Capstone Project course 1

October 3, 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.

[15]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping,␣
↪ReduceLROnPlateau, Callback
from tensorflow.keras.layers import MaxPooling2D, Conv2D, Dense, Flatten,␣
↪Dropout, BatchNormalization
from tensorflow.keras.losses import SparseCategoricalCrossentropy
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.initializers import he_uniform, zeros
from tensorflow.keras.regularizers import l2
import seaborn as sns
import pandas as pd
```

```
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt
%matplotlib inline
```

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.

[16]:
```
# 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.

1.2 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.

[17]:
```
x_train = np.moveaxis(train['X'], -1, 0) # use moveaxis to reshape matlab array
y_train = np.array(train['y'])
x_test = np.moveaxis(test['X'], -1, 0)
y_test = np.array(test['y'])
y_train[y_train[:,0] == 10] = 0
y_test[y_test[:,0] == 10] = 0

x_train = x_train/255. # normalizing
x_test = x_test/255.
```

[18]:
```
n1 = np.random.randint(1000) # random start position for 10 photos
fig, ax = plt.subplots(1, 10, figsize=(10, 1))
for i in range(10):
    ax[i].set_axis_off()
    ax[i].imshow(x_train[n1 * i])
```

[19]:
```
x_train = x_train.mean(axis=3, keepdims=True) # make images gray
x_test = x_test.mean(axis=3, keepdims=True)
```

[21]:
```
fig, ax = plt.subplots(1, 10, figsize=(10, 1))
for i in range(10):
    image = x_train[n1 * i].squeeze()
    ax[i].set_axis_off()
    ax[i].imshow(image, cmap='Greys')
```

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.

[22]:
```
# callbacks

def get_checkpoint_callback(name):
    checkpoint_path = f'checkpoint_{name}/best'
    checkpoint = ModelCheckpoint(checkpoint_path, frequency='epoch',␣
↪save_best_only=True)
    return checkpoint

def get_early_stoping():
    return EarlyStopping(monitor='val_loss', patience=10, verbose=0)

def reduce_lr(coef=0.5):
    return ReduceLROnPlateau(monitor='loss', factor=coef)
```

[23]:
```
def get_MLP_model(input_shape, wd):
    model = Sequential([
        Flatten(input_shape=input_shape),
        Dense(512, kernel_regularizer=l2(wd), activation='relu'),
        Dense(256, kernel_regularizer=l2(wd), activation='relu'),
        Dense(128, kernel_regularizer=l2(wd), activation='relu'),
        Dense(64, kernel_regularizer=l2(wd), activation='relu'),
        Dense(10, kernel_regularizer=l2(wd), activation='softmax')
    ])
    model.compile(optimizer=Adam(0.0001),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

[24]:
```
with tf.device('GPU:0'):
    model = get_MLP_model(x_train[1].shape, 1e-5)
    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, 128)               32896
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_4 (Dense)              (None, 10)                650
=================================================================
Total params: 697,930
Trainable params: 697,930
Non-trainable params: 0
_________________________________________________________________
```

[25]:
```
with tf.device('GPU:0'):
    history_mlp = model.fit(x_train,
                            y_train,
                            epochs=20,
                            batch_size=16,
                            validation_split=.15,
                            callbacks=[get_early_stoping(),
                                       get_checkpoint_callback('mlp'),
                                       reduce_lr()])
```

```
Epoch 1/20
3875/3892 [============================>.] - ETA: 0s - loss: 1.8505 - accuracy:
0.3585WARNING:tensorflow:From
c:\users\79689\appdata\local\programs\python\python37\lib\site-
packages\tensorflow\python\training\tracking\tracking.py:111:
Model.state_updates (from tensorflow.python.keras.engine.training) is deprecated
and will be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied
automatically.
WARNING:tensorflow:From
c:\users\79689\appdata\local\programs\python\python37\lib\site-
packages\tensorflow\python\training\tracking\tracking.py:111: Layer.updates
(from tensorflow.python.keras.engine.base_layer) is deprecated and will be
removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied
automatically.
INFO:tensorflow:Assets written to: checkpoint_mlp\best\assets
3892/3892 [==============================] - 12s 3ms/step - loss: 1.8483 -
accuracy: 0.3595 - val_loss: 1.5321 - val_accuracy: 0.4917
Epoch 2/20
```

