpoints
62268/62268 [==============================] - 12s 189us/sample - loss: 1.2576 - acc: 0.5975 -
Epoch 5/30
61696/62268 [============================>.] - ETA: 0s - loss: 1.1937 - acc: 0.6215
Epoch 00005: val_loss improved from 1.28550 to 1.22646, saving model to mlp_checkpoints
62268/62268 [==============================] - 12s 188us/sample - loss: 1.1932 - acc: 0.6218 -
Epoch 6/30
61696/62268 [============================>.] - ETA: 0s - loss: 1.1495 - acc: 0.6407
Epoch 00006: val_loss improved from 1.22646 to 1.13084, saving model to mlp_checkpoints
62268/62268 [==============================] - 12s 189us/sample - loss: 1.1489 - acc: 0.6411 -
Epoch 7/30
62080/62268 [============================>.] - ETA: 0s - loss: 1.1021 - acc: 0.6587
Epoch 00007: val_loss improved from 1.13084 to 1.10991, saving model to mlp_checkpoints
62268/62268 [==============================] - 12s 190us/sample - loss: 1.1020 - acc: 0.6587 -
Epoch 8/30
62080/62268 [============================>.] - ETA: 0s - loss: 1.0602 - acc: 0.6730
Epoch 00008: val_loss improved from 1.10991 to 1.08857, saving model to mlp_checkpoints
62268/62268 [==============================] - 12s 189us/sample - loss: 1.0603 - acc: 0.6730 -
Epoch 9/30
61952/62268 [============================>.] - ETA: 0s - loss: 1.0288 - acc: 0.6835
Epoch 00009: val_loss improved from 1.08857 to 1.03047, saving model to mlp_checkpoints
62268/62268 [==============================] - 12s 189us/sample - loss: 1.0283 - acc: 0.6837 -
Epoch 10/30
61952/62268 [============================>.] - ETA: 0s - loss: 0.9905 - acc: 0.6972
Epoch 00010: val_loss improved from 1.03047 to 0.99725, saving model to mlp_checkpoints
62268/62268 [==============================] - 12s 190us/sample - loss: 0.9897 - acc: 0.6975 -
Epoch 11/30
62080/62268 [============================>.] - ETA: 0s - loss: 0.9800 - acc: 0.7001
Epoch 00011: val_loss improved from 0.99725 to 0.97685, saving model to mlp_checkpoints
62268/62268 [==============================] - 12s 191us/sample - loss: 0.9799 - acc: 0.7002 -
Epoch 12/30
62080/62268 [============================>.] - ETA: 0s - loss: 0.9624 - acc: 0.7059
Epoch 00012: val_loss did not improve from 0.97685
```

