Note to Peer Reviewer: Coursera notebook platform's kernel keep dying at times, hence cannot generate full results

Capstone Project

Image classifier for the SVHN dataset

Instructions

In this notebook, you will create a neural network that classifies real-world images digits. You will use concepts from throughout this 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.

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.

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
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Flatten, BatchNormalization, MaxPool2D,
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
%matplotlib inline
```

SVHN overview image 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. 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]:
```
X_train = train['X']
X_test = test['X']
y_train = train['y']
y_test = test['y']
```

In [4]:
```
X_train.shape, X_test.shape
```

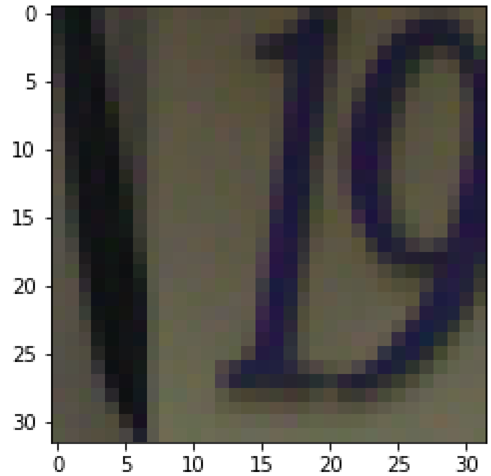
Out[4]:
```
((32, 32, 3, 73257), (32, 32, 3, 26032))
```

In [5]:
```
X_train = np.moveaxis(X_train, -1, 0)
X_test = np.moveaxis(X_test, -1 , 0)
```

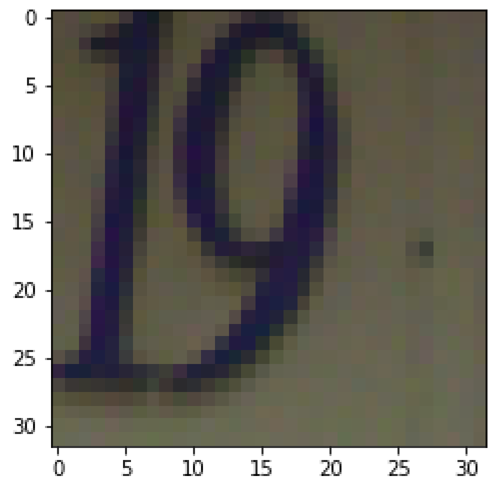
In [6]:
```
X_train.shape, X_test.shape
```

Out[6]:
```
((73257, 32, 32, 3), (26032, 32, 32, 3))
```

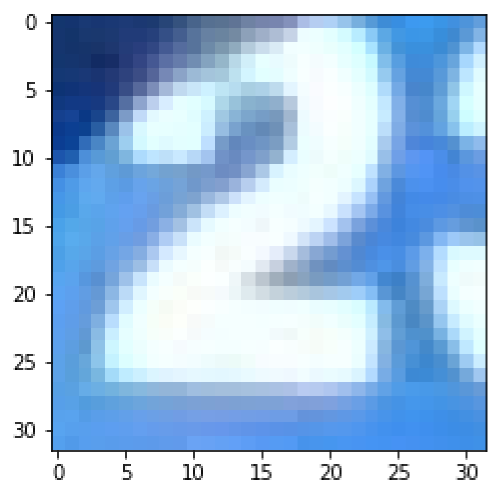
In [7]:
```
for i in range(10):
    plt.imshow(X_train[i, :, :, :,])
    plt.show()
    print(y_train[i])
```



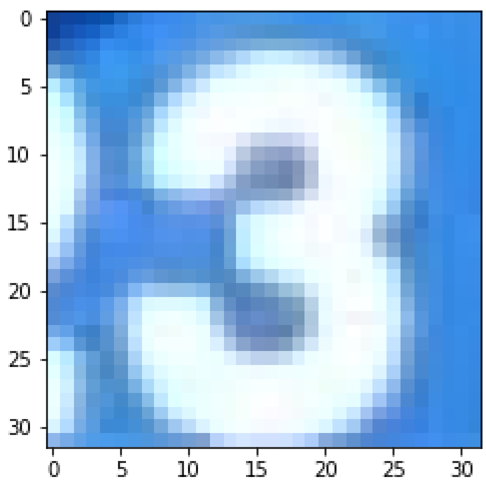
[1]



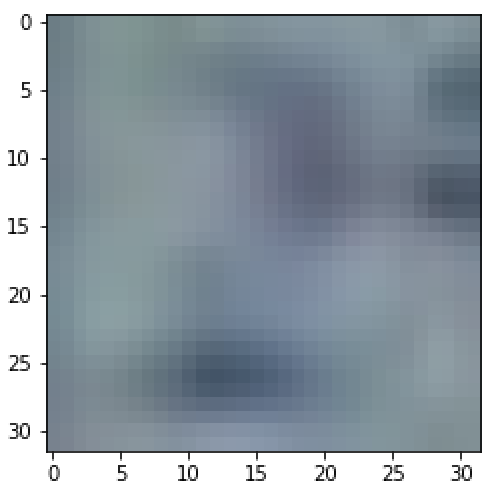
[9]



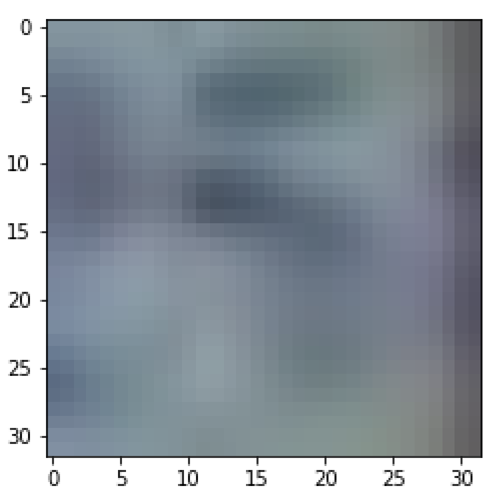
[2]



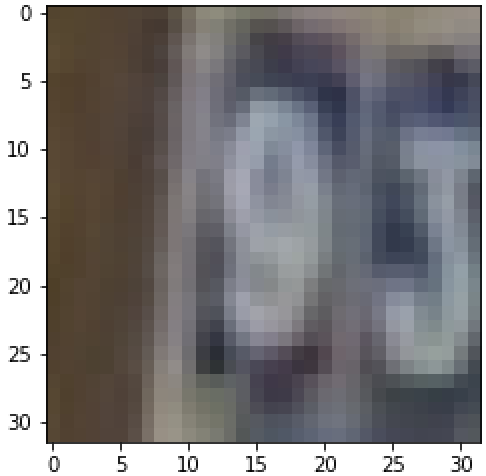
[3]



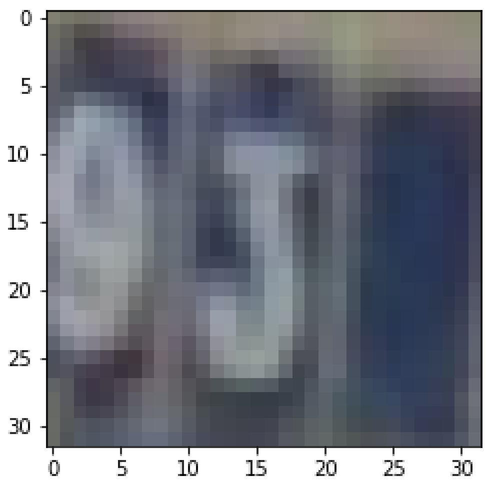
[2]



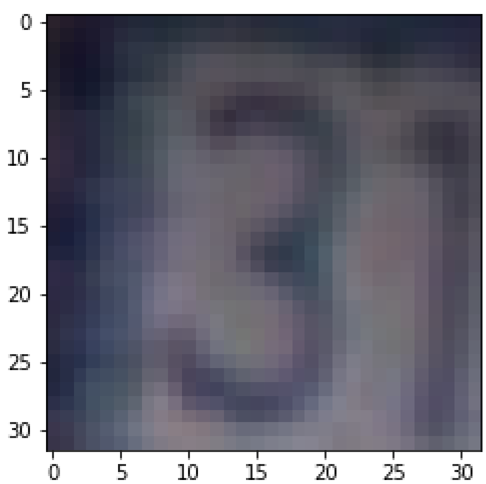
[5]



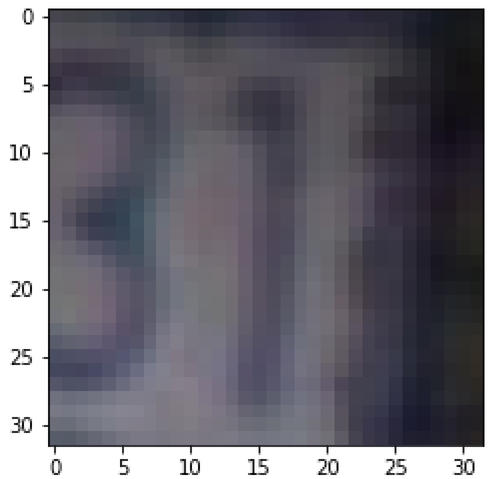
[9]



[3]



[3]



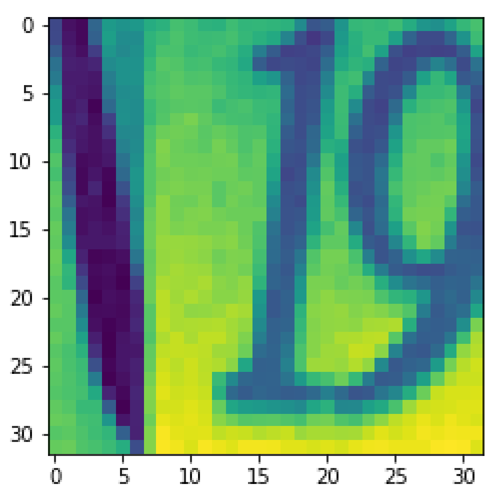
[1]

In [8]:
```
X_train_gs = np.mean(X_train, 3).reshape(73257, 32, 32, 1)/255
X_test_gs = np.mean(X_test,3).reshape(26032, 32,32 ,1)/255
X_train_for_plotting = np.mean(X_train,3)
```

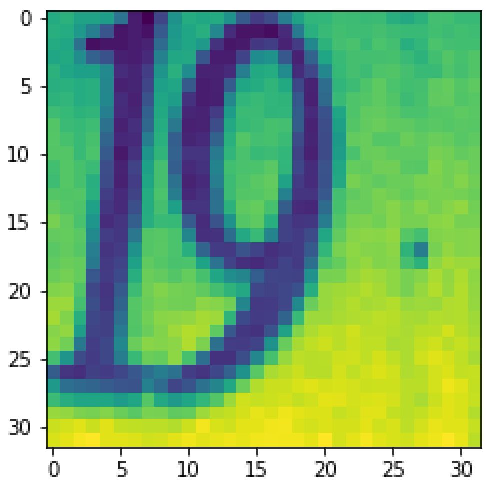
In [9]:
```
X_train_gs.shape