```
3882/3892 [============================>.] - ETA: 0s - loss: 1.3220 - accuracy:
0.5780INFO:tensorflow:Assets written to: checkpoint_mlp\best\assets
3892/3892 [==============================] - 12s 3ms/step - loss: 1.3218 -
accuracy: 0.5781 - val_loss: 1.1998 - val_accuracy: 0.6326
Epoch 3/20
3882/3892 [============================>.] - ETA: 0s - loss: 1.1334 - accuracy:
0.6485INFO:tensorflow:Assets written to: checkpoint_mlp\best\assets
3892/3892 [==============================] - 12s 3ms/step - loss: 1.1331 -
accuracy: 0.6486 - val_loss: 1.1470 - val_accuracy: 0.6466
Epoch 4/20
3888/3892 [============================>.] - ETA: 0s - loss: 1.0235 - accuracy:
0.6870INFO:tensorflow:Assets written to: checkpoint_mlp\best\assets
3892/3892 [==============================] - 11s 3ms/step - loss: 1.0233 -
accuracy: 0.6870 - val_loss: 0.9669 - val_accuracy: 0.6999
Epoch 5/20
3892/3892 [==============================] - 10s 3ms/step - loss: 0.9492 -
accuracy: 0.7106 - val_loss: 0.9811 - val_accuracy: 0.6943
Epoch 6/20
3892/3892 [==============================] - 10s 3ms/step - loss: 0.8902 -
accuracy: 0.7305 - val_loss: 0.9687 - val_accuracy: 0.6985
Epoch 7/20
3884/3892 [============================>.] - ETA: 0s - loss: 0.8413 - accuracy:
0.7436INFO:tensorflow:Assets written to: checkpoint_mlp\best\assets
3892/3892 [==============================] - 11s 3ms/step - loss: 0.8412 -
accuracy: 0.7436 - val_loss: 0.8199 - val_accuracy: 0.7526
Epoch 8/20
3886/3892 [============================>.] - ETA: 0s - loss: 0.8042 - accuracy:
0.7550INFO:tensorflow:Assets written to: checkpoint_mlp\best\assets
3892/3892 [==============================] - 12s 3ms/step - loss: 0.8041 -
accuracy: 0.7550 - val_loss: 0.8016 - val_accuracy: 0.7548
Epoch 9/20
3872/3892 [============================>.] - ETA: 0s - loss: 0.7701 - accuracy:
0.7680INFO:tensorflow:Assets written to: checkpoint_mlp\best\assets
3892/3892 [==============================] - 11s 3ms/step - loss: 0.7704 -
accuracy: 0.7680 - val_loss: 0.7788 - val_accuracy: 0.7648
Epoch 10/20
3888/3892 [============================>.] - ETA: 0s - loss: 0.7422 - accuracy:
0.7756INFO:tensorflow:Assets written to: checkpoint_mlp\best\assets
3892/3892 [==============================] - 11s 3ms/step - loss: 0.7422 -
accuracy: 0.7755 - val_loss: 0.7275 - val_accuracy: 0.7808
Epoch 11/20
3892/3892 [==============================] - 10s 3ms/step - loss: 0.7174 -
accuracy: 0.7844 - val_loss: 0.7450 - val_accuracy: 0.7740
Epoch 12/20
3885/3892 [============================>.] - ETA: 0s - loss: 0.6924 - accuracy:
0.7912INFO:tensorflow:Assets written to: checkpoint_mlp\best\assets
3892/3892 [==============================] - 12s 3ms/step - loss: 0.6926 -
accuracy: 0.7912 - val_loss: 0.7088 - val_accuracy: 0.7881
```

