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 Dense, Flatten, Conv2D, MaxPooling2D, Dropout,BatchN
ormalization
from tensorflow.keras.callbacks import ModelCheckpoint,EarlyStopping
```

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

In [5]:

```
import matplotlib.pyplot as plt
import numpy as np
fig, axs = plt.subplots(nrows =4, ncols=5, figsize=(10,8), subplot_kw={'xticks': [], 'yt
icks':[]})
list_labels = []
for i in range(1, 21):
    rnd_num = np.random.randint(X_train.shape[3], size=1)
    img = X_train[:,:,:,rnd_num]
    fig.add_subplot(4, 5, i)
    plt.imshow(img.squeeze())
    plt.axis('off')
    list_labels.append(y_train[rnd_num])

for i, ax in enumerate(axs.reshape(-1)):
    title = str(list_labels[i]).strip("[]")
    ax.set_title(title)

plt.show()
```

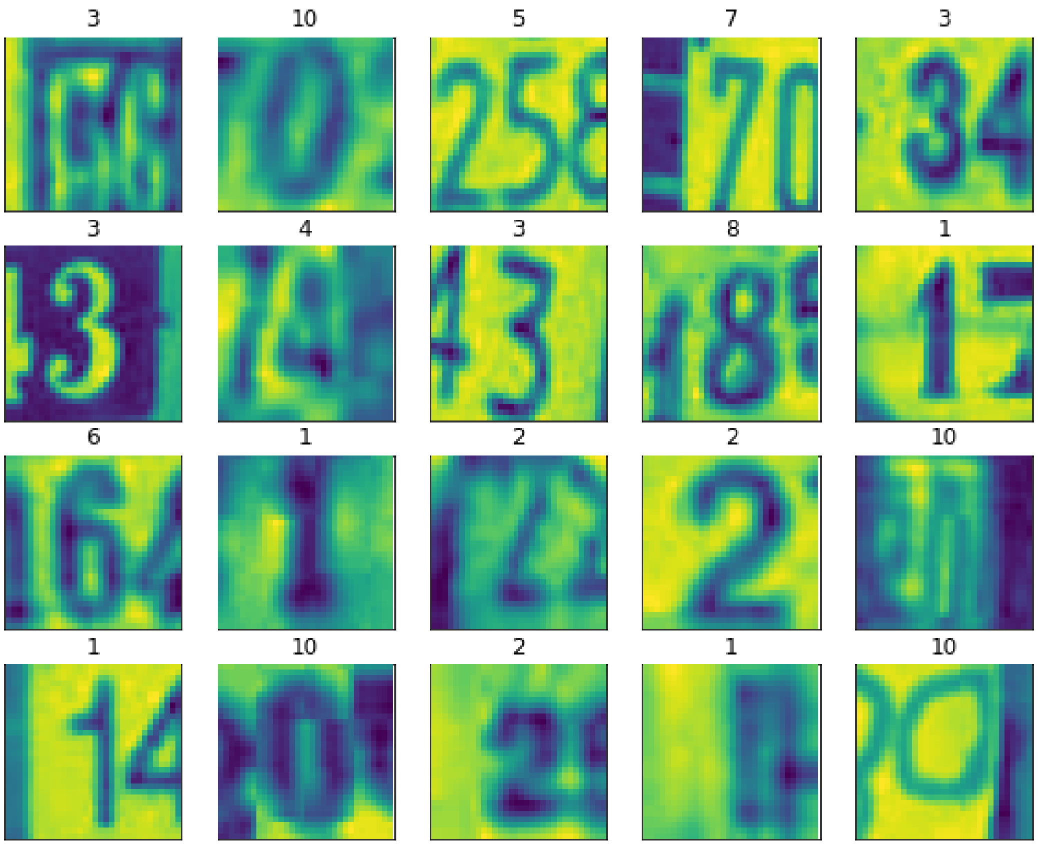
In [6]:

```
X_train = X_train.mean(axis=2, dtype="float32").reshape(32, 32, 1, -1)
X_test = X_test.mean(axis=2, dtype="float32").reshape(32, 32, 1, -1)
print(X_train.shape)
print(X_test.shape)
```

```
(32, 32, 1, 73257)
(32, 32, 1, 26032)
```

In [7]:

```
fig, axs = plt.subplots(nrows =4, ncols=5, figsize=(10,8), subplot_kw={'xticks': [], 'yt
icks':[]})
list_labels = []
for i in range(1, 21):
    rnd_num = np.random.randint(X_train.shape[3], size=1)
    img = X_train[:,:,:,rnd_num]
    fig.add_subplot(4, 5, i)
    plt.imshow(img.squeeze())
    plt.axis('off')
    list_labels.append(y_train[rnd_num])

for i, ax in enumerate(axs.reshape(-1)):
    title = str(list_labels[i]).strip("[]")
    ax.set_title(title)

