Assignment 2 (key)

1. Load and inspect the **load_digits** dataset.

   a. enable the GPU:

```
import tensorflow as tf

# display tf version and test if GPU is active
tf.__version__, tf.test.gpu_device_name()
```

```
('2.3.0', '/device:GPU:0')
```

   b. load the datset:

```
from sklearn.datasets import load_digits

digits = load_digits()
```

   c. display keys:

```
digits.keys()
```

   d. assign image and target data, and target names to variables for convenience:

   (note: use the flattened image data contained in **digits.images**)

```
images = digits.images
targets = digits.target
target_names = digits.target_names
```

   e. display image and target shapes, and target names:

```
print (images.shape)
print (targets.shape)
print (target_names)
```

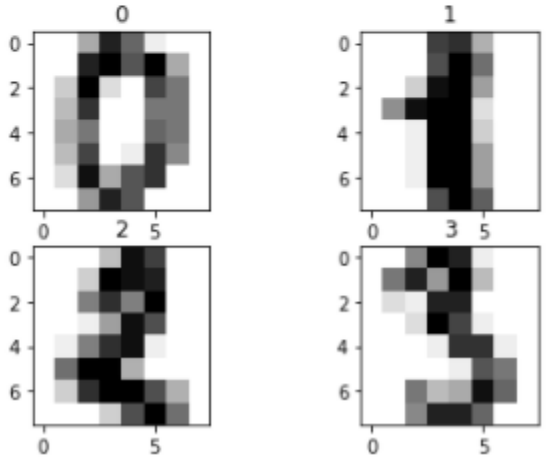
```
(1797, 8, 8)
(1797,)
[0 1 2 3 4 5 6 7 8 9]
```

   f. display the first four images and targets:

   * to make it look fancy, consult:

https://matplotlib.org/3.1.0/gallery/subplots_axes_and_figures/subplots_demo.html
https://stackoverflow.com/questions/41793931/plotting-images-side-by-side-using-matplotlib

```
import matplotlib.pyplot as plt

f, axs = plt.subplots(2,2)
axs[0, 0].set_title(targets[0])
axs[0, 1].set_title(targets[1])
axs[1, 0].set_title(targets[2])
axs[1, 1].set_title(targets[3])
axs[0,0].imshow(images[0], cmap='binary')
axs[0,1].imshow(images[1], cmap='binary')
axs[1,0].imshow(images[2], cmap='binary')
axs[1,1].imshow(images[3], cmap='binary')
plt.show()
```



2. Split data into train and test sets.

   a. use 75% for training and 25% for validation:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    images, targets, test_size=0.25, random_state=0)
```

   b. display train and test shapes:

```
print (X_train.shape, y_train.shape)
print (X_test.shape, y_test.shape)
```

3. Build the input pipeline:

   a. scale feature data:

```
# scale by dividing by the number of pixels in an image

s_train = X_train / 255.0
s_test = X_test / 255.0
```

b. prepare data for TensorFlow consumption:

```
train_dataset = tf.data.Dataset.from_tensor_slices((s_train,
                                                    y_train))
test_dataset = tf.data.Dataset.from_tensor_slices((s_test,
                                                   y_test))
```

c. display a train example with the **take** method:

```
for feature, label in train_dataset.take(1):
  print (feature[0])
  print ('label:', label)
```

```
tf.Tensor(
[0.         0.01176471 0.05098039 0.0627451  0.03529412 0.
 0.         0.        ], shape=(8,), dtype=float64)
label: tf.Tensor(2, shape=(), dtype=int64)
```

4. Batch and shuffle data.

   a. establish batch and buffer size:

```
BATCH_SIZE = 64
SHUFFLE_BUFFER_SIZE = 100
```

   b. batch and shuffle:

```
train_ds = train_dataset\
.shuffle(SHUFFLE_BUFFER_SIZE)\
.batch(BATCH_SIZE)

test_ds = test_dataset.batch(BATCH_SIZE)
```

5. Create the model.

   a. define input shape:

```
for item in train_ds.take(1):
  s = item[0].shape

in_shape = s[1:]
in_shape
```

```
TensorShape([8, 8])
```

b. import libraries:

```
# import libraries to simplify model layout

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
```

c. build model:

```
model = Sequential([
    Flatten(input_shape=in_shape),
    Dense(256, activation='relu'),
    Dense(10, activation='softmax')
])
```

d. display model summary:

```
model.summary()
```

6. Compile, train, and generalize:

a. compile and train:

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

```
history = model.fit(train_ds, epochs=60,
                    validation_data=(test_ds))
```

b. generalize:

```
model.evaluate(test_ds)
```

```
8/8 [==============================] - 0s 3ms/step - loss: 0.1909 - accuracy: 0.9511
[0.19088159501552582, 0.9511111378669739]
```

7. Visualize.

a. assign history object to a variable:

```
history_dict = history.history
```

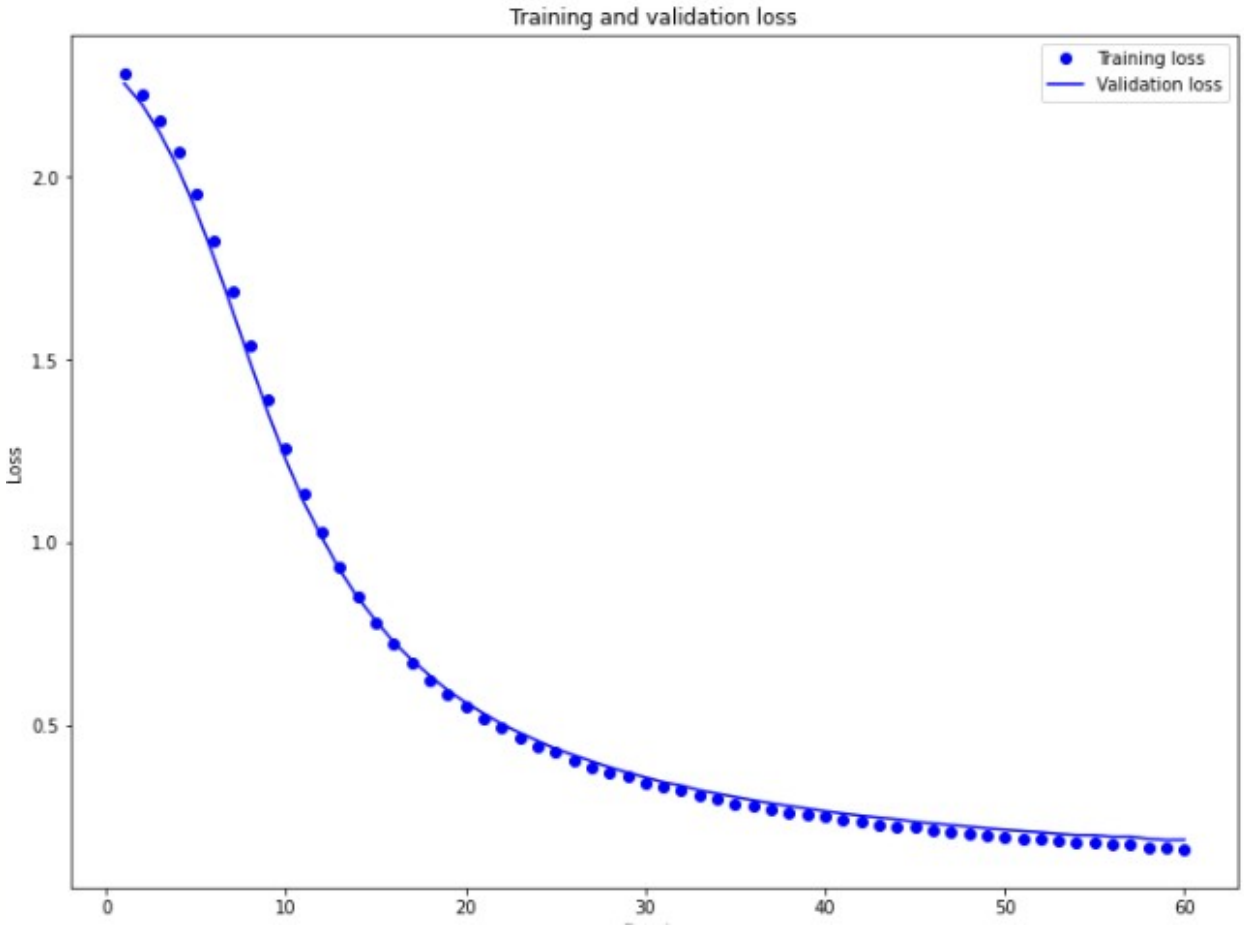
b. display keys in history object:

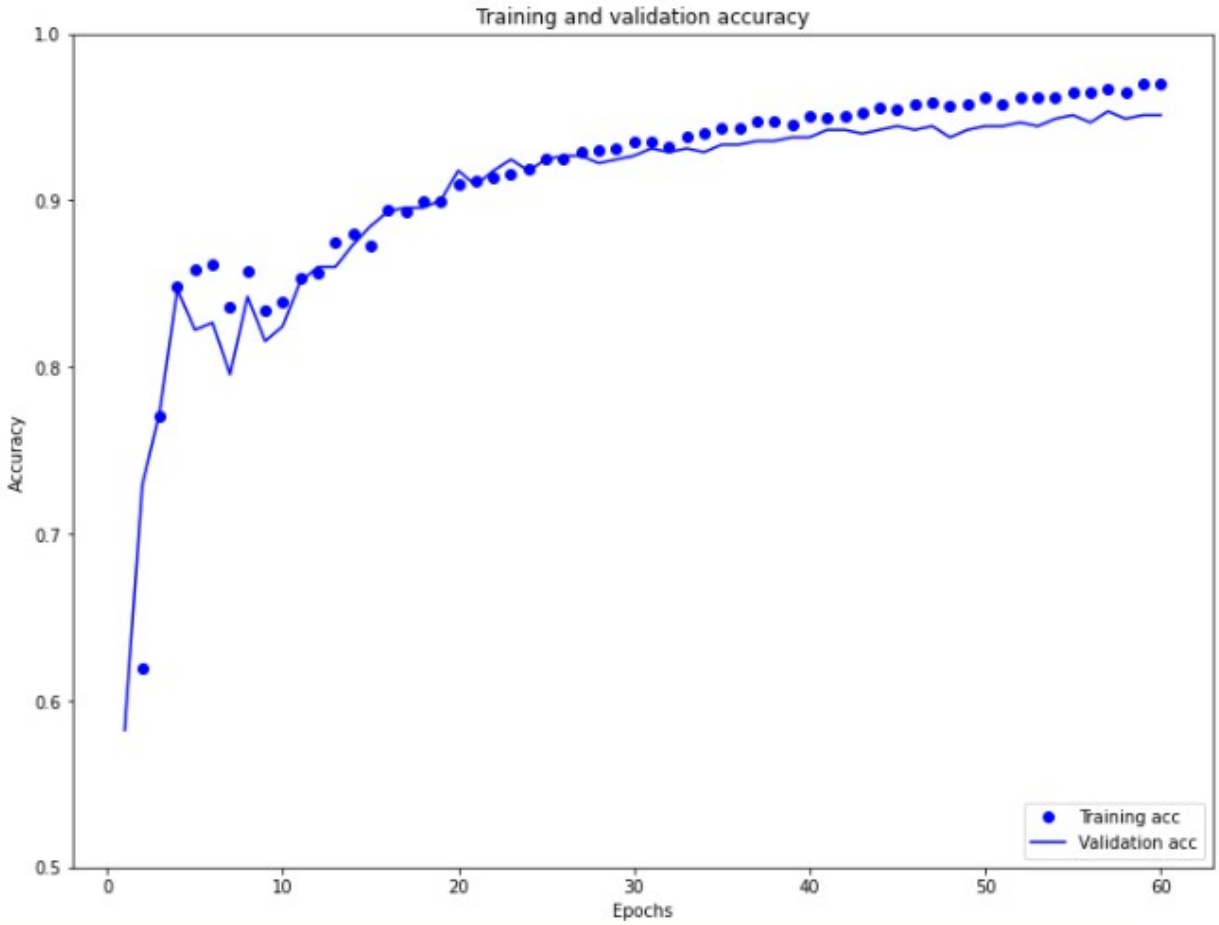
```
history_dict.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

