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

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout, BatchNor
malization
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
```

SVHN overview image

For the capstone project, you will use the [SVHN dataset (http://ufldl.stanford.edu/housenumbers/)](http://ufldl.stanford.edu/housenumbers/). 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]:

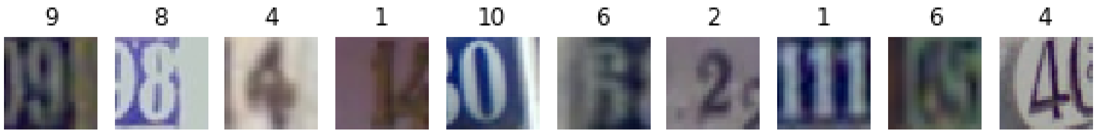
```
X_train = train['X']
y_train = train['y']
```

In [4]:

```
X_test = test['X']
y_test = test['y']
```

In [6]:

```
# Select a random sample of images and corresponding labels from the dataset (at least 10), and display them in a figure.

# get the random 10 samples of images
num = 10
random_visual_list = np.random.randint(0, X_train.shape[3], num)

# visualizations
fig, ax = plt.subplots(1, num, figsize=(num, 1))
for n,i in enumerate(random_visual_list):
    ax[n].set_axis_off()
    ax[n].imshow(X_train[:,:,:,i])
    ax[n].set_title(y_train[i][0])
```

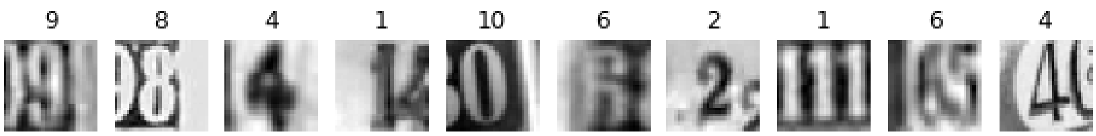


In [7]:

```
# Convert the training and test images to grayscale by taking the average across all colour channels for each pixel.

X_train_grayscaled = np.mean(X_train, axis = 2)/255.0
X_test_grayscaled = np.mean(X_test, axis = 2)/255.0
```

In [8]:

```
# Select a random sample of the grayscale images and corresponding labels from the dataset (at least 10), and display them in a figure.

# visualizations
fig, ax = plt.subplots(1, num, figsize=(num, 1))
for n,i in enumerate(random_visual_list):
    ax[n].set_axis_off()
    ax[n].imshow(X_train_grayscaled[:,:,i], 'gray')
    ax[n].set_title(y_train[i][0])
```



In [8]:

```
# do some changes for inputs to feed into model
# change the shape of inputs
X_train_grayscaled_shaped = X_train_grayscaled.transpose(2,0,1)[:,:,:,np.newaxis]
X_test_grayscaled_shaped = X_test_grayscaled.transpose(2,0,1)[:,:,:,np.newaxis]
print('X train shape: ',X_train_grayscaled_shaped.shape)
print('X test shape: ',X_test_grayscaled_shaped.shape)
```

```
X train shape:  (73257, 32, 32, 1)
X test shape:  (26032, 32, 32, 1)
```

In [9]:

```
# change the output value range from [1,10] to [0,9]
y_train_ranged = y_train - 1
y_test_ranged = y_test -1
print('y range: ', y_train.min(), y_train.max())
print('change y range to :', y_train_ranged.min(), y_train_ranged.max())
```

```
y range:  1 10
change y range to : 0 9
```

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 [10]:

```
def get_mlp_model(input_shape):
    model = Sequential([
        Flatten(input_shape = input_shape, name = 'flatten',),
        Dense(2046, activation = 'relu', name = 'dense0'),
        Dense(1024, activation = 'relu', name = 'dense1'),
        Dense(128, activation = 'relu', name = 'dense2'),
        Dense(32, activation = 'relu', name = 'dense3'),
        Dense(10, activation = 'softmax', name = 'dense4'),
    ])

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

In [53]:

```
# get model
model = get_mlp_model((32,32,1))
```

In [54]:

```
#Print out the model summary (using the summary() method)
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten (Flatten)            (None, 1024)              0         
_________________________________________________________________
dense0 (Dense)               (None, 2046)              2097150   
_________________________________________________________________
dense1 (Dense)               (None, 1024)              2096128   
_________________________________________________________________
dense2 (Dense)               (None, 128)               131200    
_________________________________________________________________
dense3 (Dense)               (None, 32)                4128      
_________________________________________________________________
dense4 (Dense)               (None, 10)                330       
=================================================================
Total params: 4,328,936
Trainable params: 4,328,936
Non-trainable params: 0
_________________________________________________________________
```

In [14]:

```
# 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.
mlp_checkpoint_path = 'mlp_model_best_checkpoints'
checkpoint = ModelCheckpoint(filepath= mlp_checkpoint_path,
                             frequency = 'epoch',
                             save_weights_only= True,
                             save_best_only= True,
                             verbose= 1)
earlystopping = EarlyStopping(patience= 2)
```