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 [8]:

```
X_train = X_train.T.reshape(-1, 32, 32, 1)
X_train = np.swapaxes(X_train, axis1=1, axis2=2)
X_test = X_test.T.reshape(-1, 32, 32, 1)
X_test = np.swapaxes(X_test, axis1=1, axis2=2)

y_train = y_train.squeeze()
y_test = y_test.squeeze()
print(X_train.shape)
print(X_test.shape)
```

```
(73257, 32, 32, 1)
(26032, 32, 32, 1)
```

In [9]:

```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

In [10]:

```
list_labels = np.unique(y_train)
list_labels
```

Out[10]:

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10], dtype=uint8)
```

In [12]:

```
from tensorflow.keras.utils import to_categorical
y_train_truth_value = to_categorical(y_train - 1, num_classes=10)
y_test_truth_value = to_categorical(y_test - 1, num_classes=10)
```

In [14]:

```
w1 = tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.05)
model = Sequential([
    Flatten(input_shape=(32,32,1)),
    Dense(512, activation="relu", kernel_initializer=w1),
    Dense(256, activation="relu", kernel_initializer=w1),
    Dense(256, activation="relu", kernel_initializer=w1),
    Dense(10, activation="softmax")
])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 512)               524800
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_3 (Dense)              (None, 10)                2570
=================================================================
Total params: 724,490
Trainable params: 724,490
Non-trainable params: 0
_________________________________________________________________
```

In [25]:

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

In [26]:

```
checkpoint_best = ModelCheckpoint(filepath = 'model_checkpoints_best/checkpoint',
                                  save_weights_only=True,
                                  save_freq='epoch',
                                  monitor='loss',
                                  mode='min',
                                  save_best_only=True,
                                  verbose=1)
early = EarlyStopping(monitor='loss', patience=3, verbose=1)
```

In [16]:

```
history = model.fit(X_train, y_train_truth_value, batch_size=128, epochs=30, validation_
split=0.15, callbacks=[checkpoint_best, early])
```