```
62268/62268 [==============================] - 12s 189us/sample - loss: 0.9627 - acc: 0.7058 -
Epoch 13/30
61824/62268 [============================>.] - ETA: 0s - loss: 0.9420 - acc: 0.7134
Epoch 00013: val_loss improved from 0.97685 to 0.95821, saving model to mlp_checkpoints
62268/62268 [==============================] - 12s 190us/sample - loss: 0.9419 - acc: 0.7135 -
Epoch 14/30
62080/62268 [============================>.] - ETA: 0s - loss: 0.9170 - acc: 0.7201
Epoch 00014: val_loss improved from 0.95821 to 0.92171, saving model to mlp_checkpoints
62268/62268 [==============================] - 12s 189us/sample - loss: 0.9173 - acc: 0.7200 -
Epoch 15/30
61824/62268 [============================>.] - ETA: 0s - loss: 0.9040 - acc: 0.7261
Epoch 00015: val_loss did not improve from 0.92171
62268/62268 [==============================] - 12s 187us/sample - loss: 0.9037 - acc: 0.7262 -
Epoch 16/30
61952/62268 [============================>.] - ETA: 0s - loss: 0.8916 - acc: 0.7286
Epoch 00016: val_loss did not improve from 0.92171
62268/62268 [==============================] - 12s 189us/sample - loss: 0.8918 - acc: 0.7285 -
Epoch 17/30
61824/62268 [============================>.] - ETA: 0s - loss: 0.8741 - acc: 0.7332
Epoch 00017: val_loss improved from 0.92171 to 0.91353, saving model to mlp_checkpoints
62268/62268 [==============================] - 12s 190us/sample - loss: 0.8730 - acc: 0.7336 -
Epoch 18/30
61952/62268 [============================>.] - ETA: 0s - loss: 0.8567 - acc: 0.7380
Epoch 00018: val_loss improved from 0.91353 to 0.88141, saving model to mlp_checkpoints
62268/62268 [==============================] - 12s 189us/sample - loss: 0.8566 - acc: 0.7381 -
Epoch 19/30
61824/62268 [============================>.] - ETA: 0s - loss: 0.8410 - acc: 0.7450
Epoch 00019: val_loss did not improve from 0.88141
62268/62268 [==============================] - 12s 187us/sample - loss: 0.8412 - acc: 0.7449 -
Epoch 20/30
62080/62268 [============================>.] - ETA: 0s - loss: 0.8322 - acc: 0.7468
Epoch 00020: val_loss did not improve from 0.88141
62268/62268 [==============================] - 12s 188us/sample - loss: 0.8321 - acc: 0.7468 -
Epoch 21/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.8274 - acc: 0.7474
Epoch 00021: val_loss improved from 0.88141 to 0.83501, saving model to mlp_checkpoints
62268/62268 [==============================] - 12s 193us/sample - loss: 0.8275 - acc: 0.7474 -
Epoch 22/30
61952/62268 [============================>.] - ETA: 0s - loss: 0.8103 - acc: 0.7547
Epoch 00022: val_loss did not improve from 0.83501
62268/62268 [==============================] - 12s 188us/sample - loss: 0.8106 - acc: 0.7546 -
Epoch 23/30
61824/62268 [============================>.] - ETA: 0s - loss: 0.8024 - acc: 0.7567
Epoch 00023: val_loss did not improve from 0.83501
62268/62268 [==============================] - 12s 188us/sample - loss: 0.8028 - acc: 0.7565 -
Epoch 24/30
61952/62268 [============================>.] - ETA: 0s - loss: 0.7958 - acc: 0.7579
Epoch 00024: val_loss did not improve from 0.83501
```