```
Epoch 13/20
3892/3892 [==============================] - 10s 3ms/step - loss: 0.6744 -
accuracy: 0.7969 - val_loss: 0.7263 - val_accuracy: 0.7819
Epoch 14/20
3873/3892 [============================>.] - ETA: 0s - loss: 0.6557 - accuracy:
0.8034INFO:tensorflow:Assets written to: checkpoint_mlp\best\assets
3892/3892 [==============================] - 11s 3ms/step - loss: 0.6561 -
accuracy: 0.8033 - val_loss: 0.7019 - val_accuracy: 0.7963
Epoch 15/20
3881/3892 [============================>.] - ETA: 0s - loss: 0.6370 - accuracy:
0.8090INFO:tensorflow:Assets written to: checkpoint_mlp\best\assets
3892/3892 [==============================] - 12s 3ms/step - loss: 0.6369 -
accuracy: 0.8090 - val_loss: 0.6863 - val_accuracy: 0.7952
Epoch 16/20
3889/3892 [============================>.] - ETA: 0s - loss: 0.6189 - accuracy:
0.8140INFO:tensorflow:Assets written to: checkpoint_mlp\best\assets
3892/3892 [==============================] - 12s 3ms/step - loss: 0.6191 -
accuracy: 0.8139 - val_loss: 0.6835 - val_accuracy: 0.8010
Epoch 17/20
3872/3892 [============================>.] - ETA: 0s - loss: 0.6067 - accuracy:
0.8177INFO:tensorflow:Assets written to: checkpoint_mlp\best\assets
3892/3892 [==============================] - 11s 3ms/step - loss: 0.6065 -
accuracy: 0.8178 - val_loss: 0.6817 - val_accuracy: 0.7997
Epoch 18/20
3885/3892 [============================>.] - ETA: 0s - loss: 0.5879 - accuracy:
0.8227INFO:tensorflow:Assets written to: checkpoint_mlp\best\assets
3892/3892 [==============================] - 11s 3ms/step - loss: 0.5878 -
accuracy: 0.8227 - val_loss: 0.6625 - val_accuracy: 0.8031
Epoch 19/20
3892/3892 [==============================] - 10s 3ms/step - loss: 0.5754 -
accuracy: 0.8244 - val_loss: 0.6638 - val_accuracy: 0.8078
Epoch 20/20
3887/3892 [============================>.] - ETA: 0s - loss: 0.5647 - accuracy:
0.8286INFO:tensorflow:Assets written to: checkpoint_mlp\best\assets
3892/3892 [==============================] - 11s 3ms/step - loss: 0.5647 -
accuracy: 0.8285 - val_loss: 0.6130 - val_accuracy: 0.8230
```

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.

[26]:
```
def get_CNN_model(input_shape):
    model = Sequential([
        Conv2D(64, (3,3), activation='relu', input_shape=input_shape),
        MaxPooling2D((2,2)),
        Conv2D(32, (3,3), activation='relu'),
        MaxPooling2D((2,2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(64, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),
        Dense(64, activation='relu'),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer=Adam(0.0001),
                  loss=SparseCategoricalCrossentropy(),
                  metrics=['accuracy'])
    return model
```

[27]:
```
with tf.device('GPU:0'):
    cnn_model = get_CNN_model(x_train[0].shape)
    cnn_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 64)        640
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 32)        18464
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_5 (Dense)              (None, 128)               147584
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
```

```
_________________________________________________________________
dense_6 (Dense)              (None, 64)                8256
_________________________________________________________________
batch_normalization (BatchNo (None, 64)                256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_7 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_8 (Dense)              (None, 10)                650
=================================================================
Total params: 180,010
Trainable params: 179,882
Non-trainable params: 128
_________________________________________________________________
```

[28]:

```
with tf.device('GPU:0'):
    history_cnn = cnn_model.fit(x_train,
                                y_train,
                                epochs=20,
                                batch_size=16,
                                validation_split=.15,
                                callbacks=[get_early_stoping(),
                                           get_checkpoint_callback('cnn'),
                                           reduce_lr()])
```

