Experiment 1

Aim: Design a Dense Net for temperature conversion

Software used: Jupyter notebook

Theory:

A Dense Net is a type of convolutional neural network that utilises dense connections between layers, through Dense Blocks, where we connect all layers (with matching feature-map sizes) directly with each other. Dense Net(Densely Connected Convolutional Networks) is one of the latest neural networks for visual object recognition. Dense Net is the concept that is similar to ResNet, but rather than adding the convolution output with the identity, we concatenate the identity connection to the convolution output.

Code and Output:

In [1]:
```
import tensorflow as tf
```

In [2]:
```
import numpy as np
import logging
logger = tf.get_logger()
logger.setLevel(logging.ERROR)
```

In [3]:
```
#training data
cel = np.array([-40,-10,0,8,15,22,38], dtype = float)
fahr = np.array([-40,14,32,46,59,72,100], dtype=float)
for i,c in enumerate(cel):
    print("{} degree Celsius = {} degree Fahrenheit".format(c, fahr[i]))
```

```
-40.0 degree Celsius = -40.0 degree Fahrenheit
-10.0 degree Celsius = 14.0 degree Fahrenheit
0.0 degree Celsius = 32.0 degree Fahrenheit
8.0 degree Celsius = 46.0 degree Fahrenheit
15.0 degree Celsius = 59.0 degree Fahrenheit
22.0 degree Celsius = 72.0 degree Fahrenheit
38.0 degree Celsius = 100.0 degree Fahrenheit
```

In [5]:
```
#building a model
layer = tf.keras.layers.Dense(units = 1, input_shape=[1])
#input_shape = [1] This specifies that the input to this layer is a single value. That is, the shape is a one-dimensional array w
#units=1 — This specifies the number of neurons in the layer. The number of neurons defines how many internal variables the layer
```

In [6]:
```
model = tf.keras.Sequential([layer])
```

In [8]:
```
#compile the model, with loss and optimizer functions
#Loss function — A way of measuring how far off predictions are from the desired outcome. (The measured difference is called the
#Optimizer function — A way of adjusting internal values in order to reduce the loss.
model.compile(loss = 'mean_squared_error', optimizer = tf.keras.optimizers.Adam(0.1))
```

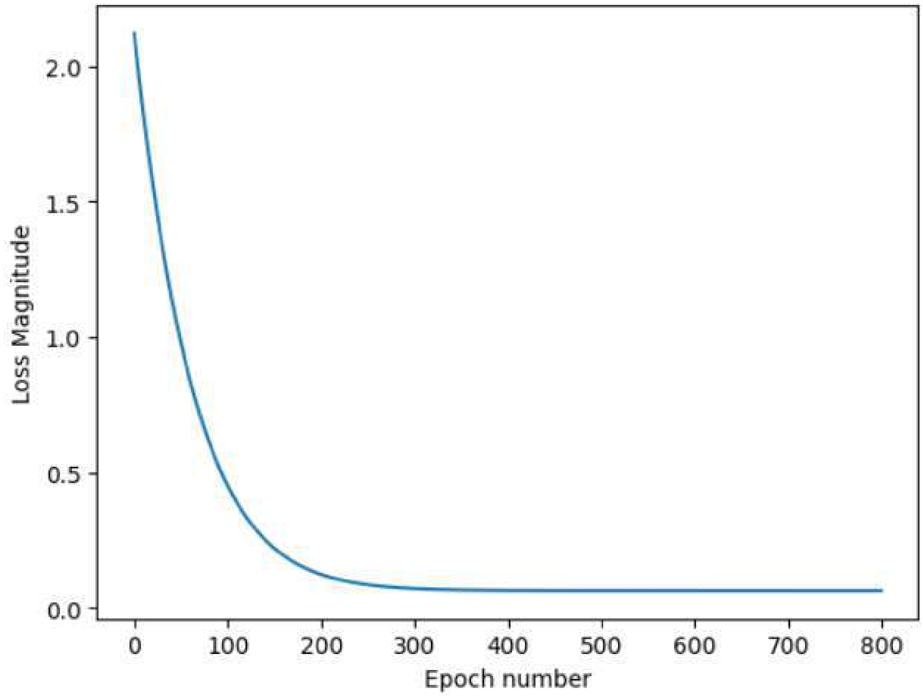
In [9]:
```
#the learning rate (0.1 in the code above). This is the step size taken when adjusting values in the model. If the value is too s
```

In [18]:
```
#train the model
history = model.fit(cel, fahr, epochs = 800, verbose = False)
#The epochs argument specifies how many times this cycle should be run, and the verbose argument controls how much output the met
print("Finished training the model")
```

```
Finished training the model
```

In [19]:
```
#visualizaton
import matplotlib.pyplot as plt
plt.xlabel("Epoch number")
plt.ylabel("Loss Magnitude")
plt.plot(history.history["loss"])
```

```
Out[19]: [<matplotlib.lines.Line2D at 0x1d78ea3b8b0>]
```




```
In [20]: #test
print(model.predict([100.0]))

1/1 [==============================] - 0s 73ms/step
[[211.74742]]
```

Conclusion: implementation was successful.

EXPERIMENT 2

Aim: design a densenet for classifying images using fashion MNIST dataset

Software used: Google Collab

Theory:

A Dense Net is a type of convolutional neural network that utilises dense connections between layers, through Dense Blocks, where we connect all layers (with matching feature-map sizes) directly with each other. Dense Net(Densely Connected Convolutional Networks) is one of the latest neural networks for visual object recognition. Dense Net is the concept that is similar to ResNet, but rather than adding the convolution output with the identity, we concatenate the identity connection to the convolution output.

Code and Output:

```
import numpy as np
import matplotlib as pd
import math
import matplotlib.pyplot as plt
import tensorflow as tf
```

```
[ ] import tensorflow_datasets as tfds
    tfds.disable_progress_bar()
```

```
[ ] !pip install -U tensorflow_datasets
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tensorflow_datasets in /usr/local/lib/python3.8/dist-packages (4.8.2)
Requirement already satisfied: protobuf>=3.12.2 in /usr/local/lib/python3.8/dist-packages (from tensorflow_datasets) (3.19.6)
Requirement already satisfied: importlib-resources in /usr/local/lib/python3.8/dist-packages (from tensorflow_datasets) (5.10.2)
Requirement already satisfied: toml in /usr/local/lib/python3.8/dist-packages (from tensorflow_datasets) (0.10.2)
Requirement already satisfied: etils[enp,epath]>=0.9.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow_datasets) (1.0.0)
Requirement already satisfied: click in /usr/local/lib/python3.8/dist-packages (from tensorflow_datasets) (7.1.2)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow_datasets) (2.25.1)
Requirement already satisfied: absl-py in /usr/local/lib/python3.8/dist-packages (from tensorflow_datasets) (1.4.0)
Requirement already satisfied: dill in /usr/local/lib/python3.8/dist-packages (from tensorflow_datasets) (0.3.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.8/dist-packages (from tensorflow_datasets) (4.64.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from tensorflow_datasets) (1.21.6)
Requirement already satisfied: wrapt in /usr/local/lib/python3.8/dist-packages (from tensorflow_datasets) (1.14.1)
Requirement already satisfied: tensorflow-metadata in /usr/local/lib/python3.8/dist-packages (from tensorflow_datasets) (1.12.0)
Requirement already satisfied: promise in /usr/local/lib/python3.8/dist-packages (from tensorflow_datasets) (2.3)
Requirement already satisfied: psutil in /usr/local/lib/python3.8/dist-packages (from tensorflow_datasets) (5.4.8)
Requirement already satisfied: dm-tree in /usr/local/lib/python3.8/dist-packages (from tensorflow_datasets) (0.1.8)
Requirement already satisfied: termcolor in /usr/local/lib/python3.8/dist-packages (from tensorflow_datasets) (2.2.0)
```

```
dataset, metadata = tfds.load('fashion_mnist', as_supervised=True, with_info=True)
train_dataset, test_dataset = dataset['train'], dataset['test']
```

```
class_names = metadata.features['label'].names
print("Class names: {}".format(class_names))
```

```
Class names: ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

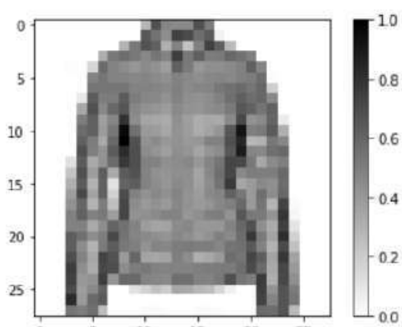
```
num_train_examples = metadata.splits['train'].num_examples
num_test_examples = metadata.splits['test'].num_examples
print("Number of training examples: {}".format(num_train_examples))
print("Number of test examples:     {}".format(num_test_examples))
```

```
Number of training examples: 60000
Number of test examples:     10000
```

```
def normalize(images, labels):
  images = tf.cast(images, tf.float32)
  images /= 255
  return images, labels

# The map function applies the normalize function to each element in the train
# and test datasets
train_dataset =  train_dataset.map(normalize)
test_dataset  =  test_dataset.map(normalize)

# The first time you use the dataset, the images will be loaded from disk
```

```
# The first time you use the dataset, the images will be loaded from disk
# Caching will keep them in memory, making training faster
train_dataset =  train_dataset.cache()
test_dataset  =  test_dataset.cache()
```

```
# Take a single image, and remove the color dimension by reshaping
for image, label in test_dataset.take(1):
  break
image = image.numpy().reshape((28,28))

# Plot the image - voila a piece of fashion clothing
plt.figure()
plt.imshow(image, cmap=plt.cm.binary)
plt.colorbar()
plt.grid(False)
plt.show()
```



```
plt.figure(figsize=(10,10))
for i, (image, label) in enumerate(train_dataset.take(25)):
    image = image.numpy().reshape((28,28))
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(image, cmap=plt.cm.binary)
    plt.xlabel(class_names[label])
plt.show()
```


```
[ ] model = tf.keras.Sequential([
        tf.keras.layers.Flatten(input_shape=(28, 28, 1)),
        tf.keras.layers.Dense(128, activation=tf.nn.relu),
        tf.keras.layers.Dense(10, activation=tf.nn.softmax)
    ])
```

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

```
[ ] BATCH_SIZE = 32
    train_dataset = train_dataset.cache().repeat().shuffle(num_train_examples).batch(BATCH_SIZE)
    test_dataset = test_dataset.cache().batch(BATCH_SIZE)
```

```
[ ] model.fit(train_dataset, epochs=5, steps_per_epoch=math.ceil(num_train_examples/BATCH_SIZE))

    Epoch 1/5
    1875/1875 [==============================] - 13s 4ms/step - loss: 0.4957 - accuracy: 0.8261
    Epoch 2/5
    1875/1875 [==============================] - 5s 3ms/step - loss: 0.3769 - accuracy: 0.8619
    Epoch 3/5
    1875/1875 [==============================] - 7s 4ms/step - loss: 0.3333 - accuracy: 0.8793
    Epoch 4/5
    1875/1875 [==============================] - 6s 3ms/step - loss: 0.3130 - accuracy: 0.8844
    Epoch 5/5
    1875/1875 [==============================] - 7s 4ms/step - loss: 0.2947 - accuracy: 0.8911
    <keras.callbacks.History at 0x7f5ca8db44c0>
```

Conclusion: Implementation was successful.

EXPERIMENT 3

Aim: Design a ConvNet for classifying images using flower dataset

Software used: Google collab

Theory:

A convolutional neural network (CNN or ConvNet) is a network architecture for deep learning that learns directly from data. CNNs are particularly useful for finding patterns in images to recognize objects, classes, and categories. They can also be quite effective for classifying audio, time-series, and signal data.

Code and Output:

```
import os
import numpy as np
import glob
import shutil

import tensorflow as tf

import matplotlib.pyplot as plt
```

```
[ ] #import packages
    import tensorflow as tf
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
[ ] _URL = "https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"

    zip_file = tf.keras.utils.get_file(origin=_URL,
                                       fname="flower_photos.tgz",
                                       extract=True)

    base_dir = os.path.join(os.path.dirname(zip_file), 'flower_photos')
```

The dataset we downloaded contains images of 5 types of flowers:

1. Rose
2. Daisy
3. Dandelion
4. Sunflowers
5. Tulips

So, let's create the labels for these 5 classes:

```
[ ] classes = ['roses', 'daisy', 'dandelion', 'sunflowers', 'tulips']
```

```
[ ] for cl in classes:
      img_path = os.path.join(base_dir, cl)
      images = glob.glob(img_path + '/*.jpg')
      print("{}: {} Images".format(cl, len(images)))
      train, val = images[:round(len(images)*0.8)], images[round(len(images)*0.8):]

      for t in train:
        if not os.path.exists(os.path.join(base_dir, 'train', cl)):
          os.makedirs(os.path.join(base_dir, 'train', cl))
        shutil.move(t, os.path.join(base_dir, 'train', cl))

      for v in val:
        if not os.path.exists(os.path.join(base_dir, 'val', cl)):
          os.makedirs(os.path.join(base_dir, 'val', cl))
        shutil.move(v, os.path.join(base_dir, 'val', cl))
```

```
roses: 641 Images
```

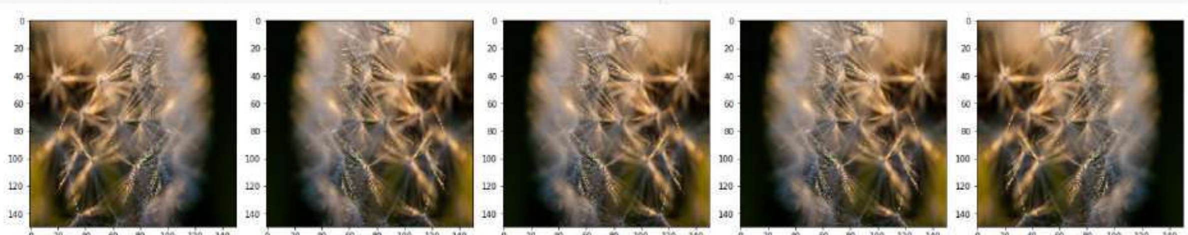
```
[ ] train_dir = os.path.join(base_dir, 'train')
    val_dir = os.path.join(base_dir, 'val')
```