```

Out[9]: (73257, 32, 32, 1)

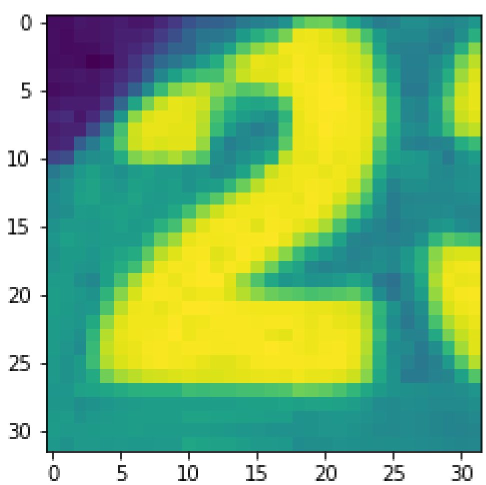
In [10]:
```
for i in range(10):
    plt.imshow(X_train_for_plotting[i, :, :,])
    plt.show()
    print(y_train[i])
```



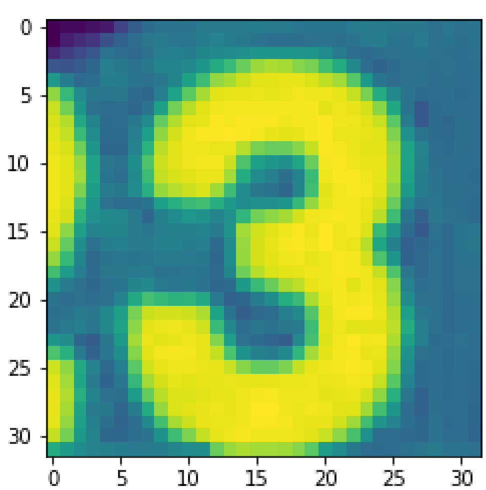
[1]



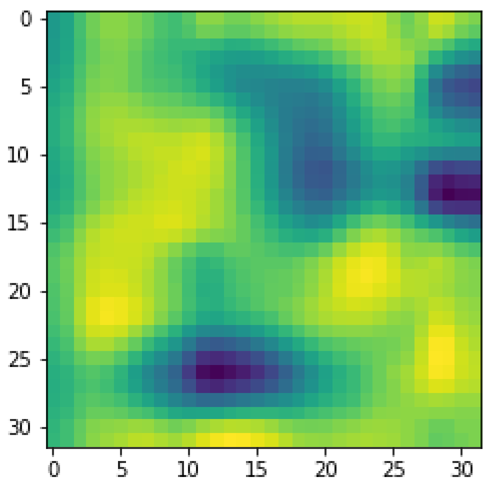
[9]



[2]

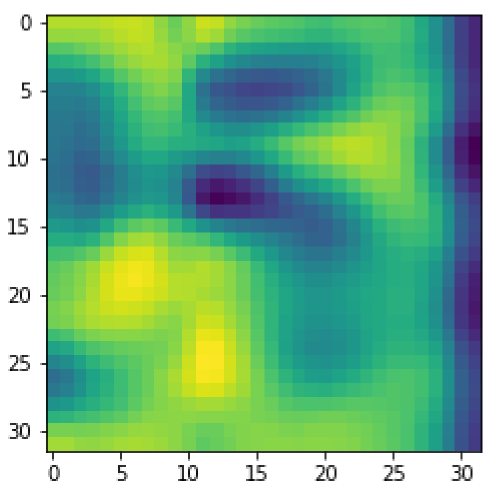


[3]



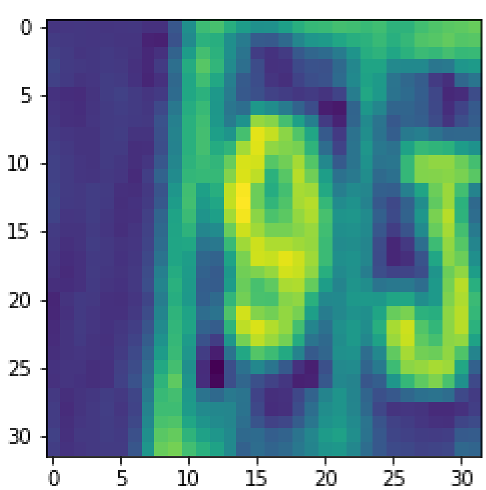
0
5
10
15
20
25
30
0
5
10
15
20
25
30

[2]



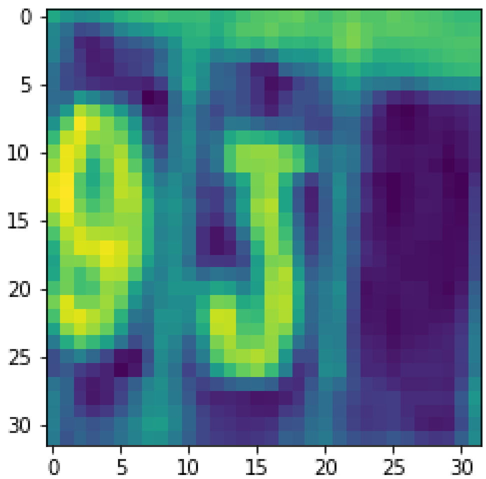
0
5
10
15
20
25
30
0
5
10
15
20
25
30

[5]

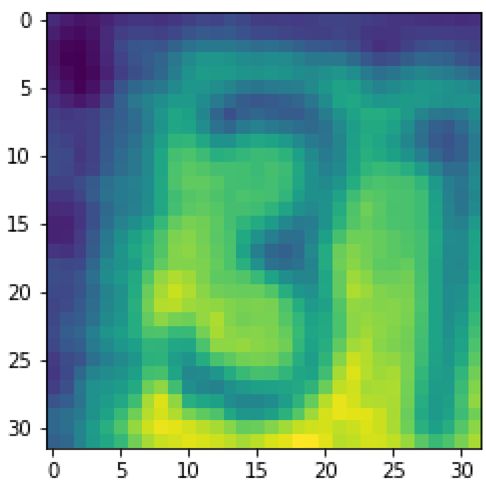


0
5
10
15
20
25
30
0
5
10
15
20
25
30

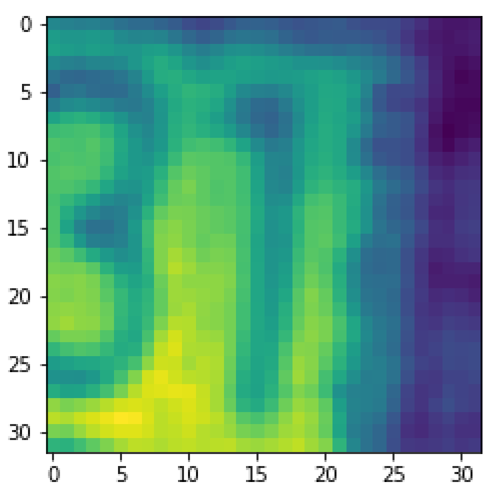
[9]



[3]



[3]



[1]

In [11]:
```
X_train[0].shape
```

Out[11]: (32, 32, 3)

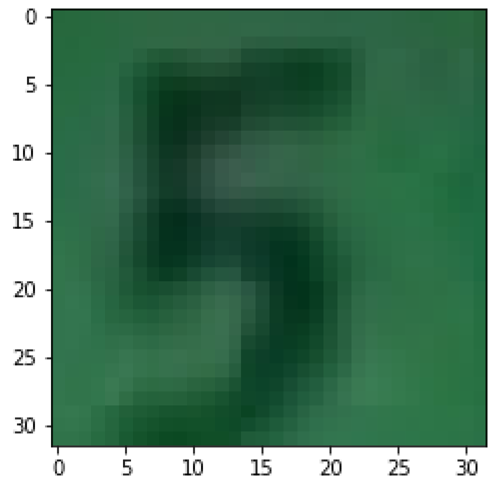
In [12]:
```
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder().fit(y_train)
```

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

In [13]:
```python
y_test_oh[0]
```

Out[13]:
```
array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0.])