```
Epoch 1/20
3892/3892 [==============================] - ETA: 0s - loss: 1.9628 - accuracy:
0.3047INFO:tensorflow:Assets written to: checkpoint_cnn\best\assets
3892/3892 [==============================] - 16s 4ms/step - loss: 1.9628 -
accuracy: 0.3047 - val_loss: 1.1491 - val_accuracy: 0.6239
Epoch 2/20
3890/3892 [============================>.] - ETA: 0s - loss: 1.2015 - accuracy:
0.5969INFO:tensorflow:Assets written to: checkpoint_cnn\best\assets
3892/3892 [==============================] - 16s 4ms/step - loss: 1.2014 -
accuracy: 0.5969 - val_loss: 0.7091 - val_accuracy: 0.7963
Epoch 3/20
3887/3892 [============================>.] - ETA: 0s - loss: 0.9349 - accuracy:
0.7015INFO:tensorflow:Assets written to: checkpoint_cnn\best\assets
3892/3892 [==============================] - 16s 4ms/step - loss: 0.9349 -
accuracy: 0.7015 - val_loss: 0.5539 - val_accuracy: 0.8373
Epoch 4/20
3884/3892 [============================>.] - ETA: 0s - loss: 0.8081 - accuracy:
0.7495INFO:tensorflow:Assets written to: checkpoint_cnn\best\assets
3892/3892 [==============================] - 16s 4ms/step - loss: 0.8079 -
accuracy: 0.7496 - val_loss: 0.5194 - val_accuracy: 0.8443
Epoch 5/20
3891/3892 [============================>.] - ETA: 0s - loss: 0.7360 - accuracy:
```

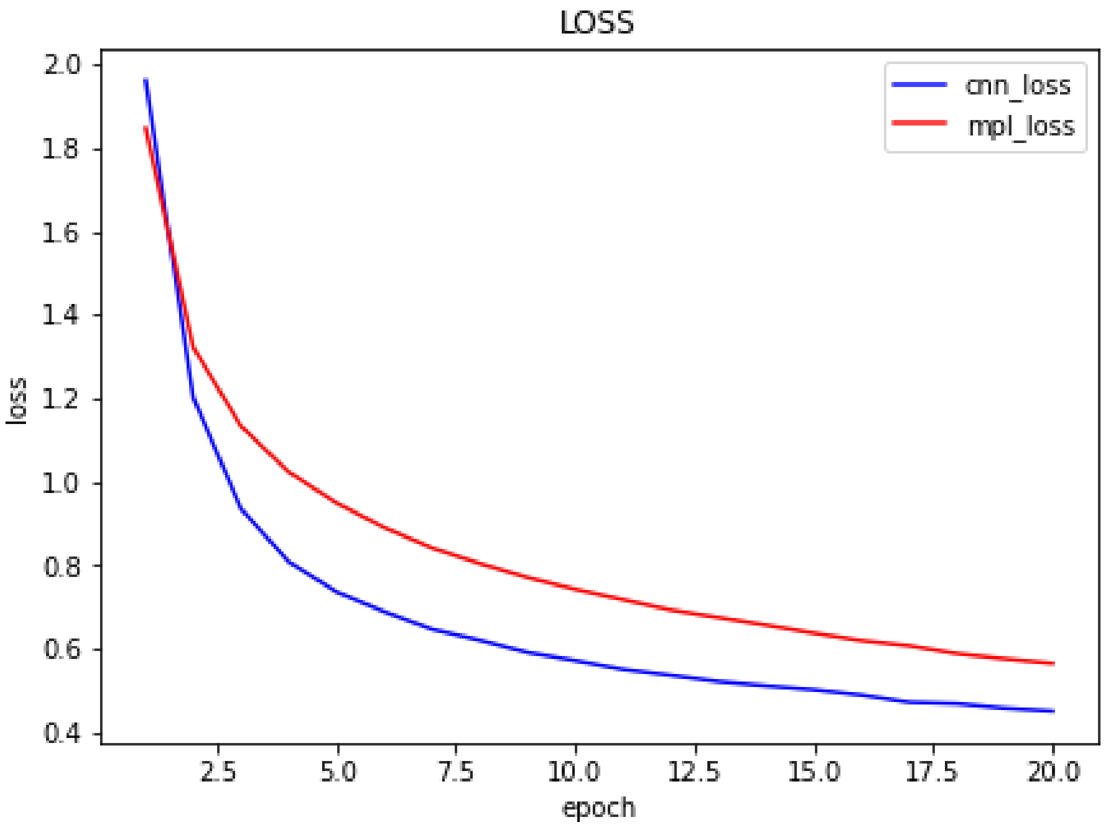
```
0.7756INFO:tensorflow:Assets written to: checkpoint_cnn\best\assets
3892/3892 [==============================] - 16s 4ms/step - loss: 0.7360 -
accuracy: 0.7756 - val_loss: 0.4874 - val_accuracy: 0.8544
Epoch 6/20
3884/3892 [============================>.] - ETA: 0s - loss: 0.6881 - accuracy:
0.7939INFO:tensorflow:Assets written to: checkpoint_cnn\best\assets
3892/3892 [==============================] - 16s 4ms/step - loss: 0.6884 -
accuracy: 0.7938 - val_loss: 0.4763 - val_accuracy: 0.8571
Epoch 7/20
3881/3892 [============================>.] - ETA: 0s - loss: 0.6464 - accuracy:
0.8059INFO:tensorflow:Assets written to: checkpoint_cnn\best\assets
3892/3892 [==============================] - 16s 4ms/step - loss: 0.6461 -