```
62268/62268 [==============================] - 12s 189us/sample - loss: 0.7960 - acc: 0.7579 -
Epoch 25/30
61952/62268 [============================>.] - ETA: 0s - loss: 0.7865 - acc: 0.7610
Epoch 00025: val_loss improved from 0.83501 to 0.81389, saving model to mlp_checkpoints
62268/62268 [==============================] - 12s 190us/sample - loss: 0.7863 - acc: 0.7611 -
Epoch 26/30
61824/62268 [============================>.] - ETA: 0s - loss: 0.7725 - acc: 0.7653
Epoch 00026: val_loss did not improve from 0.81389
62268/62268 [==============================] - 12s 189us/sample - loss: 0.7727 - acc: 0.7650 -
Epoch 27/30
61696/62268 [============================>.] - ETA: 0s - loss: 0.7718 - acc: 0.7665
Epoch 00027: val_loss did not improve from 0.81389
62268/62268 [==============================] - 12s 188us/sample - loss: 0.7721 - acc: 0.7663 -
Epoch 28/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.7608 - acc: 0.7714
Epoch 00028: val_loss improved from 0.81389 to 0.79495, saving model to mlp_checkpoints
62268/62268 [==============================] - 12s 192us/sample - loss: 0.7610 - acc: 0.7713 -
Epoch 29/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.7508 - acc: 0.7729
Epoch 00029: val_loss did not improve from 0.79495
62268/62268 [==============================] - 12s 186us/sample - loss: 0.7508 - acc: 0.7730 -
Epoch 30/30
61824/62268 [============================>.] - ETA: 0s - loss: 0.7454 - acc: 0.7744
Epoch 00030: val_loss improved from 0.79495 to 0.79111, saving model to mlp_checkpoints
62268/62268 [==============================] - 12s 190us/sample - loss: 0.7455 - acc: 0.7744 -
```

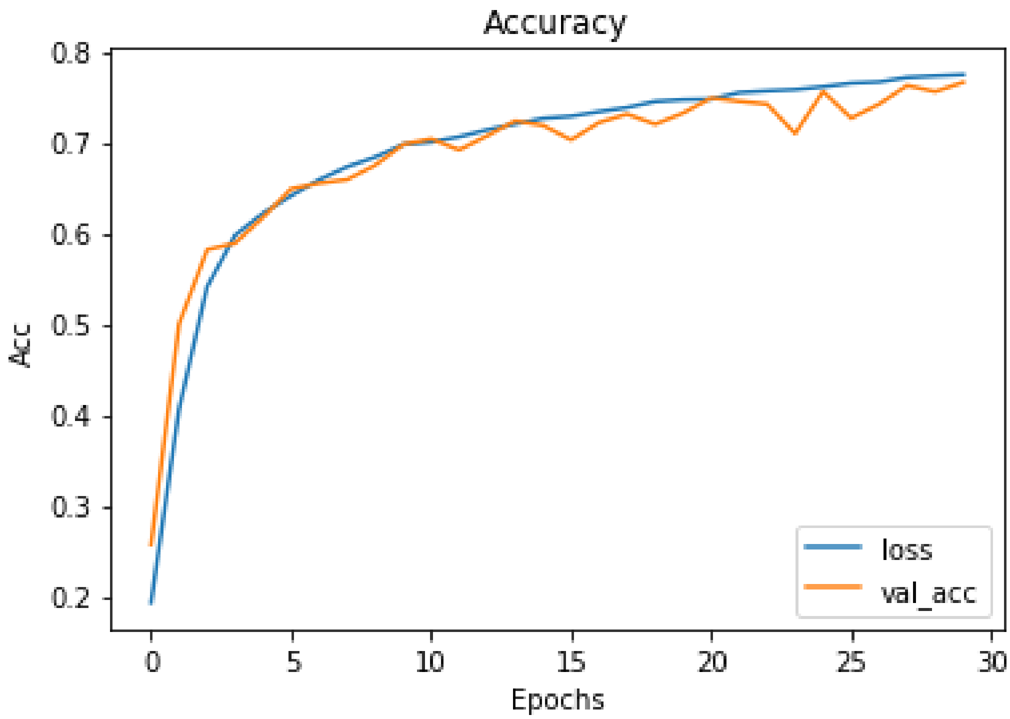
In [66]:
```python
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['loss','val_loss'], loc='upper right')
plt.title("Loss")
```

Out[66]:
```
Text(0.5, 1.0, 'Loss')
```



In [67]:
```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.xlabel('Epochs')
plt.ylabel('Acc')
plt.legend(['loss','val_acc'], loc='lower right')
plt.title("Accuracy")
```

Out[67]:
```
Text(0.5, 1.0, 'Accuracy')
```




```
In [68]: test_loss, test_accuracy = model.evaluate(test_gs, y_test_oh, verbose=2)
```

```
26032/1 - 3s - loss: 0.7973 - acc: 0.7425
```

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 [54]: checkpoint = ModelCheckpoint(filepath = 'model_checkpoints', save_best_only=True, save
         earlystop = EarlyStopping(patience=5, monitor='loss')
```

In [44]:
```
model_cnn = Sequential([
            Conv2D(filters=32, input_shape=train_gs[0].shape, kernel_size=(2, 2),
                   activation='relu'),
            MaxPool2D(pool_size= (2,2)),
            Conv2D(filters=16, kernel_size=(2, 2), activation='relu'),
            MaxPool2D(pool_size= (2,2)),
            Conv2D(filters=8, kernel_size=(2, 2), activation='relu'),
            MaxPooling2D(pool_size=(2, 2)),
            BatchNormalization(),
            Flatten(name='flatten'),
            Dense(units=32, activation='relu'),
            Dense(units=16, activation='relu'),
            Dense(units=10, activation='softmax')
        ])
```

In [45]:
```
model_cnn.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 31, 31, 32)        160
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 16)        2064
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 16)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 6, 6, 8)           520
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 3, 3, 8)           0
_________________________________________________________________
batch_normalization (BatchNo (None, 3, 3, 8)           32
_________________________________________________________________
flatten (Flatten)            (None, 72)                0
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2336
_________________________________________________________________
dense_5 (Dense)              (None, 16)                528
_________________________________________________________________
dense_6 (Dense)              (None, 10)                170
=================================================================
Total params: 5,810
Trainable params: 5,794
Non-trainable params: 16
_________________________________________________________________
```