```

In [14]:
```python
plt.imshow(X_test[0])
plt.show()
```



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 [17]:
```python
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
```

In [18]:
```python
checkpoint = ModelCheckpoint(filepath = 'SeqMode\\mySeqModel', save_best_only=True,
earlystop = EarlyStopping(patience=5, monitor='loss')
```

In [19]:
```python
model2 = Sequential([
    Flatten(input_shape=X_train[0].shape),
    Dense(128*4, activation='relu'),
    Dense(64, activation='relu'),
    BatchNormalization(),
    Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    Dense(32, activation='relu'),
    Dense(10, activation='softmax')
])
model2.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten (Flatten)            (None, 3072)              0         
_________________________________________________________________
dense (Dense)                (None, 512)               1573376   
_________________________________________________________________
dense_1 (Dense)              (None, 64)                32832     
_________________________________________________________________
batch_normalization (BatchNo (None, 64)                256       
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160      
_________________________________________________________________
dropout (Dropout)            (None, 64)                0         
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080      
_________________________________________________________________
dense_4 (Dense)              (None, 10)                330       
=================================================================
Total params: 1,613,034
Trainable params: 1,612,906
Non-trainable params: 128
_________________________________________________________________
```

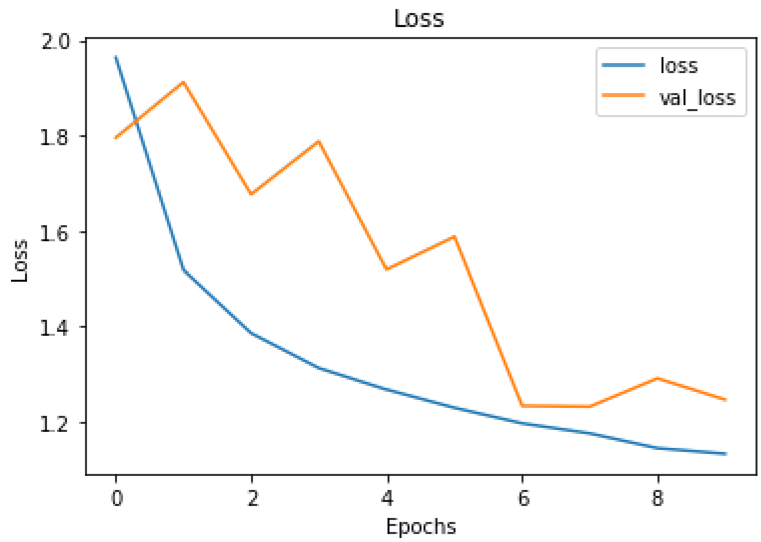
In [20]:
```python
model2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
```

In [19]:
```python
history = model2.fit(X_train, y_train_oh, callbacks=[checkpoint, earlystop], batch_
                     , epochs=10)
```