accuracy: 0.8060 - val_loss: 0.4429 - val_accuracy: 0.8674
Epoch 8/20
3892/3892 [==============================] - 14s 4ms/step - loss: 0.6200 -
accuracy: 0.8154 - val_loss: 0.4564 - val_accuracy: 0.8581
Epoch 9/20
3880/3892 [============================>.] - ETA: 0s - loss: 0.5908 - accuracy:
0.8255INFO:tensorflow:Assets written to: checkpoint_cnn\best\assets
3892/3892 [==============================] - 16s 4ms/step - loss: 0.5906 -
accuracy: 0.8255 - val_loss: 0.4384 - val_accuracy: 0.8679
Epoch 10/20
3882/3892 [============================>.] - ETA: 0s - loss: 0.5715 - accuracy:
0.8318INFO:tensorflow:Assets written to: checkpoint_cnn\best\assets
3892/3892 [==============================] - 16s 4ms/step - loss: 0.5712 -
accuracy: 0.8319 - val_loss: 0.4099 - val_accuracy: 0.8760
Epoch 11/20
3892/3892 [==============================] - 14s 4ms/step - loss: 0.5501 -
accuracy: 0.8364 - val_loss: 0.4156 - val_accuracy: 0.8751
Epoch 12/20
3891/3892 [============================>.] - ETA: 0s - loss: 0.5358 - accuracy:
0.8426INFO:tensorflow:Assets written to: checkpoint_cnn\best\assets
3892/3892 [==============================] - 16s 4ms/step - loss: 0.5357 -
accuracy: 0.8427 - val_loss: 0.3929 - val_accuracy: 0.8822
Epoch 13/20
3877/3892 [============================>.] - ETA: 0s - loss: 0.5213 - accuracy:
0.8470INFO:tensorflow:Assets written to: checkpoint_cnn\best\assets
3892/3892 [==============================] - 16s 4ms/step - loss: 0.5209 -
accuracy: 0.8472 - val_loss: 0.3747 - val_accuracy: 0.8886
Epoch 14/20
3892/3892 [==============================] - 14s 4ms/step - loss: 0.5106 -
accuracy: 0.8500 - val_loss: 0.3808 - val_accuracy: 0.8855
Epoch 15/20
3887/3892 [============================>.] - ETA: 0s - loss: 0.5014 - accuracy:
0.8538INFO:tensorflow:Assets written to: checkpoint_cnn\best\assets
3892/3892 [==============================] - 16s 4ms/step - loss: 0.5014 -
accuracy: 0.8538 - val_loss: 0.3729 - val_accuracy: 0.8893
Epoch 16/20
```