```
[ ] batch_size = 100
    IMG_SHAPE = 150
```

```
[ ] image_gen = ImageDataGenerator(rescale=1./255, horizontal_flip=True)

    train_data_gen = image_gen.flow_from_directory(
        batch_size = batch_size,
        directory = train_dir,
        shuffle = True,
        target_size = (IMG_SHAPE,IMG_SHAPE)
    )
```

```
Found 2935 images belonging to 5 classes.
```

```
[ ] # This function will plot images in the form of a grid with 1 row and 5 columns where images are placed in each column.
    def plotImages(images_arr):
        fig, axes = plt.subplots(1, 5, figsize=(20,20))
        axes = axes.flatten()
        for img, ax in zip( images_arr, axes):
            ax.imshow(img)
        plt.tight_layout()
        plt.show()


    augmented_images = [train_data_gen[0][0][0] for i in range(5)]
    plotImages(augmented_images)
```



```
[ ] image_gen = ImageDataGenerator(rescale = 1./255, rotation_range =45)

    train_data_gen = image_gen.flow_from_directory(batch_size=batch_size,
                                                    directory=train_dir,
                                                    shuffle = True,
                                                    target_size = (IMG_SHAPE,IMG_SHAPE))
```

```
Found 2935 images belonging to 5 classes.
```

Let's take 1 sample image from our training examples and repeat it 5 times so that the augmentation can be applied to the same image 5 times over randomly, to see the augmentation in action.

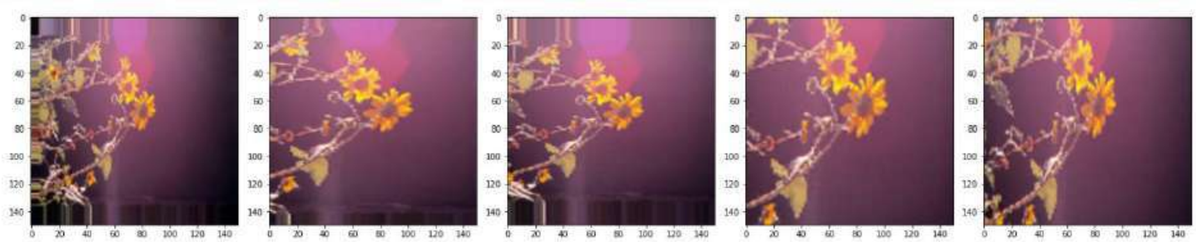
```
[ ] augmented_images = [train_data_gen[0][0][0] for i in range(5)]
    plotImages(augmented_images)
```

```
[ ] image_gen = ImageDataGenerator(rescale = 1./255, zoom_range =0.5)

    train_data_gen = image_gen.flow_from_directory(batch_size=batch_size,
                                                    directory=train_dir,
                                                    shuffle = True,
                                                    target_size = (IMG_SHAPE,IMG_SHAPE))
```

```
Found 2935 images belonging to 5 classes.
```

Let's take 1 sample image from our training examples and repeat it 5 times so that the augmentation can be applied to the same image 5 times over randomly, to see the augmentation in action.

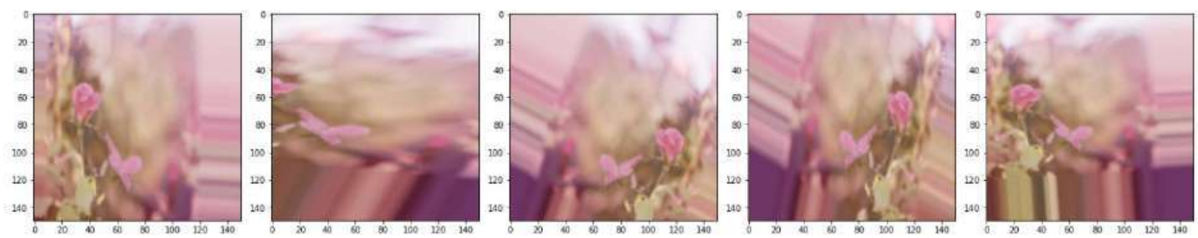
```
[ ] augmented_images = [train_data_gen[0][0][0] for i in range(5)]
    plotImages(augmented_images)
```



```
[ ]  image_gen_train = ImageDataGenerator(
                         rescale=1./255,
                         rotation_range=45,
                         width_shift_range=.15,
                         height_shift_range=.15,
                         horizontal_flip=True,
                         zoom_range=0.5
                         )


     train_data_gen = image_gen_train.flow_from_directory(
                                                  batch_size=batch_size,
                                                  directory=train_dir,
                                                  shuffle=True,
                                                  target_size=(IMG_SHAPE,IMG_SHAPE),
                                                  class_mode='sparse'
                                                  )
```

```
Found 2935 images belonging to 5 classes.
```

```
[ ]  augmented_images = [train_data_gen[0][0][0] for i in range(5)]
     plotImages(augmented_images)
```



```
[ ]  image_gen_val = ImageDataGenerator(rescale=1./255)
     val_data_gen = image_gen_val.flow_from_directory(batch_size=batch_size,
                                                      directory=val_dir,
                                                      target_size=(IMG_SHAPE, IMG_SHAPE),
                                                      class_mode='sparse')
```

```
Found 735 images belonging to 5 classes.
```

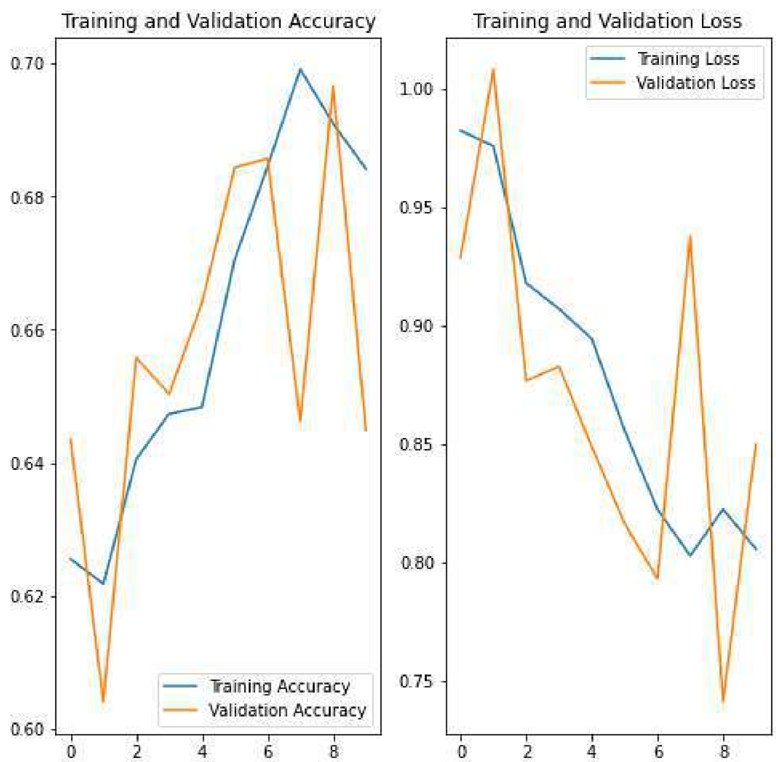
```
[ ] model = Sequential()

    model.add(Conv2D(16, 3, padding='same', activation='relu', input_shape=(IMG_SHAPE,IMG_SHAPE, 3)))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(32, 3, padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(64, 3, padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dropout(0.2))
    model.add(Dense(512, activation='relu'))

    model.add(Dropout(0.2))
    model.add(Dense(5))
```

```
[ ] # Compile the model
    model.compile(optimizer='adam',
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics=['accuracy'])
```

```
[ ] epochs = 10

    history = model.fit_generator(
        train_data_gen,
        steps_per_epoch=int(np.ceil(train_data_gen.n / float(batch_size))),
        epochs=epochs,
        validation_data=val_data_gen,
        validation_steps=int(np.ceil(val_data_gen.n / float(batch_size)))
    )
```

```
<ipython-input-24-2b255a50af7b>:3: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `M
  history = model.fit_generator(
Epoch 1/10
30/30 [==============================] - 116s 4s/step - loss: 0.9824 - accuracy: 0.6256 - val_loss: 0.9287 - val_accuracy: 0.6435
Epoch 2/10
30/30 [==============================] - 115s 4s/step - loss: 0.9759 - accuracy: 0.6218 - val_loss: 1.0084 - val_accuracy: 0.6041
Epoch 3/10
30/30 [==============================] - 125s 4s/step - loss: 0.9180 - accuracy: 0.6405 - val_loss: 0.8767 - val_accuracy: 0.6558
Epoch 4/10
30/30 [==============================] - 117s 4s/step - loss: 0.9071 - accuracy: 0.6474 - val_loss: 0.8828 - val_accuracy: 0.6503
Epoch 5/10
30/30 [==============================] - 116s 4s/step - loss: 0.8945 - accuracy: 0.6484 - val_loss: 0.8490 - val_accuracy: 0.6639
Epoch 6/10
30/30 [==============================] - 115s 4s/step - loss: 0.8558 - accuracy: 0.6705 - val_loss: 0.8165 - val_accuracy: 0.6844
Epoch 7/10
30/30 [==============================] - 118s 4s/step - loss: 0.8225 - accuracy: 0.6845 - val_loss: 0.7931 - val_accuracy: 0.6857
Epoch 8/10
30/30 [==============================] - 114s 4s/step - loss: 0.8029 - accuracy: 0.6991 - val_loss: 0.9379 - val_accuracy: 0.6463
Epoch 9/10
30/30 [==============================] - 116s 4s/step - loss: 0.8225 - accuracy: 0.6910 - val_loss: 0.7411 - val_accuracy: 0.6966
Epoch 10/10
30/30 [==============================] - 116s 4s/step - loss: 0.8057 - accuracy: 0.6842 - val_loss: 0.8499 - val_accuracy: 0.6449
```

```
[ ]  acc = history.history['accuracy']
     val_acc = history.history['val_accuracy']

     loss = history.history['loss']
     val_loss = history.history['val_loss']

     epochs_range = range(epochs)

     plt.figure(figsize=(8, 8))
     plt.subplot(1, 2, 1)
     plt.plot(epochs_range, acc, label='Training Accuracy')
     plt.plot(epochs_range, val_acc, label='Validation Accuracy')
     plt.legend(loc='lower right')
     plt.title('Training and Validation Accuracy')

     plt.subplot(1, 2, 2)
     plt.plot(epochs_range, loss, label='Training Loss')
     plt.plot(epochs_range, val_loss, label='Validation Loss')
     plt.legend(loc='upper right')
     plt.title('Training and Validation Loss')
     plt.show()
```



Conclusion: Implementation was successful.

EXPERIMENT 4

Aim: design a ConvNet for classifying dog and cat images

Software used: Google collab

Theory:

A convolutional neural network (CNN or ConvNet) is a network architecture for deep learning that learns directly from data. CNNs are particularly useful for finding patterns in images to recognize objects, classes, and categories. They can also be quite effective for classifying audio, time-series, and signal data.

Code and Output:

In [14]:
```
#importing the packages
import tensorflow as tf
import os
import re
import numpy as np
import zipfile
import matplotlib.pyplot as plt

from tensorflow.contrib import learn
from tensorflow.contrib.learn.python.learn.estimators import model_fn
from tensorflow.contrib.learn import RunConfig as run_config
from keras.callbacks import TensorBoard
slim = tf.contrib.slim
```

In [15]:
```
# Here 0 means Cat and 1 means Dog
CAT = 0
DOG = 1
#Returns whether the low memory mode is used.
IS_LOW_MEMORY_MODE = True
#current working directory of a process.
cwd = os.getcwd()
#This method is called when RandomState is initialized
np.random.seed(2124)
```

In [25]:
```
#Method to prepare a file
def prepare_file():
  file_list = ['train','test']
  flag = True
  for i in range(len(file_list)):
    filename = file_list[i] + '.zip'
    dest_filename = os.path.join(cwd,'data',filename)

    images_path = os.path.join(cwd,'data',file_list[1])
    zip = zipfile.ZipFile(dest_filename)

  return flag
```

In [28]:
```
#Method to read the image label
def read_image_label_list(folder_dir):
    dir_list = os.listdir(os.path.join(cwd,folder_dir))
    filenames = []
    labels = []

    for i, d in enumerate(dir_list):
        if re.search("train",folder_dir):
            if re.search("cat", d):
                labels.append(CAT)
            else:
                labels.append(DOG)
        else:
            labels.append(-1)
        filenames.append(os.path.join(cwd, folder_dir, d))
    return filenames, labels
```

In [29]:

```python
#Method to read the image from disk
def read_images_from_disk(input_queue):
    filename = input_queue[0]
    label = input_queue[1]

    file_contents = tf.read_file(filename)
    image = tf.image.decode_image(file_contents, channels=3)
    image.set_shape([None, None, 3])

    return image, label
```

In [30]:

```python
#Method to generate input function
def gen_input_fn(image_list, label_list, batch_size, shuffle):

    def input_fn():
        images = tf.convert_to_tensor(image_list, dtype=tf.string)
        labels = tf.convert_to_tensor(label_list, dtype=tf.int32)

        input_queue = tf.train.slice_input_producer(
            [images, labels],
            capacity=batch_size * 5,
            shuffle=shuffle,
            name="file_input_queue"
        )

        image, label = read_images_from_disk(input_queue)
        image = tf.image.resize_images(image, (224, 224), tf.image.ResizeMethod.NEAREST_NEIGHBOR)

        image_batch, label_batch = tf.train.batch(
            [image, label],
            batch_size=batch_size,
            num_threads=1,
            name="batch_queue",
            capacity=batch_size * 10,
            allow_smaller_final_batch = False
        )

        return (
            tf.identity(image_batch, name="features"),
            tf.identity(label_batch, name="label")
        )

    return input_fn
```

In [42]:

```python
#Method to train a valid input function
def train_valid_input_fn(data_dir, train_batch_size, valid_batch_size=None):
    img, labels = read_image_label_list(data_dir)
    img = np.array(img)
    labels = np.array(labels)
    data_size = img.shape[0]

    print("Data size: " + str(data_size))
    split = int(0.7 * data_size)

    random_seq = np.random.permutation(data_size)

    img = img[random_seq]
    labels = labels[random_seq]

    if valid_batch_size == None:
        valid_batch_size = train_batch_size

    return (
        gen_input_fn(img[0:split], labels[0:split], train_batch_size, shuffle = True),
        gen_input_fn(img[split:], labels[split:], valid_batch_size, shuffle = False)
          )
```

In [32]:

```python
#Method to test input function
def test_input_fn(data_dir,batch_size):
    image_list, label_list = read_image_label_list(data_dir)
    return gen_input_fn(image_list, label_list, batch_size, shuffle = False), image_list
```

In [33]:

```
if prepare_file():
    print("Files completed!")
```

```
Files completed!
```

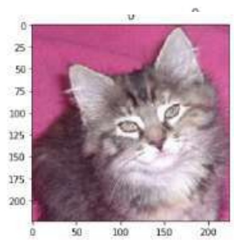
In [34]:

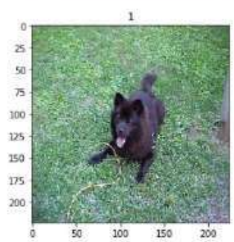
```
#Method to plot data
def plot_img(data, label=None):
    plt.ion()
    plt.figure()
    plt.imshow(data)
    if label is not None:
        plt.title(label)
```

In [35]:

```
#Method to preview data
def preview_img():

    img_preview = tf.Graph()

    with img_preview.as_default():
        tensor_train, _ = train_valid_input_fn('data/train', 5)
        result = tf.tuple(tensor_train())

    with tf.Session(graph=img_preview) as sess:
        sess.run(tf.global_variables_initializer())
        coord = tf.train.Coordinator()
        threads = tf.train.start_queue_runners(coord=coord)

        images, labels = sess.run(result)
        for i in range(len(images)):
            plot_img(images[i], str(labels[i]))

        coord.request_stop()
        coord.join(threads)

        sess.close()

preview_img()
```

```
Data size: 25000
```





Define Model

In [36]:

```python
#Cat-Dog Method Declaration
def catdog_model(inputs, is_training):
    with tf.variable_scope('catdog', values=[inputs]):
        with slim.arg_scope(
            [slim.conv2d, slim.fully_connected],
            activation_fn=tf.nn.relu6,
            weights_initializer=tf.truncated_normal_initializer(0.0, 0.01)):

            net = inputs

            if IS_LOW_MEMORY_MODE == False:
                net = slim.repeat(net, 2, slim.conv2d, 64, [3, 3], scope='conv1')
                net = slim.max_pool2d(net, [2, 2], scope='pool1')

                net = slim.repeat(net, 2, slim.conv2d, 128, [3, 3], scope='conv2')
                net = slim.max_pool2d(net, [2, 2], scope='pool2')

                net = slim.repeat(net, 4, slim.conv2d, 256, [3, 3], scope='conv3')
                net = slim.max_pool2d(net, [2, 2], scope='pool3')
                net = slim.repeat(net, 4, slim.conv2d, 512, [3, 3], scope='conv4')
                net = slim.max_pool2d(net, [2, 2], scope='pool4')
                net = slim.repeat(net, 4, slim.conv2d, 512, [3, 3], scope='conv5')
                net = slim.max_pool2d(net, [2, 2], scope='pool5')

                net = tf.reshape(net, [-1, 7 * 7 * 512])

                net = slim.fully_connected(net, 2048, scope='fc6')
                net = slim.dropout(net, 0.5, is_training=is_training, scope='dropout6')

                net = slim.fully_connected(net, 2048, scope='fc7')
                net = slim.dropout(net, 0.5, is_training=is_training, scope='dropout7')

                net = slim.fully_connected(net, 2, activation_fn=None, scope='fc8')

            else:
                # Model for my Mac T_T
                net = tf.image.resize_images(net, (72, 72), tf.image.ResizeMethod.NEAREST_NEIGHBOR)

                net = slim.repeat(net, 1, slim.conv2d, 64, [3, 3], scope='conv1')
                net = slim.max_pool2d(net, [2, 2], scope='pool1')

                net = slim.repeat(net, 1, slim.conv2d, 128, [3, 3], scope='conv2')
                net = slim.max_pool2d(net, [2, 2], scope='pool2')

                net = slim.repeat(net, 2, slim.conv2d, 256, [3, 3], scope='conv3')
                net = slim.max_pool2d(net, [2, 2], scope='pool3')

                net = tf.reshape(net, [-1, 9 * 9 * 256])

                net = slim.fully_connected(net, 1024, scope='fc4')
                net = slim.dropout(net, 0.5, is_training=is_training, scope='dropout4')
                net = slim.dropout(net, 0.5, is_training=is_training, scope='dropout4')

                net = slim.fully_connected(net, 1024, scope='fc5')
                net = slim.dropout(net, 0.5, is_training=is_training, scope='dropout5')

                net = slim.fully_connected(net, 2, activation_fn=None, scope='fc6')

        return net
```

In [37]:

```python
#Cat-Dog Model function
def catdog_model_fn(features, labels, mode, params):

    is_training = False
    if mode == learn.ModeKeys.TRAIN:
        is_training = True

    output = catdog_model(features, is_training)

    log_loss = None
    train_op = None
    eval_metric_ops = None

    softmax_predictions = tf.nn.softmax(output)

    if mode != learn.ModeKeys.INFER:
        onehot_labels = tf.one_hot(
            tf.cast(labels, tf.int32),
            depth = 2
        )
        log_loss = tf.identity(
            tf.losses.log_loss(
                onehot_labels,
                tf.nn.softmax(output),
                reduction = tf.losses.Reduction.MEAN
            ),
            name = "log_loss_tensor"
        )
        eval_metric_ops = {
            "log_loss": log_loss
        }

    if mode == learn.ModeKeys.TRAIN:
        train_op = tf.contrib.layers.optimize_loss(
            loss = log_loss,
            global_step = tf.contrib.framework.get_global_step(),
            learning_rate = params['learning_rate'],
            optimizer = "Adam"
        )

    predictions = {
        'predict': softmax_predictions
    }
```

```python
    return model_fn.ModelFnOps(
        mode = mode,
        predictions = predictions,
        loss = log_loss,
        train_op = train_op,
        eval_metric_ops = eval_metric_ops
    )
```

In [38]:

```python
#Feature Engineering function
def feature_engineering_fn(features, labels):
    features = tf.to_float(features)
    features = tf.map_fn(tf.image.per_image_standardization, features)
    return features, labels

tf.logging.set_verbosity(tf.logging.ERROR)

model_path = 'model' if IS_LOW_MEMORY_MODE else 'model'
classifier = learn.Estimator(
    model_fn = catdog_model_fn,
    model_dir = model_path,
    config = run_config(
        save_summary_steps = 10,
        keep_checkpoint_max = 3,
        save_checkpoints_steps = 75
    ),
    feature_engineering_fn = feature_engineering_fn,
    params = {
        'learning_rate': 0.01
    }
)

train_input_fn, validate_input_fn = train_valid_input_fn('data/train', 32, 64)

logging_hook = tf.train.LoggingTensorHook(
    tensors = {
        'log_loss': 'log_loss_tensor'
    },
    every_n_iter = 3
)

validation_monitor = tf.contrib.learn.monitors.ValidationMonitor(
    input_fn = validate_input_fn,
    eval_steps = 30,
    every_n_steps = 100,
    name = 'Validatation'
)
```

```
Data size: 25000
```

In [39]:

```
tf.logging.set_verbosity(tf.logging.INFO)
classifier.fit(
    input_fn = train_input_fn,
    steps = 100,
    monitors = [logging_hook, validation_monitor]
)
```

```
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Restoring parameters from model/model.ckpt-200
INFO:tensorflow:Saving checkpoints for 201 into model/model.ckpt.
INFO:tensorflow:log_loss = 6.5479765
INFO:tensorflow:Starting evaluation at 2018-02-06-15:59:45
INFO:tensorflow:Restoring parameters from model/model.ckpt-201
INFO:tensorflow:Evaluation [1/30]
INFO:tensorflow:Evaluation [2/30]
INFO:tensorflow:Evaluation [3/30]
INFO:tensorflow:Evaluation [4/30]
INFO:tensorflow:Evaluation [5/30]
INFO:tensorflow:Evaluation [6/30]
INFO:tensorflow:Evaluation [7/30]
INFO:tensorflow:Evaluation [8/30]
INFO:tensorflow:Evaluation [9/30]
INFO:tensorflow:Evaluation [10/30]
INFO:tensorflow:Evaluation [11/30]
INFO:tensorflow:Evaluation [12/30]
INFO:tensorflow:Evaluation [13/30]
INFO:tensorflow:Evaluation [14/30]
INFO:tensorflow:Evaluation [15/30]
INFO:tensorflow:Evaluation [16/30]
INFO:tensorflow:Evaluation [17/30]
INFO:tensorflow:Evaluation [18/30]
INFO:tensorflow:Evaluation [19/30]
INFO:tensorflow:Evaluation [20/30]
INFO:tensorflow:Evaluation [21/30]
INFO:tensorflow:Evaluation [22/30]
INFO:tensorflow:Evaluation [23/30]
INFO:tensorflow:Evaluation [24/30]
INFO:tensorflow:Evaluation [25/30]
INFO:tensorflow:Evaluation [26/30]
INFO:tensorflow:Evaluation [27/30]
INFO:tensorflow:Evaluation [28/30]
INFO:tensorflow:Evaluation [29/30]
INFO:tensorflow:Evaluation [30/30]
INFO:tensorflow:Finished evaluation at 2018-02-06-16:01:05
INFO:tensorflow:Saving dict for global step 201: global_step = 201, log_loss = 7.555359, loss = 8.210157
INFO:tensorflow:Validation (step 201): log_loss = 7.555359, loss = 8.210157, global_step = 201
INFO:tensorflow:loss = 6.5479765, step = 201
INFO:tensorflow:log_loss = 7.555357 (92.375 sec)
INFO:tensorflow:log_loss = 8.059048 (11.834 sec)
```

ut[39]: Estimator(params={'learning_rate': 0.01})

n [40]:

```
#Model Evaluation
classifier.evaluate(
    input_fn = validate_input_fn,
    steps = 75
)
```

```
INFO:tensorflow:Starting evaluation at 2018-02-06-16:08:01
INFO:tensorflow:Restoring parameters from model/model.ckpt-300
INFO:tensorflow:Evaluation [1/75]
INFO:tensorflow:Evaluation [2/75]
INFO:tensorflow:Evaluation [3/75]
INFO:tensorflow:Evaluation [4/75]
INFO:tensorflow:Evaluation [5/75]
INFO:tensorflow:Evaluation [6/75]
INFO:tensorflow:Evaluation [7/75]
INFO:tensorflow:Evaluation [8/75]
INFO:tensorflow:Evaluation [9/75]
INFO:tensorflow:Evaluation [10/75]
INFO:tensorflow:Evaluation [11/75]
INFO:tensorflow:Evaluation [12/75]
INFO:tensorflow:Evaluation [13/75]
INFO:tensorflow:Evaluation [14/75]
INFO:tensorflow:Evaluation [15/75]
INFO:tensorflow:Evaluation [16/75]
INFO:tensorflow:Evaluation [17/75]
INFO:tensorflow:Evaluation [18/75]
INFO:tensorflow:Evaluation [19/75]
INFO:tensorflow:Evaluation [20/75]
INFO:tensorflow:Evaluation [21/75]
INFO:tensorflow:Evaluation [22/75]
INFO:tensorflow:Evaluation [23/75]
INFO:tensorflow:Evaluation [24/75]
INFO:tensorflow:Evaluation [25/75]
INFO:tensorflow:Evaluation [26/75]
INFO:tensorflow:Evaluation [27/75]
INFO:tensorflow:Evaluation [28/75]
INFO:tensorflow:Evaluation [29/75]
INFO:tensorflow:Evaluation [30/75]
INFO:tensorflow:Evaluation [31/75]
INFO:tensorflow:Evaluation [32/75]
INFO:tensorflow:Evaluation [33/75]
INFO:tensorflow:Evaluation [34/75]
```

```
INFO:tensorflow:Evaluation [75/75]
INFO:tensorflow:Finished evaluation at 2018-02-06-16:11:15
INFO:tensorflow:Saving dict for global step 300: global_step = 300, log_loss = 9.066431, loss = 8.163145
```

Out[40]:

```
{'global_step': 300, 'log_loss': 9.066431, 'loss': 8.163145}
```

Final Prediction

In [41]:

```
#Print the result
test_fn, image_test_list = test_input_fn('data/test',32)
test_n = len(image_test_list)

print("Test size: %d" % test_n)

result_file = open(os.path.join(cwd, 'result/result.txt'),'w+')
result_file.write('id,label\n')

predictions = classifier.predict(input_fn = test_fn, as_iterable=True)
for i, p in enumerate(predictions):
    if i >= test_n:
        break

    id = image_test_list[i].split("/")[-1]
    id = id.split(".")[0]

    if i % 100 == 0:
        print("Predict %d %s: %f" % (i,image_test_list[i],p["predict"][1]))

    result_file.write("%s,%f\n" % (id, p["predict"][1]))

result_file.flush()
result_file.close()
print('Finish!!')
```

```
Test size: 12500
INFO:tensorflow:Restoring parameters from model/model.ckpt-300
Predict 0 /home/kunal/Documents/Data_Science_Practice/cat-vs-dog-classification-master/src/Graph/data/test/361.jpg: 1.000000
Predict 100 /home/kunal/Documents/Data_Science_Practice/cat-vs-dog-classification-master/src/Graph/data/test/7579.jpg: 1.000000
Predict 200 /home/kunal/Documents/Data_Science_Practice/cat-vs-dog-classification-master/src/Graph/data/test/2517.jpg: 1.000000
Predict 300 /home/kunal/Documents/Data_Science_Practice/cat-vs-dog-classification-master/src/Graph/data/test/84.jpg: 1.000000
Predict 400 /home/kunal/Documents/Data_Science_Practice/cat-vs-dog-classification-master/src/Graph/data/test/1895.jpg: 1.000000
Predict 500 /home/kunal/Documents/Data_Science_Practice/cat-vs-dog-classification-master/src/Graph/data/test/4914.jpg: 1.000000
Predict 600 /home/kunal/Documents/Data_Science_Practice/cat-vs-dog-classification-master/src/Graph/data/test/9553.jpg: 1.000000
Predict 700 /home/kunal/Documents/Data_Science_Practice/cat-vs-dog-classification-master/src/Graph/data/test/4363.jpg: 1.000000
Predict 800 /home/kunal/Documents/Data_Science_Practice/cat-vs-dog-classification-master/src/Graph/data/test/10603.jpg: 1.000000
Predict 900 /home/kunal/Documents/Data_Science_Practice/cat-vs-dog-classification-master/src/Graph/data/test/8116.jpg: 1.000000
Predict 1000 /home/kunal/Documents/Data_Science_Practice/cat-vs-dog-classification-master/src/Graph/data/test/2670.jpg: 1.000000
Predict 1100 /home/kunal/Documents/Data_Science_Practice/cat-vs-dog-classification-master/src/Graph/data/test/1503.jpg: 1.000000
Predict 1200 /home/kunal/Documents/Data_Science_Practice/cat-vs-dog-classification-master/src/Graph/data/test/4862.jpg: 1.000000
Predict 1300 /home/kunal/Documents/Data_Science_Practice/cat-vs-dog-classification-master/src/Graph/data/test/2896.jpg: 1.000000
Predict 1400 /home/kunal/Documents/Data_Science_Practice/cat-vs-dog-classification-master/src/Graph/data/test/716.jpg: 1.000000
Predict 1500 /home/kunal/Documents/Data_Science_Practice/cat-vs-dog-classification-master/src/Graph/data/test/1924.jpg: 1.000000
Predict 1600 /home/kunal/Documents/Data_Science_Practice/cat-vs-dog-classification-master/src/Graph/data/test/7924.jpg: 1.000000
```

Conclusion: Implementation was successful.

EXPERIMENT 5

Aim: image classification using transfer learning for dogs vs cats dataset

Software used: Google collab

Theory:

Transfer learning means taking the relevant parts of a pre-trained machine learning model and applying it to a new but similar problem. This will usually be the core information for the model to function, with new aspects added to the model to solve a specific task.

Code and Output:

In [1]:
```
import os, cv2, random
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from tqdm import tqdm
from random import shuffle
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
from keras.utils import plot_model
from tensorflow.python.keras.applications import ResNet50
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense, Flatten, GlobalAveragePooling2D
%matplotlib inline
```

In [2]:
```
TEST_SIZE = 0.5
RANDOM_STATE = 2018
BATCH_SIZE = 64
NO_EPOCHS = 20
NUM_CLASSES = 2
SAMPLE_SIZE = 20000
PATH = '/kaggle/input/dogs-vs-cats-redux-kernels-edition/'
TRAIN_FOLDER = './train/'
TEST_FOLDER =  './test/'
IMG_SIZE = 224
RESNET_WEIGHTS_PATH = '/kaggle/input/resnet50/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5'
```

In [3]:
```
train_image_path = os.path.join(PATH, "train.zip")
test_image_path = os.path.join(PATH, "test.zip")
```

In [4]:
```
import zipfile
with zipfile.ZipFile(train_image_path,"r") as z:
    z.extractall(".")
```

In [5]:
```
with zipfile.ZipFile(test_image_path,"r") as z:
    z.extractall(".")
```

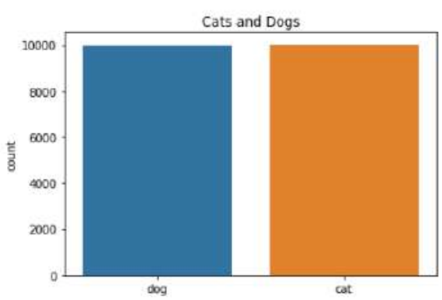
In [6]:
```
train_image_list = os.listdir("./train/")[0:SAMPLE_SIZE]
test_image_list = os.listdir("./test/")
```

In [8]:

```
def process_data(data_image_list, DATA_FOLDER, isTrain=True):
    data_df = []
    for img in tqdm(data_image_list):
        path = os.path.join(DATA_FOLDER, img)
        if(isTrain):
            label = label_pet_image_one_hot_encoder(img)
        else:
            label = img.split('.')[0]
        img = cv2.imread(path,cv2.IMREAD_COLOR)
        img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
        data_df.append([np.array(img),np.array(label)])
    shuffle(data_df)
    return data_df
```

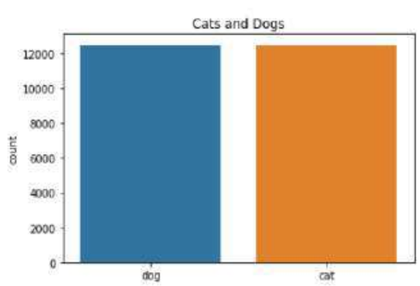
In [9]:

```
def plot_image_list_count(data_image_list):
    labels = []
    for img in data_image_list:
        labels.append(img.split('.')[-3])
    sns.countplot(labels)
    plt.title('Cats and Dogs')

plot_image_list_count(train_image_list)
```



In [10]:

```
plot_image_list_count(os.listdir(TRAIN_FOLDER))
```



In [11]:

```
train = process_data(train_image_list, TRAIN_FOLDER)
```

```
100%|██████████| 20000/20000 [00:43<00:00, 457.45it/s]
```

Then, we plot the image selection.

In [12]:

```
def show_images(data, isTest=False):
    f, ax = plt.subplots(5,5, figsize=(15,15))
    for i,data in enumerate(data[:25]):
        img_num = data[1]
        img_data = data[0]
        label = np.argmax(img_num)
        if label  == 1:
            str_label='Dog'
        elif label == 0:
            str_label='Cat'
        if(isTest):
            str_label="None"
        ax[i//5, i%5].imshow(img_data)
        ax[i//5, i%5].axis('off')
        ax[i//5, i%5].set_title("Label: {}".format(str_label))
    plt.show()

show_images(train)
```

In [13]:

```
test = process_data(test_image_list, TEST_FOLDER, False)
```

```
100%|██████████| 12500/12500 [00:26<00:00, 476.31it/s]
```

Then, we show a selection of the test set.

In [14]:

```
show_images(test,True)
```

In [15]:

```
X = np.array([i[0] for i in train]).reshape(-1,IMG_SIZE,IMG_SIZE,3)
y = np.array([i[1] for i in train])
```

In [16]:

```
model = Sequential()
model.add(ResNet50(include_top=False, pooling='max', weights=RESNET_WEIGHTS_PATH))
model.add(Dense(NUM_CLASSES, activation='softmax'))
# ResNet-50 model is already trained, should not be trained
model.layers[0].trainable = True
```

In [17]:

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

In [18]:

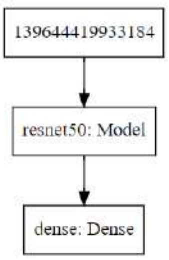
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Model)             (None, 2048)              23587712
_________________________________________________________________
dense (Dense)                (None, 2)                 4098
=================================================================
Total params: 23,591,810
Trainable params: 23,538,690
Non-trainable params: 53,120
_________________________________________________________________
```

In [19]:

```
plot_model(model, to_file='model.png')
SVG(model_to_dot(model).create(prog='dot', format='svg'))
```

Out[19]:



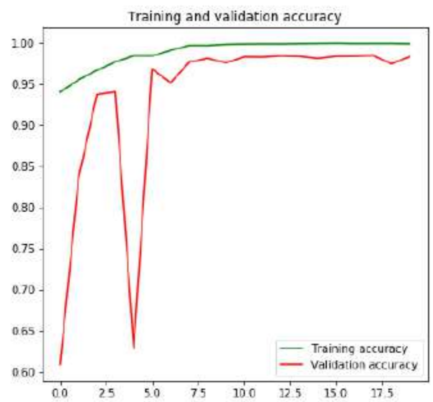
In [21]:

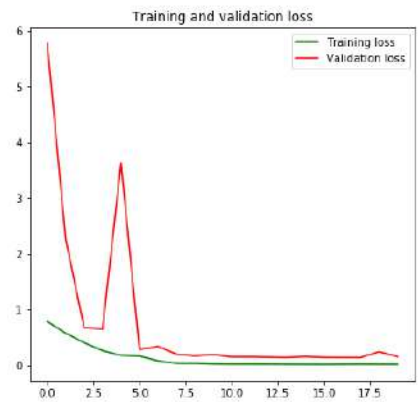
```
train_model = model.fit(X_train, y_train,
                  batch_size=BATCH_SIZE,
                  epochs=NO_EPOCHS,
                  verbose=1,
                  validation_data=(X_val, y_val))
```

```
Train on 10000 samples, validate on 10000 samples
Epoch 1/20
10000/10000 [==============================] - 105s 11ms/step - loss: 0.7823 - acc: 0.9403 - val_
loss: 5.7698 - val_acc: 0.6090
Epoch 2/20
10000/10000 [==============================] - 93s 9ms/step - loss: 0.5717 - acc: 0.9551 - val_lo
ss: 2.2545 - val_acc: 0.8356
Epoch 3/20
10000/10000 [==============================] - 93s 9ms/step - loss: 0.4005 - acc: 0.9667 - val_lo
ss: 0.6695 - val_acc: 0.9375
Epoch 4/20
10000/10000 [==============================] - 93s 9ms/step - loss: 0.2591 - acc: 0.9768 - val_lo
ss: 0.6475 - val_acc: 0.9405
Epoch 5/20
10000/10000 [==============================] - 93s 9ms/step - loss: 0.1732 - acc: 0.9843 - val_lo
ss: 3.6218 - val_acc: 0.6283
```

In [22]:

```
def plot_accuracy_and_loss(train_model):
    hist = train_model.history
    acc = hist['acc']
    val_acc = hist['val_acc']
    loss = hist['loss']
    val_loss = hist['val_loss']
    epochs = range(len(acc))
    f, ax = plt.subplots(1,2, figsize=(14,6))
    ax[0].plot(epochs, acc, 'g', label='Training accuracy')
    ax[0].plot(epochs, val_acc, 'r', label='Validation accuracy')
    ax[0].set_title('Training and validation accuracy')
    ax[0].legend()
    ax[1].plot(epochs, loss, 'g', label='Training loss')
    ax[1].plot(epochs, val_loss, 'r', label='Validation loss')
    ax[1].set_title('Training and validation loss')
    ax[1].legend()
    plt.show()
plot_accuracy_and_loss(train_model)
```





In [23]:

```
score = model.evaluate(X_val, y_val, verbose=0)
print('Validation loss:', score[0])
print('Validation accuracy:', score[1])
```

```
Validation loss: 0.15023201193418587
Validation accuracy: 0.9832
```

Conclusion: Implementation was successful.

EXPERIMENT 6

Aim: implement Action Recognition with an Inflated 3D CNN

Software used: Google collab

Theory:

The architecture of inflated 3D CNN model goes something like this – input is a video, 3D input as in 2-dimensional frame with time as the third dimension. It contains Convolutional(CNN) layers with stride 2, after which there is a max-pooling layer and multiple Inception modules (conv. Layers with one max pooling layer, concatenation is the main task). Inflated because of the reason that we are having these modules (described in the paper) dilated into the middle of the model. These modules can have different mini architectures in them like LSTM.

Code and Output:

```
!pip install -q imageio
!pip install -q opencv-python
!pip install -q git+https://github.com/tensorflow/docs
```

```
Preparing metadata (setup.py) ... done
Building wheel for tensorflow-docs (setup.py) ... done
```

▾ Import the necessary modules

```
[2] #@title Import the necessary modules
# TensorFlow and TF-Hub modules.
from absl import logging

import tensorflow as tf
import tensorflow_hub as hub
from tensorflow_docs.vis import embed

logging.set_verbosity(logging.ERROR)

# Some modules to help with reading the UCF101 dataset.
import random
import re
import os
import tempfile
import ssl
import cv2
import numpy as np

# Some modules to display an animation using imageio.
import imageio
from IPython import display

from urllib import request  # requires python3
```

```
[3] #@title Helper functions for the UCF101 dataset

# Utilities to fetch videos from UCF101 dataset
UCF_ROOT = "https://www.crcv.ucf.edu/THUMOS14/UCF101/UCF101/"
_VIDEO_LIST = None
_CACHE_DIR = tempfile.mkdtemp()
# As of July 2020, crcv.ucf.edu doesn't use a certificate accepted by the
# default Colab environment anymore.
unverified_context = ssl._create_unverified_context()

def list_ucf_videos():
  """Lists videos available in UCF101 dataset."""
  global _VIDEO_LIST
  if not _VIDEO_LIST:
    index = request.urlopen(UCF_ROOT, context=unverified_context).read().decode("utf-8")
    videos = re.findall("(v_[\w_]+\.avi)", index)
    _VIDEO_LIST = sorted(set(videos))
  return list(_VIDEO_LIST)

def fetch_ucf_video(video):
  """Fetchs a video and cache into local filesystem."""
  cache_path = os.path.join(_CACHE_DIR, video)
  if not os.path.exists(cache_path):
    urlpath = request.urljoin(UCF_ROOT, video)
    print("Fetching %s => %s" % (urlpath, cache_path))
    data = request.urlopen(urlpath, context=unverified_context).read()
    open(cache_path, "wb").write(data)
  return cache_path

# Utilities to open video files using CV2
def crop_center_square(frame):
  y, x = frame.shape[0:2]
  min_dim = min(y, x)
  start_x = (x // 2) - (min_dim // 2)
  start_y = (y // 2) - (min_dim // 2)
  return frame[start_y:start_y+min_dim,start_x:start_x+min_dim]

def load_video(path, max_frames=0, resize=(224, 224)):
  cap = cv2.VideoCapture(path)
```

```
  cap = cv2.VideoCapture(path)
  frames = []
  try:
    while True:
      ret, frame = cap.read()
      if not ret:
        break
      frame = crop_center_square(frame)
      frame = cv2.resize(frame, resize)
      frame = frame[:, :, [2, 1, 0]]
      frames.append(frame)

      if len(frames) == max_frames:
        break
  finally:
    cap.release()
  return np.array(frames) / 255.0

def to_gif(images):
  converted_images = np.clip(images * 255, 0, 255).astype(np.uint8)
  imageio.mimsave('./animation.gif', converted_images, fps=25)
  return embed.embed_file('./animation.gif')
```