```
Train on 73257 samples, validate on 26032 samples
Epoch 1/10
73216/73257 [============================>.] - ETA: 0s - loss: 1.9638 - acc: 0.3001
Epoch 00001: val_loss improved from inf to 1.79571, saving model to SeqMode\mySeqMod
el
73257/73257 [==============================] - 81s 1ms/sample - loss: 1.9637 - acc:
0.3002 - val_loss: 1.7957 - val_acc: 0.3811
Epoch 2/10
73216/73257 [============================>.] - ETA: 0s - loss: 1.5185 - acc: 0.4844
Epoch 00002: val_loss did not improve from 1.79571
73257/73257 [==============================] - 78s 1ms/sample - loss: 1.5186 - acc:
0.4844 - val_loss: 1.9122 - val_acc: 0.3762
Epoch 3/10
73216/73257 [============================>.] - ETA: 0s - loss: 1.3865 - acc: 0.5435
Epoch 00003: val_loss improved from 1.79571 to 1.67678, saving model to SeqMode\mySe
qModel
73257/73257 [==============================] - 78s 1ms/sample - loss: 1.3866 - acc:
0.5434 - val_loss: 1.6768 - val_acc: 0.4510
Epoch 4/10
73216/73257 [============================>.] - ETA: 0s - loss: 1.3135 - acc: 0.5771
Epoch 00004: val_loss did not improve from 1.67678
73257/73257 [==============================] - 81s 1ms/sample - loss: 1.3134 - acc:
0.5771 - val_loss: 1.7880 - val_acc: 0.5576
Epoch 5/10
73216/73257 [============================>.] - ETA: 0s - loss: 1.2681 - acc: 0.5928
Epoch 00005: val_loss improved from 1.67678 to 1.51952, saving model to SeqMode\mySe
qModel
73257/73257 [==============================] - 81s 1ms/sample - loss: 1.2682 - acc:
0.5928 - val_loss: 1.5195 - val_acc: 0.6017
Epoch 6/10
73216/73257 [============================>.] - ETA: 0s - loss: 1.2299 - acc: 0.6086
Epoch 00006: val_loss did not improve from 1.51952
73257/73257 [==============================] - 90s 1ms/sample - loss: 1.2300 - acc:
0.6086 - val_loss: 1.5881 - val_acc: 0.5265
Epoch 7/10
73216/73257 [============================>.] - ETA: 0s - loss: 1.1975 - acc: 0.6197
Epoch 00007: val_loss improved from 1.51952 to 1.23404, saving model to SeqMode\mySe
qModel
73257/73257 [==============================] - 91s 1ms/sample - loss: 1.1974 - acc:
0.6197 - val_loss: 1.2340 - val_acc: 0.6263
Epoch 8/10
73216/73257 [============================>.] - ETA: 0s - loss: 1.1765 - acc: 0.6277
Epoch 00008: val_loss improved from 1.23404 to 1.23249, saving model to SeqMode\mySe
qModel
73257/73257 [==============================] - 78s 1ms/sample - loss: 1.1763 - acc:
0.6277 - val_loss: 1.2325 - val_acc: 0.6300
Epoch 9/10
73216/73257 [============================>.] - ETA: 0s - loss: 1.1453 - acc: 0.6407
Epoch 00009: val_loss did not improve from 1.23249
73257/73257 [==============================] - 79s 1ms/sample - loss: 1.1453 - acc:
0.6407 - val_loss: 1.2917 - val_acc: 0.5895
Epoch 10/10
73216/73257 [============================>.] - ETA: 0s - loss: 1.1337 - acc: 0.6438
Epoch 00010: val_loss did not improve from 1.23249
73257/73257 [==============================] - 93s 1ms/sample - loss: 1.1336 - acc:
0.6438 - val_loss: 1.2471 - val_acc: 0.6276
```

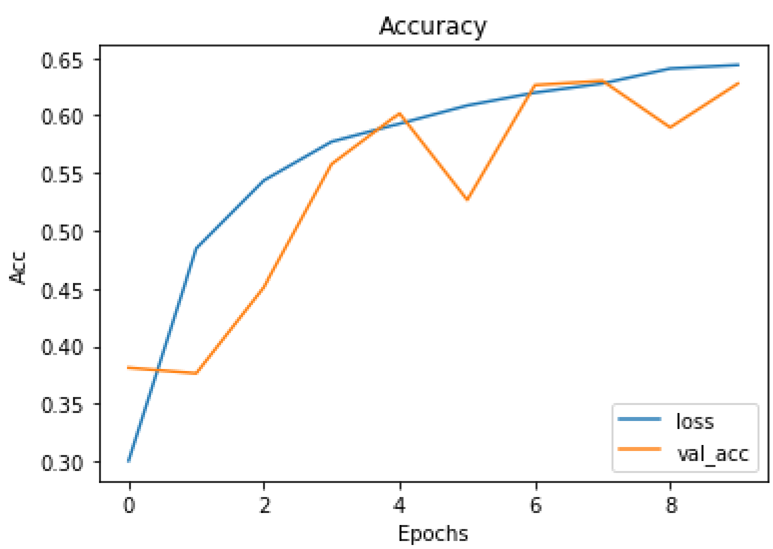
In [20]:
```python
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['loss','val_loss'], loc='upper right')
plt.title("Loss")
```

Out[20]: Text(0.5, 1.0, 'Loss')



In [21]:
```python
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[21]: Text(0.5, 1.0, 'Accuracy')