In [15]:

```
# train the model (we recommend a maximum of 30 epochs),
# making use of both training and validation sets during the training run.
history = model.fit(x = X_train_grayscaled_shaped,
                    y = y_train_ranged,
                    batch_size = 128,
                    epochs= 30,
                    validation_data = [X_test_grayscaled_shaped, y_test_ranged],
                    callbacks = [earlystopping, checkpoint]
          )
```

```
Train on 73257 samples, validate on 26032 samples
Epoch 1/30
73216/73257 [============================>.] - ETA: 0s - loss: 2.1577 - accuracy:
0.2213
Epoch 00001: val_loss improved from inf to 1.89073, saving model to mlp_model_best
_checkpoints
73257/73257 [==============================] - 198s 3ms/sample - loss: 2.1576 - ac
curacy: 0.2214 - val_loss: 1.8907 - val_accuracy: 0.3300
Epoch 2/30
73216/73257 [============================>.] - ETA: 0s - loss: 1.5211 - accuracy:
0.4777
Epoch 00002: val_loss improved from 1.89073 to 1.37018, saving model to mlp_model_
best_checkpoints
73257/73257 [==============================] - 185s 3ms/sample - loss: 1.5210 - ac
curacy: 0.4778 - val_loss: 1.3702 - val_accuracy: 0.5621
Epoch 3/30
73216/73257 [============================>.] - ETA: 0s - loss: 1.2177 - accuracy:
0.6081
Epoch 00003: val_loss improved from 1.37018 to 1.26064, saving model to mlp_model_
best_checkpoints
73257/73257 [==============================] - 195s 3ms/sample - loss: 1.2178 - ac
curacy: 0.6080 - val_loss: 1.2606 - val_accuracy: 0.6056
Epoch 4/30
73216/73257 [============================>.] - ETA: 0s - loss: 1.1026 - accuracy:
0.6520
Epoch 00004: val_loss improved from 1.26064 to 1.17421, saving model to mlp_model_
best_checkpoints
73257/73257 [==============================] - 193s 3ms/sample - loss: 1.1025 - ac
curacy: 0.6520 - val_loss: 1.1742 - val_accuracy: 0.6423
Epoch 5/30
73216/73257 [============================>.] - ETA: 0s - loss: 1.0389 - accuracy:
0.6731
Epoch 00005: val_loss improved from 1.17421 to 1.11096, saving model to mlp_model_
best_checkpoints
73257/73257 [==============================] - 190s 3ms/sample - loss: 1.0388 - ac
curacy: 0.6732 - val_loss: 1.1110 - val_accuracy: 0.6616
Epoch 6/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.9635 - accuracy:
0.6975
Epoch 00006: val_loss improved from 1.11096 to 1.02903, saving model to mlp_model_
best_checkpoints
73257/73257 [==============================] - 192s 3ms/sample - loss: 0.9634 - ac
curacy: 0.6975 - val_loss: 1.0290 - val_accuracy: 0.6851
Epoch 7/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.9091 - accuracy:
0.7158
Epoch 00007: val_loss improved from 1.02903 to 1.02219, saving model to mlp_model_
best_checkpoints
73257/73257 [==============================] - 195s 3ms/sample - loss: 0.9091 - ac
curacy: 0.7158 - val_loss: 1.0222 - val_accuracy: 0.6914
Epoch 8/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.8704 - accuracy:
0.7285
Epoch 00008: val_loss improved from 1.02219 to 0.97826, saving model to mlp_model_
best_checkpoints
73257/73257 [==============================] - 192s 3ms/sample - loss: 0.8703 - ac
curacy: 0.7286 - val_loss: 0.9783 - val_accuracy: 0.6976
Epoch 9/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.8360 - accuracy:
0.7375
Epoch 00009: val_loss improved from 0.97826 to 0.97380, saving model to mlp_model_
```

