Assignment 6 (key)

1. Enable 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')
```

2. Load 'cats_vs_dogs' TFDS with info object:

```
import tensorflow_datasets as tfds

data, info = tfds.load(name='cats_vs_dogs', with_info=True)
```

3. Display **info** object:

```
info
```

4. Split data.

   a. split into train and test sets:

```
(train_ds, test_ds), info = tfds.load(
    'cats_vs_dogs',
    split=['train[:80%]', 'train[80%:]'],
    with_info=True, shuffle_files=True,
    as_supervised=True,
    try_gcs=True
)
```

   b. find number of train and test examples:

```
num_train = len([i for i, example in enumerate(train_ds)])
num_test = len([i for i, example in enumerate(test_ds)])
```

   c. display number of train and test examples:

```
print ('train images:', num_train)
print ('test images:', num_test)
```

```
train images: 18610
test images: 4652
```

5. Inspect data:

   a. display examples:

```
fig = tfds.show_examples(train_ds, info)
```

   b. extract class labels:

```
class_labels = info.features['label'].names
class_labels
```

   c. extract number of classes:

```
num_classes = info.features['label'].num_classes
num_classes
```

```
2
```

   d. prepare 30 images and labels for display:

   (hint: take 30 examples from **train** set, squeeze out 1 dimension, and build lists)

```
num = 30
images, labels = [], []
for img, lbl in train_ds.take(num):
  image, label = img, lbl
  images.append(tf.squeeze(image.numpy()))
  labels.append(label.numpy())
```

   e. create a function to plot images and labels:

```
def display_grid(feature, target, n_rows, n_cols, cl):
  plt.figure(figsize=(n_cols * 1.5, n_rows * 1.5))
  for row in range(n_rows):
    for col in range(n_cols):
      index = n_cols * row + col
      plt.subplot(n_rows, n_cols, index + 1)
      plt.imshow(feature[index], cmap='binary',
                 interpolation='nearest')
      plt.axis('off')
      plt.title(cl[target[index]], fontsize=12)
  plt.subplots_adjust(wspace=0.2, hspace=0.5)
```

f. plot images and labels:

```
import matplotlib.pyplot as plt

rows, cols = 5, 6
display_grid(images, labels, rows, cols, class_labels)
```

6. Build the input pipeline.

   a. resize and scale images:

```
IMAGE_RES = 150

def format_image(image, label):
  image = tf.image.resize(image, (IMAGE_RES, IMAGE_RES))/255.0
  return image, label
```

   b. prepare train and test data for TensorFlow consumption:

```
BATCH_SIZE = 200
SHUFFLE_SIZE = 500

train_batches = train_ds.shuffle(SHUFFLE_SIZE).\
map(format_image).batch(BATCH_SIZE).cache().prefetch(1)

validation_batches = test_ds.\
map(format_image).batch(BATCH_SIZE).cache().prefetch(1)
```

   c. inspect train and test data:

```
train_batches.element_spec
```

```
(TensorSpec(shape=(None, 150, 150, 3), dtype=tf.float32, name=None),
 TensorSpec(shape=(None,), dtype=tf.int64, name=None))
```

7. Model data:

a. import libraries:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D,\
Dense, Flatten
```

b. clear previous models and generate seed:

```
import numpy as np

tf.keras.backend.clear_session()
np.random.seed(0)
tf.random.set_seed(0)
```

c. get input shape:

```
for img, lbl in train_ds.take(1):
  img.shape

in_shape = img.shape[1:]
in_shape
```

d. create model:

```
model = Sequential([
  Conv2D(32, (3, 3), activation = 'relu',
         input_shape=in_shape, strides=1,
         kernel_regularizer='l1_l2'),
  MaxPooling2D(2, 2),
  Conv2D(64, (3, 3), activation='relu'),
  MaxPooling2D(2, 2),
  Conv2D(128, (3, 3), activation='relu'),
  MaxPooling2D(2),
  Conv2D(128, (3, 3), activation='relu'),
  MaxPooling2D(2, 2),
  Flatten(),
  Dense(512, activation='relu'),
  Dense(2, activation='sigmoid')
])
```

e. inspect model:

```
model.summary()
```

f. compile:

```
loss = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True)

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

g. train for 10 epochs:

```
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)

epochs = 10
history = model.fit(train_batches, epochs=epochs,
                    verbose=1, validation_data=validation_batches)
```

h. generalize:

```
model.evaluate(validation_batches)
```

8. Visualize performance:

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
plt.show()

plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.ylim([0.5, 1.0])
plt.legend(loc='lower right')
plt.show()
```