```
3880/3892 [============================>.] - ETA: 0s - loss: 0.4887 - accuracy:
0.8567INFO:tensorflow:Assets written to: checkpoint_cnn\best\assets
3892/3892 [==============================] - 16s 4ms/step - loss: 0.4884 -
accuracy: 0.8568 - val_loss: 0.3677 - val_accuracy: 0.8915
Epoch 17/20
3887/3892 [============================>.] - ETA: 0s - loss: 0.4711 - accuracy:
0.8635INFO:tensorflow:Assets written to: checkpoint_cnn\best\assets
3892/3892 [==============================] - 16s 4ms/step - loss: 0.4710 -
accuracy: 0.8635 - val_loss: 0.3647 - val_accuracy: 0.8918
Epoch 18/20
3892/3892 [==============================] - 14s 4ms/step - loss: 0.4678 -
accuracy: 0.8640 - val_loss: 0.3780 - val_accuracy: 0.8883
Epoch 19/20
3886/3892 [============================>.] - ETA: 0s - loss: 0.4572 - accuracy:
0.8662INFO:tensorflow:Assets written to: checkpoint_cnn\best\assets
3892/3892 [==============================] - 16s 4ms/step - loss: 0.4571 -
accuracy: 0.8662 - val_loss: 0.3575 - val_accuracy: 0.8961
Epoch 20/20
3892/3892 [==============================] - 14s 4ms/step - loss: 0.4503 -
accuracy: 0.8701 - val_loss: 0.3678 - val_accuracy: 0.8927
```

[34]:
```
df_cnn = pd.DataFrame(history_cnn.history)
df_mlp = pd.DataFrame(history_mlp.history)
```

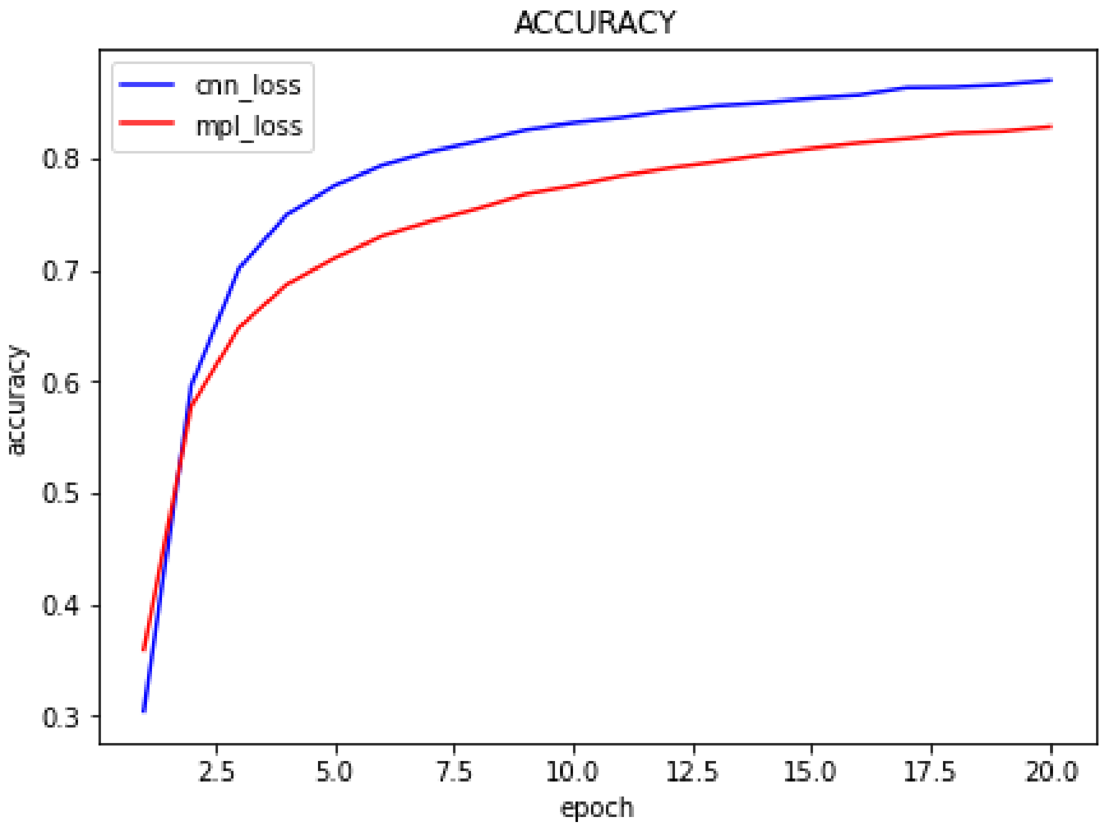
[57]:
```
plt.figure(figsize=(7, 5))
plt.plot(range(1, 21), df_cnn['loss'], c='b', label='cnn_loss')
plt.plot(range(1, 21), df_mlp['loss'], c='r', label='mlp_loss')
plt.legend(['cnn_loss', 'mpl_loss'])
plt.title('LOSS')
plt.xlabel('epoch')
plt.ylabel('loss')
```

[57]:
```
Text(0, 0.5, 'loss')
```



[58]:
```
plt.figure(figsize=(7, 5))
plt.plot(range(1, 21), df_cnn['accuracy'], c='b', label='cnn_accuracy')
plt.plot(range(1, 21), df_mlp['accuracy'], c='r', label='mlp_accuracy')
plt.legend(['cnn_loss', 'mpl_loss'])
plt.title('ACCURACY')
plt.xlabel('epoch')
plt.ylabel('accuracy')
```

[58]:
```
Text(0, 0.5, 'accuracy')
```