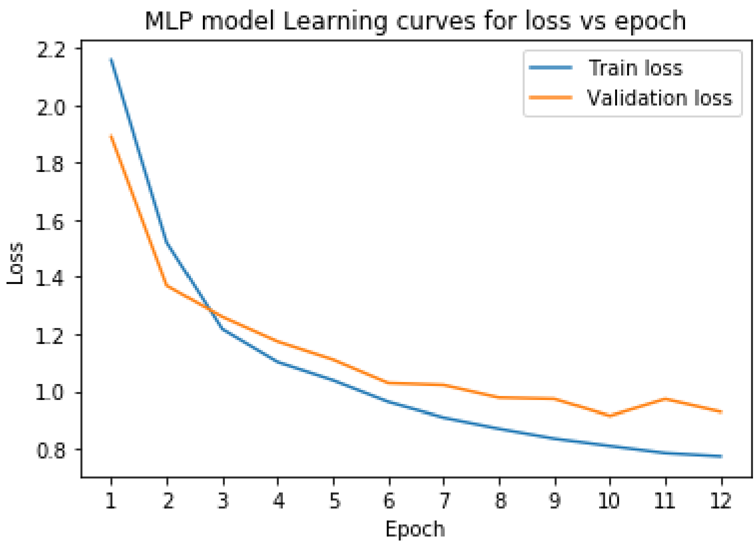
```
best_checkpoints
73257/73257 [==============================] - 194s 3ms/sample - loss: 0.8358 - ac
curacy: 0.7375 - val_loss: 0.9738 - val_accuracy: 0.7012
Epoch 10/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.8100 - accuracy:
0.7459
Epoch 00010: val_loss improved from 0.97380 to 0.91507, saving model to mlp_model_
best_checkpoints
73257/73257 [==============================] - 194s 3ms/sample - loss: 0.8101 - ac
curacy: 0.7459 - val_loss: 0.9151 - val_accuracy: 0.7169
Epoch 11/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.7853 - accuracy:
0.7541
Epoch 00011: val_loss did not improve from 0.91507
73257/73257 [==============================] - 193s 3ms/sample - loss: 0.7854 - ac
curacy: 0.7540 - val_loss: 0.9735 - val_accuracy: 0.6994
Epoch 12/30
73216/73257 [============================>.] - ETA: 0s - loss: 0.7741 - accuracy:
0.7555
Epoch 00012: val_loss did not improve from 0.91507
73257/73257 [==============================] - 194s 3ms/sample - loss: 0.7741 - ac
curacy: 0.7555 - val_loss: 0.9291 - val_accuracy: 0.7160
```

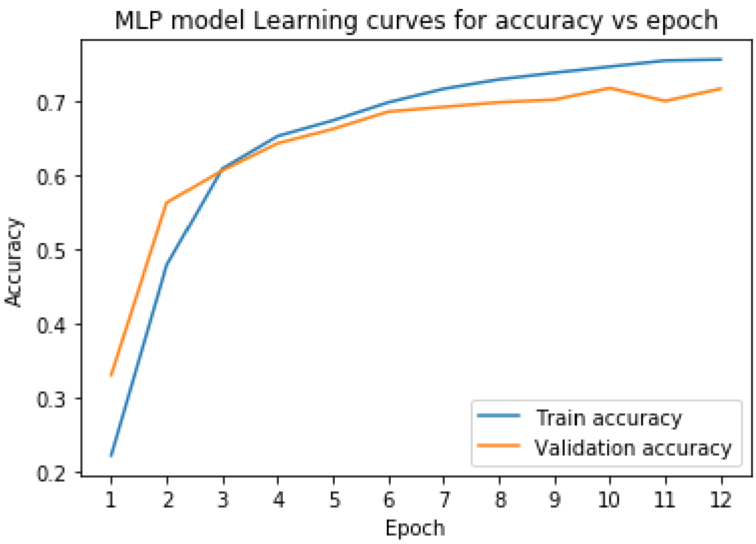
In [16]:

```
# Plot the learning curves for loss vs epoch and accuracy vs epoch for both training and validation sets.
plt.figure()
plt.plot( history.history['loss'], label = 'Train loss')
plt.plot( history.history['val_loss'], label = 'Validation loss')
plt.xticks(np.arange(0,len(history.history['loss'])), np.arange(1,len(history.history['loss'])+1) )
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('MLP model Learning curves for loss vs epoch')
plt.legend()


plt.figure()
plt.plot( history.history['accuracy'], label = 'Train accuracy')
plt.plot( history.history['val_accuracy'], label = 'Validation accuracy')
plt.xticks(np.arange(0,len(history.history['accuracy'])), np.arange(1,len(history.history['accuracy'])+1) )
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('MLP model Learning curves for accuracy vs epoch')
plt.legend()
```

Out[16]:

```
<matplotlib.legend.Legend at 0x7ff59c22fe10>
```





In [61]:

```
# Compute and display the loss and accuracy of the trained model on the test set.
def get_test_loss_accuracy(model, x_test, y_test):
    test_loss, test_acc = model.evaluate(x=x_test, y=y_test, verbose=0)
    print('Test accuracy: {:0.3f}, Test loss: {:0.3f}'.format(test_acc, test_loss))
```

In [17]:

```
# mlp model
get_test_loss_accuracy(model, X_test_grayscaled_shaped, y_test_ranged)
```

```
Test accuracy: 0.716, Test loss: 0.929
```

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 [11]:

```
def get_cnn_model(input_shape):
    model = Sequential([
        Conv2D(filters=32, input_shape=input_shape, kernel_size=(3, 3),  #64
               activation='relu', name='conv_1'),
        Conv2D(filters=16, kernel_size=(3, 3), activation='relu', name='conv_2'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(4, 4), name='pool_1'),
        Flatten(name='flatten'),
        Dropout(0.3),
        Dense(units=32, activation='relu', name='dense_1'),
        Dropout(0.3),
        Dense(units=10, activation='softmax', name='dense_2')
    ])

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

In [56]:

```
model2 = get_cnn_model((32, 32, 1))
model2.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv_1 (Conv2D)              (None, 30, 30, 32)        320       
_________________________________________________________________
conv_2 (Conv2D)              (None, 28, 28, 16)        4624      
_________________________________________________________________
batch_normalization (BatchNo (None, 28, 28, 16)        64        
_________________________________________________________________
pool_1 (MaxPooling2D)        (None, 7, 7, 16)          0         
_________________________________________________________________
flatten (Flatten)            (None, 784)               0         
_________________________________________________________________
dropout (Dropout)            (None, 784)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 32)                25120     
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                330       
=================================================================
Total params: 30,458
Trainable params: 30,426
Non-trainable params: 32
_________________________________________________________________
```

In [57]:

```
cnn_checkpoint_path = 'cnn_model_best_checkpoints'
cnn_checkpoint = ModelCheckpoint(filepath= cnn_checkpoint_path,
                                 frequency = 'epoch',
                                 save_weights_only= True,
                                 save_best_only= True,
                                 verbose= 2)
cnn_earlystopping = EarlyStopping(patience= 2)
```

In [58]:

```
history2 = model2.fit(x = X_train_grayscaled_shaped,
                        y = y_train_ranged,
                        batch_size = 128,
                        epochs= 20,
                        validation_data = [X_test_grayscaled_shaped, y_test_ranged],
                        callbacks = [cnn_earlystopping, cnn_checkpoint]
            )
```

```
Train on 73257 samples, validate on 26032 samples
Epoch 1/20
73216/73257 [============================>.] - ETA: 0s - loss: 1.8418 - accuracy:
0.3577
Epoch 00001: val_loss improved from inf to 1.66589, saving model to cnn_model_best
_checkpoints
73257/73257 [==============================] - 565s 8ms/sample - loss: 1.8417 - ac
curacy: 0.3578 - val_loss: 1.6659 - val_accuracy: 0.4176
Epoch 2/20
73216/73257 [============================>.] - ETA: 0s - loss: 1.4479 - accuracy:
0.5088
Epoch 00002: val_loss improved from 1.66589 to 1.03856, saving model to cnn_model_
best_checkpoints
73257/73257 [==============================] - 547s 7ms/sample - loss: 1.4478 - ac
curacy: 0.5088 - val_loss: 1.0386 - val_accuracy: 0.6867
Epoch 3/20
73216/73257 [============================>.] - ETA: 0s - loss: 1.2248 - accuracy:
0.5927
Epoch 00003: val_loss improved from 1.03856 to 1.03418, saving model to cnn_model_
