# Laboratory Practical File
# on
# Intro to Deep Learning [CSE321]


## Bachelor of Technology 2020-2024

## Semester-6

## Department of Artificial Intelligence

## Amity School of Engineering and Technology
## Amity University, Noida, Uttar Pradesh, India

**Submitted to:**
Dr. Krishna Kant Singh

**Submitted by:**

Mr. Janya Verma
A023119820023
6AI-1

## Table of Contents

| S. No. | Category of Assignment | Code | Exp. No. | Name of Experiment | Date of Allotment of experiment | Signature of Faculty |
|---|---|---|---|---|---|---|
| **1.** | **Mandatory Experiment*** | **LR (10)** | **1** | design a densenet for temperature conversion | **12/01/23** | |
| **2.** | **Mandatory Experiment*** | | **2** | design a densenet for classifying images using fashion MNIST dataset | **19/01/23** | |
| **3.** | **Mandatory Experiment*** | | **3** | design a convnet for classifying images using fashion MNIST dataset | **02/02/23** | |
| **4.** | **Mandatory Experiment*** | | **4** | design a convnet for classifying dog and cat images | **09/02/23** | |
| **5.** | **Mandatory Experiment*** | | **5** | image classification using transfer learning for dogs vs cats dataset | **16/02/23** | |
| **6.** | **Mandatory Experiment*** | | **6** | implementation of inflated 3D CNN for action recognition | **23/02/23** | |
| 7 | **Mandatory Experiment*** | | 7 | object detection using CNN | **02/03/23** | |
| **8.** | **Mandatory Experiment*** | | **8** | generating images with Big GAN | **09/03/23** | |
| **9.** | **Mandatory Experiment*** | | **9** | implement transformer network for | **16/03/23** | |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | translating language | | |
| **10.** | **Mandatory Experiment*** | | **10** | MLops | **23/03/23** | |
| | | | | | | |
| **11.** | **Design Based Open Ended experiment**** | **PR (10)** | | | | |
| **12.** | **Viva** | **Viva (5)** | | | | |

# Experiment 1

## Aim

Design a densenet for temperature conversion.

## Software Requirements

Google Colab

## 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))
```

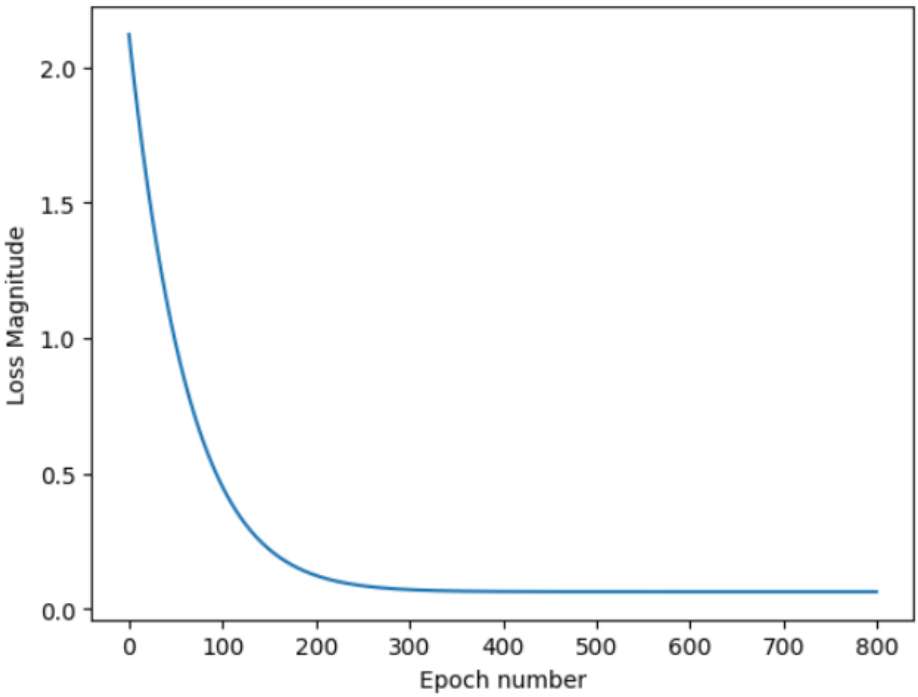
In [9]:
```
#the learning rate (0.1 in the code above). This is the step size taken when adjusting values in the model. If the value is too s
```

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 Colab

## 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']
```

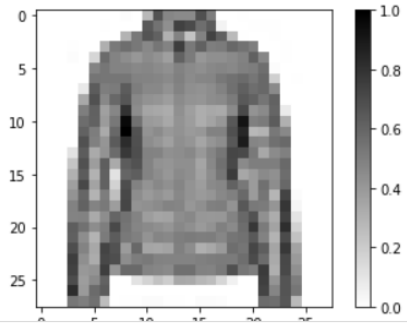
```
[ ] num_train_examples = metadata.splits['train'].num_examples
    num_test_examples = metadata.splits['test'].num_examples
    print("Number of training examples: {}".format(num_train_examples))
    print("Number of test examples:     {}".format(num_test_examples))
```

```
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 CNN for classifying images using flower dataset

## Software Requirements

Google Colab

## 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
```

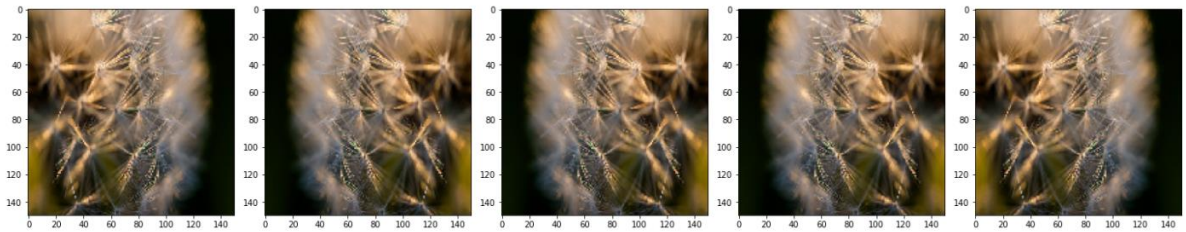
```
[ ]  train_dir = os.path.join(base_dir, 'train')
     val_dir = os.path.join(base_dir, 'val')
```

```
[ ]  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)
```

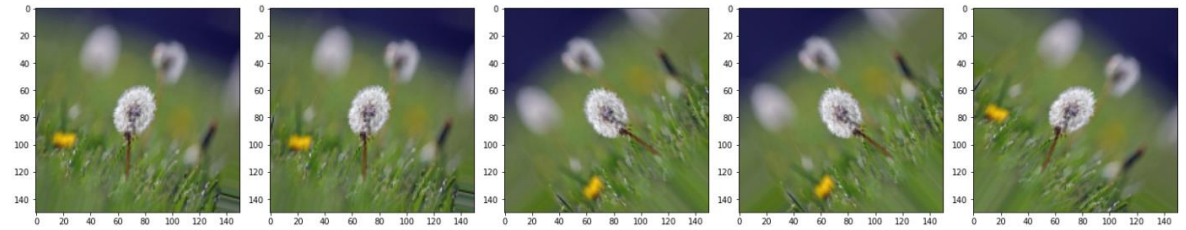


```
[ ]  image_gen = ImageDataGenerator(rescale = 1./255, rotation_range =45)

     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)
```

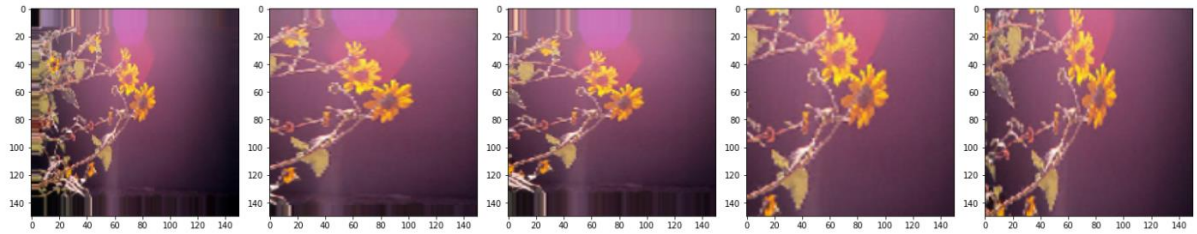


```
[ ]  image_gen = ImageDataGenerator(rescale = 1./255, zoom_range =0.5)

     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.

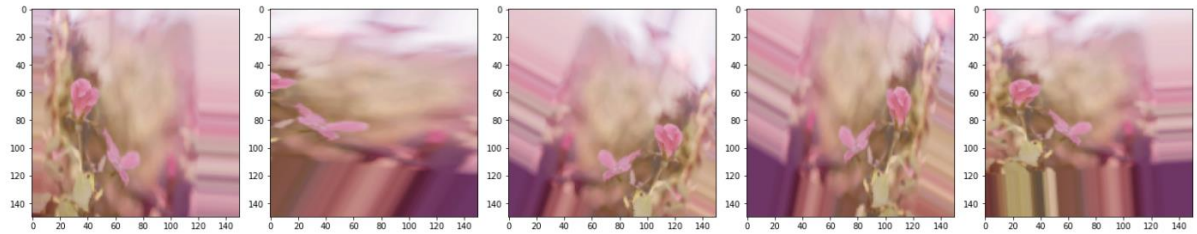
```
[ ]  augmented_images = [train_data_gen[0][0][0] for i in range(5)]
     plotImages(augmented_images)
```



```
[ ]  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.
```

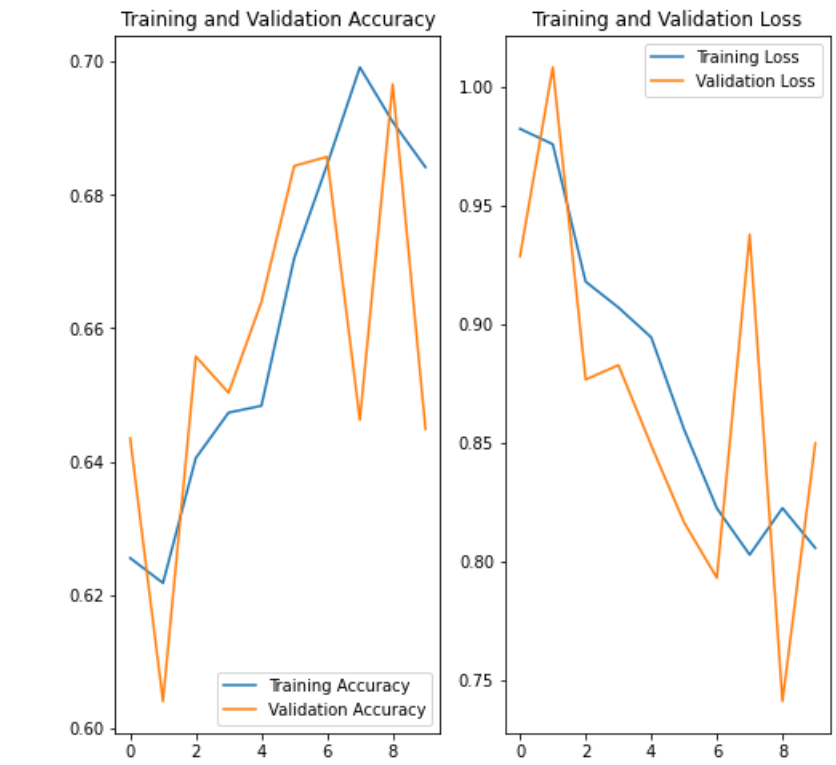
```
[ ]  model = Sequential()

     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 CNN for classifying dog and cat image

## Software Requirements

Google Colab

## 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[i])
    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]:
```
#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]:
```
#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]:
```
#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]:
```
#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
```

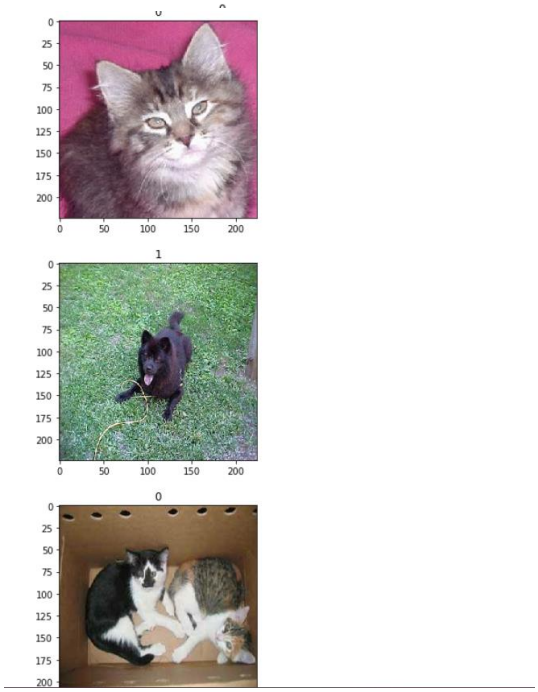
In [33]:
```
if prepare_file():
    print("Files completed!")
```

```
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]:
```
#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]:
```
#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
    }
```

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

In [38]:

```
#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)
```

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

```
In [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/2679.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 Requirements

Google Colab

## 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(".")
```

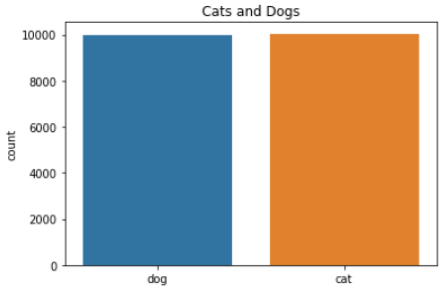
In [6]:
```
train_image_list = os.listdir("./train/")[0:SAMPLE_SIZE]
test_image_list = os.listdir("./test/")
```

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
```

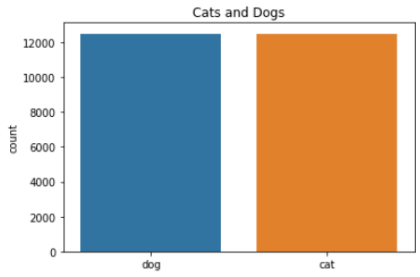
In [9]:

```
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
```

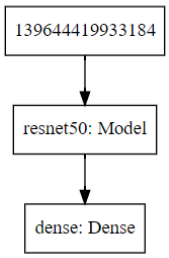
In [17]:
```
model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])
```

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'))
```

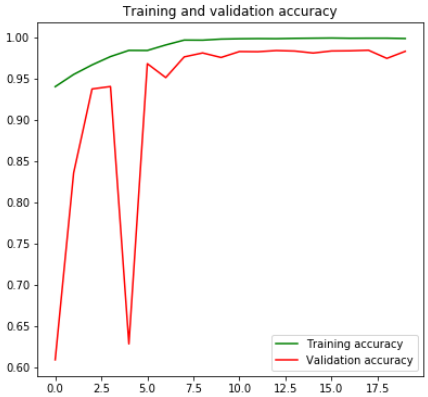
Out[19]:

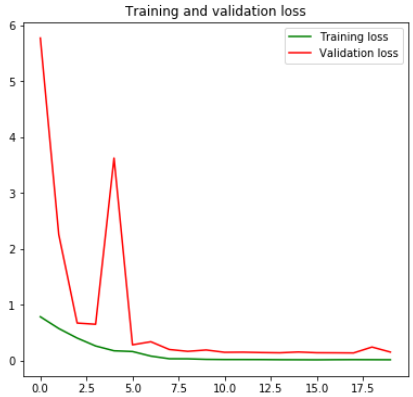



In [21]:
```
train_model = model.fit(X_train, y_train,
                  batch_size=BATCH_SIZE,
                  epochs=NO_EPOCHS,
                  verbose=1,
                  validation_data=(X_val, y_val))
```

```
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 Requirements

Google Colab

## 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

```
#@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 Requirements**

Google Colab

**Code and Output:**

Save the date! TensorFlow is back at Google I/O on May 10

://io.google/2023/?utm_source=devsite-hpp&utm_medium=embedded_marketing&utm_campaign=tf_ba

# Convolutional Neural Network (CNN)

Run in Google Colab (https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials

This tutorial demonstrates training a simple Convolutional Neural Network (https://developers.google.com/machine-learning/glossary/#convolutional_neural_network) (CNN) to classify CIFAR images (https://www.cs.toronto.edu/%7Ekriz/cifar.html). Because this tutorial uses the Keras Sequential API (https://www.tensorflow.org/guide/keras/overview), creating and training your model will take just a few lines of code.

## Import TensorFlow

```
import tensorflow as tf

from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

```
2022-12-14 02:35:18.952623: W tensorflow/compiler/xla/stream_executor/platfor
2022-12-14 02:35:18.952732: W tensorflow/compiler/xla/stream_executor/platfor
2022-12-14 02:35:18.952748: W tensorflow/compiler/tf2tensorrt/utils/py_utils.
```

## Download and prepare the CIFAR10 dataset

The CIFAR10 dataset contains 60,000 color images in 10 classes, with 6,000 images in each class. The dataset is divided into 50,000 training images and 10,000 testing images. The classes are mutually exclusive and there is no overlap between them.