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 [15]:

```
model3 = Sequential([
    Conv2D(filters= 16, kernel_size= 3, activation='relu', input_shape=X_train[0].s
    MaxPool2D(pool_size= (3,3), strides=1),
    Conv2D(filters= 32, kernel_size = 3, padding='valid', strides=1, activation='re
    MaxPool2D(pool_size = (1,1), strides = 3),
    BatchNormalization(),
    Conv2D(filters= 32, kernel_size = 3, padding='valid', strides=2, activation='re
    tf.keras.layers.Dropout(0.5),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(32, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    Dense(10, activation='softmax')
])
```

In [16]:

```
model3.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 30, 30, 16)        448       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 28, 28, 16)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 26, 26, 32)        4640      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 9, 9, 32)          0         
_________________________________________________________________
batch_normalization (BatchNo (None, 9, 9, 32)          128       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 32)          9248      
_________________________________________________________________
dropout (Dropout)            (None, 4, 4, 32)          0         
_________________________________________________________________
flatten (Flatten)            (None, 512)               0         
_________________________________________________________________
dense (Dense)                (None, 128)               65664     
_________________________________________________________________
dense_1 (Dense)              (None, 32)                4128      
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                330       
=================================================================
Total params: 84,586
Trainable params: 84,522
Non-trainable params: 64
_________________________________________________________________
```