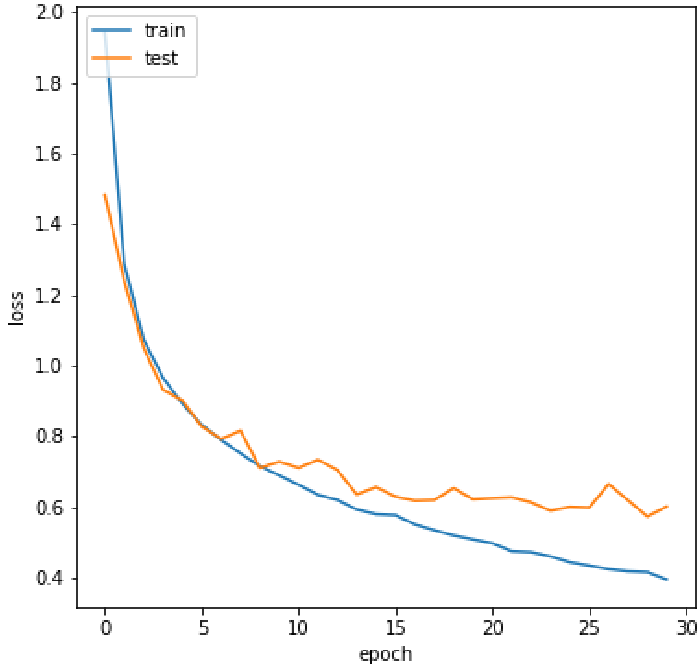
```
Train on 62268 samples, validate on 10989 samples
Epoch 1/30
62208/62268 [============================>.] - ETA: 0s - loss: 1.9417 - accuracy: 0.3058
ETA: 1s - loss: 1.9553
Epoch 00001: loss improved from inf to 1.94128, saving model to model_checkpoints_best/ch
eckpoint
62268/62268 [==============================] - 38s 618us/sample - loss: 1.9413 - accuracy
: 0.3060 - val_loss: 1.4821 - val_accuracy: 0.4977
Epoch 2/30
62080/62268 [============================>.] - ETA: 0s - loss: 1.2895 - accuracy: 0.5783
Epoch 00002: loss improved from 1.94128 to 1.28850, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 36s 579us/sample - loss: 1.2885 - accuracy
: 0.5786 - val_loss: 1.2391 - val_accuracy: 0.5981
Epoch 3/30
62080/62268 [============================>.] - ETA: 0s - loss: 1.0743 - accuracy: 0.6601
Epoch 00003: loss improved from 1.28850 to 1.07424, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 36s 575us/sample - loss: 1.0742 - accuracy
: 0.6601 - val_loss: 1.0500 - val_accuracy: 0.6639
Epoch 4/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.9654 - accuracy: 0.6984
Epoch 00004: loss improved from 1.07424 to 0.96564, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 36s 575us/sample - loss: 0.9656 - accuracy
: 0.6984 - val_loss: 0.9319 - val_accuracy: 0.7063
Epoch 5/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.8921 - accuracy: 0.7223
Epoch 00005: loss improved from 0.96564 to 0.89224, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 36s 573us/sample - loss: 0.8922 - accuracy
: 0.7223 - val_loss: 0.9014 - val_accuracy: 0.7155
Epoch 6/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.8322 - accuracy: 0.7413
Epoch 00006: loss improved from 0.89224 to 0.83236, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 35s 568us/sample - loss: 0.8324 - accuracy
: 0.7413 - val_loss: 0.8277 - val_accuracy: 0.7448
Epoch 7/30
62080/62268 [============================>.] - ETA: 0s - loss: 0.7903 - accuracy: 0.7539
Epoch 00007: loss improved from 0.83236 to 0.79002, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 36s 576us/sample - loss: 0.7900 - accuracy
: 0.7540 - val_loss: 0.7928 - val_accuracy: 0.7512
Epoch 8/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.7522 - accuracy: 0.7647
Epoch 00008: loss improved from 0.79002 to 0.75207, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 36s 574us/sample - loss: 0.7521 - accuracy
: 0.7647 - val_loss: 0.8161 - val_accuracy: 0.7484
Epoch 9/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.7162 - accuracy: 0.7769
Epoch 00009: loss improved from 0.75207 to 0.71611, saving model to model_checkpoints_bes
```

```
Epoch 00009: loss improved from 0.73207 to 0.71611, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 35s 569us/sample - loss: 0.7161 - accuracy
: 0.7769 - val_loss: 0.7110 - val_accuracy: 0.7772
Epoch 10/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.6897 - accuracy: 0.7845
Epoch 00010: loss improved from 0.71611 to 0.68981, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 35s 569us/sample - loss: 0.6898 - accuracy
: 0.7845 - val_loss: 0.7288 - val_accuracy: 0.7751
Epoch 11/30
62080/62268 [============================>.] - ETA: 0s - loss: 0.6627 - accuracy: 0.7923
Epoch 00011: loss improved from 0.68981 to 0.66303, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 35s 570us/sample - loss: 0.6630 - accuracy
: 0.7923 - val_loss: 0.7110 - val_accuracy: 0.7758
Epoch 12/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.6349 - accuracy: 0.8008
Epoch 00012: loss improved from 0.66303 to 0.63511, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 35s 568us/sample - loss: 0.6351 - accuracy
: 0.8008 - val_loss: 0.7343 - val_accuracy: 0.7642
Epoch 13/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.6204 - accuracy: 0.8053
Epoch 00013: loss improved from 0.63511 to 0.62040, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 36s 574us/sample - loss: 0.6204 - accuracy
: 0.8053 - val_loss: 0.7048 - val_accuracy: 0.7730
Epoch 14/30
62080/62268 [============================>.] - ETA: 0s - loss: 0.5935 - accuracy: 0.8140
Epoch 00014: loss improved from 0.62040 to 0.59347, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 36s 574us/sample - loss: 0.5935 - accuracy
: 0.8140 - val_loss: 0.6360 - val_accuracy: 0.8057
Epoch 15/30
62080/62268 [============================>.] - ETA: 0s - loss: 0.5806 - accuracy: 0.8156
Epoch 00015: loss improved from 0.59347 to 0.58028, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 35s 569us/sample - loss: 0.5803 - accuracy
: 0.8158 - val_loss: 0.6564 - val_accuracy: 0.7963
Epoch 16/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.5777 - accuracy: 0.8163
Epoch 00016: loss improved from 0.58028 to 0.57765, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 36s 576us/sample - loss: 0.5776 - accuracy
: 0.8163 - val_loss: 0.6298 - val_accuracy: 0.8067
Epoch 17/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.5512 - accuracy: 0.8247
Epoch 00017: loss improved from 0.57765 to 0.55123, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 35s 570us/sample - loss: 0.5512 - accuracy
: 0.8247 - val_loss: 0.6186 - val_accuracy: 0.8097
Epoch 18/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.5348 - accuracy: 0.8306
Epoch 00018: loss improved from 0.55123 to 0.53498, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 35s 569us/sample - loss: 0.5350 - accuracy
: 0.8306 - val_loss: 0.6201 - val_accuracy: 0.8118
Epoch 19/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.5200 - accuracy: 0.8342
Epoch 00019: loss improved from 0.53498 to 0.51997, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 36s 572us/sample - loss: 0.5200 - accuracy
: 0.8342 - val_loss: 0.6536 - val_accuracy: 0.7989
Epoch 20/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.5085 - accuracy: 0.8367
ETA: 0s - loss: 0.5082 - accuracy: 0.
Epoch 00020: loss improved from 0.51997 to 0.50875, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 36s 570us/sample - loss: 0.5087 - accuracy
: 0.8367 - val_loss: 0.6224 - val_accuracy: 0.8072
Epoch 21/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.4978 - accuracy: 0.8405
```

```
62208/62268 [============================>.] - ETA: 0s - loss: 0.4978 - accuracy: 0.8405
Epoch 00021: loss improved from 0.50875 to 0.49771, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 35s 569us/sample - loss: 0.4977 - accuracy
: 0.8406 - val_loss: 0.6256 - val_accuracy: 0.8067
Epoch 22/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.4757 - accuracy: 0.8473
Epoch 00022: loss improved from 0.49771 to 0.47544, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 36s 574us/sample - loss: 0.4754 - accuracy
: 0.8474 - val_loss: 0.6280 - val_accuracy: 0.8094
Epoch 23/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.4730 - accuracy: 0.8496
Epoch 00023: loss improved from 0.47544 to 0.47293, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 36s 575us/sample - loss: 0.4729 - accuracy
: 0.8497 - val_loss: 0.6136 - val_accuracy: 0.8187
Epoch 24/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.4607 - accuracy: 0.8535
ETA: 0s - loss: 0.4615
Epoch 00024: loss improved from 0.47293 to 0.46080, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 36s 571us/sample - loss: 0.4608 - accuracy
: 0.8535 - val_loss: 0.5900 - val_accuracy: 0.8226
Epoch 25/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.4446 - accuracy: 0.8580
ETA: 0s - loss: 0.4448 - accuracy: 0.85
Epoch 00025: loss improved from 0.46080 to 0.44449, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 36s 572us/sample - loss: 0.4445 - accuracy
: 0.8580 - val_loss: 0.6006 - val_accuracy: 0.8220
Epoch 26/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.4351 - accuracy: 0.8611
Epoch 00026: loss improved from 0.44449 to 0.43495, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 36s 571us/sample - loss: 0.4349 - accuracy
: 0.8612 - val_loss: 0.5984 - val_accuracy: 0.8224
Epoch 27/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.4245 - accuracy: 0.8629
Epoch 00027: loss improved from 0.43495 to 0.42474, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 35s 564us/sample - loss: 0.4247 - accuracy
: 0.8628 - val_loss: 0.6652 - val_accuracy: 0.8023
Epoch 28/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.4187 - accuracy: 0.8664
Epoch 00028: loss improved from 0.42474 to 0.41875, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 35s 565us/sample - loss: 0.4188 - accuracy
: 0.8664 - val_loss: 0.6199 - val_accuracy: 0.8146
Epoch 29/30
62080/62268 [============================>.] - ETA: 0s - loss: 0.4166 - accuracy: 0.8661
Epoch 00029: loss improved from 0.41875 to 0.41651, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 35s 562us/sample - loss: 0.4165 - accuracy
: 0.8661 - val_loss: 0.5737 - val_accuracy: 0.8320
Epoch 30/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.3955 - accuracy: 0.8730
Epoch 00030: loss improved from 0.41651 to 0.39560, saving model to model_checkpoints_bes
t/checkpoint
62268/62268 [==============================] - 35s 557us/sample - loss: 0.3956 - accuracy
: 0.8730 - val_loss: 0.6015 - val_accuracy: 0.8244
```