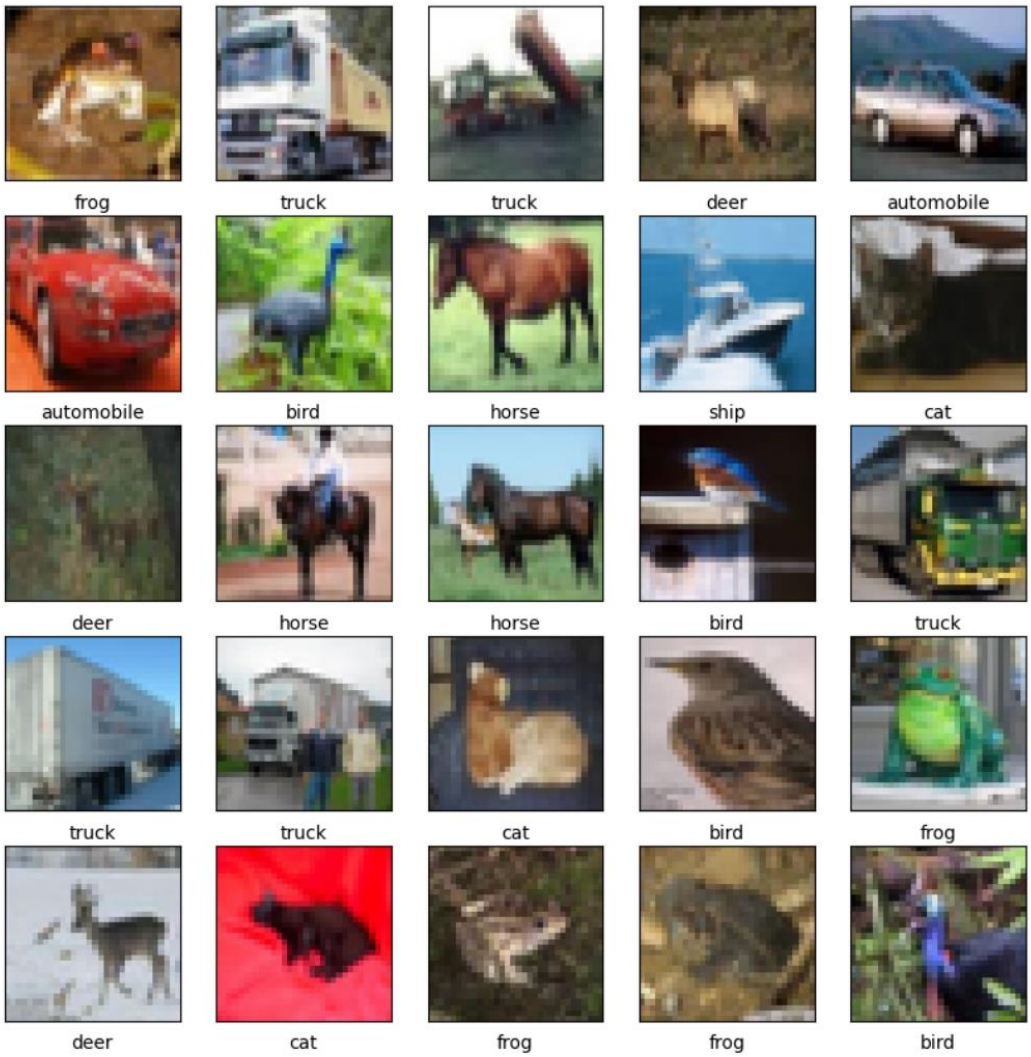
```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.l

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 3s 0us/step
```

## Verify the data

To verify that the dataset looks correct, let's plot the first 25 images from the training set and display the class name below each image:

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```




## Create the convolutional base

The 6 lines of code below define the convolutional base using a common pattern: a stack of Conv2D (https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D) and MaxPooling2D (https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D) layers.

As input, a CNN takes tensors of shape (image_height, image_width, color_channels), ignoring the batch size. If you are new to these dimensions, color_channels refers to (R,G,B). In this example, you will configure your CNN to process inputs of shape (32, 32, 3), which is the format of CIFAR images. You can do this by passing the argument `input_shape` to your first layer.

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

Let's display the architecture of your model so far:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)
```

Above, you can see that the output of every Conv2D and MaxPooling2D layer is a 3D tensor of shape (height, width, channels). The width and height dimensions tend to shrink as you go deeper in the network. The number of output channels for each Conv2D layer is controlled by the first argument (e.g., 32 or 64). Typically, as the width and height shrink, you can afford (computationally) to add more output channels in each Conv2D layer.

## Add Dense layers on top

To complete the model, you will feed the last output tensor from the convolutional base (of shape (4, 4, 64)) into one or more Dense layers to perform classification. Dense layers take vectors as input (which are 1D), while the current output is a 3D tensor. First, you will flatten (or unroll) the 3D output to 1D, then add one or more Dense layers on top. CIFAR has 10 output classes, so you use a final Dense layer with 10 outputs.

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

Here's the complete architecture of your model:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)
```

The network summary shows that (4, 4, 64) outputs were flattened into vectors of shape (1024) before going through two Dense layers.

## Compile and train the model

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

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

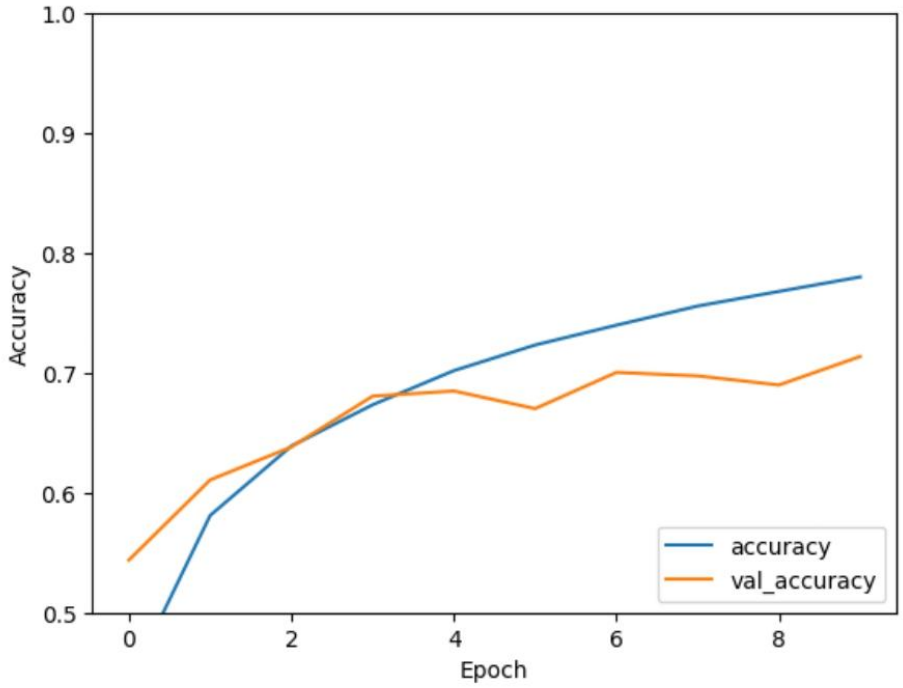
```
Epoch 1/10
1563/1563 [==============================] - 10s 4ms/step - loss: 1.5316 - ac
Epoch 2/10
```

```
1563/1563 [==============================] - 6s 4ms/step - loss: 1.1786 - acc
Epoch 3/10
1563/1563 [==============================] - 6s 4ms/step - loss: 1.0249 - acc
Epoch 4/10
1563/1563 [==============================] - 6s 4ms/step - loss: 0.9256 - acc
Epoch 5/10
1563/1563 [==============================] - 6s 4ms/step - loss: 0.8491 - acc
Epoch 6/10
1563/1563 [==============================] - 7s 4ms/step - loss: 0.7900 - acc
```

## Evaluate the model

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

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 1s - loss: 0.8744 - accuracy: 0.7137 - 652ms/epoch - 2ms/step
```




```
print(test_acc)
```

```
0.713699996471405
```

Your simple CNN has achieved a test accuracy of over 70%. Not bad for a few lines of code! For another CNN style, check out the TensorFlow 2 quickstart for experts (https://www.tensorflow.org/tutorials/quickstart/advanced) example that uses the Keras subclassing API and `tf.GradientTape` (https://www.tensorflow.org/api_docs/python/tf/GradientTape).



## Conclusion

Implementation was successful.

## Experiment 8

### Aim

Generating Images with BigGAN

### Software Requirements

Google Colab

### 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
```

```
[3]  import tensorflow.compat.v1 as tf
     tf.disable_v2_behavior()

     import os
     import io
     import IPython.display
     import numpy as np
     import PIL.Image
     from scipy.stats import truncnorm
     import tensorflow_hub as hub
```

```
WARNING:tensorflow:From /usr/local/lib/python3.9/dist-packages/tensorflow/python/compat/v2_compat.py:107: disable_resource_variables (from tensorflow.python.ops.variable_scope)
Instructions for updating:
non-resource variables are not supported in the long term
```

[4] 1m
```
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] 0s
```
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]   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]
      return grid
```

```
[5]   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]  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,
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 Requirements**

Google Colab

**Code and Output:**

Save the date! TensorFlow is back at Google I/O on May 10

://io.google/2023/?utm_source=devsite-hpp&utm_medium=embedded_marketing&utm_campaign=tf_ba

# Neural machine translation with a Transformer and Keras



Run in Google Colab (https://colab.research.google.com/github/tensorflow/text/blob/master/docs/tutorials/tr

This tutorial demonstrates how to create and train a sequence-to-sequence (https://developers.google.com/machine-learning/glossary#sequence-to-sequence-task) Transformer (https://developers.google.com/machine-learning/glossary#Transformer) model to translate Portuguese into English (https://www.tensorflow.org/datasets/catalog/ted_hrlr_translate#ted_hrlr_translatept_to_en). The Transformer was originally proposed in "Attention is all you need" (https://arxiv.org/abs/1706.03762) by Vaswani et al. (2017).

Transformers are deep neural networks that replace CNNs and RNNs with self-attention (https://developers.google.com/machine-learning/glossary#self-attention). Self attention allows Transformers to easily transmit information across the input sequences.

As explained in the Google AI Blog post (https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html):

> Neural networks for machine translation typically contain an encoder reading the input sentence and generating a representation of it. A decoder then generates the output sentence word by word while consulting the representation generated by the encoder. The Transformer starts by generating initial representations, or embeddings, for each word... Then, using self-attention, it aggregates information from all of the other words, generating a new representation per word informed by the entire context, represented by the filled balls. This step is then repeated multiple times in parallel for all words, successively generating new representations.

Figure 1: Applying the Transformer to machine translation. Source: Google AI Blog (https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html).

That's a lot to digest, the goal of this tutorial is to break it down into easy to understand parts. In this tutorial you will:

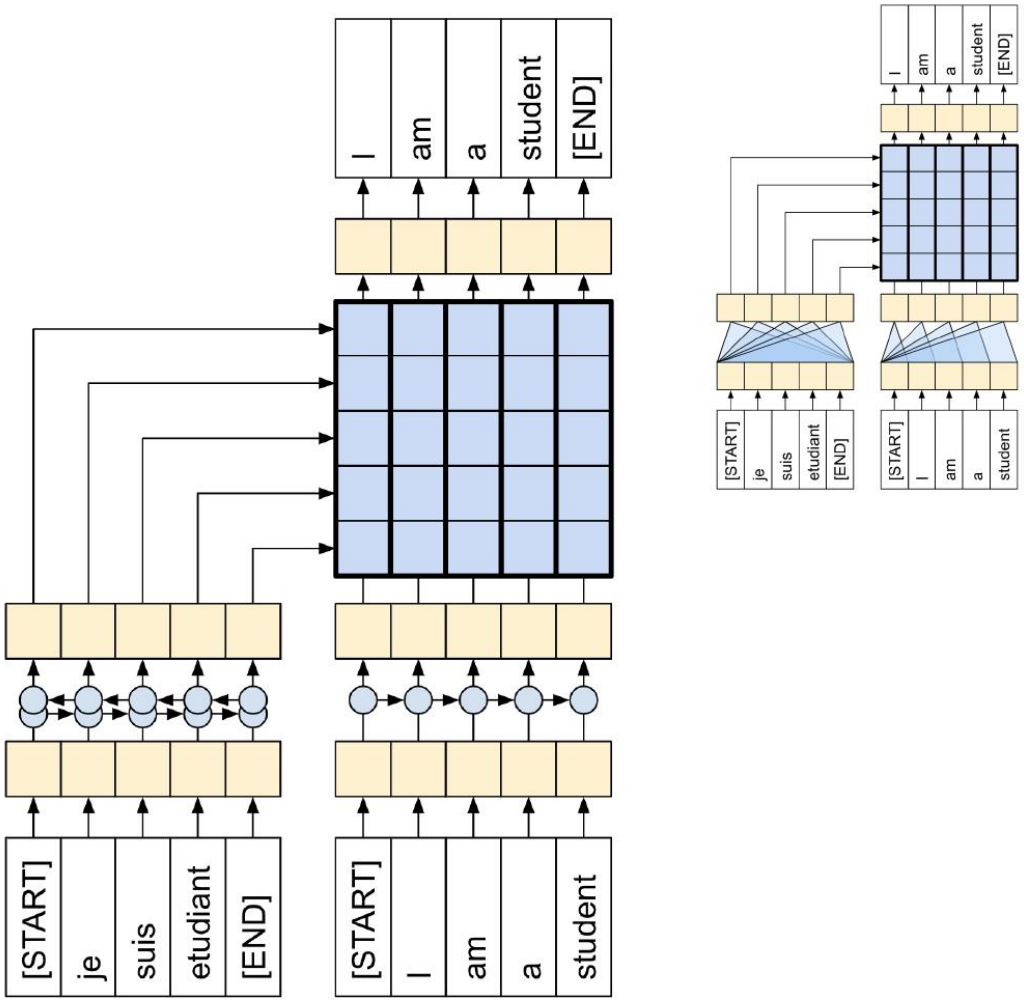
- Prepare the data.
- Implement necessary components:
    - Positional embeddings.
    - Attention layers.
    - The encoder and decoder.
- Build & train the Transformer.
- Generate translations.

- Export the model.

To get the most out of this tutorial, it helps if you know about the basics of text generation (https://www.tensorflow.org/text/tutorials/text_generation) and attention mechanisms.

A Transformer is a sequence-to-sequence encoder-decoder model similar to the model in the NMT with attention tutorial (https://www.tensorflow.org/text/tutorials/nmt_with_attention). A single-layer Transformer takes a little more code to write, but is almost identical to that encoder-decoder RNN model. The only difference is that the RNN layers are replaced with self attention layers. This tutorial builds a 4-layer Transformer which is larger and more powerful, but not fundamentally more complex.

| The RNN+Attention model (https://www.tensorflow.org/text/tutorials/nmt_with_attention) | A 1-layer transformer |
| --- | --- |



After training the model in this notebook, you will be able to input a Portuguese sentence and return the English translation.

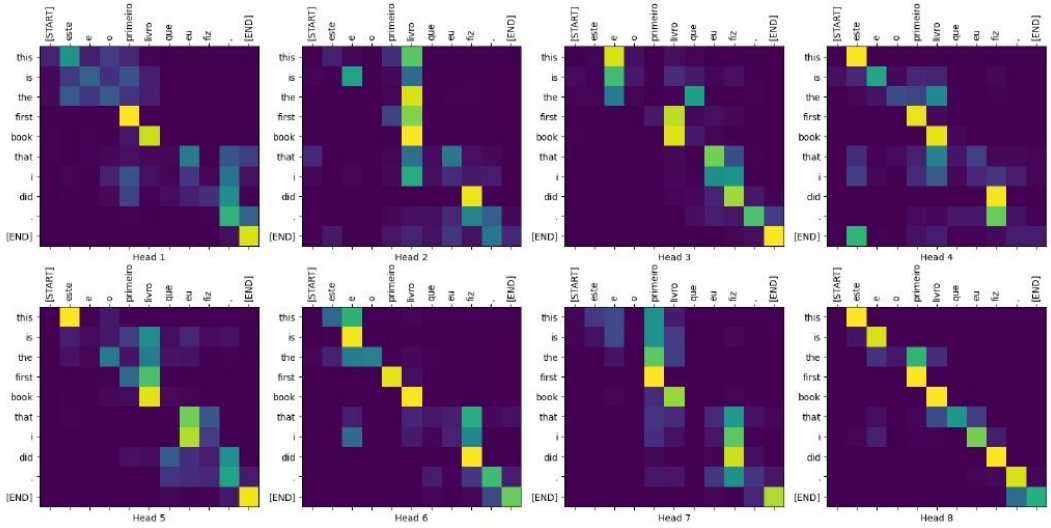



Figure 2: Visualized attention weights that you can generate at the end of this tutorial.

## Why Transformers are significant

- Transformers excel at modeling sequential data, such as natural language.
- Unlike the recurrent neural networks (RNNs) (https://www.tensorflow.org/text/tutorials/text_generation), Transformers are parallelizable. This makes them efficient on hardware like GPUs and TPUs. The main reasons is that Transformers replaced recurrence with attention, and computations can happen simultaneously. Layer outputs can be computed in parallel, instead of a series like an RNN.
- Unlike RNNs (https://www.tensorflow.org/guide/keras/rnn) (like seq2seq, 2014 (https://arxiv.org/abs/1409.3215)) or convolutional neural networks (CNNs) (https://www.tensorflow.org/tutorials/images/cnn) (for example, ByteNet (https://arxiv.org/abs/1610.10099)), Transformers are able to capture distant or long-range contexts and dependencies in the data between distant positions in the input or output sequences. Thus, longer connections can be learned. Attention allows each location to have access to the entire input at each layer, while in RNNs and CNNs, the information needs to pass through many processing steps to move a long distance, which makes it harder to learn.

- Transformers make no assumptions about the temporal/spatial relationships across the data. This is ideal for processing a set of objects (for example, StarCraft units (https://www.deepmind.com/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii)).

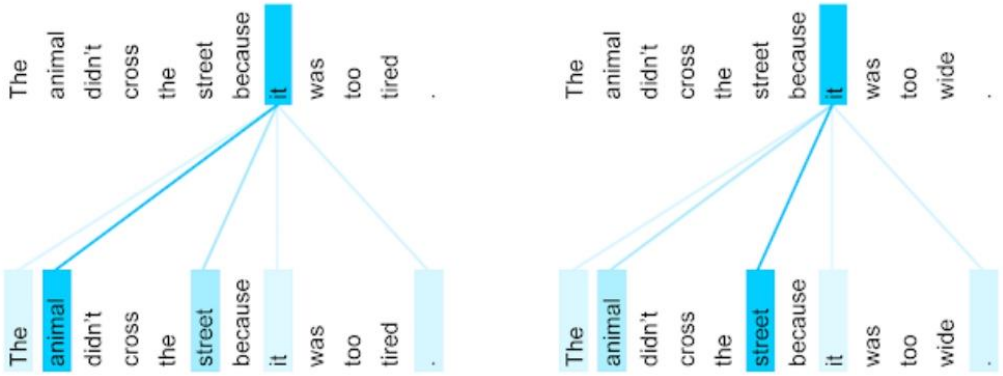



Figure 3: The encoder self-attention distribution for the word “it” from the 5th to the 6th layer of a Transformer trained on English-to-French translation (one of eight attention heads). Source: Google AI Blog (https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html).

## Setup

Begin by installing TensorFlow Datasets (https://tensorflow.org/datasets) for loading the dataset and TensorFlow Text (https://www.tensorflow.org/text) for text preprocessing:

```
$ # 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
```

## Data handling

This section downloads the dataset and the subword tokenizer, from this tutorial (https://www.tensorflow.org/text/guide/subwords_tokenizer), then wraps it all up in a `tf.data.Dataset` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset) for training.