Get the kinetics-400 labels

```
#@title Get the kinetics-400 labels
# Get the kinetics-400 action labels from the GitHub repository.
KINETICS_URL = "https://raw.githubusercontent.com/deepmind/kinetics-i3d/master/data/label_map.txt"
with request.urlopen(KINETICS_URL) as obj:
  labels = [line.decode("utf-8").strip() for line in obj.readlines()]
print("Found %d labels." % len(labels))
```

```
Found 400 labels.
```

```
[5] # Get the list of videos in the dataset.
    ucf_videos = list_ucf_videos()

    categories = {}
    for video in ucf_videos:
      category = video[2:-12]
      if category not in categories:
        categories[category] = []
      categories[category].append(video)
    print("Found %d videos in %d categories." % (len(ucf_videos), len(categories)))

    for category, sequences in categories.items():
      summary = ", ".join(sequences[:2])
      print("%-20s %4d videos (%s, ...)" % (category, len(sequences), summary))
```

```
IceDancing           158 videos (v_IceDancing_g01_c01.avi, v_IceDancing_g01_c02.avi, ...)
JavelinThrow         117 videos (v_JavelinThrow_g01_c01.avi, v_JavelinThrow_g01_c02.avi, ...)
JugglingBalls        121 videos (v_JugglingBalls_g01_c01.avi, v_JugglingBalls_g01_c02.avi, ...)
JumpRope             144 videos (v_JumpRope_g01_c01.avi, v_JumpRope_g01_c02.avi, ...)
JumpingJack          123 videos (v_JumpingJack_g01_c01.avi, v_JumpingJack_g01_c02.avi, ...)
Kayaking             141 videos (v_Kayaking_g01_c01.avi, v_Kayaking_g01_c02.avi, ...)
Knitting             123 videos (v_Knitting_g01_c01.avi, v_Knitting_g01_c02.avi, ...)
LongJump             131 videos (v_LongJump_g01_c01.avi, v_LongJump_g01_c02.avi, ...)
Lunges               127 videos (v_Lunges_g01_c01.avi, v_Lunges_g01_c02.avi, ...)
MilitaryParade       125 videos (v_MilitaryParade_g01_c01.avi, v_MilitaryParade_g01_c02.avi, ...)
Mixing               136 videos (v_Mixing_g01_c01.avi, v_Mixing_g01_c02.avi, ...)
MoppingFloor         110 videos (v_MoppingFloor_g01_c01.avi, v_MoppingFloor_g01_c02.avi, ...)
Nunchucks            132 videos (v_Nunchucks_g01_c01.avi, v_Nunchucks_g01_c02.avi, ...)
```

```
[6] # Get a sample cricket video.
    video_path = fetch_ucf_video("v_CricketShot_g04_c02.avi")
    sample_video = load_video(video_path)
```

```
Fetching https://www.crcv.ucf.edu/THUMOS14/UCF101/UCF101/v_CricketShot_g04_c02.avi => /tmp/tmpghywt42l/v_CricketShot_g04_c02.avi
```

```
[7] sample_video.shape
```

```
(116, 224, 224, 3)
```

```
[8] i3d = hub.load("https://tfhub.dev/deepmind/i3d-kinetics-400/1").signatures['default']
```

Run the id3 model and print the top-5 action predictions.

```
[9] def predict(sample_video):
      # Add a batch axis to the sample video.
      model_input = tf.constant(sample_video, dtype=tf.float32)[tf.newaxis, ...]

      logits = i3d(model_input)['default'][0]
      probabilities = tf.nn.softmax(logits)

      print("Top 5 actions:")
      for i in np.argsort(probabilities)[::-1][:5]:
        print(f"  {labels[i]:22}: {probabilities[i] * 100:5.2f}%")
```

```
predict(sample_video)
```

```
Top 5 actions:
  playing cricket      : 97.77%
  skateboarding        :  0.71%
  robot dancing        :  0.56%
  roller skating       :  0.56%
  golf putting         :  0.13%
```

```
[11] !curl -O https://upload.wikimedia.org/wikipedia/commons/8/86/End_of_a_jam.ogv
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 55.0M  100 55.0M    0     0  24.7M      0  0:00:02  0:00:02 --:--:-- 24.7M
```

```
[12] video_path = "End_of_a_jam.ogv"
```

```
[13] sample_video = load_video(video_path)[:100]
     sample_video.shape
```

```
(100, 224, 224, 3)
```

```
[14] to_gif(sample_video)
```

```
[15] predict(sample_video)
```

```
Top 5 actions:
  roller skating       : 96.85%
  playing volleyball   :  1.63%
  skateboarding        :  0.21%
  playing ice hockey   :  0.20%
  playing basketball   :  0.16%
```

Conclusion: Implementation was successful.

EXPERIMENT 7

Aim: implement Object-Detection-using-CNN

Software used: Google collab

Code and Output:

1. import tensorflow as tf
2. import tensorflow_hub as hub
3. import matplotlib.pyplot as plt
4. import tempfile
5. from six.moves.urllib.request import urlopen
6. from six import BytesIO
7. import numpy as np
8. from PIL import Image
9. from PIL import ImageColor
10. from PIL import ImageDraw
11. from PIL import ImageFont
12. from PIL import ImageOps
13. import time
14. print(tf.__version__)
15. print("The following GPU devices are available: %s" % tf.test.gpu_device_name())
16. 2.12.0
17. The following GPU devices are available:
18. def display_image(image):
19. fig = plt.figure(figsize=(20, 15))
20. plt.grid(False)
21. plt.imshow(image)
22. def download_and_resize_image(url, new_width=256, new_height=256, display=False):
23. _, filename = tempfile.mkstemp(suffix=".jpg")
24. response = urlopen(url)
25. image_data = response.read()
26. image_data = BytesIO(image_data)
27. pil_image = Image.open(image_data)
28. pil_image = ImageOps.fit(pil_image, (new_width, new_height), Image.ANTIALIAS)
29. pil_image_rgb = pil_image.convert("RGB")
30. pil_image_rgb.save(filename, format="JPEG", quality=90)
31. print("Image downloaded to %s." % filename)
32. if display:
33. display_image(pil_image)
34. return filename
35. def draw_bounding_box_on_image(image, ymin, xmin, ymax, xmax, color, font, thickness=4, display_str_list=()):
36. draw = ImageDraw.Draw(image)
37. im_width, im_height = image.size
38. (left, right, top, bottom) = (xmin * im_width, xmax * im_width, ymin * im_height, ymax * im_height)

39. draw.line([(left, top), (left, bottom), (right, bottom), (right, top), (left, top)], width=thickness, fill=color)
40. display_str_heights = [font.getsize(ds)[1] for ds in display_str_list]
41. total_display_str_height = (1 + 2 * 0.05) * sum(display_str_heights)
42. if top > total_display_str_height:
43. text_bottom = top
44. else:
45. text_bottom = top + total_display_str_height
46. for display_str in display_str_list[::-1]:
47. text_width, text_height = font.getsize(display_str)
48. margin = np.ceil(0.05 * text_height)
49. draw.rectangle([(left, text_bottom - text_height - 2 * margin), (left + text_width, text_bottom)], fill=color)
50. draw.text((left + margin, text_bottom - text_height - margin), display_str, fill="black", font=font)
51. text_bottom -= text_height - 2 * margin
52. def draw_boxes(image, boxes, class_names, scores, max_boxes=10, min_score=0.1):
53. """Overlay labeled boxes on an image with formatted scores and label names."""
54. colors = list(ImageColor.colormap.values())
55. try:
56. font = ImageFont.truetype("/usr/share/fonts/truetype/liberation/LiberationSansNarrow-Regular.ttf", 25)
57. except IOError:
58. print("Font not found, using default font.")
59. font = ImageFont.load_default()
60. for i in range(min(boxes.shape[0], max_boxes)):
61. if scores[i] >= min_score:
62. ymin, xmin, ymax, xmax = tuple(boxes[i])
63. display_str = "{}: {}%".format(class_names[i].decode("ascii"), int(100 * scores[i]))
64. color = colors[hash(class_names[i]) % len(colors)]
65. image_pil = Image.fromarray(np.uint8(image)).convert("RGB")
66. draw_bounding_box_on_image(image_pil, ymin, xmin, ymax, xmax, color, font, display_str_list=[display_str])
67. np.copyto(image, np.array(image_pil))
68. return image
69. image_url = "https://upload.wikimedia.org/wikipedia/commons/6/60/Naxos_Taverna.jpg" #@param
70. downloaded_image_path = download_and_resize_image(image_url, 1280, 856, True)
71. module_handle = "https://tfhub.dev/google/faster_rcnn/openimages_v4/inception_resnet_v2/1" #@param
72. ["https://tfhub.dev/google/openimages_v4/ssd/mobilenet_v2/1",
73. "https://tfhub.dev/google/faster_rcnn/openimages_v4/inception_resnet_v2/1"]


```
74. detector = hub.load(module_handle).signatures['default']
75. def load_img(path):
76. img = tf.io.read_file(path)
77. img = tf.image.decode_jpeg(img, channels=3)
78. return img
79. def run_detector(detector, path):
80. img = load_img(path)
81. converted_img = tf.image.convert_image_dtype(img, tf.float32)[tf.newaxis, ...]
82. start_time = time.time()
83. result = detector(converted_img)
84. end_time = time.time()
85. result = {key:value.numpy() for key,value in result.items()}
86. print("Found %d objects." % len(result["detection_scores"]))
87. print("Inference time: ", end_time-start_time)
88. image_with_boxes = draw_boxes(
89. img.numpy(), result["detection_boxes"],
90. result["detection_class_entities"], result["detection_scores"])
91. display_image(image_with_boxes)
92. run_detector(detector, downloaded_image_path)
93. Found 100 objects.
94. Inference time: 86.00339889526367
```

Conclusion: Implementation was successfully

EXPERIMENT 8

Aim: Generating Images with BigGAN

Software used: Google collab

Theory:

BigGAN is a type of generative adversarial network that was designed for scaling generation to high-resolution, high-fidelity images. It includes a number of incremental changes and innovations. BigGan models are conditional GANs, meaning they take the class index as an input to generate images from the same category.

Code and Output:

```
[2] # BigGAN-deep models
    # module_path = 'https://tfhub.dev/deepmind/biggan-deep-128/1'  # 128x128 BigGAN-deep
    module_path = 'https://tfhub.dev/deepmind/biggan-deep-256/1'  # 256x256 BigGAN-deep
    # module_path = 'https://tfhub.dev/deepmind/biggan-deep-512/1'  # 512x512 BigGAN-deep

    # BigGAN (original) models
    # module_path = 'https://tfhub.dev/deepmind/biggan-128/2'  # 128x128 BigGAN
    # module_path = 'https://tfhub.dev/deepmind/biggan-256/2'  # 256x256 BigGAN
    # module_path = 'https://tfhub.dev/deepmind/biggan-512/2'  # 512x512 BigGAN
```

```
[4] tf.reset_default_graph()
    print('Loading BigGAN module from:', module_path)
    module = hub.Module(module_path)
    inputs = {k: tf.placeholder(v.dtype, v.get_shape().as_list(), k)
              for k, v in module.get_input_info_dict().items()}
    output = module(inputs)

    print()
    print('Inputs:\n', '\n'.join(
        '  {}: {}'.format(*kv) for kv in inputs.items()))
    print()
    print('Output:', output)
```

```
Loading BigGAN module from: https://tfhub.dev/deepmind/biggan-deep-256/1

Inputs:
   y: Tensor("y:0", shape=(?, 1000), dtype=float32)
  z: Tensor("z:0", shape=(?, 128), dtype=float32)
  truncation: Tensor("truncation:0", shape=(), dtype=float32)

Output: Tensor("module_apply_default/G_trunc_output:0", shape=(?, 256, 256, 3), dtype=float32)
```

```
[5] input_z = inputs['z']
    input_y = inputs['y']
    input_trunc = inputs['truncation']

    dim_z = input_z.shape.as_list()[1]
    vocab_size = input_y.shape.as_list()[1]

    def truncated_z_sample(batch_size, truncation=1., seed=None):
      state = None if seed is None else np.random.RandomState(seed)
      values = truncnorm.rvs(-2, 2, size=(batch_size, dim_z), random_state=state)
      return truncation * values

    def one_hot(index, vocab_size=vocab_size):
      index = np.asarray(index)
      if len(index.shape) == 0:
        index = np.asarray([index])
      assert len(index.shape) == 1
      num = index.shape[0]
      output = np.zeros((num, vocab_size), dtype=np.float32)
      output[np.arange(num), index] = 1
      return output

    def one_hot_if_needed(label, vocab_size=vocab_size):
      label = np.asarray(label)
      if len(label.shape) <= 1:
        label = one_hot(label, vocab_size)
      assert len(label.shape) == 2
      return label

    def sample(sess, noise, label, truncation=1., batch_size=8,
               vocab_size=vocab_size):
      noise = np.asarray(noise)
      label = np.asarray(label)
      num = noise.shape[0]
      if len(label.shape) == 0:
        label = np.asarray([label] * num)
      if label.shape[0] != num:
        raise ValueError('Got # noise samples ({}) != # label samples ({})'
                         .format(noise.shape[0], label.shape[0]))
```