```
[59]: with tf.device('GPU:0'):
          cnn_dict_results = cnn_model.evaluate(x_test, y_test)
```

```
814/814 [==============================] - 2s 2ms/step - loss: 0.4210 -
accuracy: 0.8786
```

```
[60]: with tf.device('GPU:0'):
          mlp_dict_results = model.evaluate(x_test, y_test)
```

```
814/814 [==============================] - 1s 2ms/step - loss: 0.7169 -
accuracy: 0.8001
```

```
[63]: print('cnn results:\n\tloss:{}\n\tacc:{}'.format(cnn_dict_results[0],␣
       ↪cnn_dict_results[1]))
      print('mlp results:\n\tloss:{}\n\tacc:{}'.format(mlp_dict_results[0],␣
       ↪mlp_dict_results[1]))
```

```
cnn results:
        loss:0.421041339635849
        acc:0.8785725235939026
mlp results:
```

```
loss:0.7169337868690491
acc:0.8000922203063965
```

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.

[125]:
```
#loading models
loaded_mlp = load_model('checkpoint_mlp/best')
loaded_cnn = load_model('checkpoint_cnn/best')
```

[126]:
```
#selecting 5 random pictures with tags
random_set_x = []
random_set_y = []
for i in range(5):
    el = np.random.randint(len(x_test))
    random_set_x.append(x_test[el])
    random_set_y.append(y_test[el][0])