Toggle section

### Download the dataset

Use TensorFlow Datasets to load the Portuguese-English translation dataset (https://www.tensorflow.org/datasets/catalog/ted_hrlr_translate#ted_hrlr_translatept_to_en)D Talks Open Translation Project. This dataset contains approximately 52,000 training, 1,200 validation and 1,800 test examples.

```
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` (https://www.tensorflow.org/api_docs/python/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'))
```

```
> Examples in Portuguese:
e quando melhoramos a procura , tiramos a única vantagem da impressão , que é
mas e se estes fatores fossem ativos ?
mas eles não tinham a curiosidade de me testar .

> Examples in English:
and when you improve searchability , you actually take away the one advantage
but what if it were active ?
but they did n't test for curiosity .
```

## Set up the tokenizer

Now that you have loaded the dataset, you need to tokenize the text, so that each element is represented as a token (https://developers.google.com/machine-learning/glossary#token) or token ID (a numeric representation).

Tokenization is the process of breaking up text, into "tokens". Depending on the tokenizer, these tokens can represent sentence-pieces, words, subwords, or characters. To learn more about tokenization, visit this guide (https://www.tensorflow.org/text/guide/tokenizers).

This tutorial uses the tokenizers built in the subword tokenizer (https://www.tensorflow.org/text/guide/subwords_tokenizer) tutorial. That tutorial optimizes two `text.BertTokenizer` (https://www.tensorflow.org/text/api_docs/python/text/BertTokenizer) objects (one for English, one for Portuguese) for **this dataset** and exports them in a TensorFlow `saved_model` format.

> **Note:** This is different from the original paper (https://arxiv.org/pdf/1706.03762.pdf), section 5.1, where they used a single byte-pair tokenizer for both the source and target with a vocabulary-size of 37000.

Download, extract, and import the `saved_model`:

```
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_nar
    cache_dir='.', cache_subdir='', extract=True
)
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/m
184801/184801 [==============================] - 0s 0us/step
'./ted_hrlr_translate_pt_en_converter.zip'
```

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

The **`tf.saved_model`** (https://www.tensorflow.org/api_docs/python/tf/saved_model) contains two text tokenizers, one for English and one for Portuguese. Both have the same methods:

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

```
['detokenize',
 'get_reserved_tokens',
 'get_vocab_path',
 'get_vocab_size',
 'lookup',
 'tokenize',
 'tokenizer',
 'vocab']
```

The **`tokenize`** method converts a batch of strings to a padded-batch of token IDs. This method splits punctuation, lowercases and unicode-normalizes the input before tokenizing. That standardization is not visible here because the input data is already standardized.

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

```
> This is a batch of strings:
and when you improve searchability , you actually take away the one advantage
```

```
but what if it were active ?
but they did n't test for curiosity .
```

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

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

```
> This is a padded-batch of token IDs:
[2, 72, 117, 79, 1259, 1491, 2362, 13, 79, 150, 184, 311, 71, 103, 2308, 74, 2
[2, 87, 90, 107, 76, 129, 1852, 30, 3]
[2, 87, 83, 149, 50, 9, 56, 664, 85, 2512, 15, 3]
```

The `detokenize` method attempts to convert these token IDs back to human-readable text:

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

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

```
> This is human-readable text:
and when you improve searchability , you actually take away the one advantage
but what if it were active ?
but they did n ' t test for curiosity .
```

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

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

```
> This is the text split into tokens:
<tf.RaggedTensor [[b'[START]', b'and', b'when', b'you', b'improve', b'search'
  b',', b'you', b'actually', b'take', b'away', b'the', b'one', b'advantage',
  b'of', b'print', b',', b'which', b'is', b's', b'##ere', b'##nd', b'##ip',
  b'##ity', b'.', b'[END]']                                              ,
 [b'[START]', b'but', b'what', b'if', b'it', b'were', b'active', b'?',
  b'[END]']                                                            ,
 [b'[START]', b'but', b'they', b'did', b'n', b"'", b't', b'test', b'for',
  b'curiosity', b'.', b'[END]']                                         ]>
```

The output demonstrates the "subword" aspect of the subword tokenization.

For example, the word `'searchability'` is decomposed into `'search'` and `'##ability'`, and the word `'serendipity'` into `'s'`, `'##ere'`, `'##nd'`, `'##ip'` and `'##ity'`.

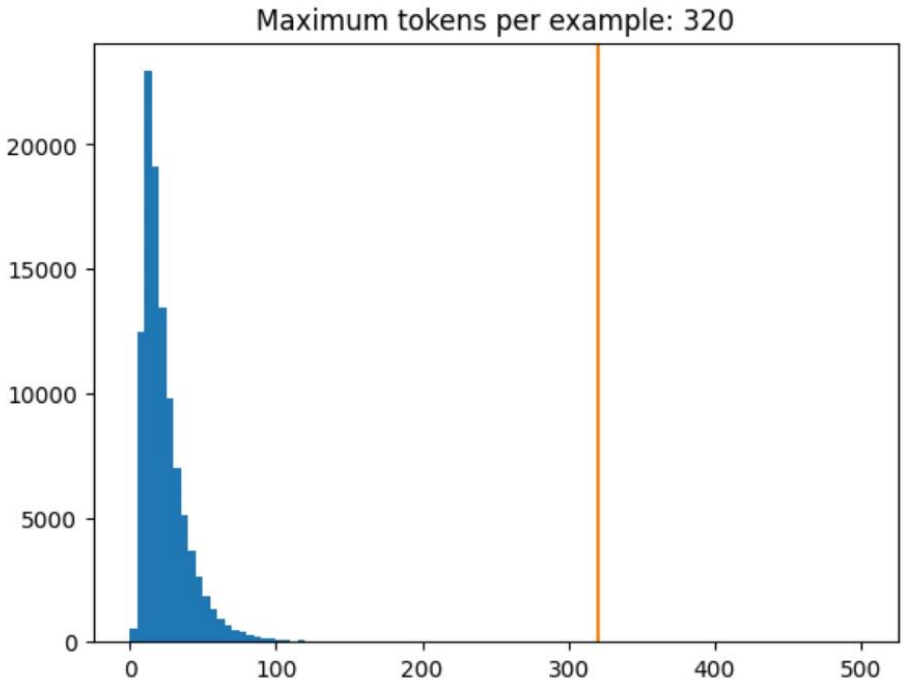
Note that the tokenized text includes `'[START]'` and `'[END]'` tokens.

The distribution of tokens per example in the dataset is as follows:

```
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)
```

```
................................................
```

```
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}');
```




## Set up a data pipeline with `tf.data` (https://www.tensorflow.org/api_docs/python/tf/data

The following function takes batches of text as input, and converts them to a format suitable for training.

1. It tokenizes them into ragged batches.
2. It trims each to be no longer than `MAX_TOKENS`.
3. It splits the target (English) tokens into inputs and labels. These are shifted by one step so that at each input location the `label` is the id of the next token.
4. It converts the `RaggedTensor`s to padded dense `Tensor`s.
5. It returns an `(inputs, labels)` pair.

```
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
```

The function below converts a dataset of text examples into data of batches for training.

1. It tokenizes the text, and filters out the sequences that are too long. (The `batch`/`unbatch` is included because the tokenizer is much more efficient on large batches).
2. The `cache` method ensures that that work is only executed once.
3. Then `shuffle` and, `dense_to_ragged_batch` randomize the order and assemble batches of examples.
4. Finally `prefetch` runs the dataset in parallel with the model to ensure that data is available when needed. See Better performance with the `tf.data` (https://www.tensorflow.org/guide/data_performance.ipynb) for details.

```
BUFFER_SIZE = 20000
BATCH_SIZE = 64
```

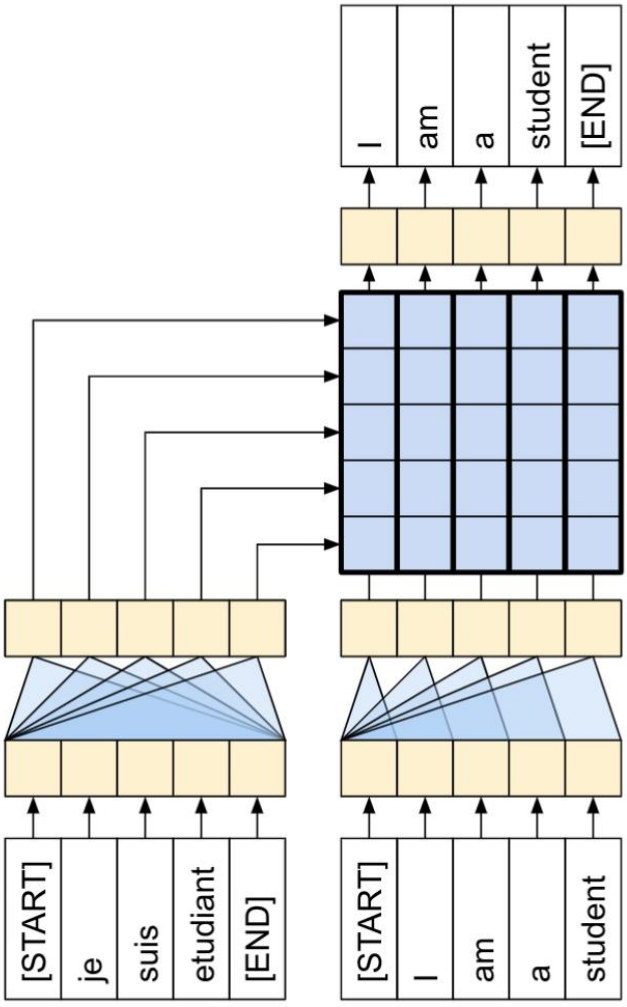
```
def make_batches(ds):
  return (
      ds
      .shuffle(BUFFER_SIZE)
      .batch(BATCH_SIZE)
      .map(prepare_batch, tf.data.AUTOTUNE)
      .prefetch(buffer_size=tf.data.AUTOTUNE))
```

## Test the Dataset

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

The resulting `tf.data.Dataset` (https://www.tensorflow.org/api_docs/python/tf/data/Dataset) objects are setup for training with Keras. Keras `Model.fit` (https://www.tensorflow.org/api_docs/python/tf/keras/Model#fit) training expects `(inputs, labels)` pairs. The `inputs` are pairs of tokenized Portuguese and English sequences, `(pt, en)`. The `labels` are the same English sequences shifted by 1. This shift is so that at each location input `en` sequence, the `label` in the next token.

**Inputs at the bottom, labels at the top.**




This is the same as the text generation tutorial (/text/tutorials/text_generation), except here you have additional input "context" (the Portuguese sequence) that the model is "conditioned" on.

This setup is called "teacher forcing" because regardless of the model's output at each timestep, it gets the true value as input for the next timestep. This is a simple and efficient way to train a text generation model. It's efficient because you don't need to run the model sequentially, the outputs at the different sequence locations can be computed in parallel.

You might have expected the `input, output`, pairs to simply be the `Portuguese, English` sequences. Given the Portuguese sequence, the model would try to generate the English sequence.

It's possible to train a model that way. You'd need to write out the inference loop and pass the model's output back to the input. It's slower (time steps can't run in parallel), and a harder task to learn (the model can't get the end of a sentence right until it gets the beginning right), but it can give a more stable model because the model has to learn to correct its own errors during training.

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

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

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

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

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

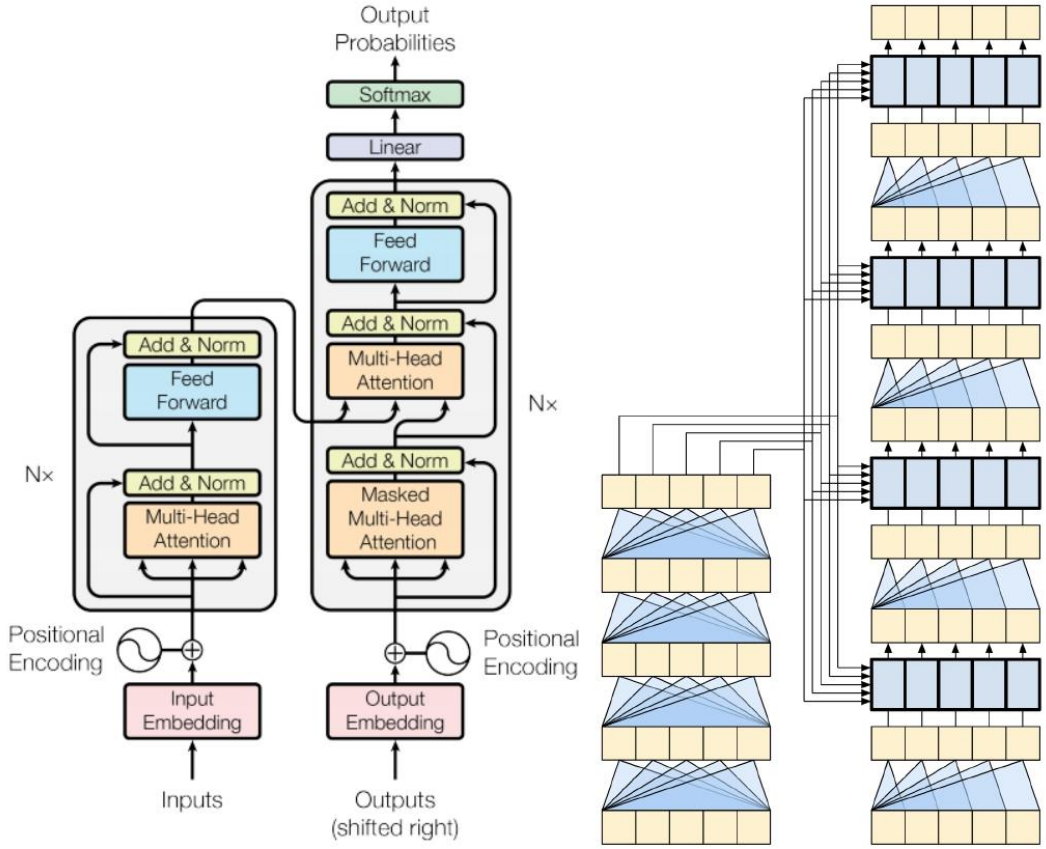
```
tf.Tensor([   2  476 2569 2626 6010   52 2564 1915  188   15], shape=(10,), d
tf.Tensor([ 476 2569 2626 6010   52 2564 1915  188   15    3], shape=(10,), d
```

## Define the components

There's a lot going on inside a Transformer. The important things to remember are:

1. It follows the same general pattern as a standard sequence-to-sequence model with an encoder and a decoder.
2. If you work through it step by step it will all make sense.

| The original Transformer diagram | A representation of a 4-layer Transformer |
|---|---|

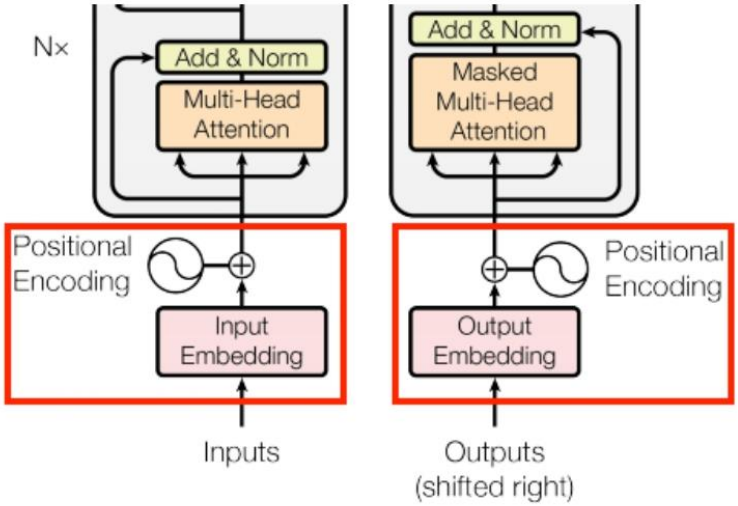



Each of the components in these two diagrams will be explained as you progress through the tutorial.

## The embedding and positional encoding layer

The inputs to both the encoder and decoder use the same embedding and positional encoding logic.

| The embedding and positional encoding layer |
|---|




Given a sequence of tokens, both the input tokens (Portuguese) and target tokens (English) have to be converted to vectors using a `tf.keras.layers.Embedding` (https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding) layer.

The attention layers used throughout the model see their input as a set of vectors, with no order. Since the model doesn't contain any recurrent or convolutional layers. It needs some way to identify word order, otherwise it would see the input sequence as a bag of words (https://developers.google.com/machine-learning/glossary#bag-of-words) instance, `how are you`, `how you are`, `you how are`, and so on, are indistinguishable.

A Transformer adds a "Positional Encoding" to the embedding vectors. It uses a set of sines and cosines at different frequencies (across the sequence). By definition nearby elements will have similar position encodings.