[5]
```python
  label = one_hot_if_needed(label, vocab_size)
  ims = []
  for batch_start in range(0, num, batch_size):
    s = slice(batch_start, min(num, batch_start + batch_size))
    feed_dict = {input_z: noise[s], input_y: label[s], input_trunc: truncation}
    ims.append(sess.run(output, feed_dict=feed_dict))
  ims = np.concatenate(ims, axis=0)
  assert ims.shape[0] == num
  ims = np.clip(((ims + 1) / 2.0) * 256, 0, 255)
  ims = np.uint8(ims)
  return ims

def interpolate(A, B, num_interps):
  if A.shape != B.shape:
    raise ValueError('A and B must have the same shape to interpolate.')
  alphas = np.linspace(0, 1, num_interps)
  return np.array([(1-a)*A + a*B for a in alphas])

def imgrid(imarray, cols=5, pad=1):
  if imarray.dtype != np.uint8:
    raise ValueError('imgrid input imarray must be uint8')
  pad = int(pad)
  assert pad >= 0
  cols = int(cols)
  assert cols >= 1
  N, H, W, C = imarray.shape
  rows = N // cols + int(N % cols != 0)
  batch_pad = rows * cols - N
  assert batch_pad >= 0
  post_pad = [batch_pad, pad, pad, 0]
  pad_arg = [[0, p] for p in post_pad]
  imarray = np.pad(imarray, pad_arg, 'constant', constant_values=255)
  H += pad
  W += pad
  grid = (imarray
          .reshape(rows, cols, H, W, C)
          .transpose(0, 2, 1, 3, 4)
          .reshape(rows*H, cols*W, C))
  if pad:
    grid = grid[:-pad, :-pad]
```

[5]
```python
    grid = grid[:-pad, :-pad]
  return grid

def imshow(a, format='png', jpeg_fallback=True):
  a = np.asarray(a, dtype=np.uint8)
  data = io.BytesIO()
  PIL.Image.fromarray(a).save(data, format)
  im_data = data.getvalue()
  try:
    disp = IPython.display.display(IPython.display.Image(im_data))
  except IOError:
    if jpeg_fallback and format != 'jpeg':
      print(('Warning: image was too large to display in format "{}"; '
             'trying jpeg instead.').format(format))
      return imshow(a, format='jpeg')
    else:
      raise
  return disp
```

[6]
```python
initializer = tf.global_variables_initializer()
sess = tf.Session()
sess.run(initializer)
```

Category-conditional sampling

```
#@title Category-conditional sampling { display-mode: "form", run: "auto" }

num_samples = 10 #@param {type:"slider", min:1, max:20, step:1}
truncation = 0.4 #@param {type:"slider", min:0.02, max:1, step:0.02}
noise_seed = 0 #@param {type:"slider", min:0, max:100, step:1}
category = "933) cheeseburger" #@param ["0) tench, Tinca tinca", "1) goldfish, Carassius
auratus", "2) great white shark, white shark, man-eater, man-eating shark, Carcharodon
carcharias", "3) tiger shark, Galeocerdo cuvieri", "4) hammerhead, hammerhead shark", "5)
electric ray, crampfish, numbfish, torpedo", "6) stingray", "7) cock", "8) hen", "9)
ostrich, Struthio camelus", "10) brambling, Fringilla montifringilla", "11) goldfinch,
Carduelis carduelis", "12) house finch, linnet, Carpodacus mexicanus", "13) junco,
snowbird", "14) indigo bunting, indigo finch, indigo bird, Passerina cyanea", "15) robin,
calceolus, Cypripedium parviflorum", "987) corn", "988) acorn", "989) hip, rose hip,
rosehip", "990) buckeye, horse chestnut, conker", "991) coral fungus", "992) agaric",
"993) gyromitra", "994) stinkhorn, carrion fungus", "995) earthstar", "996)
hen-of-the-woods, hen of the woods, Polyporus frondosus, Grifola frondosa", "997) bolete",
"998) ear, spike, capitulum", "999) toilet tissue, toilet paper, bathroom tissue"]

z = truncated_z_sample(num_samples, truncation, noise_seed)
y = int(category.split(')')[0])

ims = sample(sess, z, y, truncation=truncation)
imshow(imgrid(ims, cols=min(num_samples, 5)))
```

num_samples: 10

truncation: 0.4

noise_seed: 0

category: 933) cheeseburger

Interpolation

```
#@title Interpolation { display-mode: "form", run: "auto" }

num_samples = 2 #@param {type:"slider", min:1, max:5, step:1}
num_interps = 5 #@param {type:"slider", min:2, max:10, step:1}
truncation = 0.2 #@param {type:"slider", min:0.02, max:1, step:0.02}
noise_seed_A = 0 #@param {type:"slider", min:0, max:100, step:1}
category_A = "207) golden retriever" #@param ["0) tench, Tinca tinca", "1) goldfish,
Carassius auratus", "2) great white shark, white shark, man-eater, man-eating shark,
Carcharodon carcharias", "3) tiger shark, Galeocerdo cuvieri", "4) hammerhead, hammerhead
shark", "5) electric ray, crampfish, numbfish, torpedo", "6) stingray", "7) cock", "8)
hen", "9) ostrich, Struthio camelus", "10) brambling, Fringilla montifringilla", "11)
goldfinch, Carduelis carduelis", "12) house finch, linnet, Carpodacus mexicanus", "13)
junco, snowbird", "14) indigo bunting, indigo finch, indigo bird, Passerina cyanea", "15)
robin, American robin, Turdus migratorius", "16) bulbul", "17) jay", "18) magpie", "19)
chickadee", "20) water ouzel, dipper", "21) kite", "22) bald eagle, American eagle,
Haliaeetus leucocephalus", "23) vulture", "24) great grey owl, great gray owl, Strix
nebulosa", "25) European fire salamander, Salamandra salamandra", "26) common newt,
Triturus vulgaris", "27) eft", "28) spotted salamander, Ambystoma maculatum", "29)
axolotl, mud puppy, Ambystoma mexicanum", "30) bullfrog, Rana catesbeiana", "31) tree
```

num_samples: 2

num_interps: 5

truncation: 0.2

noise_seed_A: 0

category_A: 207) golden retriever

noise_seed_B: 0

category_B: 8) hen

```
def interpolate_and_shape(A, B, num_interps):
  interps = interpolate(A, B, num_interps)
  return (interps.transpose(1, 0, *range(2, len(interps.shape)))
                 .reshape(num_samples * num_interps, *interps.shape[2:]))

z_A, z_B = [truncated_z_sample(num_samples, truncation, noise_seed)
            for noise_seed in [noise_seed_A, noise_seed_B]]
y_A, y_B = [one_hot([int(category.split(')')[0])] * num_samples)
            for category in [category_A, category_B]]

z_interp = interpolate_and_shape(z_A, z_B, num_interps)
y_interp = interpolate_and_shape(y_A, y_B, num_interps)

ims = sample(sess, z_interp, y_interp, truncation=truncation)
imshow(imgrid(ims, cols=num_interps))
```

Conclusion: Implementation was successful.

EXPERIMENT 9

Aim: implement transformer network for translating language

Software used: Google collab

Theory:

Transformer neural Network works on self-attention mechanism. It was first discussed in 2017 in the paper of “Attention is all you need.” It basically replaces the traditional methods used in NLP that is RNN and LSTM.

Code and Output:

```
[ ] # Install the most re version of TensorFlow to use the improved
    # masking support for `tf.keras.layers.MultiHeadAttention`.
    !apt install --allow-change-held-packages libcudnn8=8.1.0.77-1+cuda11.2
    !pip uninstall -y -q tensorflow keras tensorflow-estimator tensorflow-text
    !pip install protobuf~=3.20.3
    !pip install -q tensorflow_datasets
    !pip install -q -U tensorflow-text tensorflow
```

Import the necessary modules:

```
[ ] import logging
    import time

    import numpy as np
    import matplotlib.pyplot as plt

    import tensorflow_datasets as tfds
    import tensorflow as tf

    import tensorflow_text
```

```
[ ] examples, metadata = tfds.load('ted_hrlr_translate/pt_to_en',
                                   with_info=True,
                                   as_supervised=True)

    train_examples, val_examples = examples['train'], examples['validation']
```

The `tf.data.Dataset` object returned by TensorFlow Datasets yields pairs of text examples:

```
[ ] for pt_examples, en_examples in train_examples.batch(3).take(1):
      print('> Examples in Portuguese:')
      for pt in pt_examples.numpy():
        print(pt.decode('utf-8'))
      print()

      print('> Examples in English:')
      for en in en_examples.numpy():
        print(en.decode('utf-8'))
```

```python
model_name = 'ted_hrlr_translate_pt_en_converter'
tf.keras.utils.get_file(
    f'{model_name}.zip',
    f'https://storage.googleapis.com/download.tensorflow.org/models/{model_name}.zip',
    cache_dir='.', cache_subdir='', extract=True
)
```

```python
tokenizers = tf.saved_model.load(model_name)
```

The `tf.saved_model` contains two text tokenizers, one for English and one for Portuguese. Both have the same methods:

```python
[item for item in dir(tokenizers.en) if not item.startswith('_')]
```

```python
print('> This is a batch of strings:')
for en in en_examples.numpy():
  print(en.decode('utf-8'))
```

```python
encoded = tokenizers.en.tokenize(en_examples)

print('> This is a padded-batch of token IDs:')
for row in encoded.to_list():
  print(row)
```

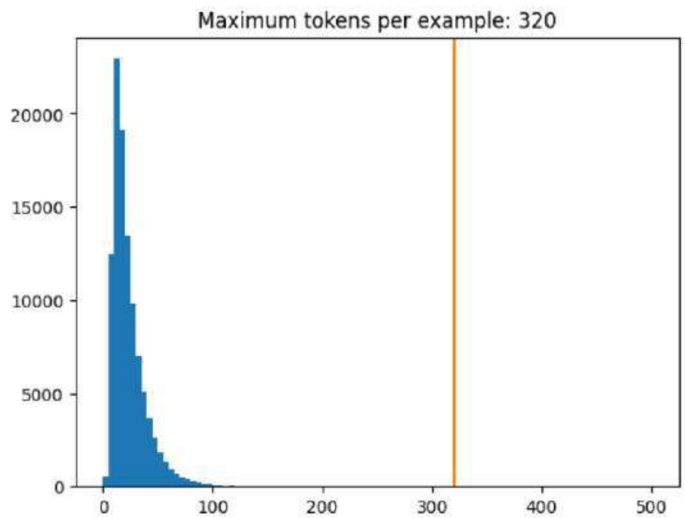
```python
round_trip = tokenizers.en.detokenize(encoded)

print('> This is human-readable text:')
for line in round_trip.numpy():
  print(line.decode('utf-8'))
```

The lower level `lookup` method converts from token-IDs to token text:

```python
print('> This is the text split into tokens:')
tokens = tokenizers.en.lookup(encoded)
tokens
```

```python
lengths = []

for pt_examples, en_examples in train_examples.batch(1024):
  pt_tokens = tokenizers.pt.tokenize(pt_examples)
  lengths.append(pt_tokens.row_lengths())

  en_tokens = tokenizers.en.tokenize(en_examples)
  lengths.append(en_tokens.row_lengths())
  print('.', end='', flush=True)
```

```python
all_lengths = np.concatenate(lengths)

plt.hist(all_lengths, np.linspace(0, 500, 101))
plt.ylim(plt.ylim())
max_length = max(all_lengths)
plt.plot([max_length, max_length], plt.ylim())
plt.title(f'Maximum tokens per example: {max_length}');
```




```
MAX_TOKENS=128
def prepare_batch(pt, en):
    pt = tokenizers.pt.tokenize(pt)      # Output is ragged.
    pt = pt[:, :MAX_TOKENS]    # Trim to MAX_TOKENS.
    pt = pt.to_tensor()  # Convert to 0-padded dense Tensor

    en = tokenizers.en.tokenize(en)
    en = en[:, :(MAX_TOKENS+1)]
    en_inputs = en[:, :-1].to_tensor()  # Drop the [END] tokens
    en_labels = en[:, 1:].to_tensor()   # Drop the [START] tokens

    return (pt, en_inputs), en_labels
```

```
[ ] BUFFER_SIZE = 20000
    BATCH_SIZE = 64
```

```
def make_batches(ds):
  return (
      ds
      .shuffle(BUFFER_SIZE)
      .batch(BATCH_SIZE)
      .map(prepare_batch, tf.data.AUTOTUNE)
      .prefetch(buffer_size=tf.data.AUTOTUNE))
```

```
[ ] # Create training and validation set batches.
    train_batches = make_batches(train_examples)
    val_batches = make_batches(val_examples)
```

```
[ ] for (pt, en), en_labels in train_batches.take(1):
      break

    print(pt.shape)
    print(en.shape)
    print(en_labels.shape)
```

The `en` and `en_labels` are the same, just shifted by 1:

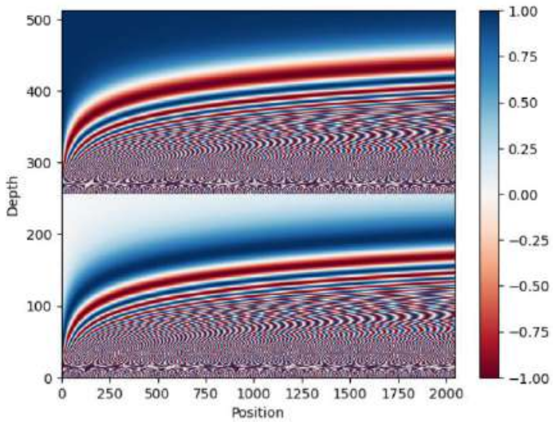
```
[ ] print(en[0][:10])
    print(en_labels[0][:10])
```

```
(64, 86)
(64, 81)
(64, 81)
```

```
tf.Tensor([   2  476 2569 2626 6010   52 2564 1915  188   15], shape=(10,), d
tf.Tensor([ 476 2569 2626 6010   52 2564 1915  188   15    3], shape=(10,), d
```