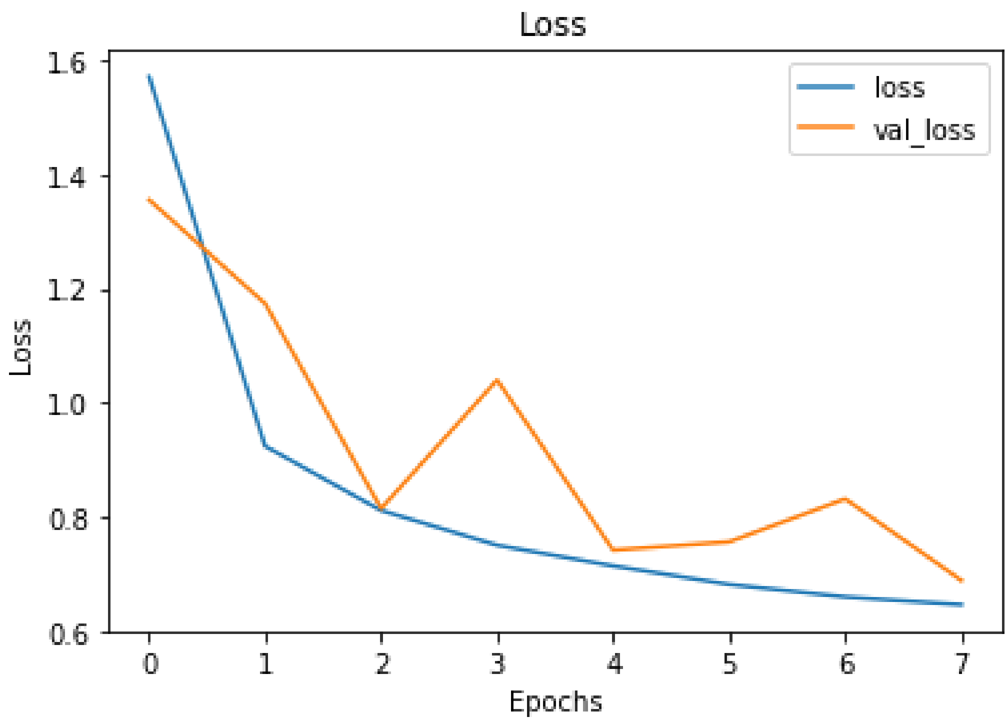
In [46]:
```
model_cnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
```

In [47]:
```
history_cnn = model_cnn.fit(train_gs, y_train_oh, callbacks=[checkpoint, earlystop], 
```

```
Train on 62268 samples, validate on 10989 samples
Epoch 1/8
62208/62268 [============================>.] - ETA: 0s - loss: 1.5713 - acc: 0.4593
Epoch 00001: val_loss did not improve from 0.75925
62268/62268 [==============================] - 159s 3ms/sample - loss: 1.5710 - acc: 0.4594 - 
Epoch 2/8
62208/62268 [============================>.] - ETA: 0s - loss: 0.9241 - acc: 0.7069
Epoch 00002: val_loss did not improve from 0.75925
62268/62268 [==============================] - 156s 3ms/sample - loss: 0.9242 - acc: 0.7069 - 
Epoch 3/8
62208/62268 [============================>.] - ETA: 0s - loss: 0.8119 - acc: 0.7454
Epoch 00003: val_loss did not improve from 0.75925
62268/62268 [==============================] - 162s 3ms/sample - loss: 0.8117 - acc: 0.7455 - 
Epoch 4/8
62208/62268 [============================>.] - ETA: 0s - loss: 0.7508 - acc: 0.7654
Epoch 00004: val_loss did not improve from 0.75925
62268/62268 [==============================] - 159s 3ms/sample - loss: 0.7507 - acc: 0.7654 - 
Epoch 5/8
62208/62268 [============================>.] - ETA: 0s - loss: 0.7145 - acc: 0.7779
Epoch 00005: val_loss improved from 0.75925 to 0.74222, saving model to model_checkpoints
62268/62268 [==============================] - 163s 3ms/sample - loss: 0.7145 - acc: 0.7779 - 
Epoch 6/8
62208/62268 [============================>.] - ETA: 0s - loss: 0.6824 - acc: 0.7862
Epoch 00006: val_loss did not improve from 0.74222
62268/62268 [==============================] - 160s 3ms/sample - loss: 0.6823 - acc: 0.7862 - 
Epoch 7/8
62208/62268 [============================>.] - ETA: 0s - loss: 0.6606 - acc: 0.7932
Epoch 00007: val_loss did not improve from 0.74222
62268/62268 [==============================] - 161s 3ms/sample - loss: 0.6604 - acc: 0.7933 - 
Epoch 8/8
62208/62268 [============================>.] - ETA: 0s - loss: 0.6469 - acc: 0.7972
Epoch 00008: val_loss improved from 0.74222 to 0.68932, saving model to model_checkpoints
62268/62268 [==============================] - 160s 3ms/sample - loss: 0.6469 - acc: 0.7972 - 
```