The formula for calculating the positional encoding (implemented in Python below) is as follows:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

```
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)
```

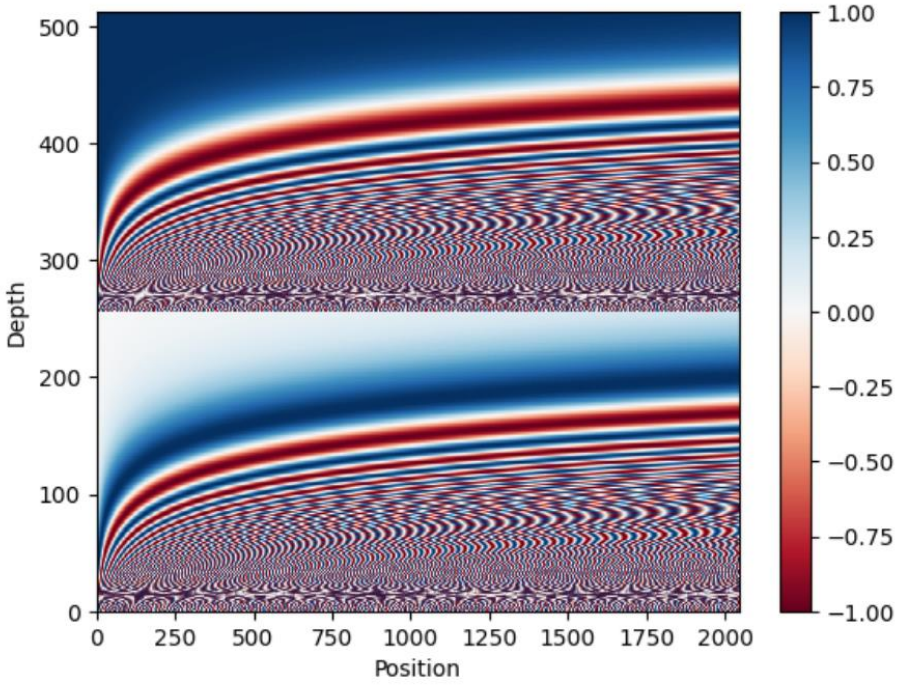
```
  angle_rates = 1 / (10000**depths)         # (1, depth)
  angle_rads = positions * angle_rates      # (pos, depth)

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

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

The position encoding function is a stack of sines and cosines that vibrate at different frequencies depending on their location along the depth of the embedding vector. They vibrate across the position axis.

Toggle code

```
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()
```

```
(2048, 512)
```

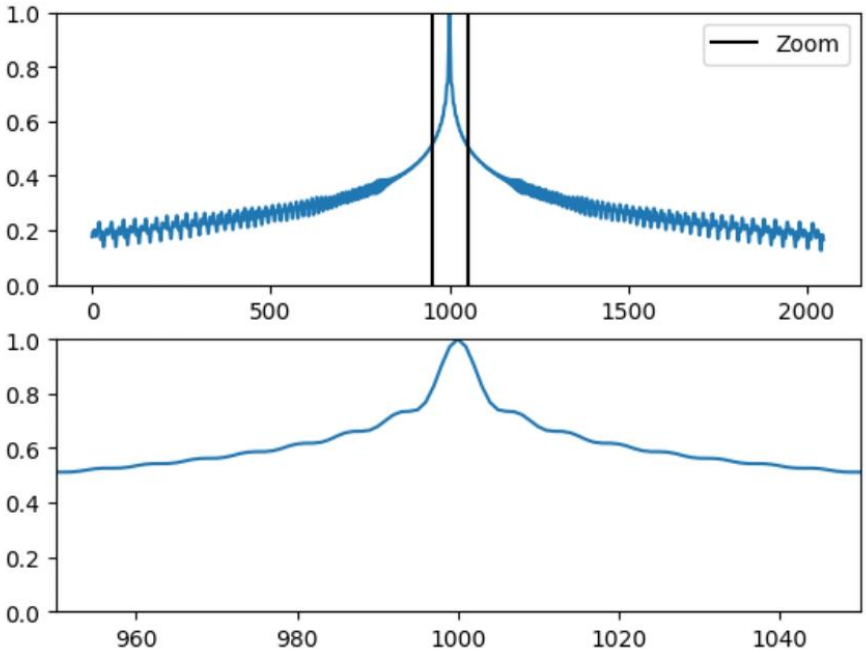



By definition these vectors align well with nearby vectors along the position axis. Below the position encoding vectors are normalized and the vector from position `1000` is compared, by dot-product, to all the others:

Toggle code

```
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)
```




So use this to create a `PositionEmbedding` layer that looks-up a token's embedding vector and adds the position vector:

```
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_enco
    x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
    x = x + self.pos_encoding[tf.newaxis, :length, :]
    return x
```

**Note:** The original paper (https://arxiv.org/pdf/1706.03762.pdf), section 3.4 and 5.1, uses a single tokenizer and weight matrix for both the source and target languages. This tutorial uses two separate tokenizers and weight matrices.

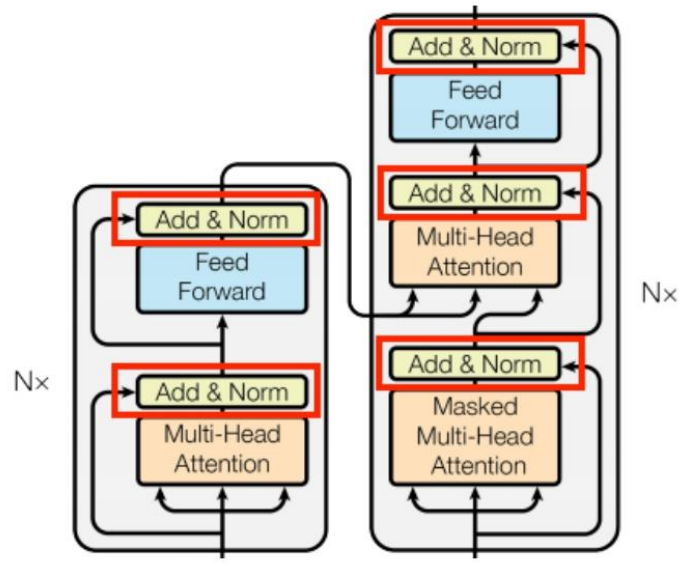
```
embed_pt = PositionalEmbedding(vocab_size=tokenizers.pt.get_vocab_size(), d_m
embed_en = PositionalEmbedding(vocab_size=tokenizers.en.get_vocab_size(), d_m

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]])>
```

## Add and normalize

**Add and normalize**




These "Add & Norm" blocks are scattered throughout the model. Each one joins a residual connection and runs the result through a `LayerNormalization` layer.

The easiest way to organize the code is around these residual blocks. The following sections will define custom layer classes for each.

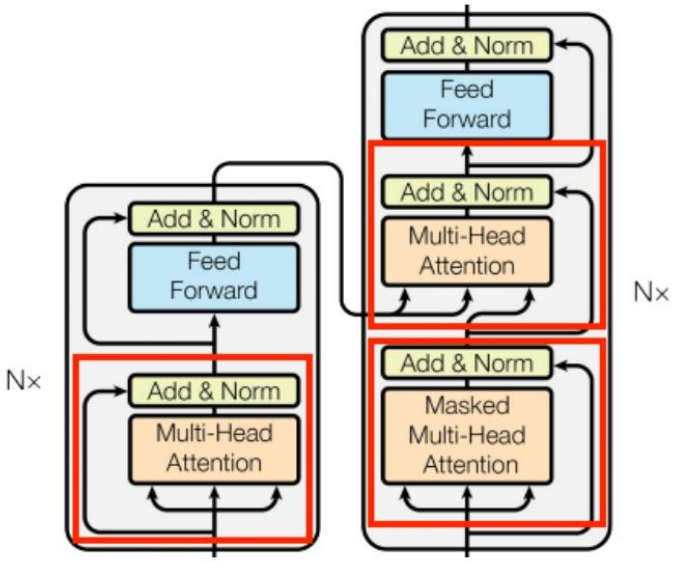
The residual "Add & Norm" blocks are included so that training is efficient. The residual connection provides a direct path for the gradient (and ensures that vectors are **updated** by the attention layers instead of **replaced**), while the normalization maintains a reasonable scale for the outputs.

**Note:** The implementations, below, use the `Add` layer to ensure that Keras masks are propagated (the `+` operator does not).

## The base attention layer

Attention layers are used throughout the model. These are all identical except for how the attention is configured. Each one contains a `layers.MultiHeadAttention` (https://www.tensorflow.org/addons/api_docs/python/tfa/layers/MultiHeadAttention), a `layers.LayerNormalization` (https://www.tensorflow.org/api_docs/python/tf/keras/layers/LayerNormalization) and a `layers.Add` (https://www.tensorflow.org/api_docs/python/tf/keras/layers/Add).

**The base attention layer**



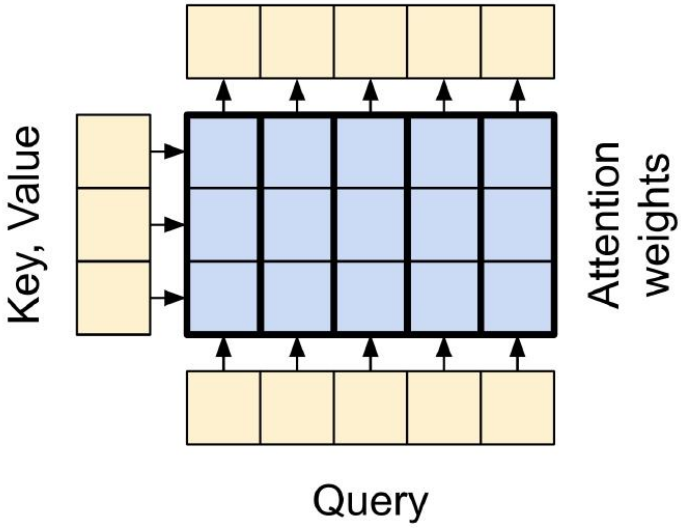

To implement these attention layers, start with a simple base class that just contains the component layers. Each use-case will be implemented as a subclass. It's a little more code to write this way, but it keeps the intention clear.

```
class BaseAttention(tf.keras.layers.Layer):
  def __init__(self, **kwargs):
    super().__init__()
    self.mha = tf.keras.layers.MultiHeadAttention(**kwargs)
    self.layernorm = tf.keras.layers.LayerNormalization()
    self.add = tf.keras.layers.Add()
```

**Attention refresher**

Before you get into the specifics of each usage, here is a quick refresher on how attention works:

**The base attention layer**




There are two inputs:

1. The query sequence; the sequence being processed; the sequence doing the attending (bottom).
2. The context sequence; the sequence being attended to (left).

The output has the same shape as the query-sequence.

The common comparison is that this operation is like a dictionary lookup. A **fuzzy**, **differentiable**, **vectorized** dictionary lookup.

Here's a regular python dictionary, with 3 keys and 3 values being passed a single query.

```
d = {'color': 'blue', 'age': 22, 'type': 'pickup'}
result = d['color']
```

- The `query`s is what you're trying to find.
- The `key`s what sort of information the dictionary has.
- The `value` is that information.

When you look up a `query` in a regular dictionary, the dictionary finds the matching `key`, and returns its associated `value`. The `query` either has a matching `key` or it doesn't. You can imagine a **fuzzy** dictionary where the keys don't have to match perfectly. If you looked up

`d["species"]` in the dictionary above, maybe you'd want it to return `"pickup"` since that's the best match for the query.

An attention layer does a fuzzy lookup like this, but it's not just looking for the best key. It combines the `values` based on how well the `query` matches each `key`.

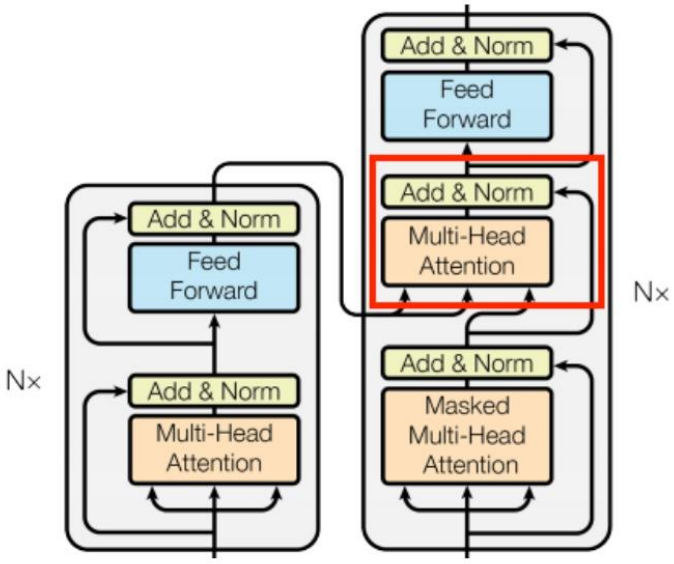
How does that work? In an attention layer the `query`, `key`, and `value` are each vectors. Instead of doing a hash lookup the attention layer combines the `query` and `key` vectors to determine how well they match, the "attention score". The layer returns the average across all the `values`, weighted by the "attention scores".

Each location the query-sequence provides a `query` vector. The context sequence acts as the dictionary. At each location in the context sequence provides a `key` and `value` vector. The input vectors are not used directly, the `layers.MultiHeadAttention` (https://www.tensorflow.org/addons/api_docs/python/tfa/layers/MultiHeadAttention) layer includes `layers.Dense` (https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense) layers to project the input vectors before using them.

## The cross attention layer

At the literal center of the Transformer is the cross-attention layer. This layer connects the encoder and decoder. This layer is the most straight-forward use of attention in the model, it performs the same task as the attention block in the NMT with attention tutorial (https://www.tensorflow.org/text/tutorials/nmt_with_attention).

**The cross attention layer**



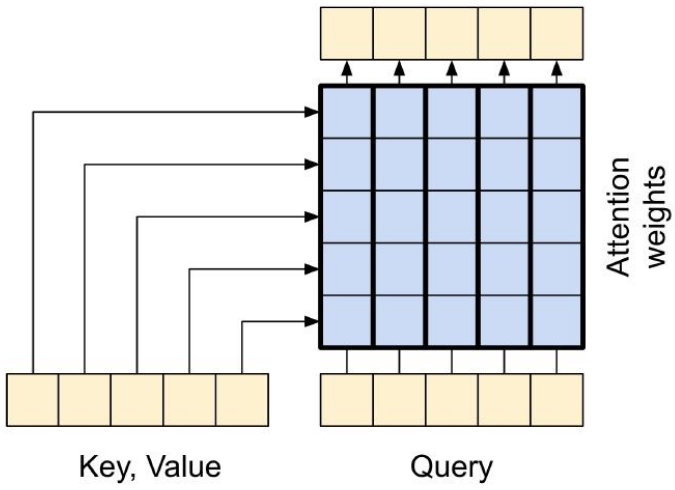
To implement this you pass the target sequence `x` as the `query` and the `context` sequence as the `key/value` when calling the `mha` layer:

```
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
```

The caricature below shows how information flows through this layer. The columns represent the weighted sum over the context sequence.

For simplicity the residual connections are not shown.

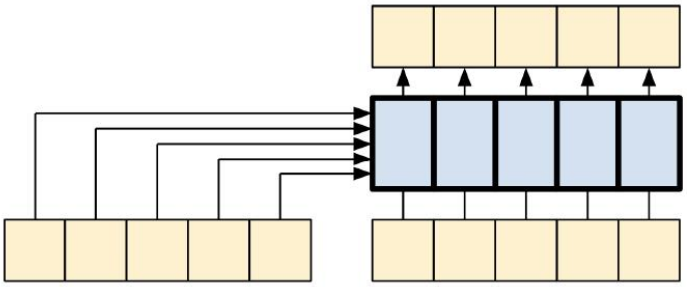
**The cross attention layer**



The output length is the length of the `query` sequence, and not the length of the context `key/value` sequence.

The diagram is further simplified, below. There's no need to draw the entire "Attention weights" matrix. The point is that each `query` location can see all the `key/value` pairs in the context, but no information is exchanged between the queries.

**Each query sees the whole context.**



Test run it on sample inputs:

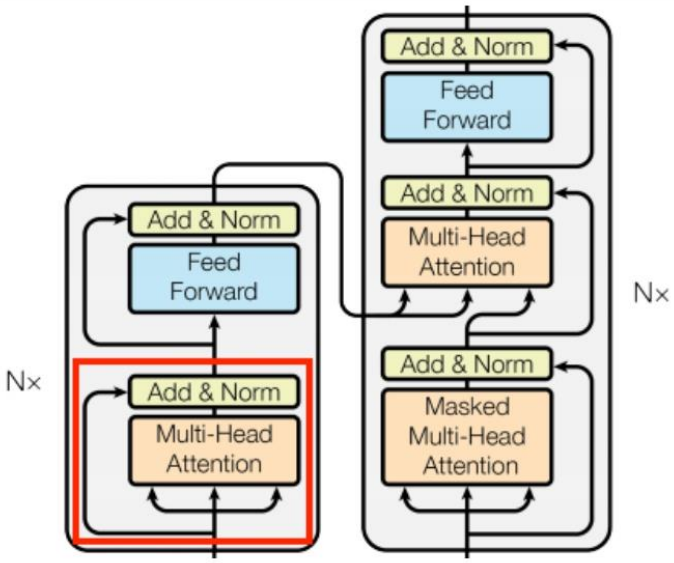
```
sample_ca = CrossAttention(num_heads=2, key_dim=512)

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)
```

## The global self attention layer

This layer is responsible for processing the context sequence, and propagating information along its length:

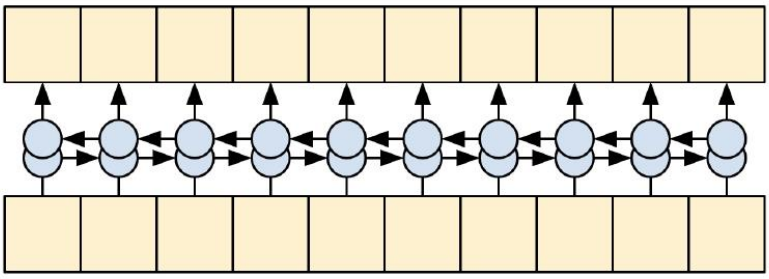
**The global self attention layer**

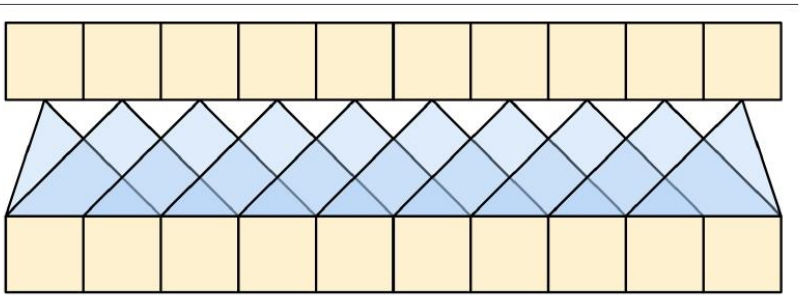



Since the context sequence is fixed while the translation is being generated, information is allowed to flow in both directions.

Before Transformers and self attention, models commonly used RNNs or CNNs to do this task:

**Bidirectional RNNs and CNNs**





RNNs and CNNs have their limitations.

- The RNN allows information to flow all the way across the sequence, but it passes through many processing steps to get there (limiting gradient flow). These RNN steps have to be run sequentially and so the RNN is less able to take advantage of modern parallel devices.
- In the CNN each location can be processed in parallel, but it only provides a limited receptive field. The receptive field only grows linearly with the number of CNN layers, You need to stack a number of Convolution layers to transmit information across the sequence (Wavenet (https://arxiv.org/abs/1609.03499) reduces this problem by using dilated convolutions).

The global self attention layer on the other hand lets every sequence element directly access every other sequence element, with only a few operations, and all the outputs can be computed in parallel.

To implement this layer you just need to pass the target sequence, `x`, as both the `query`, and `value` arguments to the `mha` layer:

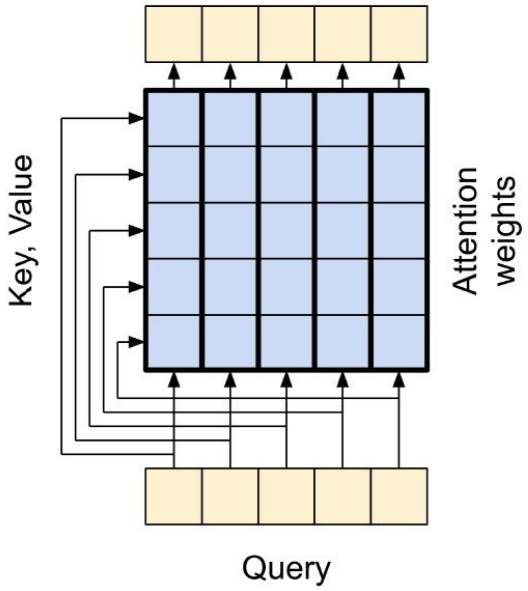
```
class GlobalSelfAttention(BaseAttention):
  def call(self, x):
    attn_output = self.mha(
        query=x,
        value=x,
        key=x)
    x = self.add([x, attn_output])
    x = self.layernorm(x)
    return x
```