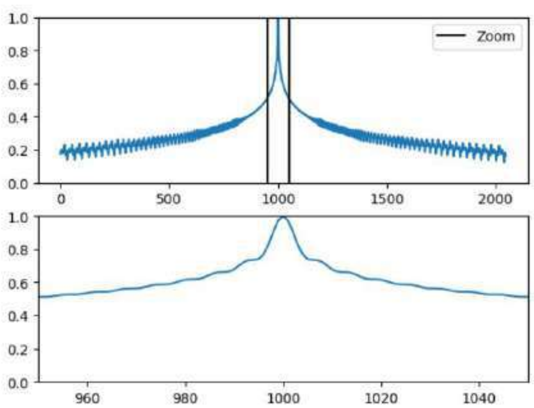
```
def positional_encoding(length, depth):
  depth = depth/2

  positions = np.arange(length)[:, np.newaxis]     # (seq, 1)
  depths = np.arange(depth)[np.newaxis, :]/depth   # (1, depth)

  angle_rates = 1 / (10000**depths)         # (1, depth)
  angle_rads = positions * angle_rates      # (pos, depth)

  pos_encoding = np.concatenate(
      [np.sin(angle_rads), np.cos(angle_rads)],
      axis=-1)

  return tf.cast(pos_encoding, dtype=tf.float32)
```

```
[ ] #@title
    pos_encoding = positional_encoding(length=2048, depth=512)

    # Check the shape.
    print(pos_encoding.shape)

    # Plot the dimensions.
    plt.pcolormesh(pos_encoding.numpy().T, cmap='RdBu')
    plt.ylabel('Depth')
    plt.xlabel('Position')
    plt.colorbar()
    plt.show()
```




```
#@title
pos_encoding/=tf.norm(pos_encoding, axis=1, keepdims=True)
p = pos_encoding[1000]
dots = tf.einsum('pd,d -> p', pos_encoding, p)
plt.subplot(2,1,1)
plt.plot(dots)
plt.ylim([0,1])
plt.plot([950, 950, float('nan'), 1050, 1050],
         [0,1,float('nan'),0,1], color='k', label='Zoom')
plt.legend()
plt.subplot(2,1,2)
plt.plot(dots)
plt.xlim([950, 1050])
plt.ylim([0,1])
```

```
(0.0, 1.0)
```



```
class PositionalEmbedding(tf.keras.layers.Layer):
  def __init__(self, vocab_size, d_model):
    super().__init__()
    self.d_model = d_model
    self.embedding = tf.keras.layers.Embedding(vocab_size, d_model, mask_zero=
    self.pos_encoding = positional_encoding(length=2048, depth=d_model)

  def compute_mask(self, *args, **kwargs):
    return self.embedding.compute_mask(*args, **kwargs)

  def call(self, x):
    length = tf.shape(x)[1]
    x = self.embedding(x)
    # This factor sets the relative scale of the embedding and positonal_encod
    x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
    x = x + self.pos_encoding[tf.newaxis, :length, :]
    return x
```

```
embed_pt = PositionalEmbedding(vocab_size=tokenizers.pt.get_vocab_size(), d_m
embed_en = PositionalEmbedding(vocab_size=tokenizers.en.get_vocab_size(), d_m

pt_emb = embed_pt(pt)
en_emb = embed_en(en)
```

```
en_emb._keras_mask
```

```
<tf.Tensor: shape=(64, 81), dtype=bool, numpy=
array([[ True,  True,  True, ..., False, False, False],
       [ True,  True,  True, ..., False, False, False],
       [ True,  True,  True, ..., False, False, False],
       ...,
       [ True,  True,  True, ..., False, False, False],
       [ True,  True,  True, ..., False, False, False],
       [ True,  True,  True, ..., False, False, False]])>
```

```
class CrossAttention(BaseAttention):
  def call(self, x, context):
    attn_output, attn_scores = self.mha(
        query=x,
        key=context,
        value=context,
        return_attention_scores=True)

    # Cache the attention scores for plotting later.
    self.last_attn_scores = attn_scores

    x = self.add([x, attn_output])
    x = self.layernorm(x)

    return x
```

```
sample_ca = CrossAttention(num_heads=2, key_dim=512)

print(pt_emb.shape)
print(en_emb.shape)
print(sample_ca(en_emb, pt_emb).shape)
```

```
(64, 86, 512)
(64, 81, 512)
(64, 81, 512)
```

```
class GlobalSelfAttention(BaseAttention):
  def call(self, x):
    attn_output = self.mha(
        query=x,
        value=x,
        key=x)
    x = self.add([x, attn_output])
    x = self.layernorm(x)
    return x
```

```
sample_gsa = GlobalSelfAttention(num_heads=2, key_dim=512)

print(pt_emb.shape)
print(sample_gsa(pt_emb).shape)
```

```
(64, 86, 512)
(64, 86, 512)
```

```
class CausalSelfAttention(BaseAttention):
  def call(self, x):
    attn_output = self.mha(
        query=x,
        value=x,
        key=x,
        use_causal_mask = True)
    x = self.add([x, attn_output])
    x = self.layernorm(x)
    return x
```

```
sample_csa = CausalSelfAttention(num_heads=2, key_dim=512)

print(en_emb.shape)
print(sample_csa(en_emb).shape)
```

```
(64, 81, 512)
(64, 81, 512)
```

The output for early sequence elements doesn't depend on later elements, so it shouldn't matter if you trim elements before or after applying the layer:

```
out1 = sample_csa(embed_en(en[:, :3]))
out2 = sample_csa(embed_en(en))[:, :3]

tf.reduce_max(abs(out1 - out2)).numpy()
```

```
4.7683716e-07
```

```
class FeedForward(tf.keras.layers.Layer):
  def __init__(self, d_model, dff, dropout_rate=0.1):
    super().__init__()
    self.seq = tf.keras.Sequential([
      tf.keras.layers.Dense(dff, activation='relu'),
      tf.keras.layers.Dense(d_model),
      tf.keras.layers.Dropout(dropout_rate)
    ])
    self.add = tf.keras.layers.Add()
    self.layer_norm = tf.keras.layers.LayerNormalization()

  def call(self, x):
    x = self.add([x, self.seq(x)])
    x = self.layer_norm(x)
    return x
```

Test the layer, the output is the same shape as the input:

```
sample_ffn = FeedForward(512, 2048)

print(en_emb.shape)
```

```
print(sample_ffn(en_emb).shape)
```

```
(64, 81, 512)
(64, 81, 512)
```

```
class Encoder(tf.keras.layers.Layer):
  def __init__(self, *, num_layers, d_model, num_heads,
               dff, vocab_size, dropout_rate=0.1):
    super().__init__()
```

```
    self.d_model = d_model
    self.num_layers = num_layers

    self.pos_embedding = PositionalEmbedding(
        vocab_size=vocab_size, d_model=d_model)

    self.enc_layers = [
        EncoderLayer(d_model=d_model,
                     num_heads=num_heads,
                     dff=dff,
                     dropout_rate=dropout_rate)
        for _ in range(num_layers)]
    self.dropout = tf.keras.layers.Dropout(dropout_rate)

  def call(self, x):
    # `x` is token-IDs shape: (batch, seq_len)
    x = self.pos_embedding(x)  # Shape `(batch_size, seq_len, d_model)`.

    # Add dropout.
    x = self.dropout(x)

    for i in range(self.num_layers):
      x = self.enc_layers[i](x)

    return x  # Shape `(batch_size, seq_len, d_model)`.
```

```
# Instantiate the encoder.
sample_encoder = Encoder(num_layers=4,
                         d_model=512,
                         num_heads=8,
                         dff=2048,
                         vocab_size=8500)

sample_encoder_output = sample_encoder(pt, training=False)

# Print the shape.
print(pt.shape)
print(sample_encoder_output.shape)  # Shape `(batch_size, input_seq_len, d_mo
```

```
(64, 86)
```

```
(64, 86, 512)
```

```
class DecoderLayer(tf.keras.layers.Layer):
  def __init__(self,
               *,
               d_model,
               num_heads,
               dff,
               dropout_rate=0.1):
    super(DecoderLayer, self).__init__()

    self.causal_self_attention = CausalSelfAttention(
        num_heads=num_heads,
        key_dim=d_model,
        dropout=dropout_rate)

    self.cross_attention = CrossAttention(
        num_heads=num_heads,
        key_dim=d_model,
        dropout=dropout_rate)

    self.ffn = FeedForward(d_model, dff)

  def call(self, x, context):
    x = self.causal_self_attention(x=x)
    x = self.cross_attention(x=x, context=context)

    # Cache the last attention scores for plotting later
    self.last_attn_scores = self.cross_attention.last_attn_scores

    x = self.ffn(x)  # Shape `(batch_size, seq_len, d_model)`.
    return x
```

Test the decoder layer:

```
sample_decoder_layer = DecoderLayer(d_model=512, num_heads=8, dff=2048)

sample_decoder_layer_output = sample_decoder_layer(
    x=en_emb, context=pt_emb)

print(en_emb.shape)
print(pt_emb.shape)
print(sample_decoder_layer_output.shape)  # `(batch_size, seq_len, d_model)`
```

Test the decoder:

```
# Instantiate the decoder.
sample_decoder = Decoder(num_layers=4,
                         d_model=512,
                         num_heads=8,
                         dff=2048,
                         vocab_size=8000)

output = sample_decoder(
    x=en,
    context=pt_emb)

# Print the shapes.
print(en.shape)
print(pt_emb.shape)
print(output.shape)
```

```
(64, 81)
(64, 86, 512)
(64, 81, 512)
```

```
sample_decoder.last_attn_scores.shape  # (batch, heads, target_seq, input_seq
```

```
class Transformer(tf.keras.Model):
  def __init__(self, *, num_layers, d_model, num_heads, dff,
               input_vocab_size, target_vocab_size, dropout_rate=0.1):
    super().__init__()
    self.encoder = Encoder(num_layers=num_layers, d_model=d_model,
                           num_heads=num_heads, dff=dff,
                           vocab_size=input_vocab_size,
                           dropout_rate=dropout_rate)
```

```
    self.decoder = Decoder(num_layers=num_layers, d_model=d_model,
                           num_heads=num_heads, dff=dff,
                           vocab_size=target_vocab_size,
                           dropout_rate=dropout_rate)

    self.final_layer = tf.keras.layers.Dense(target_vocab_size)

  def call(self, inputs):
    # To use a Keras model with `.fit` you must pass all your inputs in the
    # first argument.
    context, x  = inputs

    context = self.encoder(context)  # (batch_size, context_len, d_model)

    x = self.decoder(x, context)  # (batch_size, target_len, d_model)

    # Final linear layer output.
    logits = self.final_layer(x)  # (batch_size, target_len, target_vocab_siz

    try:
      # Drop the keras mask, so it doesn't scale the losses/metrics.
      # b/250038731
      del logits._keras_mask
    except AttributeError:
      pass

    # Return the final output and the attention weights.
    return logits
```

```
class CustomSchedule(tf.keras.optimizers.schedules.LearningRateSchedule):
  def __init__(self, d_model, warmup_steps=4000):
    super().__init__()

    self.d_model = d_model
    self.d_model = tf.cast(self.d_model, tf.float32)

    self.warmup_steps = warmup_steps

  def __call__(self, step):
    step = tf.cast(step, dtype=tf.float32)
```

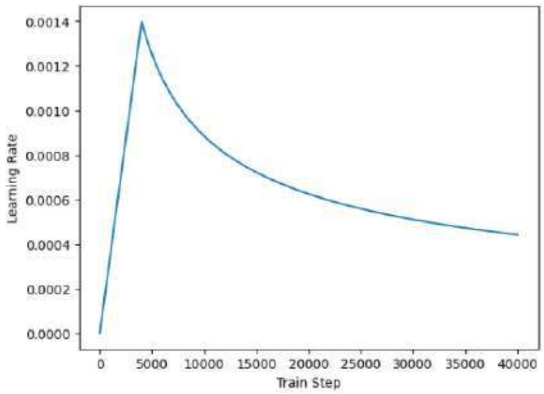
```
    arg1 = tf.math.rsqrt(step)
    arg2 = step * (self.warmup_steps ** -1.5)

    return tf.math.rsqrt(self.d_model) * tf.math.minimum(arg1, arg2)
```

Instantiate the optimizer (in this example it's `tf.keras.optimizers.Adam` (https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam)):

```
learning_rate = CustomSchedule(d_model)

optimizer = tf.keras.optimizers.Adam(learning_rate, beta_1=0.9, beta_2=0.98,
                                     epsilon=1e-9)
```

Test the custom learning rate scheduler:

```
plt.plot(learning_rate(tf.range(40000, dtype=tf.float32)))
plt.ylabel('Learning Rate')
plt.xlabel('Train Step')
```



Example 1:

```
sentence = 'este é um problema que temos que resolver.'
ground_truth = 'this is a problem we have to solve .'

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

```
Input:          : este é um problema que temos que resolver.
Prediction      : this is a problem that we have to solve .
Ground truth    : this is a problem we have to solve .
```

Example 2:

```
sentence = 'os meus vizinhos ouviram sobre esta ideia.'
ground_truth = 'and my neighboring homes heard about this idea .'

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

```
Input:          : os meus vizinhos ouviram sobre esta ideia.
Prediction      : my neighbors have heard this idea .
```

Conclusion: Implementation was successful.