random_set_x = np.array(random_set_x)
```

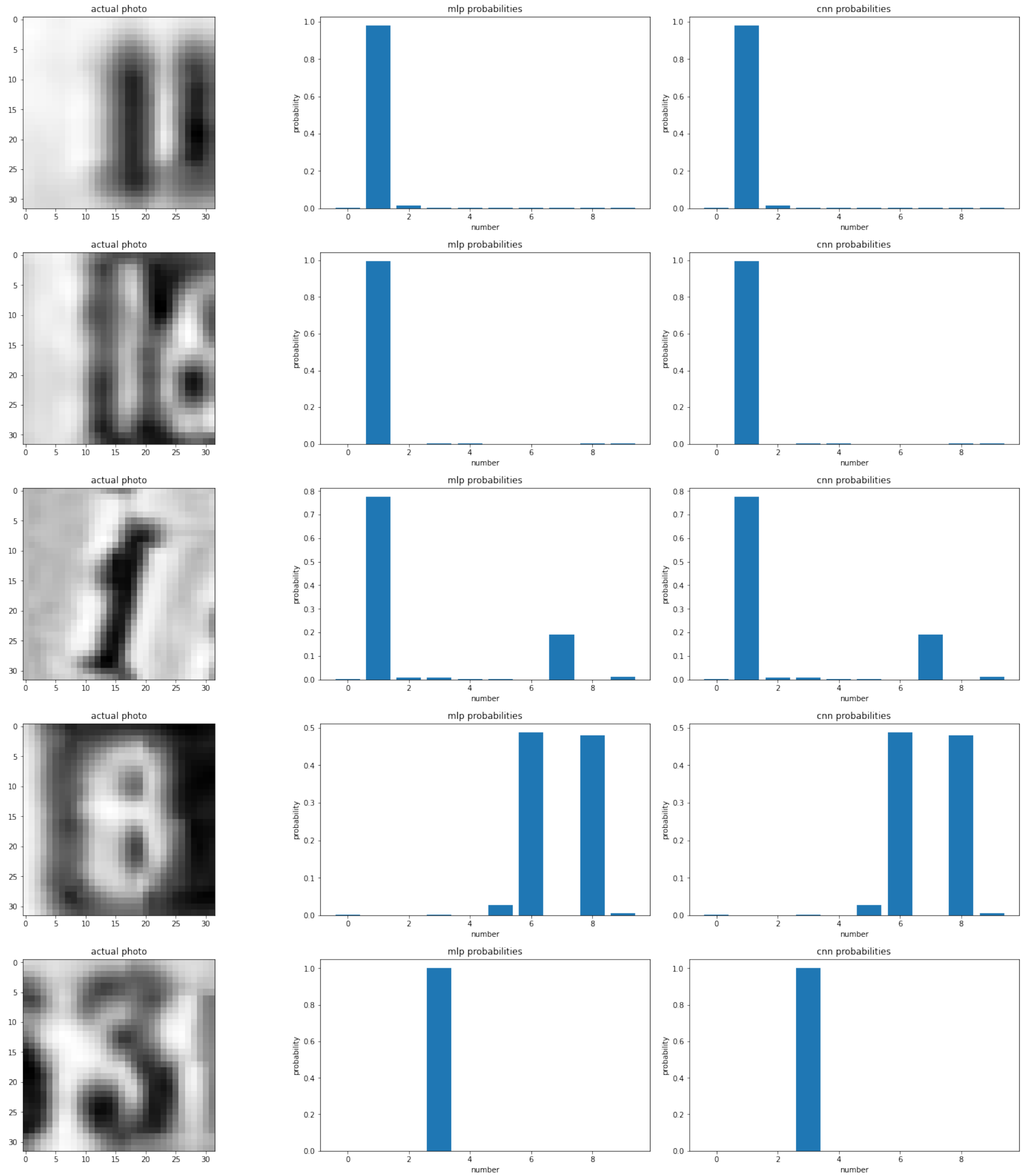
[127]:
```
with tf.device('GPU:0'):
    results = loaded_mlp.predict(random_set_x)
```

[128]:
```
with tf.device('GPU:0'):
    results_cnn = loaded_mlp.predict(random_set_x)
```

[129]:
```
fig, ax = plt.subplots(5, 3, figsize=(20, 22), constrained_layout=True)
for i in range(5):
    image = random_set_x[i].squeeze()
    ax[i, 0].title.set_text('actual photo')
    ax[i, 0].imshow(image, cmap='Greys')

    ax[i, 1].title.set_text('mlp probabilities')
    ax[i, 1].bar(x=range(10), height=results[i])
    ax[i, 1].set_xlabel('number')
    ax[i, 1].set_ylabel('probability')

    ax[i, 2].title.set_text('cnn probabilities')
    ax[i, 2].bar(x=range(10), height=results_cnn[i])
    ax[i, 2].set_xlabel('number')
    ax[i, 2].set_ylabel('probability')
```



[137]:
```
#comparing actual results
cnn_final_results = results_cnn.argmax(axis=1)
mlp_final_results = results.argmax(axis=1)
final_table = np.array([random_set_y, mlp_final_results, cnn_final_results]).T
final_res = pd.DataFrame(data=final_table, columns=['actual_res', 'mlp_res',
 ↪'cnn_res'])
final_res
```

[137]:

	actual_res	mlp_res	cnn_res
0	1	1	1
1	1	1	1
2	1	1	1
3	8	6	6
4	3	3	3

As you can see both models misinterpred number 8. Actual result is 8 but they “think” its 6. At the same time both models are not very confidence with that result (you can see that on bar plots) Probabilities of 8 and 6 are pretty close, so we need to train our models to get better results. Thanks.

[ ]:

```

```