```
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)
```

Sticking with the same style as before you could draw it like this:

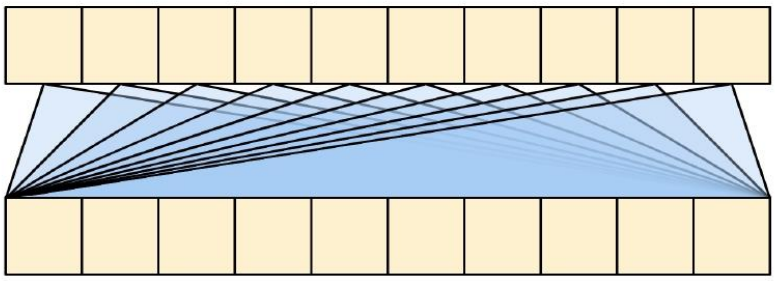
The global self attention layer




Again, the residual connections are omitted for clarity.

It's more compact, and just as accurate to draw it like this:

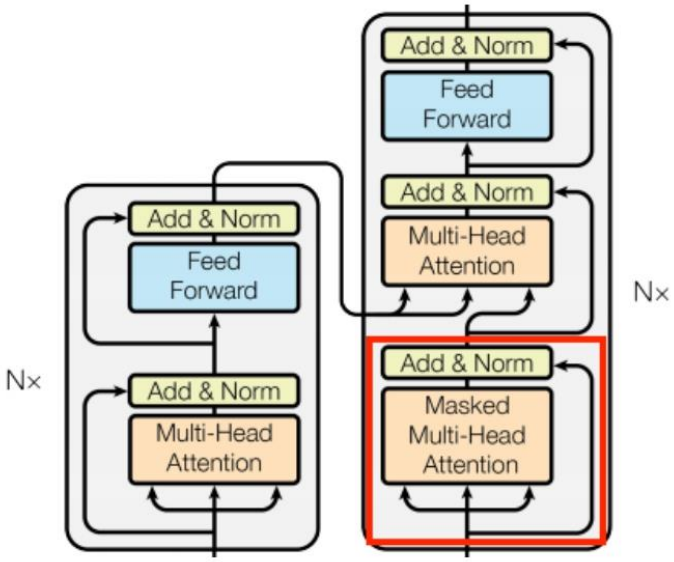
The global self attention layer



## The causal self attention layer

This layer does a similar job as the global self attention layer, for the output sequence:

The causal self attention layer

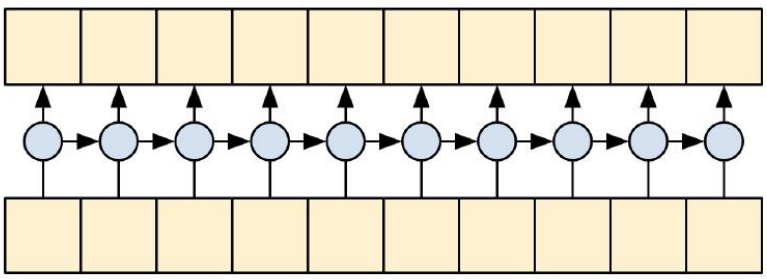


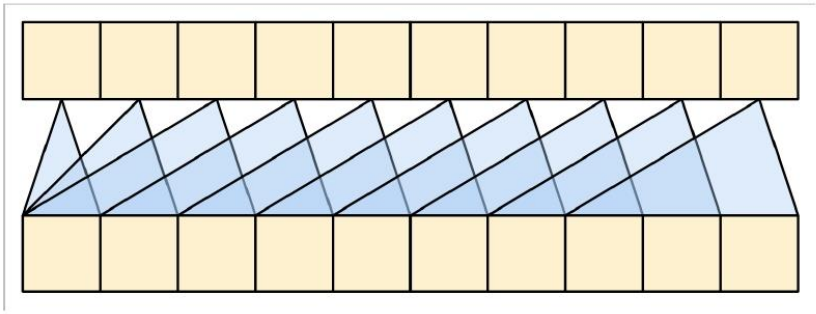

This needs to be handled differently from the encoder's global self attention layer.

Like the text generation tutorial (https://www.tensorflow.org/text/tutorials/text_generation), and the NMT with attention (https://www.tensorflow.org/text/tutorials/nmt_with_attention) tutorial, Transformers are an "autoregressive" model: They generate the text one token at a time and feed that output back to the input. To make this *efficient*, these models ensure that the output for each sequence element only depends on the previous sequence elements; the models are "causal".

A single-direction RNN is causal by definition. To make a causal convolution you just need to pad the input and shift the output so that it aligns correctly (use `layers.Conv1D(padding='causal')`).

**Causal RNNs and CNNs**





A causal model is efficient in two ways:

1. In training, it lets you compute loss for every location in the output sequence while executing the model just once.
2. During inference, for each new token generated you only need to calculate its outputs, the outputs for the previous sequence elements can be reused.
    - For an RNN you just need the RNN-state to account for previous computations (pass `return_state=True` to the RNN layer's constructor).
    - For a CNN you would need to follow the approach of Fast Wavenet (https://arxiv.org/abs/1611.09482)

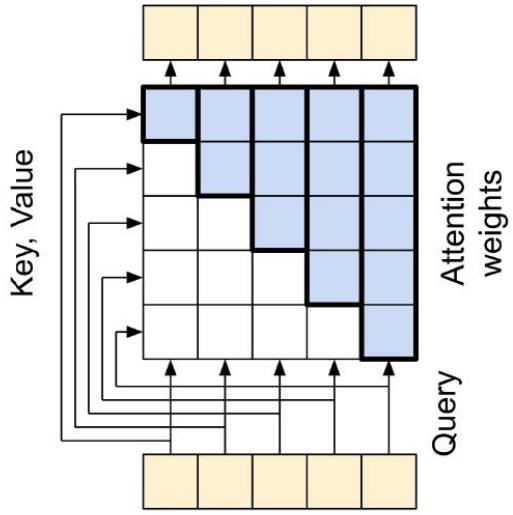
To build a causal self attention layer, you need to use an appropriate mask when computing the attention scores and summing the attention `value`s.

This is taken care of automatically if you pass `use_causal_mask = True` to the `MultiHeadAttention` layer when you call it:

```
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
```

The causal mask ensures that each location only has access to the locations that come before it:

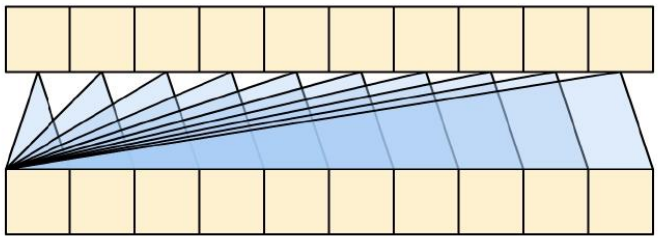
The causal self attention layer




Again, the residual connections are omitted for simplicity.

The more compact representation of this layer would be:

The causal self attention layer



Test out the layer:

```
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()
```

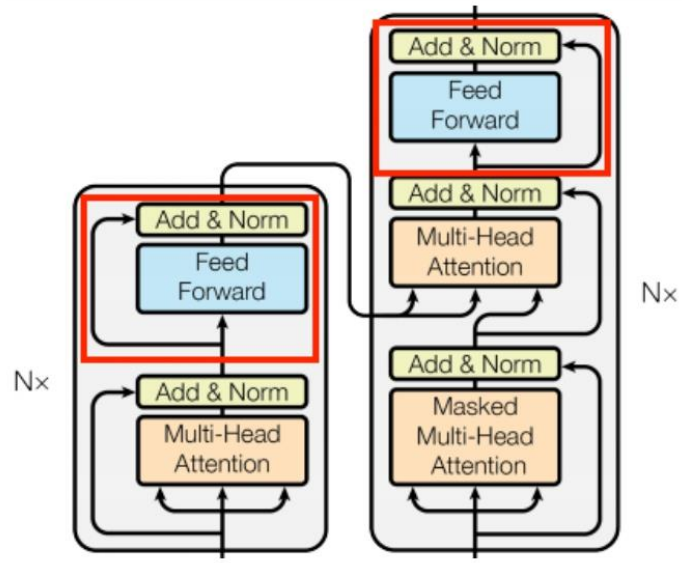
```
4.7683716e-07
```

**Note:** When using Keras masks, the output values at invalid locations are not well defined. So the above may not hold for masked regions.

## The feed forward network

The transformer also includes this point-wise feed-forward network in both the encoder and decoder:

**The feed forward network**




The network consists of two linear layers (`tf.keras.layers.Dense` (https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense)) with a ReLU activation in-between, and a dropout layer. As with the attention layers the code here also includes the residual connection and normalization:

```
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)
```

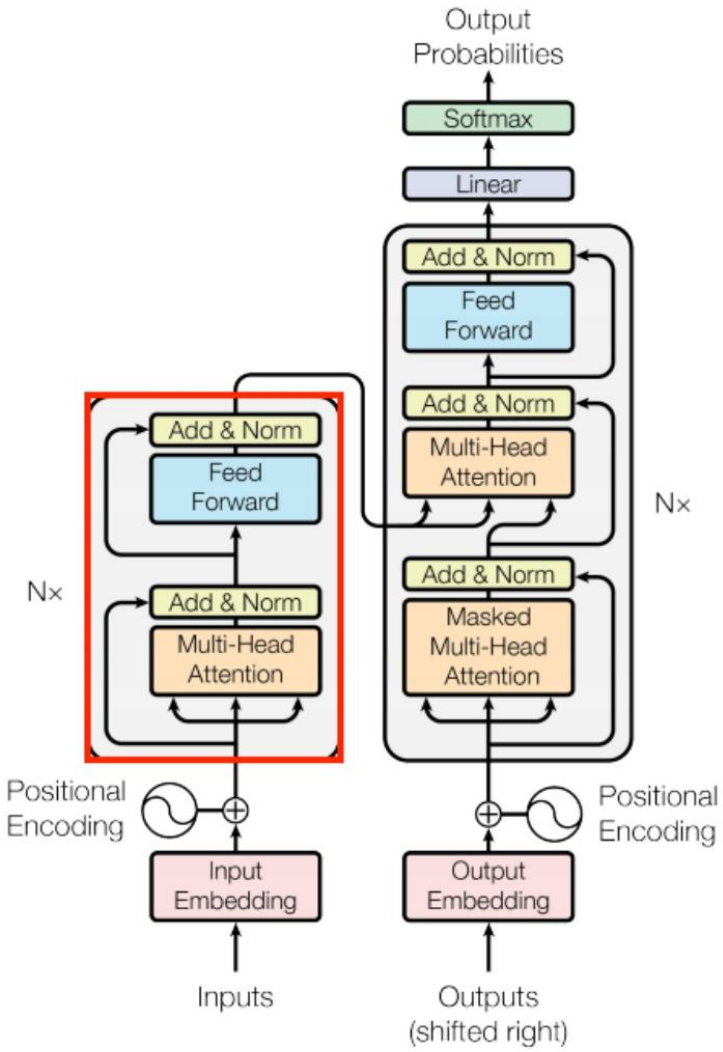
```
print(sample_ffn(en_emb).shape)
```

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

## The encoder layer

The encoder contains a stack of `N` encoder layers. Where each `EncoderLayer` contains a `GlobalSelfAttention` and `FeedForward` layer:

The encoder layer




Here is the definition of the `EncoderLayer`:

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

    self.self_attention = GlobalSelfAttention(
        num_heads=num_heads,
        key_dim=d_model,
        dropout=dropout_rate)
```

```
    self.ffn = FeedForward(d_model, dff)

  def call(self, x):
    x = self.self_attention(x)
    x = self.ffn(x)
    return x
```

And a quick test, the output will have the same shape as the input:

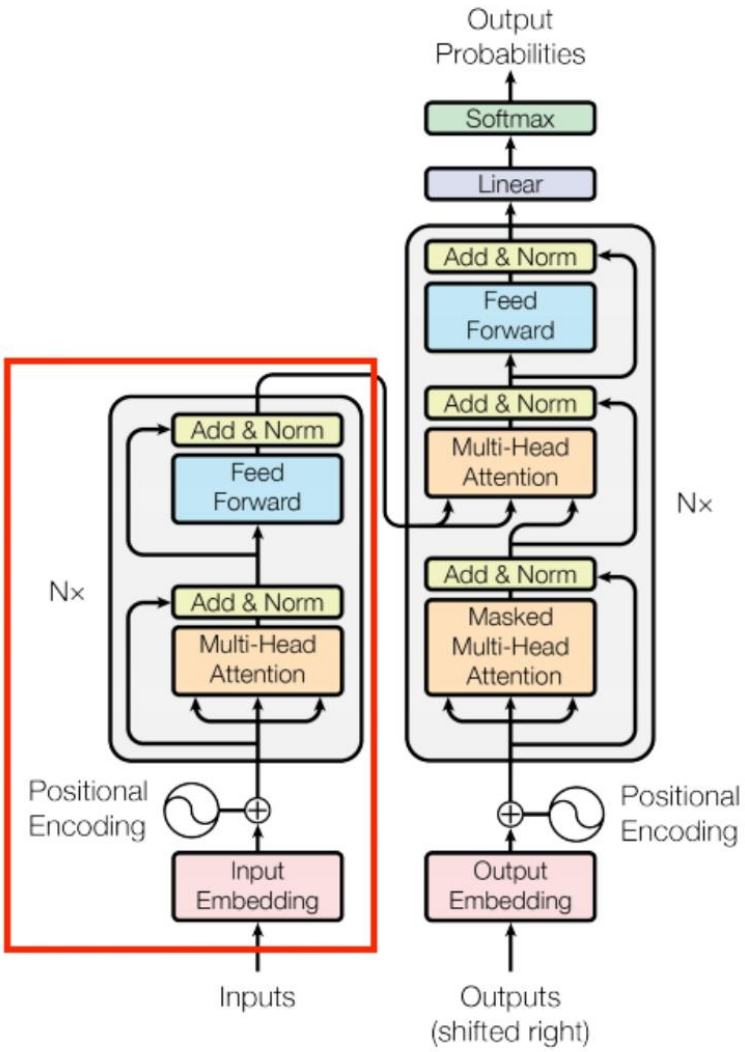
```
sample_encoder_layer = EncoderLayer(d_model=512, num_heads=8, dff=2048)

print(pt_emb.shape)
print(sample_encoder_layer(pt_emb).shape)
```

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

## The encoder

Next build the encoder.

**The encoder**




The encoder consists of:

- A `PositionalEmbedding` layer at the input.
- A stack of `EncoderLayer` layers.

```
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)`.
```

Test the encoder:

```
# 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
```

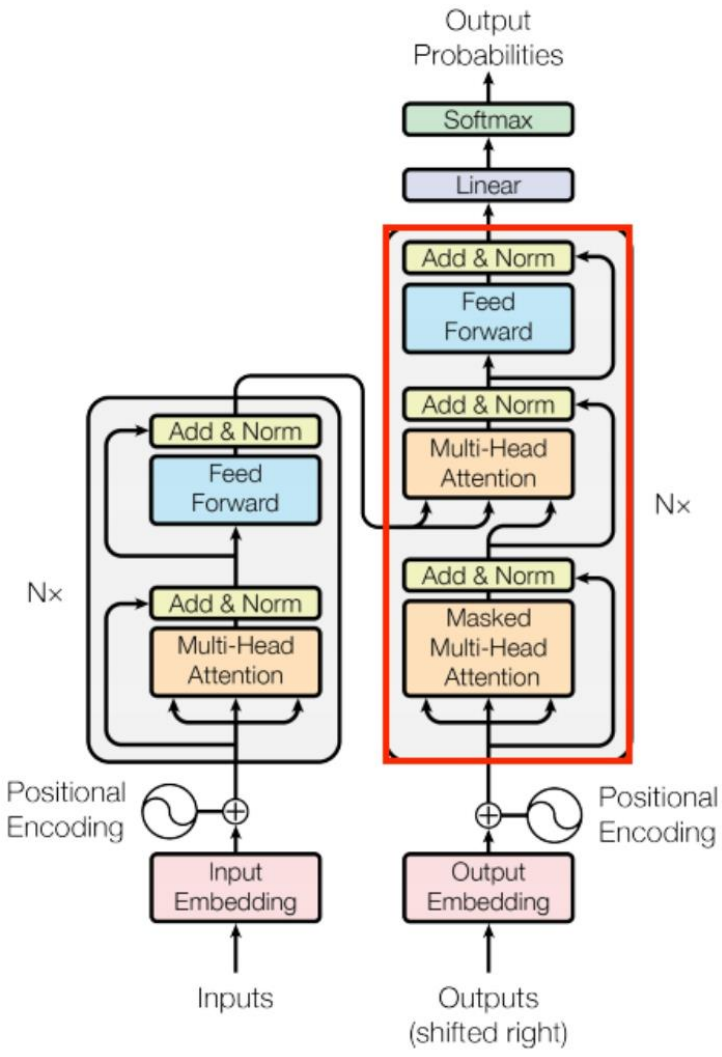
```
(64, 86)
```

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

## The decoder layer

The decoder's stack is slightly more complex, with each `DecoderLayer` containing a `CausalSelfAttention`, a `CrossAttention`, and a `FeedForward` layer:

**The decoder layer**



```
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:

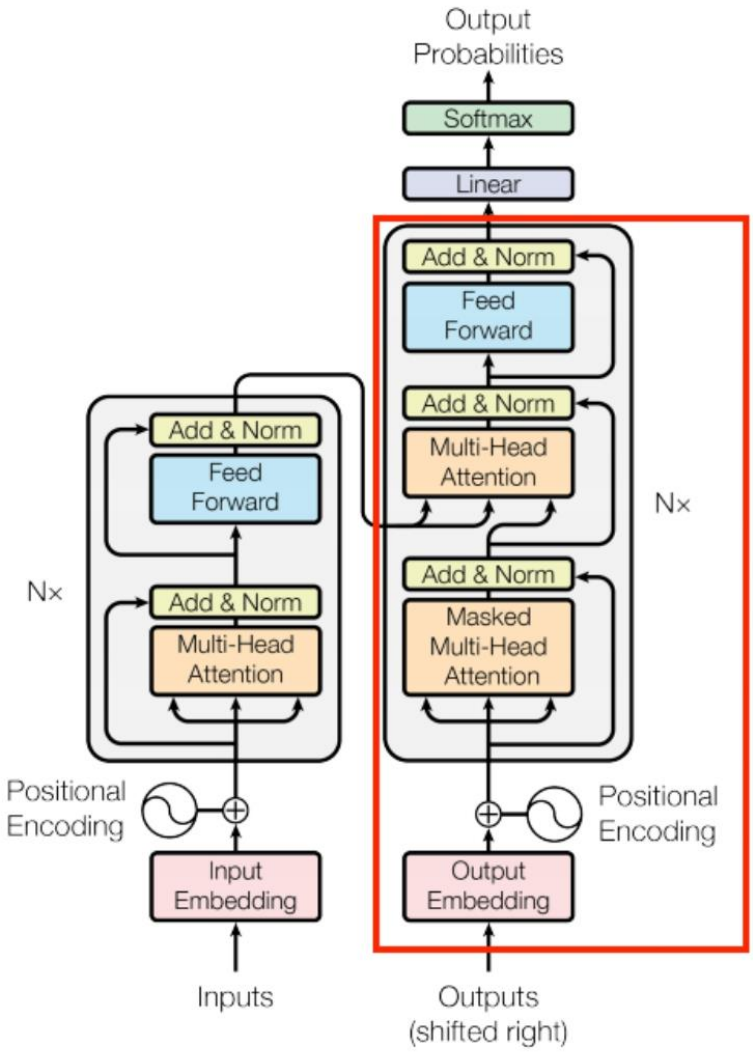
```
sample_decoder_layer = DecoderLayer(d_model=512, num_heads=8, dff=2048)

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)`
```

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

## The decoder

Similar to the `Encoder`, the `Decoder` consists of a `PositionalEmbedding`, and a stack of `DecoderLayer`s:

**The embedding and positional encoding layer**




Define the decoder by extending `tf.keras.layers.Layer`
(https://www.tensorflow.org/api_docs/python/tf/keras/layers/Layer):

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

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

```
    self.pos_embedding = PositionalEmbedding(vocab_size=vocab_size,
                                             d_model=d_model)
    self.dropout = tf.keras.layers.Dropout(dropout_rate)
    self.dec_layers = [
        DecoderLayer(d_model=d_model, num_heads=num_heads,
                     dff=dff, dropout_rate=dropout_rate)
        for _ in range(num_layers)]

    self.last_attn_scores = None

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

    x = self.dropout(x)

    for i in range(self.num_layers):
      x  = self.dec_layers[i](x, context)

    self.last_attn_scores = self.dec_layers[-1].last_attn_scores

    # The shape of x is (batch_size, target_seq_len, d_model).
    return x
```

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
```

```
TensorShape([64, 8, 81, 86])
```

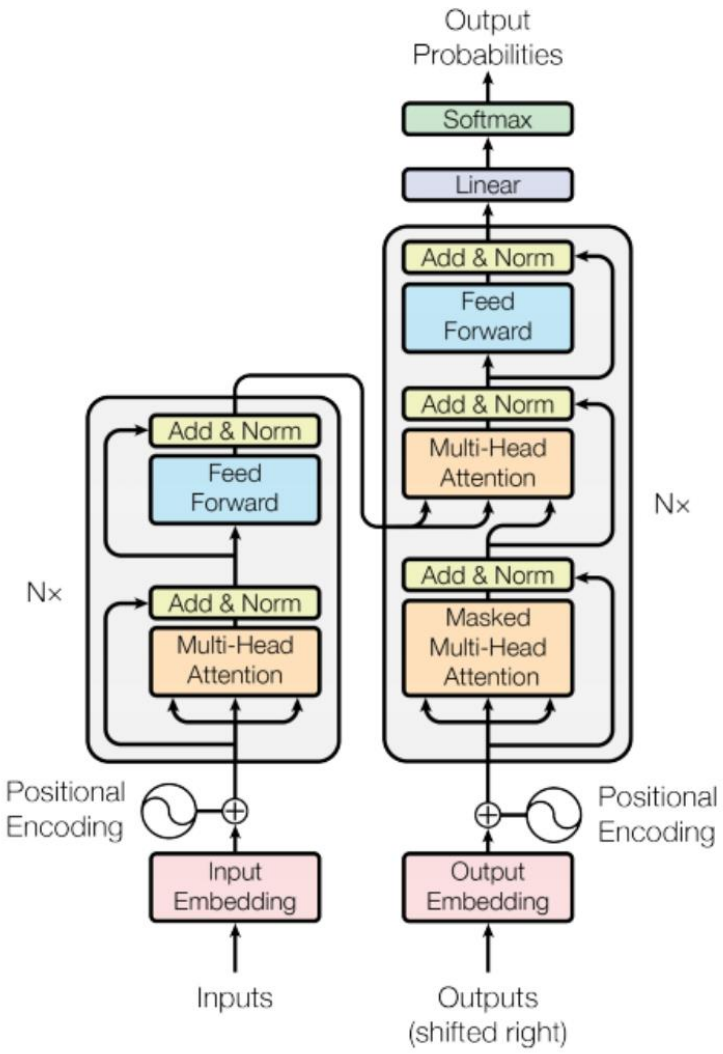
Having created the Transformer encoder and decoder, it's time to build the Transformer model and train it.

## The Transformer

You now have `Encoder` and `Decoder`. To complete the `Transformer` model, you need to put them together and add a final linear (`Dense`) layer which converts the resulting vector at each location into output token probabilities.

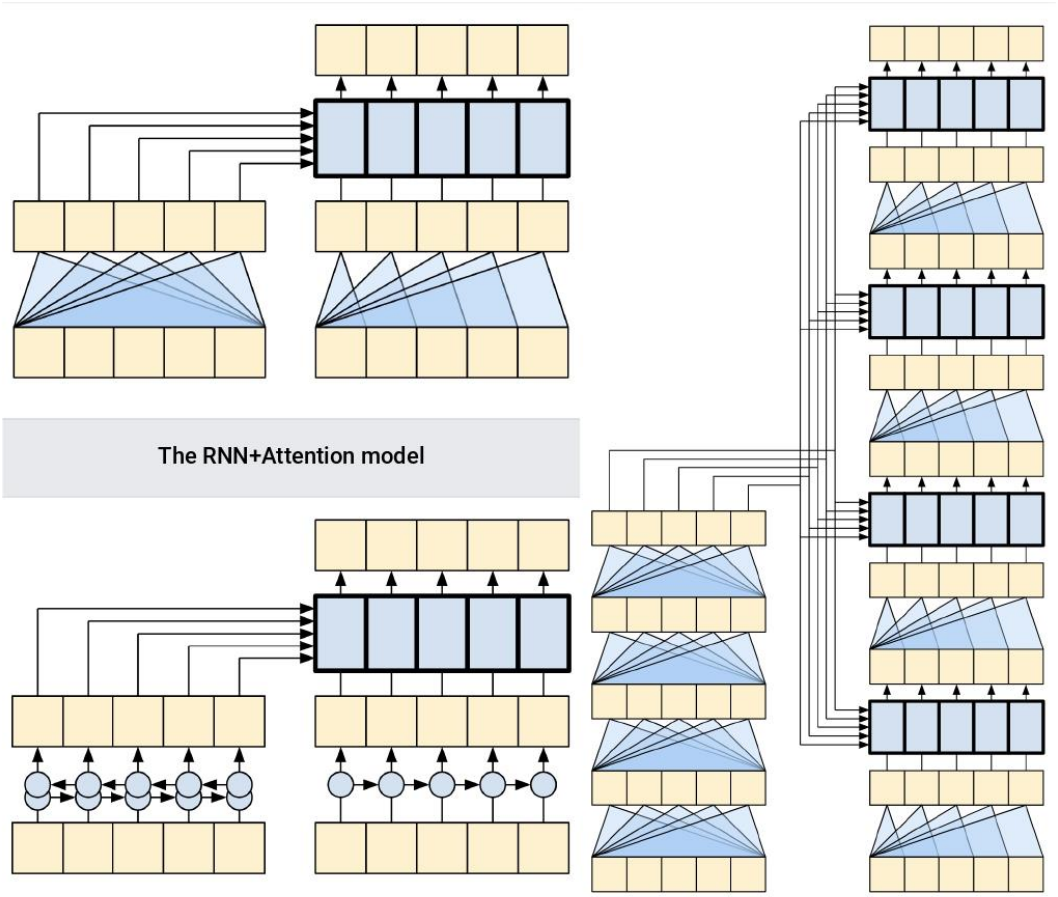
The output of the decoder is the input to this final linear layer.

**The transformer**




A `Transformer` with one layer in both the `Encoder` and `Decoder` looks almost exactly like the model from the RNN+attention tutorial (https://www.tensorflow.org/text/tutorials/nmt_with_attention). A multi-layer Transformer has more layers, but is fundamentally doing the same thing.

| A 1-layer transformer | A 4-layer transformer |
| --- | --- |



The RNN+Attention model

Create the `Transformer` by extending `tf.keras.Model` (https://www.tensorflow.org/api_docs/python/tf/keras/Model):

**Note:** The original paper (https://arxiv.org/pdf/1706.03762.pdf), section 3.4, shares the weight matrix between the embedding layer and the final linear layer. To keep things simple, this tutorial uses two separate weight matrices.

```
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
```

## Hyperparameters

To keep this example small and relatively fast, the number of layers (`num_layers`), the dimensionality of the embeddings (`d_model`), and the internal dimensionality of the `FeedForward` layer (`dff`) have been reduced.

The base model described in the original Transformer paper used `num_layers=6`, `d_model=512`, and `dff=2048`.

The number of self-attention heads remains the same (`num_heads=8`).

```
num_layers = 4
d_model = 128
dff = 512
```

```
num_heads = 8
dropout_rate = 0.1
```

## Try it out

Instantiate the `Transformer` model:

```
transformer = Transformer(
    num_layers=num_layers,
    d_model=d_model,
    num_heads=num_heads,
    dff=dff,
    input_vocab_size=tokenizers.pt.get_vocab_size().numpy(),
    target_vocab_size=tokenizers.en.get_vocab_size().numpy(),
    dropout_rate=dropout_rate)
```

Test it:

```
output = transformer((pt, en))

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

```
(64, 81)
(64, 86)
(64, 81, 7010)
```

```
attn_scores = transformer.decoder.dec_layers[-1].last_attn_scores
print(attn_scores.shape)  # (batch, heads, target_seq, input_seq)
```

```
(64, 8, 81, 86)
```

Print the summary of the model:

```
transformer.summary()
```

```
Model: "transformer"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 encoder_1 (Encoder)         multiple                  3632768

 decoder_1 (Decoder)         multiple                  5647104

 dense_38 (Dense)            multiple                  904290

=================================================================
Total params: 10,184,162
Trainable params: 10,184,162
Non-trainable params: 0
```

## Training

It's time to prepare the model and start training it.

### Set up the optimizer

Use the Adam optimizer with a custom learning rate scheduler according to the formula in the original Transformer paper (https://arxiv.org/abs/1706.03762).

$$lrate = d_{model}^{-0.5} * \min(step\_num^{-0.5}, step\_num \cdot warm$$

```
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)
```

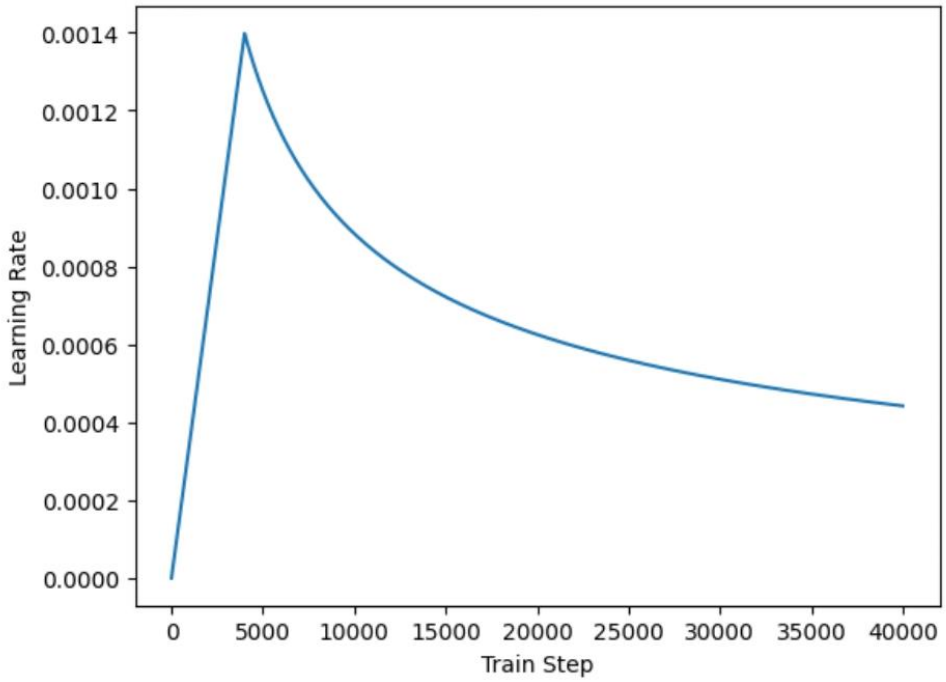
Instantiate the optimizer (in this example it's `tf.keras.optimizers.Adam` (https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam)):

```
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')
```

```
Text(0.5, 0, 'Train Step')
```




## Set up the loss and metrics

Since the target sequences are padded, it is important to apply a padding mask when calculating the loss. Use the cross-entropy loss function (`tf.keras.losses.SparseCategoricalCrossentropy` (https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy)):

```
def masked_loss(label, pred):
  mask = label != 0
  loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')
  loss = loss_object(label, pred)

  mask = tf.cast(mask, dtype=loss.dtype)
  loss *= mask

  loss = tf.reduce_sum(loss)/tf.reduce_sum(mask)
  return loss


def masked_accuracy(label, pred):
  pred = tf.argmax(pred, axis=2)
  label = tf.cast(label, pred.dtype)
```

```
  match = label == pred

  mask = label != 0

  match = match & mask

  match = tf.cast(match, dtype=tf.float32)
  mask = tf.cast(mask, dtype=tf.float32)
  return tf.reduce_sum(match)/tf.reduce_sum(mask)
```

## Train the model

With all the components ready, configure the training procedure using `model.compile`, and then run it with `model.fit`:

**Note:** This takes about an hour to train in Colab.

```
transformer.compile(
    loss=masked_loss,
    optimizer=optimizer,
    metrics=[masked_accuracy])
```

```
transformer.fit(train_batches,
                epochs=20,
                validation_data=val_batches)
```

```
Epoch 1/20
810/810 [==============================] - 221s 235ms/step - loss: 6.6041 - r
Epoch 2/20
810/810 [==============================] - 143s 176ms/step - loss: 4.5721 - r
Epoch 3/20
810/810 [==============================] - 142s 174ms/step - loss: 3.8224 - r
Epoch 4/20
810/810 [==============================] - 138s 170ms/step - loss: 3.2867 - r
Epoch 5/20
810/810 [==============================] - 138s 169ms/step - loss: 2.8953 - r
Epoch 6/20
810/810 [==============================] - 137s 169ms/step - loss: 2.5819 - r
Epoch 7/20
```

```
810/810 [==============================] - 139s 171ms/step - loss: 2.3098 - r
```

## Run inference

You can now test the model by performing a translation. The following steps are used for inference:

- Encode the input sentence using the Portuguese tokenizer (`tokenizers.pt`). This is the encoder input.
- The decoder input is initialized to the `[START]` token.
- Calculate the padding masks and the look ahead masks.
- The `decoder` then outputs the predictions by looking at the `encoder output` and its own output (self-attention).
- Concatenate the predicted token to the decoder input and pass it to the decoder.
- In this approach, the decoder predicts the next token based on the previous tokens it predicted.

**Note:** The model is optimized for *efficient training* and makes a next-token prediction for each token in the output simultaneously. This is redundant during inference, and only the last prediction is used. This model can be made more efficient for inference if you only calculate the last prediction when running in inference mode (`training=False`).

Define the `Translator` class by subclassing `tf.Module` (https://www.tensorflow.org/api_docs/python/tf/Module):

```
class Translator(tf.Module):
  def __init__(self, tokenizers, transformer):
    self.tokenizers = tokenizers
    self.transformer = transformer

  def __call__(self, sentence, max_length=MAX_TOKENS):
    # The input sentence is Portuguese, hence adding the `[START]` and `[END]
    assert isinstance(sentence, tf.Tensor)
    if len(sentence.shape) == 0:
      sentence = sentence[tf.newaxis]

    sentence = self.tokenizers.pt.tokenize(sentence).to_tensor()
```

```
    encoder_input = sentence

    # As the output language is English, initialize the output with the
    # English `[START]` token.
    start_end = self.tokenizers.en.tokenize([''])[0]
    start = start_end[0][tf.newaxis]
    end = start_end[1][tf.newaxis]

    # `tf.TensorArray` is required here (instead of a Python list), so that th
    # dynamic-loop can be traced by `tf.function`.
    output_array = tf.TensorArray(dtype=tf.int64, size=0, dynamic_size=True)
    output_array = output_array.write(0, start)

    for i in tf.range(max_length):
      output = tf.transpose(output_array.stack())
      predictions = self.transformer([encoder_input, output], training=False)

      # Select the last token from the `seq_len` dimension.
      predictions = predictions[:, -1:, :]  # Shape `(batch_size, 1, vocab_siz

      predicted_id = tf.argmax(predictions, axis=-1)

      # Concatenate the `predicted_id` to the output which is given to the
      # decoder as its input.
      output_array = output_array.write(i+1, predicted_id[0])

      if predicted_id == end:
        break

    output = tf.transpose(output_array.stack())
    # The output shape is `(1, tokens)`.
    text = tokenizers.en.detokenize(output)[0]  # Shape: `()`.

    tokens = tokenizers.en.lookup(output)[0]

    # `tf.function` prevents us from using the attention_weights that were
    # calculated on the last iteration of the loop.
    # So, recalculate them outside the loop.
    self.transformer([encoder_input, output[:,:-1]], training=False)
    attention_weights = self.transformer.decoder.last_attn_scores

    return text, tokens, attention_weights
```

**Note:** This function uses an unrolled loop, not a dynamic loop. It generates **`MAX_TOKENS`** on every call. Refer to the NMT with attention (/text/tutorials/nmt_with_attention) tutorial for an example

implementation with a dynamic loop, which can be much more efficient.

Create an instance of this `Translator` class, and try it out a few times:

```
translator = Translator(tokenizers, transformer)
```

```
def print_translation(sentence, tokens, ground_truth):
  print(f'{"Input:":15s}: {sentence}')
  print(f'{"Prediction":15s}: {tokens.numpy().decode("utf-8")}')
  print(f'{"Ground truth":15s}: {ground_truth}')
```

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 .
```

```
Ground truth   : and my neighboring homes heard about this idea .
```

Example 3:

```
sentence = 'vou então muito rapidamente partilhar convosco algumas histórias
ground_truth = "so i'll just share with you some stories very quickly of some

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

```
Input:         : vou então muito rapidamente partilhar convosco algumas histó
Prediction     : so i ' m going to share with you some magic stories that wil
Ground truth   : so i'll just share with you some stories very quickly of som
```

## Create attention plots

The `Translator` class you created in the previous section returns a dictionary of attention heatmaps you can use to visualize the internal working of the model.

For example:

```
sentence = 'este é o primeiro livro que eu fiz.'
ground_truth = "this is the first book i've ever done."