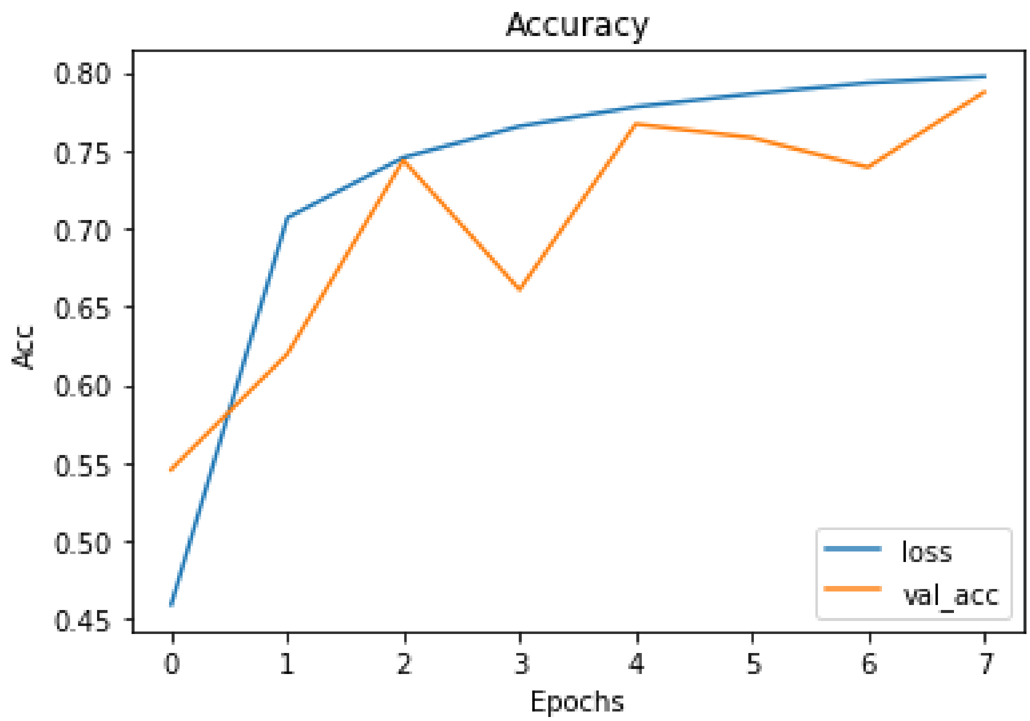
In [50]:
```
plt.plot(history_cnn.history['loss'])
plt.plot(history_cnn.history['val_loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['loss','val_loss'], loc='upper right')
plt.title("Loss")
```

Out[50]:
```
Text(0.5, 1.0, 'Loss')
```



In [51]:
```
plt.plot(history_cnn.history['acc'])
plt.plot(history_cnn.history['val_acc'])
plt.xlabel('Epochs')
plt.ylabel('Acc')
plt.legend(['loss','val_acc'], loc='lower right')
plt.title("Accuracy")
```

Out[51]:
```
Text(0.5, 1.0, 'Accuracy')
```




```
In [52]: test_loss, test_accuracy = model_cnn.evaluate(test_gs, y_test_oh, verbose=2)
```

```
26032/1 - 22s - loss: 0.6889 - acc: 0.7838
```