EXPERIMENT 10

Aim: Design BANGALORE HOUSE PRICE PREDICTION MODEL

Software used: Google collab

Theory:

Code and Output:

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
import matplotlib
matplotlib.rcParams["figure.figsize"] = (20,10)
```

LOAD DATASET

```
df1 = pd.read_csv("../input/bangalore-house-prices/bengaluru_house_prices.csv")
df1.head()
```

	area_type	availability	location	size	society	total_sq
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	10
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	26
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	NaN	14
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Soiewre	15
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	NaN	12

EXPLORATORY DATA ANALYSIS

```
df1.shape
```

```
(13320, 9)
```

```
df1.columns
```

```
Index(['area_type', 'availability', 'location', 'size', 'society',
       'total_sqft', 'bath', 'balcony', 'price'],
      dtype='object')
```

```
array(['Super built-up  Area', 'Plot  Area', 'Built-up  Area',
       'Carpet  Area'], dtype=object)
```

```
df1['area_type'].value_counts()
```

```
Super built-up  Area    8790
Built-up  Area          2418
Plot  Area              2025
Carpet  Area              87
Name: area_type, dtype: int64
```

NOTE: DROP UNNECESSARY FEATURES

```
df2 = df1.drop(['area_type','society','balcony','availability'],axis='columns')
df2.shape
```

```
(13320, 5)
```

DATA CLEANING

```
df2.isnull().sum()
```

```
location       1
size          16
total_sqft     0
bath          73
price          0
dtype: int64
```

```
df2.shape
```

```
(13320, 5)
```

```
df3 = df2.dropna()
df3.isnull().sum()
```

```
location      0
size          0
total_sqft    0
bath          0
price         0
dtype: int64
```

```
df3.shape
```

```
(13246, 5)
```

```
df3['bhk'] = df3['size'].apply(lambda x: int(x.split(' ')[0]))
df3.bhk.unique()
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:1: SettingWithCopy
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
  """Entry point for launching an IPython kernel.
array([ 2,  4,  3,  6,  1,  8,  7,  5, 11,  9, 27, 10, 19, 16, 43, 14, 12,
       13, 18])
```

EXPLORE TOTAL SQFT FEATURE

```
def is_float(x):
    try:
        float(x)
    except:
        return False
    return True
```

```
2+3
```

```
5
```

```
df3[~df3['total_sqft'].apply(is_float)].head(10)
```

	location	size	total_sqft	bath	price	bhk
30	Yelahanka	4 BHK	2100 - 2850	4.0	186.000	4
122	Hebbal	4 BHK	3067 - 8156	4.0	477.000	4
137	8th Phase JP Nagar	2 BHK	1042 - 1105	2.0	54.005	2
165	Sarjapur	2 BHK	1145 - 1340	2.0	43.490	2
188	KR Puram	2 BHK	1015 - 1540	2.0	56.800	2
410	Kengeri	1 BHK	34.46Sq. Meter	1.0	18.500	1
549	Hennur Road	2 BHK	1195 - 1440	2.0	63.770	2
648	Arekere	9 Bedroom	4125Perch	9.0	265.000	9
661	Yelahanka	2 BHK	1120 - 1145	2.0	48.130	2
672	Bettahalsoor	4 Bedroom	3090 - 5002	4.0	445.000	4

```
def convert_sqft_to_num(x):
    tokens = x.split('-')
    if len(tokens) == 2:
        return (float(tokens[0])+float(tokens[1]))/2
    try:
        return float(x)
    except:
        return None
```

```
df4 = df3.copy()
df4.total_sqft = df4.total_sqft.apply(convert_sqft_to_num)
df4 = df4[df4.total_sqft.notnull()]
df4.head(2)
```

	location	size	total_sqft	bath	price	bhk
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4

FOR ROW BELOW, IT SHOWS TOTAL SQFT AS 2475 WHICH IS AN AVERAGE OF THE RANGE 2100-2850

```
df4.loc[30]
```

```
location      Yelahanka
size              4 BHK
total_sqft       2475.0
bath                4.0
price             186.0
bhk                   4
Name: 30, dtype: object
```

```
(2100+2850)/2
```

```
2475.0
```

ADD NEW FEATURE CALLED PRICE PER SQUARE FEET

```
df5 = df4.copy()
df5['price_per_sqft'] = df5['price']*100000/df5['total_sqft']
df5.head()
```

```
df5_stats = df5['price_per_sqft'].describe()
df5_stats
```

```
count    1.320000e+04
mean     7.920759e+03
std      1.067272e+05
min      2.678298e+02
25%      4.267701e+03
50%      5.438331e+03
75%      7.317073e+03
max      1.200000e+07
Name: price_per_sqft, dtype: float64
```

```
df5.to_csv("bhp.csv",index=False)
```

EXAMINE LOCATIONS WHICH IS A CATEGORICAL VARIABLE. WE NEED TO APPLY THE DIMENSIONALITY REDUCTION TECHNIQUE HERE TO REDUCE THE NUMBER OF LOCATIONS

```
df5.location = df5.location.apply(lambda x: x.strip())
location_stats = df5['location'].value_counts(ascending=False)
location_stats
```

```
Whitefield              533
Sarjapur  Road          392
Electronic City         304
Kanakpura Road          264
Thanisandra             235
                       ...
St Thomas Town            1
Jp nagar 8th Phase .      1
Jaymahal Road             1
Chuchangatta Colony       1
Maruthi Extension         1
Name: location, Length: 1287, dtype: int64
```

```
location_stats.values.sum()
```

```
13200
```

```
len(location_stats[location_stats>10])
```

```
240
```

```
len(location_stats)
```

```
1287
```

```
len(location_stats[location_stats<=10])
```

```
1047
```

DIMENSIONALITY REDUCTIONS

ANY LOCATION HAVING LESS THAN 10 DATA PINTS SHOULD BE TAGGED AS "OTHER" LOCATION. THIS WAY NUMBER OF CATEGORIES CAN BE REDUCED BY HUGE AMOUNT. LATER ON WHEN WE DO ONE HOT ENCODING, IT WILL HELP US WITH HAVING FEWER DUMMY COLUMNS.

```
location_stats_less_than_10 = location_stats[location_stats<=10]
location_stats_less_than_10
```

```
Nagadevanahalli         10
Naganathapura           10
Basapura                10
Nagappa Reddy Layout    10
Thyagaraja Nagar        10
                        ..
St Thomas Town           1
Jp nagar 8th Phase .     1
Jaymahal Road            1
Chuchangatta Colony      1
Maruthi Extension        1
Name: location, Length: 1047, dtype: int64
```

```
len(df5.location.unique())
```

```
1287
```

```
df5.location = df5.location.apply(lambda x: 'other' if x in location_stats_less_than
len(df5.location.unique())
```

```
241
```

```
df5.head(10)
```

```
df5[df5.total_sqft/df5.bhk<300].head()
```

	location	size	total_sqft	bath	price	bhk	price_per_sqft
9	other	6 Bedroom	1020.0	6.0	370.0	6	36274.509804
45	HSR Layout	8 Bedroom	600.0	9.0	200.0	8	33333.333333
58	Murugeshpalya	6 Bedroom	1407.0	4.0	150.0	6	10660.980810
68	Devarachikkanahalli	8 Bedroom	1350.0	7.0	85.0	8	6296.296296
70	other	3 Bedroom	500.0	3.0	100.0	3	20000.000000

CHECK THE ABOVE DATA POINTS. WE HAVE 6 BHK APARTMENTS WITH 1020 SQFT. ANOTHER ONE IS 8 BHK AND THE TOTAL SQFT IS 600. THESE ARE CLEAR DATA ERRORS THAT CAN BE REMOVED SAFELY

```
df5.shape
```

```
(13200, 7)
```

```
df6 = df5[~(df5.total_sqft/df5.bhk<300)]
df6.shape
```

```
(12456, 7)
```

```
df6.price_per_sqft.describe()
```

```
count     12456.000000
mean       6308.502826
std        4168.127339
min         267.829813
25%        4210.526316
50%        5294.117647
75%        6916.666667
max      176470.588235
Name: price_per_sqft, dtype: float64
```

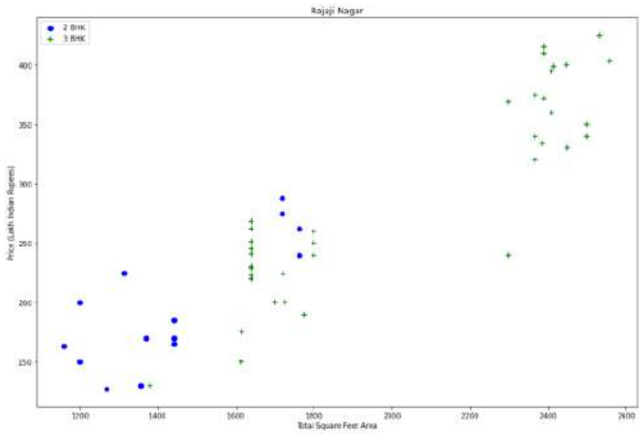
HERE WE FIND THAT MIN PRICE PER SQFT IS 267 RS/SQFT WHEREAS MAX IS 12000000, THIS SHOWS A WIDE VARIATION IN PROPERTY PRICES. WE SHOULD REMOVE OUTLIERS PER LOCATION USING MEAN AND ONE STANDARD DEVIATION

```
def remove_pps_outliers(df):
    df_out = pd.DataFrame()
    for key, subdf in df.groupby('location'):
        m = np.mean(subdf.price_per_sqft)
        st = np.std(subdf.price_per_sqft)
        reduced_df = subdf[(subdf.price_per_sqft>(m-st)) & (subdf.price_per_sqft<=(m+
        df_out = pd.concat([df_out,reduced_df],ignore_index=True)
    return df_out
df7 = remove_pps_outliers(df6)
df7.shape
```

```
(10242, 7)
```

LET'S CHECK IF FOR A GIVEN LOCATION HOW DOES THE 2 BHK AND 3 BHK PROPERTY PRICES LOOK LIKE

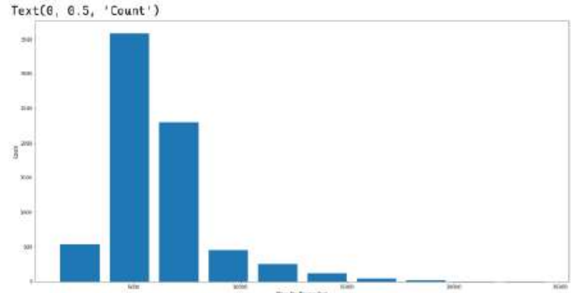
```
def plot_scatter_chart(df,location):
    bhk2 = df[(df.location==location) & (df.bhk==2)]
    bhk3 = df[(df.location==location) & (df.bhk==3)]
    matplotlib.rcParams['figure.figsize'] = (15,10)
    plt.scatter(bhk2.total_sqft,bhk2.price,color='blue',label='2 BHK', s=50)
    plt.scatter(bhk3.total_sqft,bhk3.price,marker='+', color='green',label='3 BHK', s
    plt.xlabel("Total Square Feet Area")
    plt.ylabel("Price (Lakh Indian Rupees)")
    plt.title(location)
    plt.legend()

plot_scatter_chart(df7,"Rajaji Nagar")
```



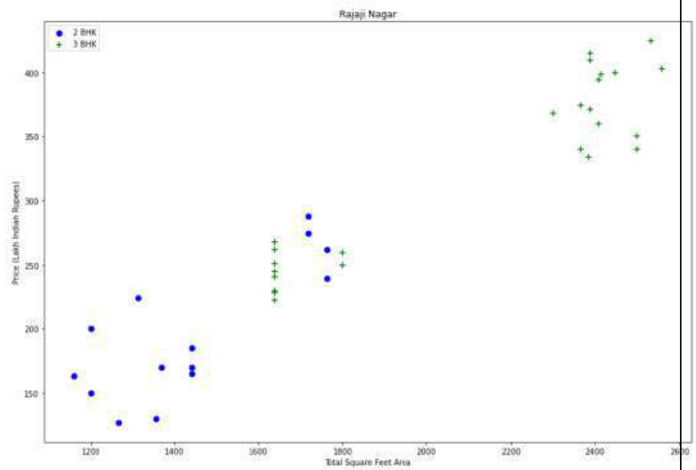

```
plot_scatter_chart(df7,"Hebbal")
```

```
import matplotlib
matplotlib.rcParams["figure.figsize"] = (20,10)
plt.hist(df8.price_per_sqft,rwidth=0.8)
plt.xlabel("Price Per Square Feet")
plt.ylabel("Count")
```

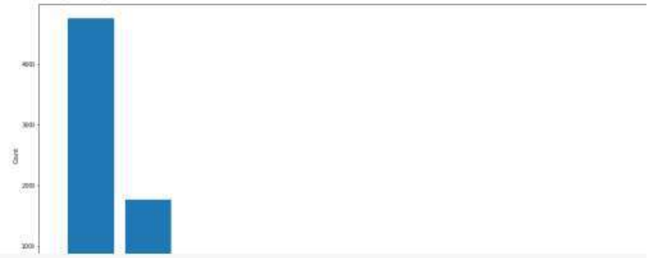
```
Text(0, 0.5, 'Count')
```



```
plot_scatter_chart(df8,"Rajaji Nagar")
```



```
Text(0, 0.5, 'Count')
```



```
df8[df8.bath>10]
```

	location	size	total_sqft	bath	price	bhk	price_per_sqft
5277	Neeladri Nagar	10 BHK	4000.0	12.0	160.0	10	4000.000000
8483	other	10 BHK	12000.0	12.0	525.0	10	4375.000000
8572	other	16 BHK	10000.0	16.0	550.0	16	5500.000000
9306	other	11 BHK	6000.0	12.0	150.0	11	2500.000000
9637	other	13 BHK	5425.0	13.0	275.0	13	5069.124424

IT IS UNUSUAL TO HAVE 2 MORE BATHROOMS THAN NUMBER OF BEDROOMS IN A HOME

```
df8[df8.bath>df8.bhk+2]
```

	location	size	total_sqft	bath	price	bhk	price_per_sqft
1626	Chikkabanavar	4 Bedroom	2460.0	7.0	80.0	4	3252.032520
5238	Nagasandra	4 Bedroom	7000.0	8.0	450.0	4	6428.571429
6711	Thanisandra	3 BHK	1806.0	6.0	116.0	3	6423.034330
8408	other	6 BHK	11338.0	9.0	1000.0	6	8819.897689

```
df9.head(2)
```

	location	size	total_sqft	bath	price	bhk	price_per_sqft
0	1st Block Jayanagar	4 BHK	2850.0	4.0	428.0	4	15017.543860
1	1st Block Jayanagar	3 BHK	1630.0	3.0	194.0	3	11901.840491

```
df10 = df9.drop(['size','price_per_sqft'],axis='columns')
df10.head(3)
```

	location	total_sqft	bath	price	bhk
0	1st Block Jayanagar	2850.0	4.0	428.0	4
1	1st Block Jayanagar	1630.0	3.0	194.0	3
2	1st Block Jayanagar	1875.0	2.0	235.0	3

USE ONE HOT ENCODING FOR LOCATION

```
dummies = pd.get_dummies(df10.location)
dummies.head(3)
```

	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	5th Phase JP Nagar	6th Phase JP Nagar	7th Phase JP Nagar	8th Phase JP Nagar
0	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0

3 rows × 241 columns

```
df11 = pd.concat([df10,dummies.drop('other',axis='columns')],axis='columns')
df11.head()
```

Conclusion: Implementation was successful.