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

```
Input:         : este é o primeiro livro que eu fiz.
Prediction     : this is the first book i did .
Ground truth   : this is the first book i've ever done.
```

Create a function that plots the attention when a token is generated:

```
def plot_attention_head(in_tokens, translated_tokens, attention):
  # The model didn't generate `<START>` in the output. Skip it.
  translated_tokens = translated_tokens[1:]

  ax = plt.gca()
  ax.matshow(attention)
  ax.set_xticks(range(len(in_tokens)))
  ax.set_yticks(range(len(translated_tokens)))

  labels = [label.decode('utf-8') for label in in_tokens.numpy()]
  ax.set_xticklabels(
      labels, rotation=90)

  labels = [label.decode('utf-8') for label in translated_tokens.numpy()]
  ax.set_yticklabels(labels)
```

```
head = 0
# Shape: `(batch=1, num_heads, seq_len_q, seq_len_k)`.
attention_heads = tf.squeeze(attention_weights, 0)
attention = attention_heads[head]
attention.shape
```

```
TensorShape([9, 11])
```

These are the input (Portuguese) tokens:

```
in_tokens = tf.convert_to_tensor([sentence])
in_tokens = tokenizers.pt.tokenize(in_tokens).to_tensor()
in_tokens = tokenizers.pt.lookup(in_tokens)[0]
in_tokens
```

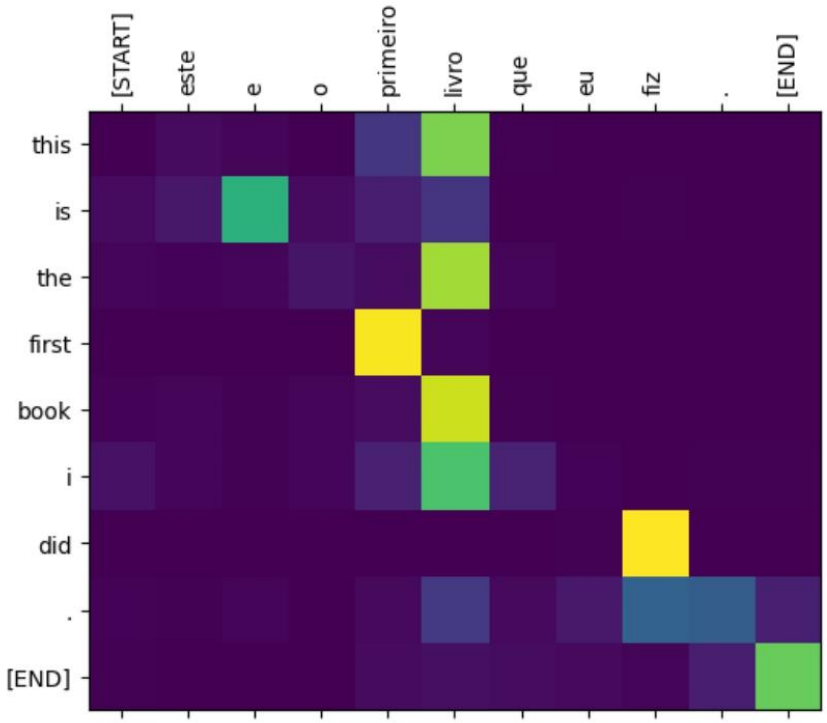
```
<tf.Tensor: shape=(11,), dtype=string, numpy=
array([b'[START]', b'este', b'e', b'o', b'primeiro', b'livro', b'que',
       b'eu', b'fiz', b'.', b'[END]'], dtype=object)>
```

And these are the output (English translation) tokens:

```
translated_tokens
```

```
<tf.Tensor: shape=(10,), dtype=string, numpy=
array([b'[START]', b'this', b'is', b'the', b'first', b'book', b'i',
       b'did', b'.', b'[END]'], dtype=object)>
```

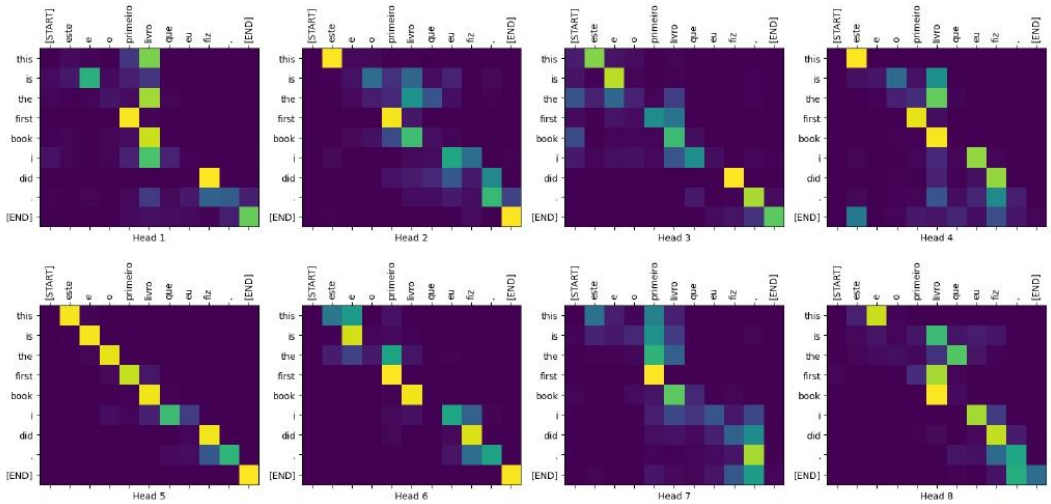
```
plot_attention_head(in_tokens, translated_tokens, attention)
```




```
def plot_attention_weights(sentence, translated_tokens, attention_heads):
  in_tokens = tf.convert_to_tensor([sentence])
  in_tokens = tokenizers.pt.tokenize(in_tokens).to_tensor()
  in_tokens = tokenizers.pt.lookup(in_tokens)[0]

  fig = plt.figure(figsize=(16, 8))

  for h, head in enumerate(attention_heads):
    ax = fig.add_subplot(2, 4, h+1)
```

```
    plot_attention_head(in_tokens, translated_tokens, head)

    ax.set_xlabel(f'Head {h+1}')

  plt.tight_layout()
  plt.show()
```

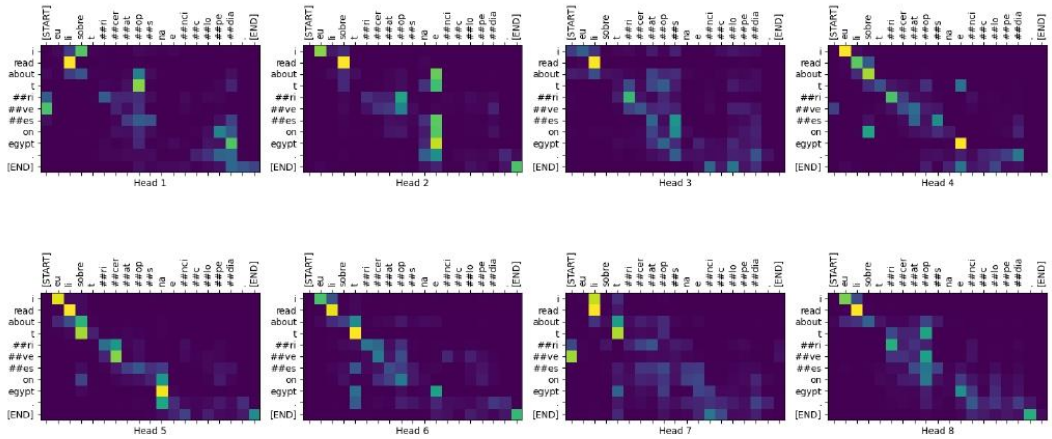
```
plot_attention_weights(sentence,
                       translated_tokens,
                       attention_weights[0])
```




The model can handle unfamiliar words. Neither `'triceratops'` nor `'encyclopédia'` are in the input dataset, and the model attempts to transliterate them even without a shared vocabulary. For example:

```
sentence = 'Eu li sobre triceratops na enciclopédia.'
ground_truth = 'I read about triceratops in the encyclopedia.'

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

plot_attention_weights(sentence, translated_tokens, attention_weights[0])
```

```
Input:         : Eu li sobre triceratops na enciclopédia.
Prediction     : i read about trivees on egypt .
Ground truth   : I read about triceratops in the encyclopedia.
```




## Export the model

You have tested the model and the inference is working. Next, you can export it as a **`tf.saved_model`** (https://www.tensorflow.org/api_docs/python/tf/saved_model). To learn about saving and loading a model in the SavedModel format, use this guide (https://www.tensorflow.org/guide/saved_model).

Create a class called `ExportTranslator` by subclassing the **`tf.Module`** (https://www.tensorflow.org/api_docs/python/tf/Module) subclass with a **`tf.function`** (https://www.tensorflow.org/api_docs/python/tf/function) on the `__call__` method:

```
class ExportTranslator(tf.Module):
  def __init__(self, translator):
    self.translator = translator

  @tf.function(input_signature=[tf.TensorSpec(shape=[], dtype=tf.string)])
  def __call__(self, sentence):
    (result,
     tokens,
     attention_weights) = self.translator(sentence, max_length=MAX_TOKENS)

    return result
```

In the above `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function) only the output sentence is returned. Thanks to the non-strict execution (https://tensorflow.org/guide/intro_to_graphs) in `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function) any unnecessary values are never computed.

Wrap `translator` in the newly created `ExportTranslator`:

```
translator = ExportTranslator(translator)
```

Since the model is decoding the predictions using `tf.argmax` (https://www.tensorflow.org/api_docs/python/tf/math/argmax) the predictions are deterministic. The original model and one reloaded from its `SavedModel` should give identical predictions:

```
translator('este é o primeiro livro que eu fiz.').numpy()
```

```
b'this is the first book i did .'
```

```
tf.saved_model.save(translator, export_dir='translator')
```

```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable ob
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable ob
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable ob
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable ob
WARNING:absl:Found untraced functions such as positional_embedding_4_layer_ca
INFO:tensorflow:Assets written to: translator/assets
INFO:tensorflow:Assets written to: translator/assets
```

```
reloaded = tf.saved_model.load('translator')
```

```
reloaded('este é o primeiro livro que eu fiz.').numpy()
```

```
b'this is the first book i did .'
```

## Conclusion

In this tutorial you learned about:

- The Transformers and their significance in machine learning
- Attention, self-attention and multi-head attention
- Positional encoding with embeddings
- The encoder-decoder architecture of the original Transformer
- Masking in self-attention
- How to put it all together to translate text

The downsides of this architecture are:

- For a time-series, the output for a time-step is calculated from the *entire history* instead of only the inputs and current hidden-state. This *may* be less efficient.
- If the input has a temporal/spatial relationship, like text or images, some positional encoding must be added or the model will effectively see a bag of words.

If you want to practice, there are many things you could try with it. For example:

- Use a different dataset to train the Transformer.
- Create the "Base Transformer" or "Transformer XL" configurations from the original paper by changing the hyperparameters.
- Use the layers defined here to create an implementation of BERT (https://arxiv.org/abs/1810.04805)
- Use Beam search to get better predictions.

There are a wide variety of Transformer-based models, many of which improve upon the 2017 version of the original Transformer with encoder-decoder, encoder-only and decoder-only architectures.

Some of these models are covered in the following research publications:

## Conclusion

Implementation was successful.

## Experiment 10

**Aim**

MLOps

**Software Requirements**

Google Colab

**Code and Output:**

# BANGALORE HOUSE PRICE PREDICTION MODEL

Data Link: https://www.kaggle.com/ameythakur20/bangalore-house-prices

## IMPORT LIBRARIES

```
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')
```

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

```
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)
```

## FEATURE ENGINEERING

```
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 |

**ABOVE DATA SHOWS THAT TOTAL SQFT CAN BE A RANGE (E.G. 2100-2850). FOR SUCH CASES WE CAN JUST TAKE AVERAGE OF MIN & MAX VALUE IN THE RANGE. THERE ARE**

**OTHER CASES WHERE VALUES ARE IN SQM WHICH CAN BE CONVERTED TO SQFT USING UNIT CONVERSION.**

```
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()
```

| | location | size | total_sqft | bath | price | bhk | price_per_sqft |
|---|---|---|---|---|---|---|---|
| 0 | Electronic City Phase II | 2 BHK | 1056.0 | 2.0 | 39.07 | 2 | 3699.810606 |
| 1 | Chikka Tirupathi | 4 Bedroom | 2600.0 | 5.0 | 120.00 | 4 | 4615.384615 |
| 2 | Uttarahalli | 3 BHK | 1440.0 | 2.0 | 62.00 | 3 | 4305.555556 |

```
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)
```

| | location | size | total_sqft | bath | price | bhk | price_per_sqft |
|---|---|---|---|---|---|---|---|
| 0 | Electronic City Phase II | 2 BHK | 1056.0 | 2.0 | 39.07 | 2 | 3699.810606 |
| 1 | Chikka Tirupathi | 4 Bedroom | 2600.0 | 5.0 | 120.00 | 4 | 4615.384615 |
| 2 | Uttarahalli | 3 BHK | 1440.0 | 2.0 | 62.00 | 3 | 4305.555556 |
| 3 | Lingadheeranahalli | 3 BHK | 1521.0 | 3.0 | 95.00 | 3 | 6245.890861 |
| 4 | Kothanur | 2 BHK | 1200.0 | 2.0 | 51.00 | 2 | 4250.000000 |

## OUTLIER REMOVAL USING BUSINESS LOGIC

**AS A DATA SCIENTIST S A DATA SCIENTIST WHEN YOU HAVE A CONVERSATION WITH YOUR BUSINESS MANAGER (WHO HAS EXPERTISE IN REAL ESTATE), HE WILL TELL YOU THAT NORMALLY SQUARE FT PER BEDROOM IS 300 (I.E. 2 BHK APARTMENT IS MINIMUM 600 SQFT. IF YOU HAVE FOR EXAMPLE 400 SQFT APARTMENT WITH 2 BHK THAN THAT SEEMS SUSPICIOUS AND CAN BE REMOVED AS AN OUTLIER. WE WILL REMOVE SUCH OUTLIERS BY KEEPING OUR MINIMUM THRESOLD PER BHK TO BE 300 SQFT**

```
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)
```

## OUTLIER REMOVAL USING STANDARD DEVIATION AND MEAN

```
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
```

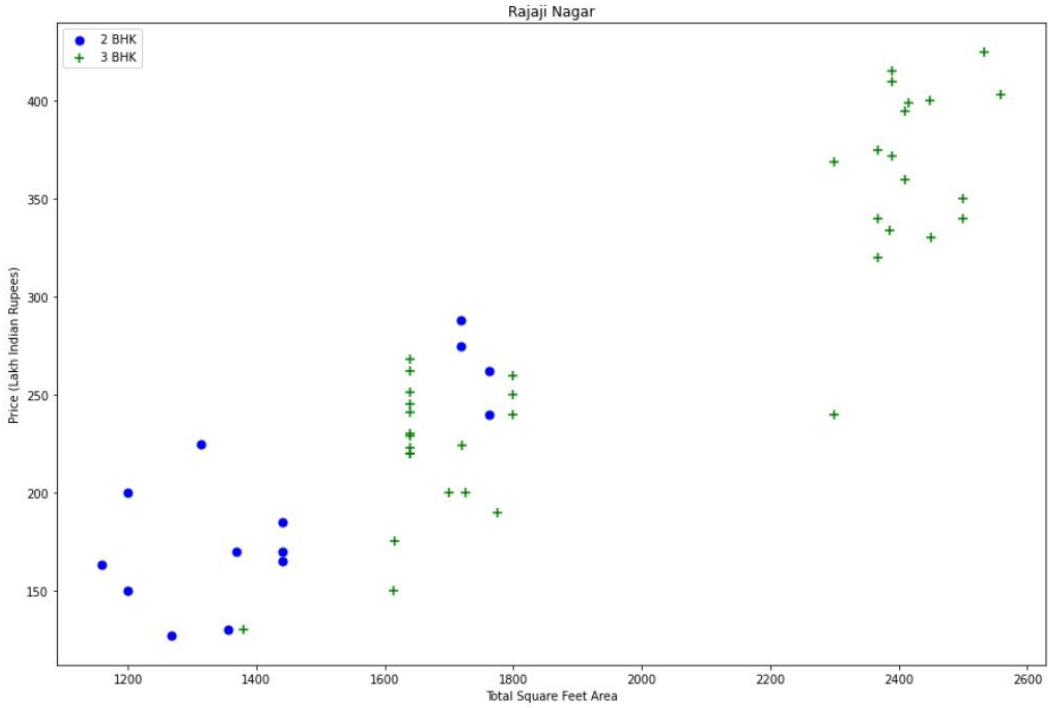
**HERE WE FIND THAT MIN PRICE PER SQFT IS 267 RS/SQFT WHEREAS MAX IS 12000000, THIS SHOWS A WIDE VARIATION IN PROPERTY PRICES. WE SHOULD REMOVE OUTLIERS PER LOCATION USING MEAN AND ONE STANDARD DEVIATION**

```
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")
```


**WE SHOULD ALSO REMOVE PROPERTIES WHERE FOR THE SAME LOCATION, THE PRICE OF (FOR EXAMPLE) A 3 BEDROOM APARTMENT IS LESS THAN A 2 BEDROOM APARTMENT (WITH THE SAME SQUARE FT AREA). WHAT WE WILL DO IS FOR A GIVEN LOCATION, WE WILL BUILD A DICTIONARY OF STATS PER BHK, I.E.**