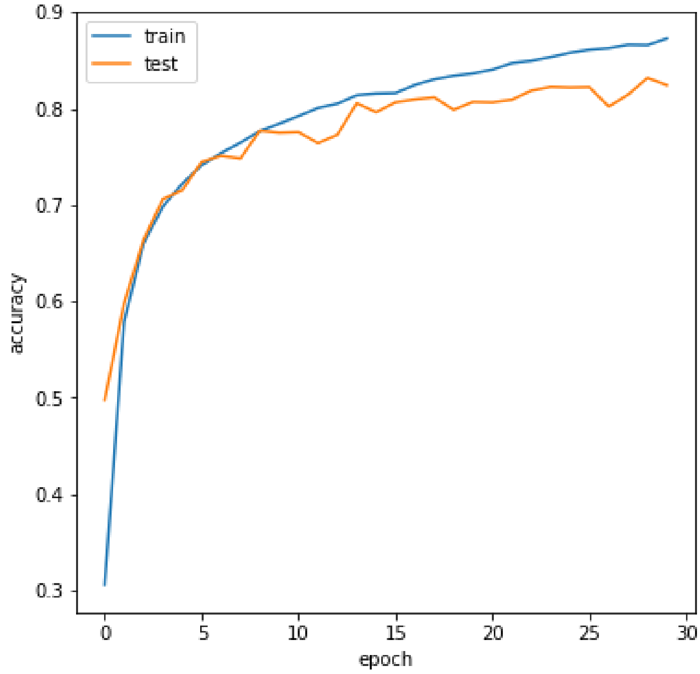
In [17]:

```
fig, axs = plt.subplots(figsize=(6,6))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'test'], loc="upper left")
plt.show()
```