In [19]:
```python
model3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])

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

In [20]:
```python
callback1 = ModelCheckpoint(filepath='CNNweights', save_best_only=True, save_weight
callback2 = EarlyStopping(monitor='loss',patience=7, verbose=1)
```

In [22]:
```python
checkpoint = ModelCheckpoint(filepath = 'SeqMode\\mySeqModel', save_best_only=True,
earlystop = EarlyStopping(patience=5, monitor='loss')
```

In [ ]:
```python
history = model3.fit(X_train, y_train_oh, callbacks=[checkpoint, earlystop],
                     batch_size=256, validation_data=(X_test, y_test_oh), epochs=10
```

```
Train on 73257 samples, validate on 26032 samples
Epoch 1/10
54272/73257 [=====================>........] - ETA: 2:00 - loss: 2.0088 - acc: 0.277
9
```

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 [15]:
```python
import random
```

In [21]:
```python
model2.load_weights('SeqMode\\mySeqModel')
```

```
---------------------------------------------------------------------------
NotFoundError                             Traceback (most recent call last)
<ipython-input-21-9f396cd752b9> in <module>
----> 1 model2.load_weights('SeqMode\\mySeqModel')

/opt/conda/lib/python3.7/site-packages/tensorflow_core/python/keras/engine/training.py in load_weights(self, filepath, by_name)
    179         raise ValueError('Load weights is not yet supported with TPUStrategy '
    180                          'with steps_per_run greater than 1.')
--> 181     return super(Model, self).load_weights(filepath, by_name)
    182 
    183   @trackable.no_automatic_dependency_tracking

/opt/conda/lib/python3.7/site-packages/tensorflow_core/python/keras/engine/network.py in load_weights(self, filepath, by_name)
   1141     else:
   1142       try:
-> 1143         pywrap_tensorflow.NewCheckpointReader(filepath)
   1144         save_format = 'tf'
   1145       except errors_impl.DataLossError:

/opt/conda/lib/python3.7/site-packages/tensorflow_core/python/pywrap_tensorflow_internal.py in NewCheckpointReader(filepattern)
    871 def NewCheckpointReader(filepattern):
    872   from tensorflow.python.util import compat
--> 873   return CheckpointReader(compat.as_bytes(filepattern))
    874 
    875 NewCheckpointReader._tf_api_names_v1 = ['train.NewCheckpointReader']

/opt/conda/lib/python3.7/site-packages/tensorflow_core/python/pywrap_tensorflow_internal.py in __init__(self, filename)
    883 
    884     def __init__(self, filename):
--> 885         this = _pywrap_tensorflow_internal.new_CheckpointReader(filename)
    886         try:
    887             self.this.append(this)

NotFoundError: Unsuccessful TensorSliceReader constructor: Failed to find any matching files for SeqMode\mySeqModel
```

In [ ]:
```python
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 = model2.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(1,11), prediction)
    axes[i, 1].set_xticks(np.arange(1,11))
    axes[i, 1].set_title("Categorical distribution. Model prediction")

plt.show()
```