best_checkpoints
73257/73257 [==============================] - 550s 8ms/sample - loss: 1.2250 - ac
curacy: 0.5927 - val_loss: 1.0342 - val_accuracy: 0.7045
Epoch 4/20
73216/73257 [============================>.] - ETA: 0s - loss: 1.1675 - accuracy:
0.6147
Epoch 00004: val_loss improved from 1.03418 to 0.96377, saving model to cnn_model_
best_checkpoints
73257/73257 [==============================] - 568s 8ms/sample - loss: 1.1674 - ac
curacy: 0.6147 - val_loss: 0.9638 - val_accuracy: 0.7145
Epoch 5/20
73216/73257 [============================>.] - ETA: 0s - loss: 1.1202 - accuracy:
0.6321
Epoch 00005: val_loss improved from 0.96377 to 0.89389, saving model to cnn_model_
best_checkpoints
73257/73257 [==============================] - 575s 8ms/sample - loss: 1.1202 - ac
curacy: 0.6321 - val_loss: 0.8939 - val_accuracy: 0.7243
Epoch 6/20
73216/73257 [============================>.] - ETA: 0s - loss: 1.0796 - accuracy:
0.6479
Epoch 00006: val_loss did not improve from 0.89389
73257/73257 [==============================] - 572s 8ms/sample - loss: 1.0797 - ac
curacy: 0.6478 - val_loss: 0.9785 - val_accuracy: 0.6957
Epoch 7/20
73216/73257 [============================>.] - ETA: 0s - loss: 1.0485 - accuracy:
0.6601
Epoch 00007: val_loss improved from 0.89389 to 0.81599, saving model to cnn_model_
best_checkpoints
73257/73257 [==============================] - 586s 8ms/sample - loss: 1.0486 - ac
curacy: 0.6601 - val_loss: 0.8160 - val_accuracy: 0.7556
Epoch 8/20
73216/73257 [============================>.] - ETA: 0s - loss: 1.0093 - accuracy:
0.6723
Epoch 00008: val_loss improved from 0.81599 to 0.81314, saving model to cnn_model_
best_checkpoints
73257/73257 [==============================] - 579s 8ms/sample - loss: 1.0093 - ac
curacy: 0.6723 - val_loss: 0.8131 - val_accuracy: 0.7480
Epoch 9/20
73216/73257 [============================>.] - ETA: 0s - loss: 0.9700 - accuracy:
0.6847
Epoch 00009: val_loss improved from 0.81314 to 0.72090, saving model to cnn_model_
best_checkpoints
```

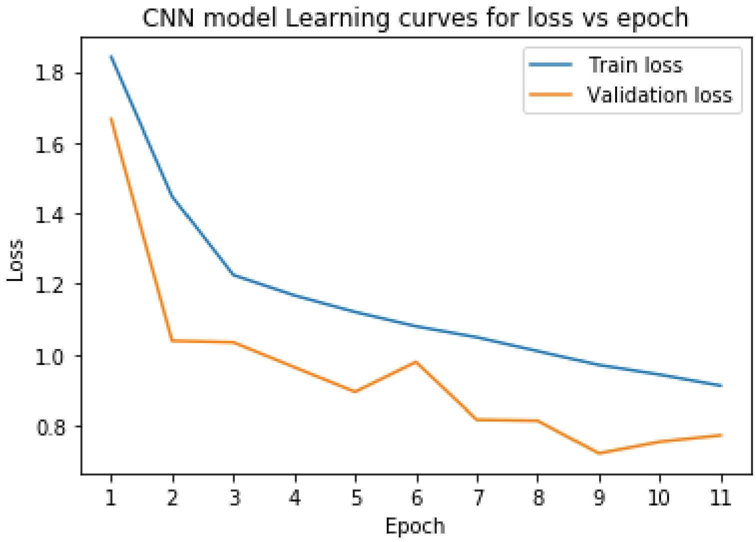
```
73257/73257 [==============================] - 566s 8ms/sample - loss: 0.9700 - ac
curacy: 0.6848 - val_loss: 0.7209 - val_accuracy: 0.7781
Epoch 10/20
73216/73257 [============================>.] - ETA: 0s - loss: 0.9427 - accuracy:
0.6977
Epoch 00010: val_loss did not improve from 0.72090
73257/73257 [==============================] - 571s 8ms/sample - loss: 0.9426 - ac
curacy: 0.6977 - val_loss: 0.7537 - val_accuracy: 0.7720
Epoch 11/20
73216/73257 [============================>.] - ETA: 0s - loss: 0.9115 - accuracy:
0.7056
Epoch 00011: val_loss did not improve from 0.72090
73257/73257 [==============================] - 567s 8ms/sample - loss: 0.9114 - ac
curacy: 0.7057 - val_loss: 0.7719 - val_accuracy: 0.7696
```

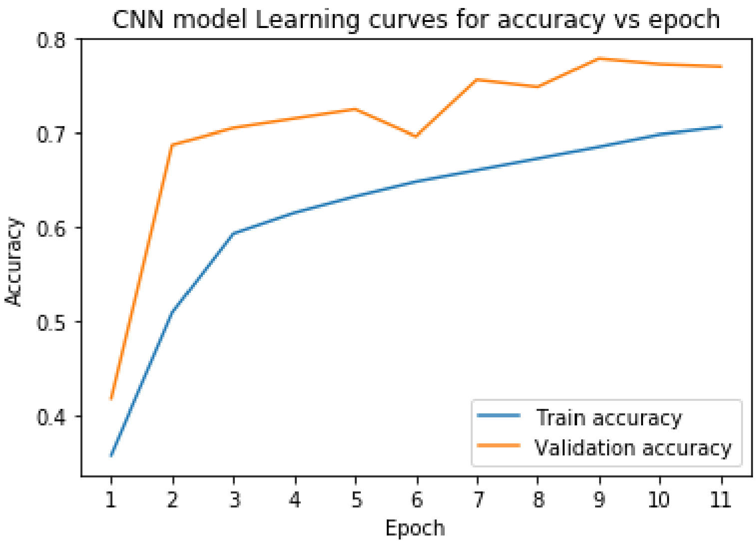
In [59]:

```
# Plot the learning curves for loss vs epoch and accuracy vs epoch for both training and validation sets.
plt.figure()
plt.plot( history2.history['loss'], label = 'Train loss')
plt.plot( history2.history['val_loss'], label = 'Validation loss')
plt.xticks(np.arange(0,len(history2.history['loss'])), np.arange(1,len(history2.history['loss'])+1) )
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('CNN model Learning curves for loss vs epoch')
plt.legend()


plt.figure()
plt.plot( history2.history['accuracy'], label = 'Train accuracy')
plt.plot( history2.history['val_accuracy'], label = 'Validation accuracy')
plt.xticks(np.arange(0,len(history2.history['accuracy'])), np.arange(1,len(history2.history['accuracy'])+1) )
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('CNN model Learning curves for accuracy vs epoch')
plt.legend()
```

Out[59]:

```
<matplotlib.legend.Legend at 0x7f86184414e0>
```





In [62]:

```
# Compute and display the loss and accuracy of the trained model on the test set.
# cnn model
get_test_loss_accuracy(model2, X_test_grayscaled_shaped, y_test_ranged)
```

```
Test accuracy: 0.770, Test loss: 0.772
```

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.

MLP model

In [13]:

```
# Load the best weights for the MLP and CNN models that you saved during the training run.
mlp_checkpoint_path = 'mlp_model_best_checkpoints'

best_mlp_model = get_mlp_model((32,32,1))
best_mlp_model.load_weights(mlp_checkpoint_path)
```

Out[13]:

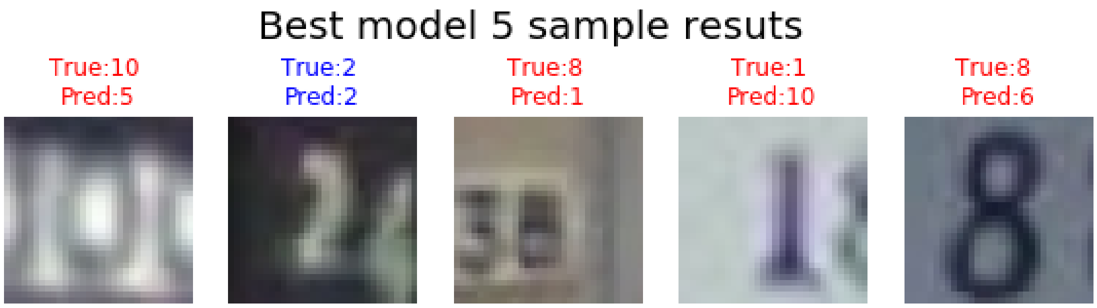
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f35dc6df0b8>
```

In [14]:

```
# get predict results
mlp_y_pred = best_mlp_model.predict(X_test_grayscaled_shaped)
mlp_y_pred_ranged = np.argmax(mlp_y_pred, axis =1) + 1  #rerange
```

In [58]:

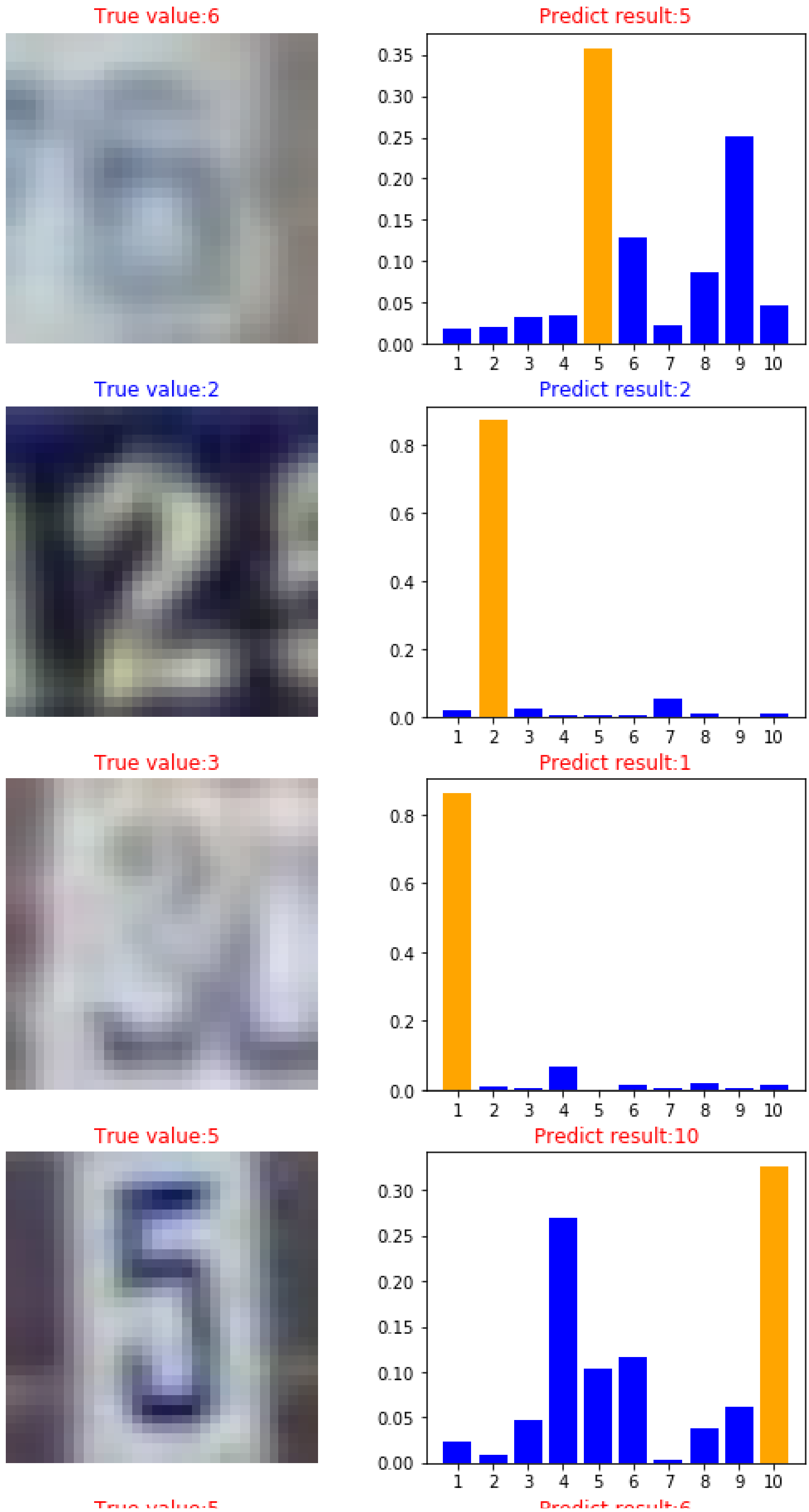
```
# Randomly select 5 images and corresponding labels from the test set and display the images wit
h their labels.
def predict_visualization(sample_number, y_true, y_pred):
    predict_list = np.random.randint(0, y_test.shape[0], sample_number)

    fig, ax = plt.subplots(1, sample_number, figsize=(sample_number*2, 3))
    for n,i in enumerate(predict_list):
        ax[n].set_axis_off()
        ax[n].imshow(X_test[:,:,:,i], 'gray')
        ax[n].set_title('True:{} ¥nPred:{}'.format(y_true[i][0], y_pred[n]),
                        color = 'red' if y_true[i][0] != y_pred[n] else 'blue' )
    fig.suptitle('Best model 5 sample resuts',fontsize = 20)

predict_visualization(5, y_test, mlp_y_pred_ranged )
```