In [18]:

```
fig, axs = plt.subplots(figsize=(6,6))
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.legend(['train', 'test'], loc="upper left")
plt.show()
```



In [30]:

```
test_loss, test_accuracy = model.evaluate(X_test, y_test_truth_value, verbose=False)
print("Loss on test samples:", np.around(test_loss, 2))
print("Accuracy on test samples:", np.around(test_accuracy*100,2), '%')
```

```
Loss on test samples: 0.74
Accuracy on test samples: 80.2 %
```

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]:

```
CNN = Sequential([
        Conv2D(filters=16, input_shape=(32,32,1), kernel_size=(3, 3),
               activation='relu',padding='SAME', name='conv_1'),
        MaxPooling2D(pool_size=(3, 3), name='pool_1'),
        Conv2D(filters=8, kernel_size=(3, 3), activation='relu',padding='SAME', name='co
nv_2'),
        BatchNormalization(),
        Flatten(name='flatten'),
        Dropout(0.3),
        Dense(units=128, activation='relu', kernel_initializer=w1, name='dense_1'),
        Dense(units=128, activation='relu',kernel_initializer=w1, name='dense_2'),
        Dense(units=10, activation='softmax', name='dense_3')
    ])
CNN.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv_1 (Conv2D)              (None, 32, 32, 16)        160       
_________________________________________________________________
pool_1 (MaxPooling2D)        (None, 10, 10, 16)        0         
_________________________________________________________________
conv_2 (Conv2D)              (None, 10, 10, 8)         1160      
_________________________________________________________________
batch_normalization_1 (Batch (None, 10, 10, 8)         32        
_________________________________________________________________
flatten (Flatten)            (None, 800)               0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 800)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 128)               102528    
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512     
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290      
=================================================================
Total params: 121,682
Trainable params: 121,666
Non-trainable params: 16
_________________________________________________________________
```

In [16]:

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

In [17]:

```
checkpoint_best_CNN = ModelCheckpoint(filepath = 'model_checkpoints_best_CNN/checkpoint',
                                      save_weights_only=True,
                                      save_freq='epoch',
                                      monitor='val_accuracy',
```

```
                                         mode='max',
                                         save_best_only=True,
                                         verbose=1)
early_CNN = EarlyStopping(monitor='val_accuracy', patience=3, verbose=1)
```

In [18]:

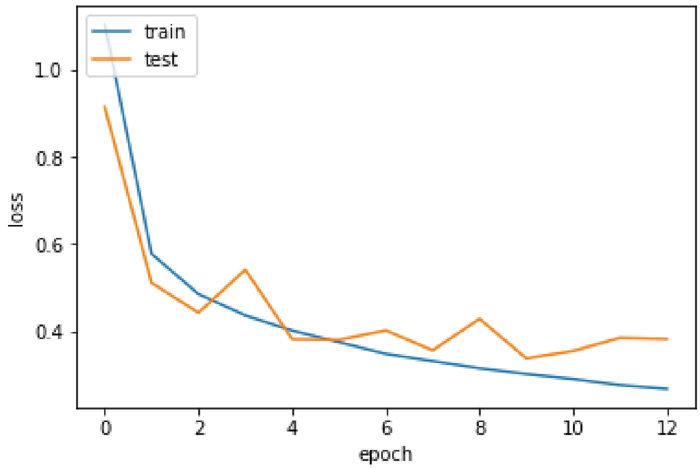
```
history_CNN = CNN.fit(X_train, y_train_truth_value, batch_size=128, epochs=30, validatio
n_split=0.15, callbacks=[checkpoint_best_CNN, early_CNN])
```

```
Train on 62268 samples, validate on 10989 samples
Epoch 1/30
62208/62268 [============================>.] - ETA: 0s - loss: 1.1054 - accuracy: 0.6344
Epoch 00001: val_accuracy improved from -inf to 0.80599, saving model to model_checkpoint
s_best_CNN/checkpoint
62268/62268 [==============================] - 124s 2ms/sample - loss: 1.1048 - accuracy:
0.6346 - val_loss: 0.9149 - val_accuracy: 0.8060
Epoch 2/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.5780 - accuracy: 0.8189
Epoch 00002: val_accuracy improved from 0.80599 to 0.84039, saving model to model_checkpo
ints_best_CNN/checkpoint
62268/62268 [==============================] - 123s 2ms/sample - loss: 0.5778 - accuracy:
0.8189 - val_loss: 0.5111 - val_accuracy: 0.8404
Epoch 3/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.4857 - accuracy: 0.8467
Epoch 00003: val_accuracy improved from 0.84039 to 0.86514, saving model to model_checkpo
ints_best_CNN/checkpoint
62268/62268 [==============================] - 123s 2ms/sample - loss: 0.4855 - accuracy:
0.8467 - val_loss: 0.4425 - val_accuracy: 0.8651
Epoch 4/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.4363 - accuracy: 0.8640
Epoch 00004: val_accuracy did not improve from 0.86514
62268/62268 [==============================] - 120s 2ms/sample - loss: 0.4364 - accuracy:
0.8640 - val_loss: 0.5412 - val_accuracy: 0.8432
Epoch 5/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.4016 - accuracy: 0.8751
Epoch 00005: val_accuracy improved from 0.86514 to 0.88452, saving model to model_checkpo
ints_best_CNN/checkpoint
62268/62268 [==============================] - 120s 2ms/sample - loss: 0.4016 - accuracy:
0.8751 - val_loss: 0.3819 - val_accuracy: 0.8845
Epoch 6/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.3751 - accuracy: 0.8828
Epoch 00006: val_accuracy improved from 0.88452 to 0.88716, saving model to model_checkpo
ints_best_CNN/checkpoint
62268/62268 [==============================] - 121s 2ms/sample - loss: 0.3750 - accuracy:
0.8828 - val_loss: 0.3804 - val_accuracy: 0.8872
Epoch 7/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.3481 - accuracy: 0.8924
Epoch 00007: val_accuracy did not improve from 0.88716
62268/62268 [==============================] - 121s 2ms/sample - loss: 0.3481 - accuracy:
0.8924 - val_loss: 0.4019 - val_accuracy: 0.8819
Epoch 8/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.3315 - accuracy: 0.8962
Epoch 00008: val_accuracy improved from 0.88716 to 0.89753, saving model to model_checkpo
ints_best_CNN/checkpoint
62268/62268 [==============================] - 121s 2ms/sample - loss: 0.3314 - accuracy:
0.8962 - val_loss: 0.3561 - val_accuracy: 0.8975
Epoch 9/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.3146 - accuracy: 0.9012
Epoch 00009: val_accuracy did not improve from 0.89753
62268/62268 [==============================] - 121s 2ms/sample - loss: 0.3148 - accuracy:
0.9012 - val_loss: 0.4286 - val_accuracy: 0.8817
Epoch 10/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.3019 - accuracy: 0.9067
Epoch 00010: val_accuracy improved from 0.89753 to 0.90172, saving model to model_checkpo
ints_best_CNN/checkpoint
62268/62268 [==============================] - 120s 2ms/sample - loss: 0.3018 - accuracy:
0.9067 - val_loss: 0.3377 - val_accuracy: 0.9017
Epoch 11/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.2904 - accuracy: 0.9089
Epoch 00011: val_accuracy did not improve from 0.90172
62268/62268 [==============================] - 121s 2ms/sample - loss: 0.2903 - accuracy:
```

```
0.9089 - val_loss: 0.3548 - val_accuracy: 0.8957
Epoch 12/30
62208/62268 [============================>.] - ETA: 0s - loss: 0.2677 - accuracy: 0.9158
Epoch 00013: val_accuracy did not improve from 0.90172
62268/62268 [==============================] - 121s 2ms/sample - loss: 0.2678 - accuracy:
0.9157 - val_loss: 0.3822 - val_accuracy: 0.8925
Epoch 00013: early stopping
```

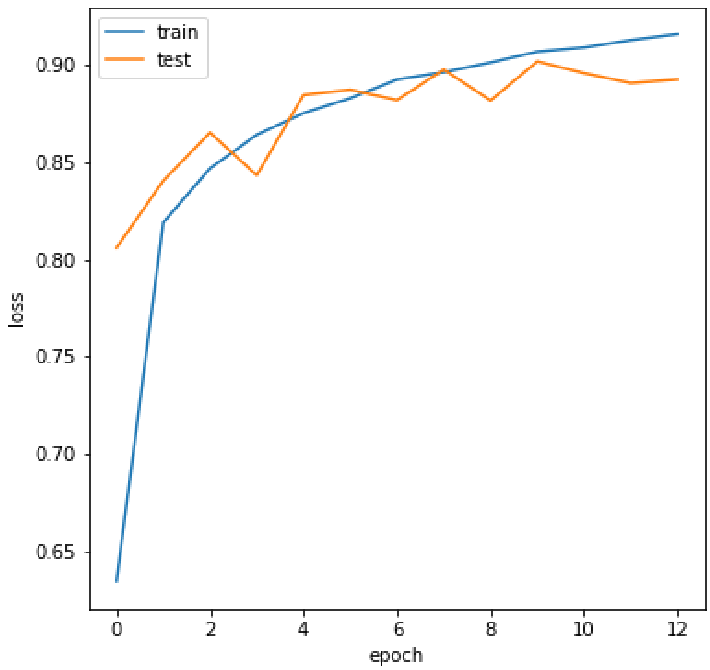
In [19]:

```
fig, axs = plt.subplots()
plt.plot(history_CNN.history['loss'])
plt.plot(history_CNN.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'test'], loc="upper left")
plt.show()
```