c. show performance plot:

```
import matplotlib.pyplot as plt

acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']

epochs = range(1, len(acc) + 1)

plt.figure(figsize=(12,9))
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# clear previous figure

plt.clf()

plt.figure(figsize=(12,9))
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.ylim((0.5,1))
plt.show()
```


Training and validation loss
Training loss
Validation loss
Loss
2.0
1.5
1.0
0.5
0
10
20
30
40
50
60



8. Make predictions.

a. get predictions from the model:

```
predictions = model.predict(test_ds)
```

b. display the first prediction:

```
predictions[0].round(2)
```

```
array([0.  , 0.  , 0.99, 0.01, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
      dtype=float32)
```

c. provide the prediction for the first images in the test set:

the index with the highest value by position based on a range from 0-9.

```
import numpy as np

first_pred = np.argmax(predictions[0]) # start at index 0
print ('predicted:', first_pred)
```

```
predicted: 2
```

d. provide the actual target for the first image in the test set:

```
print ('actual:', y_test[0])
```

```
actual: 2
```

e. was the prediction correct?

9. Save an image to your Google Drive and access it in Colab.

a. mount Google Drive to Colab:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

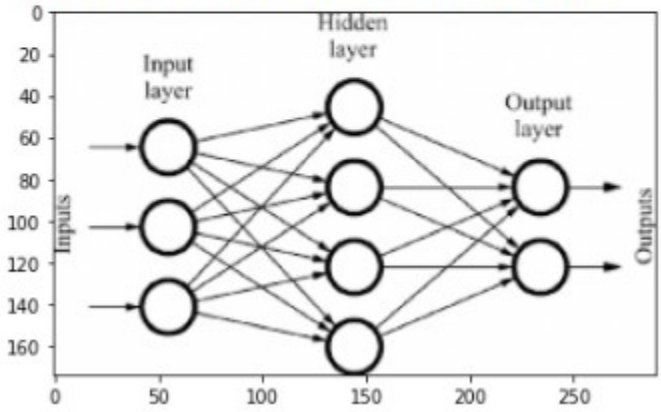
```
Mounted at /content/gdrive
```

b. save an image to your Google Drive.

c. display the image:

```
# Be sure to copy the image to this directory on Google Drive
img_path = 'gdrive/My Drive/Colab Notebooks/Figure0201.png'

from PIL import Image
import matplotlib.pyplot as plt
img  = Image.open(img_path)
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7f5ab0b33be0>
```



(note: your image name may differ)