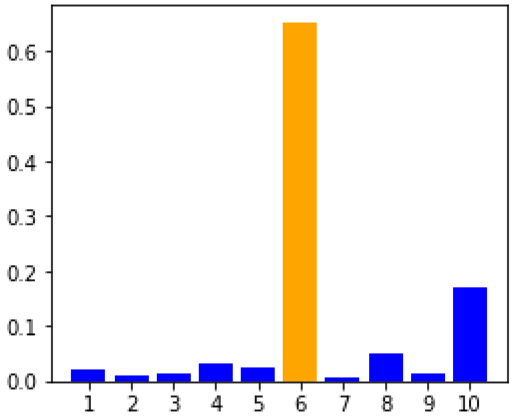


In [62]:

```
# 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.

def predict_visualization2(sample_number, y_true, y_pred, y_pred_prob):
    predict_list = np.random.randint(0, y_test.shape[0], sample_number)

    fig, ax = plt.subplots(sample_number,2, figsize=(2*4+1,sample_number*4))
    for n,i in enumerate(predict_list):
        ax[n][0].set_axis_off()
        ax[n][0].imshow(X_test[:,:,:,i], 'gray')
        ax[n][0].set_title('True value:{} '.format(y_true[i][0]),
                           color = 'red' if y_true[i][0] != y_pred[n] else 'blue' )

        ax[n][1].bar([i for i in range(1,11)],y_pred_prob[n],
                         color = ['orange' if i == np.argmax(y_pred_prob[n]) else 'blue' for i in
range(0,10)])
        ax[n][1].set_xticks([i for i in range(1,11)])
        ax[n][1].set_title('Predict result:{}'.format(y_pred[n]),
                           color = 'red' if y_true[i][0] != y_pred[n] else 'blue' )
        # if true = prediction, set title color as blue, if true != prediction, set title color
 as red


    fig.suptitle('5 sample visualization with predictive distribution',fontsize = 20)

predict_visualization2(5, y_test, mlp_y_pred_ranged, mlp_y_pred)
```

5 sample visualization with predictive distribution





CNN model

In [54]:

```
cnn_checkpoint_path = 'cnn_model_best_checkpoints'
best_cnn_model = get_cnn_model((32,32,1))
best_cnn_model.load_weights(cnn_checkpoint_path)
```

Out[54]:

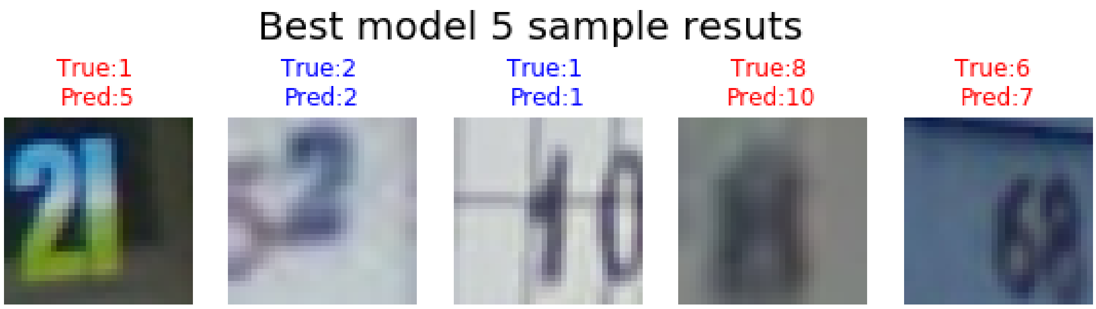
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f359c1b0f60>
```

In [55]:

```
cnn_y_pred = best_cnn_model.predict(X_test_grayscaled_shaped)
cnn_y_pred_ranged = np.argmax(cnn_y_pred, axis =1) + 1
```

In [59]:

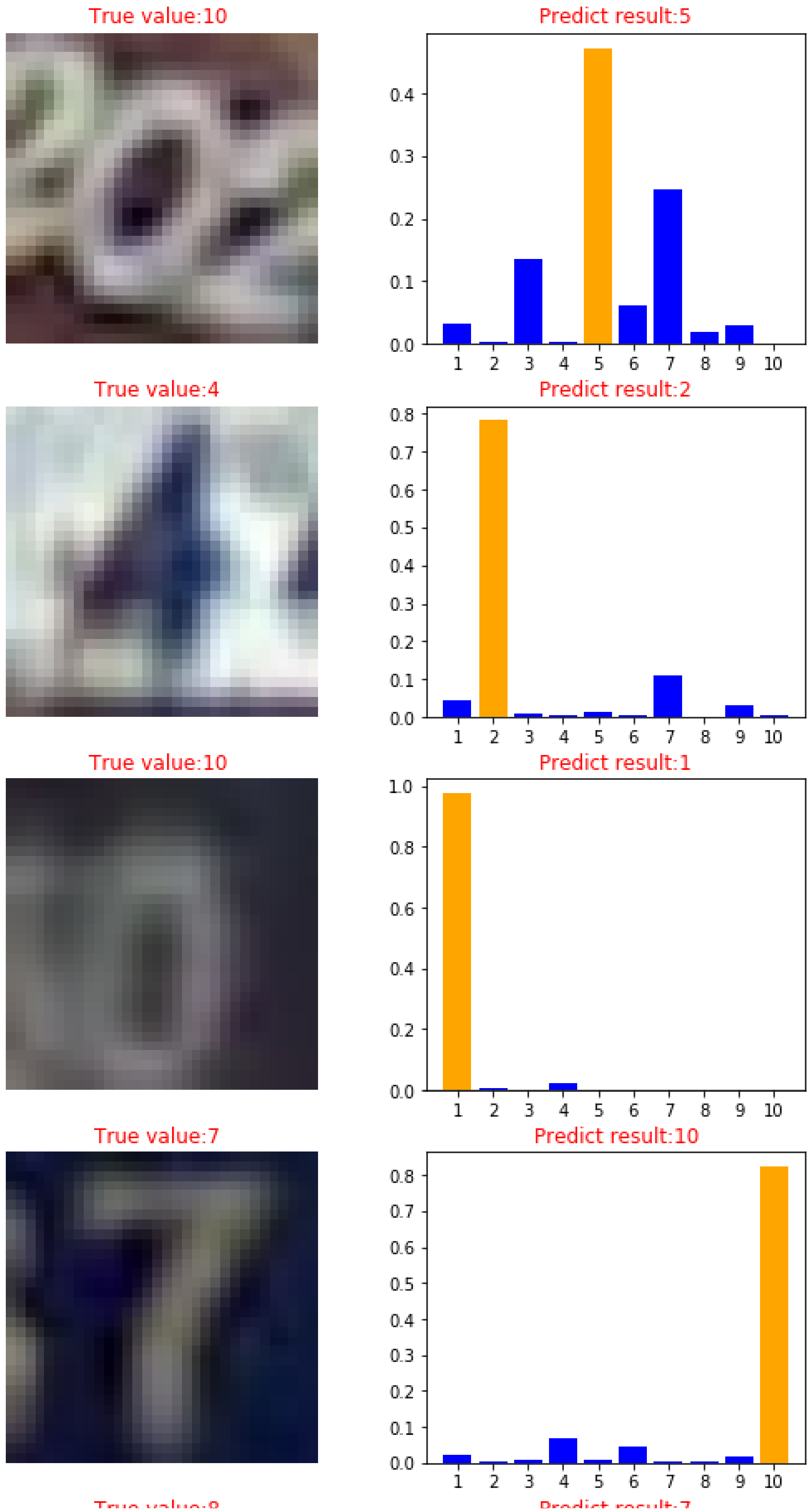
```
predict_visualization(5, y_test, cnn_y_pred_ranged )
```

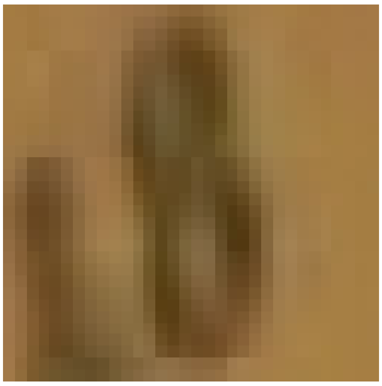


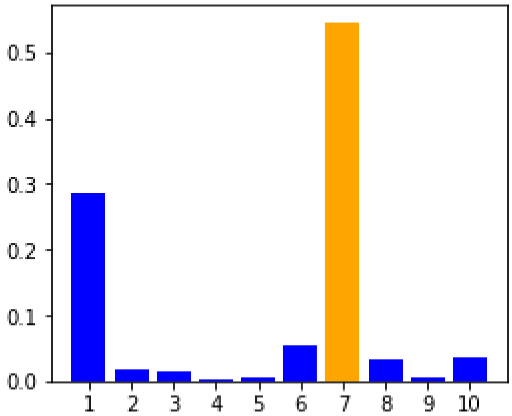
In [65]:

```
predict_visualization2(5, y_test, cnn_y_pred_ranged, cnn_y_pred)
```

5 sample visualization with predictive distribution






0.5
0.4
0.3
0.2
0.1
0.0
1 2 3 4 5 6 7 8 9 10