In [20]:

```
fig, axs = plt.subplots(figsize=(6,6))
plt.plot(history_CNN.history['accuracy'])
plt.plot(history_CNN.history['val_accuracy'])
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'test'], loc="upper left")
plt.show()
```



In [22]:

```
test_loss_CNN, test_accuracy_CNN = CNN.evaluate(X_test, y_test_truth_value, verbose=False
```

```
)
print("Loss on test samples:", np.around(test_loss_CNN, 2))
print("Accuracy on test samples:", np.around(test_accuracy_CNN*100,2), '%')
```

```
Loss on test samples: 0.42
Accuracy on test samples: 88.4 %
```

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 [23]:

```
model.load_weights('model_checkpoints_best/checkpoint')
CNN.load_weights('model_checkpoints_best_CNN/checkpoint')
```

Out[23]:

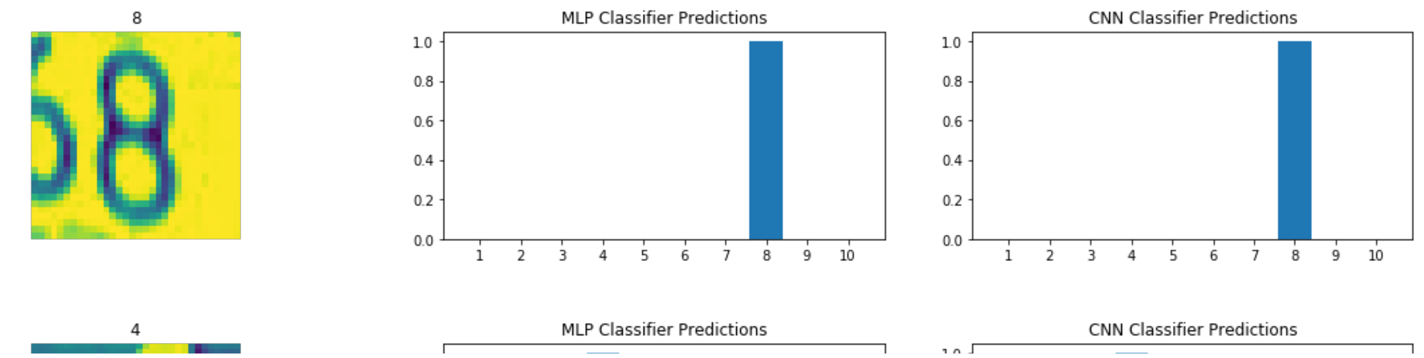
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7fbe4b9fc3c8>
```

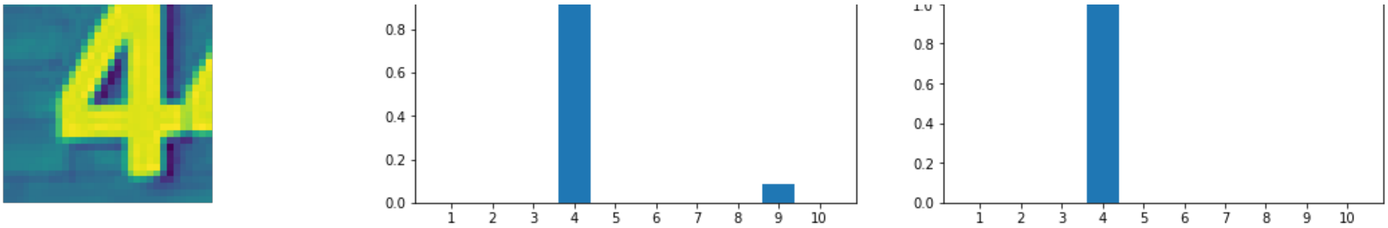
In [26]:

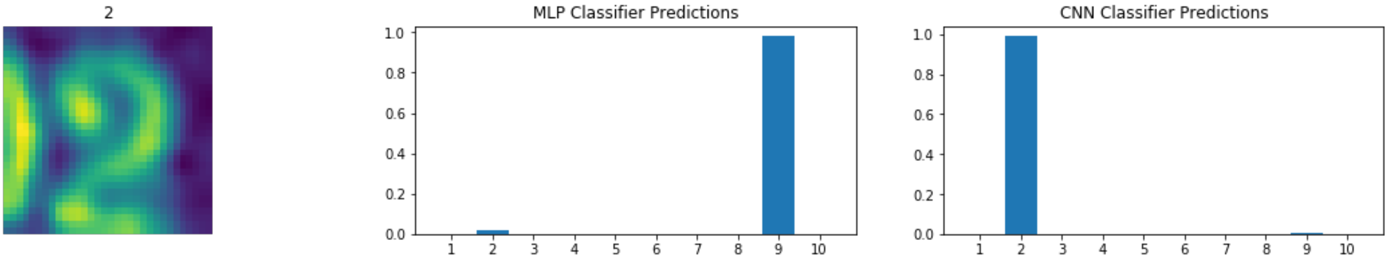
```
images = []
labels = []
predictions_MLP = []
predictions_CNN = []