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 [69]: model.load_weights('mlp_checkpoints')
```

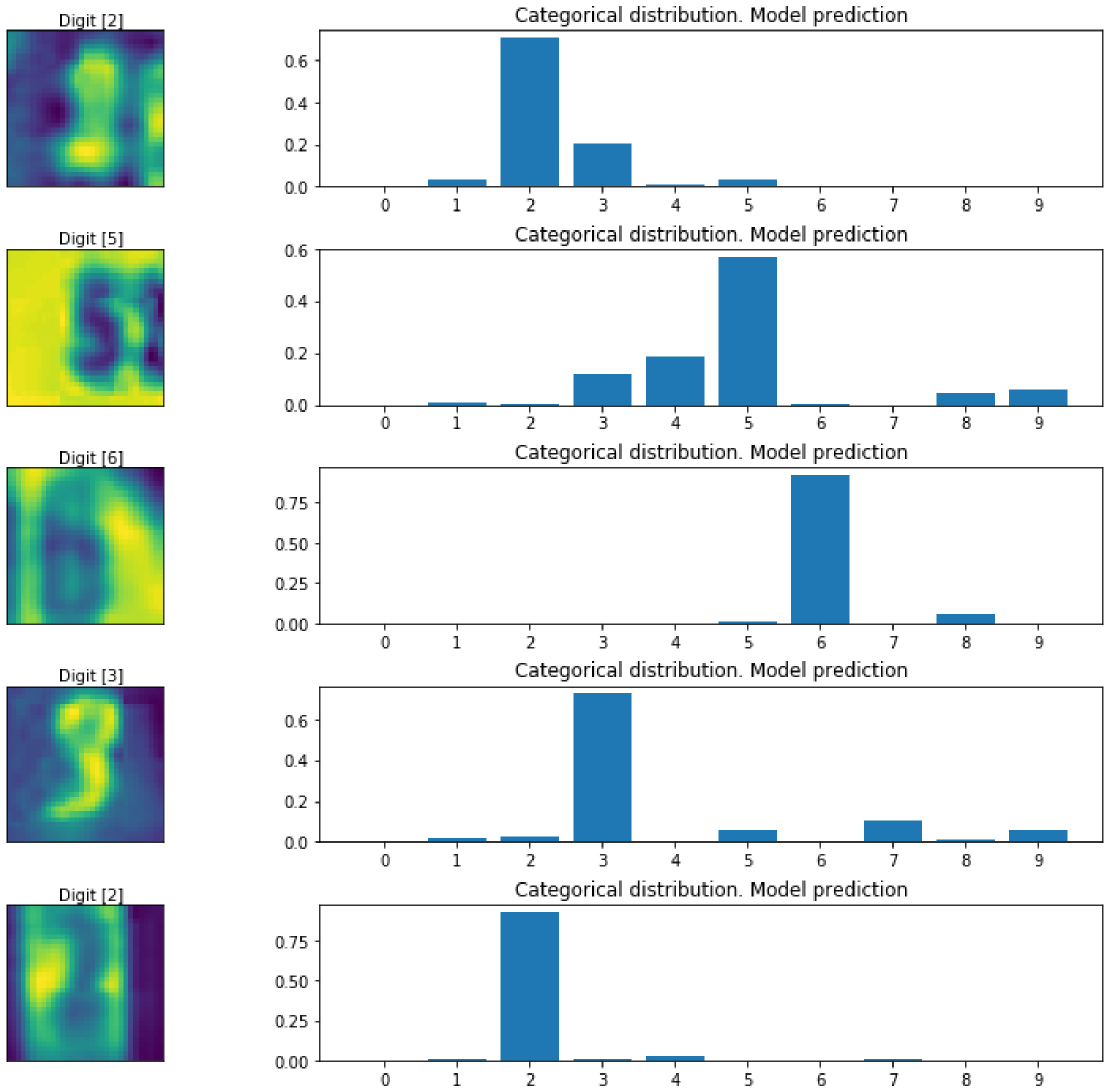
```
Out[69]: <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f4ca5b15dd8>
```