```
{
    '1' : {
        'mean': 4000,
        'std: 2000,
        'count': 34
    },
    '2' : {
        'mean': 4300,
        'std: 2300,
        'count': 22
    },
}
```

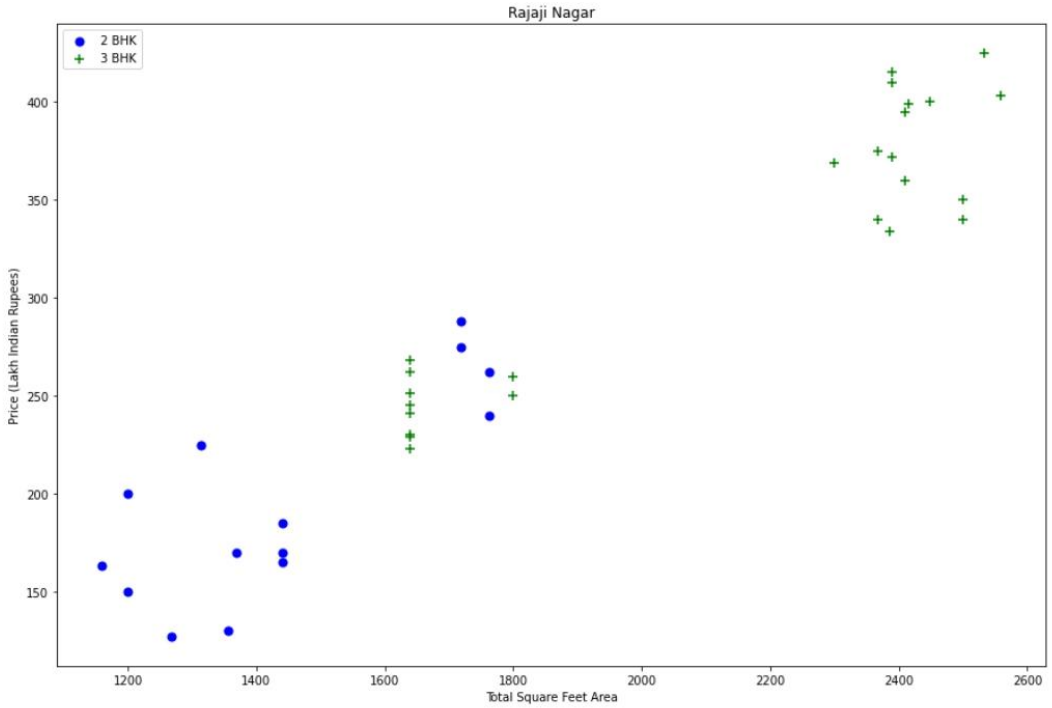
**NOW WE CAN REMOVE THOSE 2 BHK APARTMENTS WHOSE PRICE_PER_SQFT IS LESS THAN THE MEAN PRICE_PER_SQFT OF 1 BHK APARTMENT**

```
def remove_bhk_outliers(df):
    exclude_indices = np.array([])
    for location, location_df in df.groupby('location'):
        bhk_stats = {}
        for bhk, bhk_df in location_df.groupby('bhk'):
            bhk_stats[bhk] = {
                'mean': np.mean(bhk_df.price_per_sqft),
                'std': np.std(bhk_df.price_per_sqft),
                'count': bhk_df.shape[0]
            }
        for bhk, bhk_df in location_df.groupby('bhk'):
            stats = bhk_stats.get(bhk-1)
            if stats and stats['count']>5:
                exclude_indices = np.append(exclude_indices, bhk_df[bhk_df.price_per_
    return df.drop(exclude_indices,axis='index')
df8 = remove_bhk_outliers(df7)
# df8 = df7.copy()
df8.shape
```

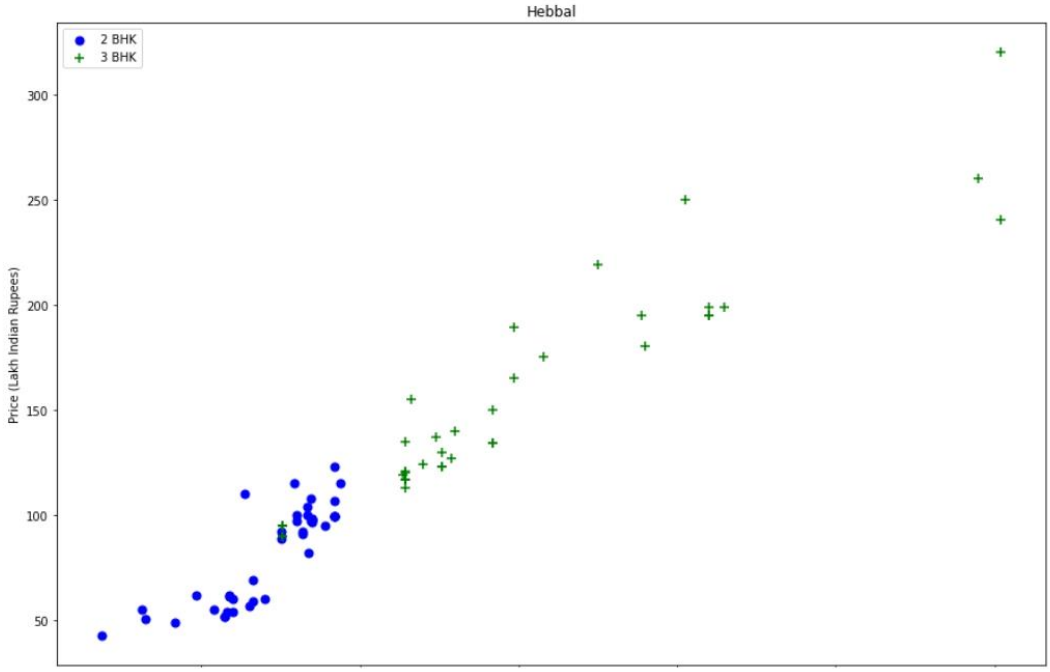
```
(7317, 7)
```

**PLOT SAME SCATTER CHART AGAIN TO VISUALIZE PRICE_PER_SQFT FOR 2 BHK AND 3 BHK PROPERTIES**

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




```
plot_scatter_chart(df8,"Hebbal")
```



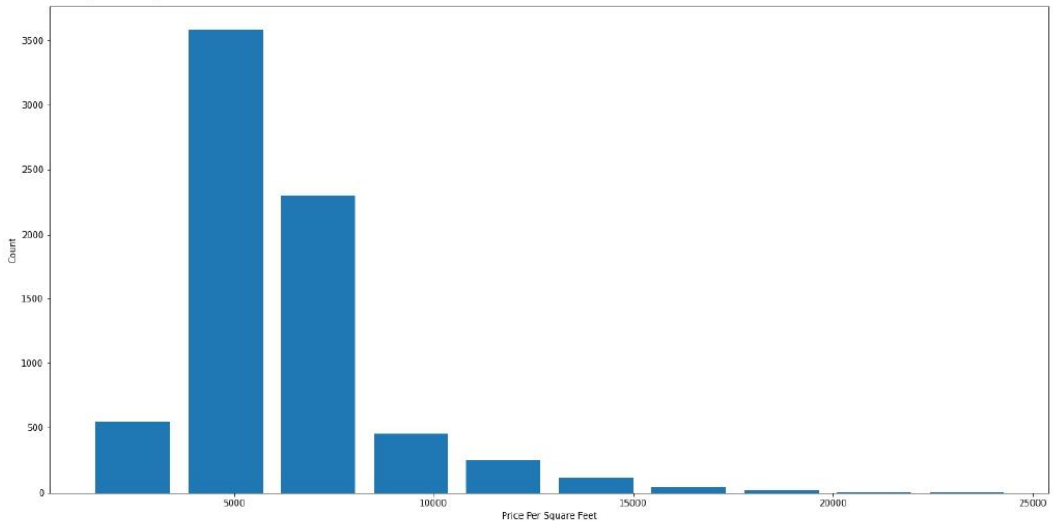

**BASED ON ABOVE CHARTS WE CAN SEE THAT DATA POINTS HIGHLIGHTED IN RED BELOW ARE OUTLIERS AND THEY ARE BEING REMOVED DUE TO REMOVE_BHK_OUTLIERS FUNCTION**

## BEFORE AND AFTER OUTLIER REMOVAL: RAJAJI NAGAR

## BEFORE AND AFTER OUTLIER REMOVAL: 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')
```



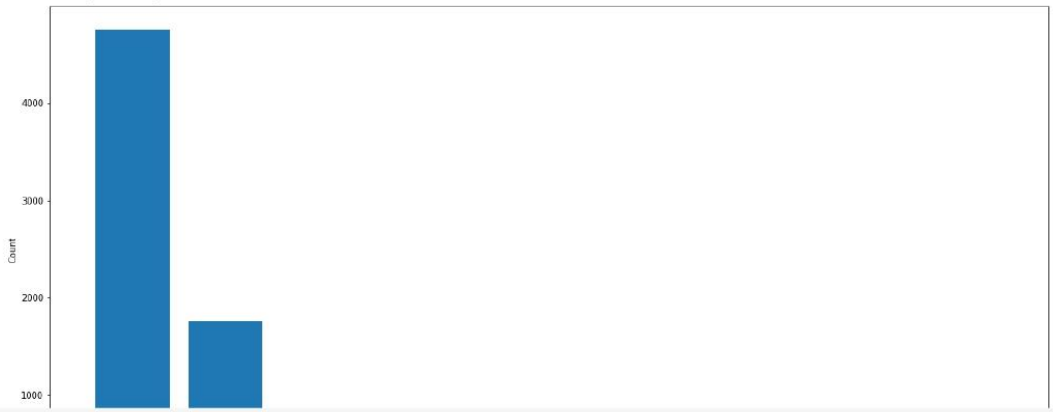

## OUTLIER REMOVAL USING BATHROOMS FEATURE

```
df8.bath.unique()
```

```
array([ 4.,  3.,  2.,  5.,  8.,  1.,  6.,  7.,  9., 12., 16., 13.])
```

```
plt.hist(df8.bath,rwidth=0.8)
plt.xlabel("Number of bathrooms")
plt.ylabel("Count")
```

```
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 |

**AGAIN THE BUSINESS MANAGER HAS A CONVERSATION WITH YOU (I.E. A DATA SCIENTIST) THAT IF YOU HAVE A 4 BEDROOM HOME AND EVEN IF YOU HAVE A BATHROOM IN ALL 4 ROOMS PLUS ONE GUEST BATHROOM, YOU WILL HAVE A TOTAL BATH = TOTAL BED + 1 MAX. ANYTHING ABOVE THAT IS AN OUTLIER OR A DATA ERROR AND CAN BE REMOVED**

```
df9 = df8[df8.bath<df8.bhk+2]
df9.shape
```

```
(7239, 7)
```

```
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 | 9 Pha Nag |
|---|---|---|---|---|---|---|---|---|---|---|
| 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()
```

| | location | total_sqft | bath | price | bhk | 1st Block Jayanagar | 1st Phase JP Nagar | 2nd Phase Judicial Layout | 2nd Stage Nagarbhavi |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1st Block Jayanagar | 2850.0 | 4.0 | 428.0 | 4 | 1 | 0 | 0 | 0 |
| **1** | 1st Block Jayanagar | 1630.0 | 3.0 | 194.0 | 3 | 1 | 0 | 0 | 0 |

```
df12 = df11.drop('location',axis='columns')
df12.head(2)
```

| | total_sqft | bath | price | bhk | 1st Block Jayanagar | 1st Phase JP Nagar | 2nd Phase Judicial Layout | 2nd Stage Nagarbhavi | 5th Block Hbr Layout |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 2850.0 | 4.0 | 428.0 | 4 | 1 | 0 | 0 | 0 | 0 |
| **1** | 1630.0 | 3.0 | 194.0 | 3 | 1 | 0 | 0 | 0 | 0 |

2 rows × 244 columns

## BUILDING A MODEL

```
df12.shape
```

```
(7239, 244)
```

```
X = df12.drop(['price'],axis='columns')
X.head(3)
```

| | total_sqft | bath | bhk | 1st Block Jayanagar | 1st Phase JP Nagar | 2nd Phase Judicial Layout | 2nd Stage Nagarbhavi | 5th Block Hbr Layout | 5th Phase JP Nagar |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 2850.0 | 4.0 | 4 | 1 | 0 | 0 | 0 | 0 | 0 |
| **1** | 1630.0 | 3.0 | 3 | 1 | 0 | 0 | 0 | 0 | 0 |
| **2** | 1875.0 | 2.0 | 3 | 1 | 0 | 0 | 0 | 0 | 0 |

3 rows × 243 columns

```
X.shape
```

```
(7239, 243)
```

```
y = df12.price
y.head(3)
```

```
0    428.0
1    194.0
2    235.0
Name: price, dtype: float64
```

```
len(y)
```

```
7239
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=10
```

```
from sklearn.linear_model import LinearRegression
lr_clf = LinearRegression()
lr_clf.fit(X_train,y_train)
lr_clf.score(X_test,y_test)
```

```
0.86291322445229449
```

## USE K FOLD CROSS VALIDATION TO MEASURE ACCURACY OF OUR LINEAR REGRESSION MODEL

```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score

cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)

cross_val_score(LinearRegression(), X, y, cv=cv)
```

```
array([0.82702546, 0.86027005, 0.85322178, 0.8436466 , 0.85481502])
```

**WE CAN SEE THAT IN 5 ITERATIONS WE GET A SCORE ABOVE 80% ALL THE TIME. THIS IS PRETTY GOOD BUT WE WANT TO TEST FEW OTHER ALGORITHMS FOR REGRESSION TO SEE IF WE CAN GET AN EVEN BETTER SCORE. WE WILL USE GRIDSEARCHCV FOR THIS PURPOSE**

## FIND BEST MODEL USING GRIDSEARCHCV

```
from sklearn.model_selection import GridSearchCV

from sklearn.linear_model import Lasso
from sklearn.tree import DecisionTreeRegressor

def find_best_model_using_gridsearchcv(X,y):
```

```
    algos = {
        'linear_regression' : {
            'model': LinearRegression(),
            'params': {
                'normalize': [True, False]
            }
        },
        'lasso': {
            'model': Lasso(),
            'params': {
                'alpha': [1,2],
                'selection': ['random', 'cyclic']
            }
        },
        'decision_tree': {
            'model': DecisionTreeRegressor(),
            'params': {
                'criterion' : ['mse','friedman_mse'],
                'splitter': ['best','random']
            }
        }
    }
    scores = []
    cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
    for algo_name, config in algos.items():
        gs =  GridSearchCV(config['model'], config['params'], cv=cv, return_train_sc
        gs.fit(X,y)
        scores.append({
            'model': algo_name,
            'best_score': gs.best_score_,
            'best_params': gs.best_params_
        })

    return pd.DataFrame(scores,columns=['model','best_score','best_params'])

find_best_model_using_gridsearchcv(X,y)
```

| | model | best_score | best_params |
|---|---|---|---|
| 0 | linear_regression | 0.847796 | {'normalize': False} |
| 1 | lasso | 0.726748 | {'alpha': 2, 'selection': 'random'} |
| 2 | decision_tree | 0.713610 | {'criterion': 'friedman_mse', 'splitter': 'best'} |

**BASED ON THE ABOVE RESULTS WE CAN SAY THAT LINEAR REGRESSION GIVES THE BEST SCORE. HENCE WE WILL USE THAT.**

## TEST THE MODEL FOR FEW PROPERTIES

```
def predict_price(location,sqft,bath,bhk):
    loc_index = np.where(X.columns==location)[0][0]

    x = np.zeros(len(X.columns))
    x[0] = sqft
    x[1] = bath
    x[2] = bhk
    if loc_index >= 0:
        x[loc_index] = 1

    return lr_clf.predict([x])[0]
```

```
predict_price('1st Phase JP Nagar',1000, 2, 2)
```

```
83.8657025831121
```

```
predict_price('1st Phase JP Nagar',1000, 3, 3)
```

```
86.08062284985986
```

```
predict_price('Indira Nagar',1000, 2, 2)
```

```
193.31197733179556
```

```
predict_price('Indira Nagar',1000, 3, 3)
```

```
195.52689759854334
```

## EXPORT THE TESTED MODEL TO A PICKLE FILE

```
import pickle
with open('banglore_home_prices_model.pickle','wb') as f:
    pickle.dump(lr_clf,f)
```

## EXPORT LOCATION AND COLUMN INFORMATION TO A FILE THAT WILL BE USEFUL LATER ON IN OUR PREDICTION APPLICATION

```
import json
columns = {
    'data_columns' : [col.lower() for col in X.columns]
}
with open("columns.json","w") as f:
    f.write(json.dumps(columns))
```

**Conclusion**

Implementation was successful.

**Experiment-1**

| Criteria | Total Marks | Marks Obtained | Comments |
|---|---|---|---|
| Concept (A) | 2 | | |
| Implementation (B) | 2 | | |
| Performance (C) | 2 | | |
| **Total** | **6** | | |

**Experiment-2**

| Criteria | Total Marks | Marks Obtained | Comments |
|---|---|---|---|
| Concept (A) | 2 | | |
| Implementation (B) | 2 | | |
| Performance (C) | 2 | | |
| **Total** | **6** | | |

**Experiment-3**

| Criteria | Total Marks | Marks Obtained | Comments |
|---|---|---|---|
| Concept (A) | 2 | | |
| Implementation (B) | 2 | | |
| Performance (C) | 2 | | |
| **Total** | **6** | | |

**Experiment-4**

| Criteria | Total Marks | Marks Obtained | Comments |
|---|---|---|---|
| Concept (A) | 2 | | |

| | | | |
|---|---|---|---|
| Implementation (B) | 2 | | |
| Performance (C) | 2 | | |
| **Total** | **6** | | |

**Experiment-5**

| Criteria | Total Marks | Marks Obtained | Comments |
|---|---|---|---|
| Concept (A) | 2 | | |
| Implementation (B) | 2 | | |
| Performance (C) | 2 | | |
| **Total** | **6** | | |

**Experiment-6**

| Criteria | Total Marks | Marks Obtained | Comments |
|---|---|---|---|
| Concept (A) | 2 | | |
| Implementation (B) | 2 | | |
| Performance (C) | 2 | | |
| **Total** | **6** | | |

**Experiment-7**

| Criteria | Total Marks | Marks Obtained | Comments |
|---|---|---|---|
| Concept (A) | 2 | | |
| Implementation (B) | 2 | | |
| Performance (C) | 2 | | |
| **Total** | **6** | | |

**Experiment-8**

| Criteria | Total Marks | Marks Obtained | Comments |
|---|---|---|---|
| Concept (A) | 2 | | |
| Implementation (B) | 2 | | |
| Performance (C) | 2 | | |
| **Total** | **6** | | |

**Experiment-9**

| Criteria | Total Marks | Marks Obtained | Comments |
|---|---|---|---|
| Concept (A) | 2 | | |
| Implementation (B) | 2 | | |
| Performance (C) | 2 | | |
| **Total** | **6** | | |

**Experiment-10**

| Criteria | Total Marks | Marks Obtained | Comments |
|---|---|---|---|
| Concept (A) | 2 | | |
| Implementation (B) | 2 | | |
| Performance (C) | 2 | | |
| **Total** | **6** | | |