for i in range(0, 5):
    rnd_num = np.random.randint(X_test.shape[0], size=1)
    img = X_test[rnd_num, :,:,:]
    images.append(img)
    labels.append(str(y_test[rnd_num]).strip('[]'))
    predictions_MLP.append(model.predict(img))
    predictions_CNN.append(CNN.predict(img))
```

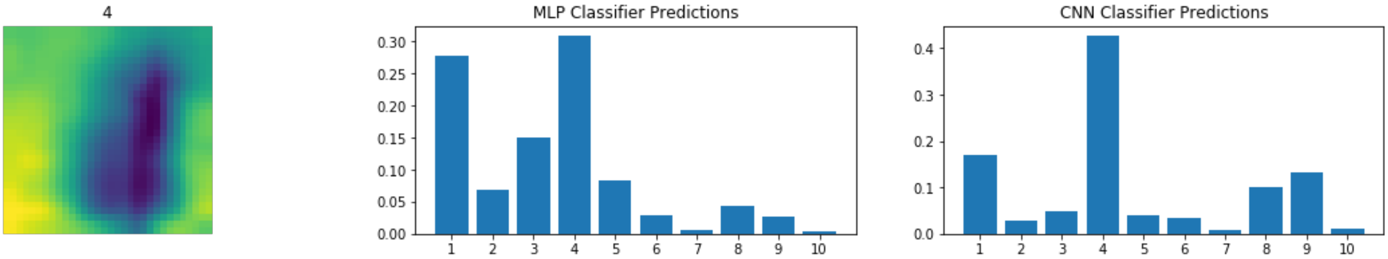
In [27]:

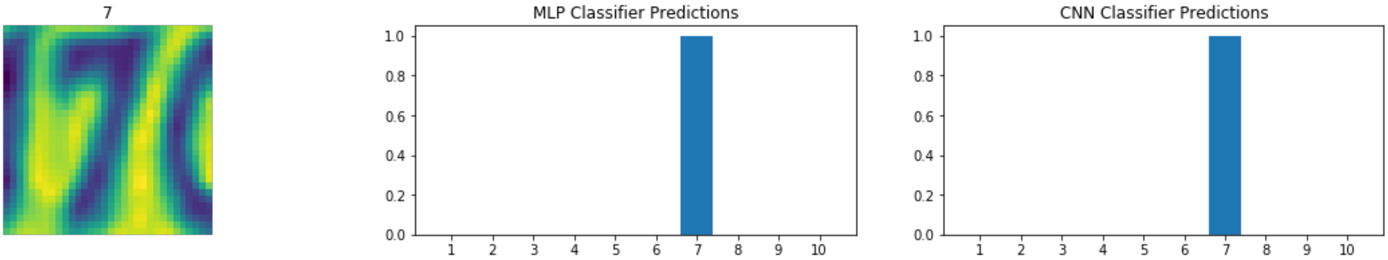
```
fig, axs = plt.subplots(nrows=5, ncols=3, figsize=(20,20))
fig.subplots_adjust(hspace=0.5, wspace=0.2)

x = np.arange(1,11)

for i, (image, label, mlp_prediction, cnn_prediction) in enumerate(zip(images, labels, p
redictions_MLP, predictions_CNN)):
    axs[i, 0].imshow(np.squeeze(image))
    axs[i, 0].set_title(label)
    axs[i, 0].axis('off')
    axs[i, 1].bar(x, mlp_prediction.reshape(10,))
    axs[i, 1].set_title('MLP Classifier Predictions')
    axs[i, 1].set_xticks(x)
    axs[i, 2].bar(x, cnn_prediction.reshape(10,))
    axs[i, 2].set_title('CNN Classifier Predictions')
    axs[i, 2].set_xticks(x)
```






2
MLP Classifier Predictions
CNN Classifier Predictions


4
MLP Classifier Predictions
CNN Classifier Predictions


7
MLP Classifier Predictions
CNN Classifier Predictions

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```