```
In [56]: model_cnn.load_weights('model_checkpoints')
```

```
Out[56]: <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f4cf0604978>
```

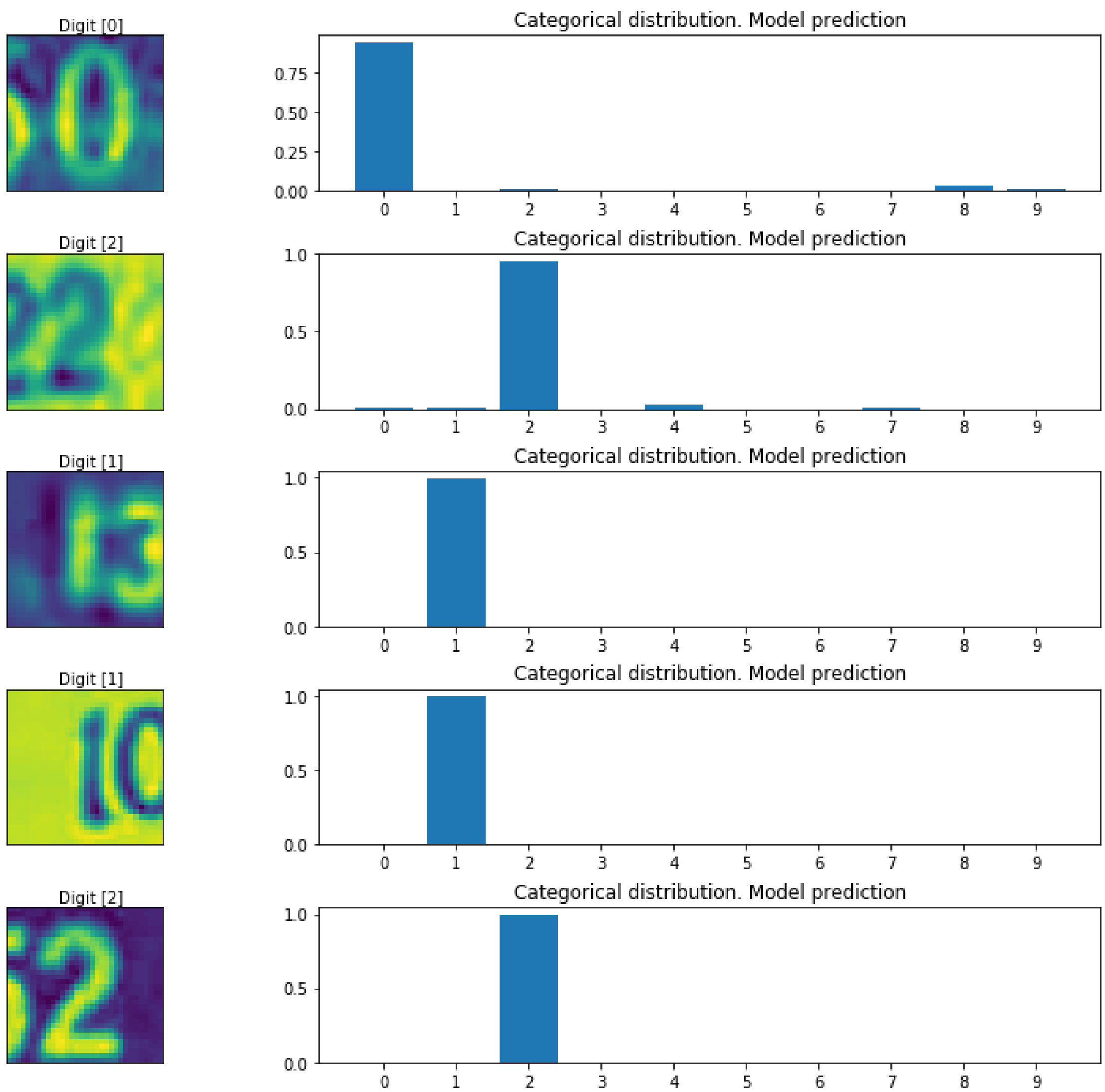
```
In [83]: num_test_images = test_gs.shape[0]

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

```
random_test_labels = y_test[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, ra
    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(0,10), prediction)
    axes[i, 1].set_xticks(np.arange(0,10))
    axes[i, 1].set_title("Categorical distribution. Model prediction")

plt.show()
```



In [82]:
```
num_test_images = test_gs.shape[0]

random_inx = np.random.choice(num_test_images, 5)
random_test_images = test_gs[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, ra
    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(0,10), prediction)
        axes[i, 1].set_xticks(np.arange(0,10))
        axes[i, 1].set_title("Categorical distribution. Model prediction")

    plt.show()
```




```
In [ ]:
```

```
In [ ]:
```