Chapter 1: Setting up an Anaconda Environment

```
import cv2
import tensorflow as tf
import dlib
import pytesseract as ptess
```

```
!pip install pytesseract
```

```
Collecting pytesseract
  Downloading https://files.pythonhosted.org/packages/47/e5/892d78db0d26372aa3
Requirement already satisfied: Pillow in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: olefile in /usr/local/lib/python3.6/dist-packag
Building wheels for collected packages: pytesseract
  Building wheel for pytesseract (setup.py) ... done
  Created wheel for pytesseract: filename=pytesseract-0.3.0-py2.py3-none-any.w
  Stored in directory: /root/.cache/pip/wheels/78/c9/ac/4cb76bd547f99700705224
Successfully built pytesseract
Installing collected packages: pytesseract
Successfully installed pytesseract-0.3.0
```

```
tf.Graph?
```

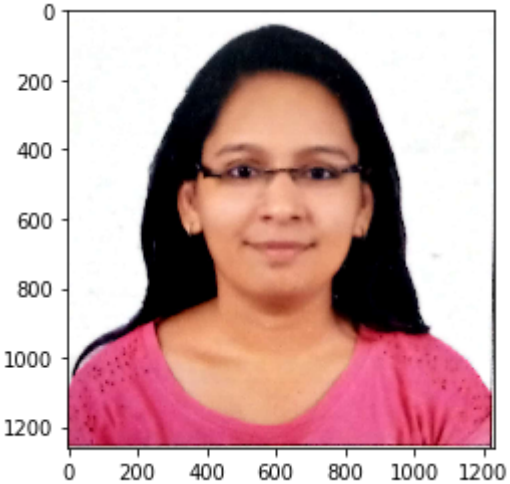
```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed

```
Saving juhipic.jpg to juhipic.jpg
```

```
testin = imread('juhipic.jpg')
figure()
imshow(testin)
```

```
<matplotlib.image.AxesImage at 0x7fba3e7ce4e0>
```




```
from google.colab import files
uploaded = files.upload()
```

```
uploaded = files.upload()
```

```
import io
import base64
from IPython.display import HTML

def playvideo(filename):
  video = io.open(filename,'r+b').read()
  encoded = base64.b64encode(video)
  return HTML(data='''<video alt="test" controls>
                <source src="data:video/mp4;base64,{0}" type="video/mp4"/>
                </video> '''.format(encoded.decode('ascii')))
playvideo('ex_video.mp4')
```

Chapter 2 : Image Captioning

```
from __future__ import absolute_import, division, print_function, unicode_literals
```

```
try:
  # %tensorflow_version only exists in Colab.
  %tensorflow_version 2.x
except Exception:
  pass
import tensorflow as tf

# You'll generate plots of attention in order to see which parts of an image
# our model focuses on during captioning
import matplotlib.pyplot as plt

# Scikit-learn includes many helpful utilities
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle

import re
import numpy as np
import os
import time
import json
from glob import glob
from PIL import Image
import pickle
```

```
TensorFlow 2.x selected.
```

Download and prepare the MS-COCO dataset

You will use the MS-COCO dataset to train our model. The dataset contains over 82,000 images, e
annotations. The code below downloads and extracts the dataset automatically.

Caution: large download ahead. You'll use the training set, which is a 13GB file.

```
annotation_zip = tf.keras.utils.get_file('captions.zip',
                                         cache_subdir=os.path.abspath('.'),
                                         origin = 'http://images.cocodataset.org/annot
                                         extract = True)
annotation_file = os.path.dirname(annotation_zip)+'/annotations/captions_train2014.jso

name_of_zip = 'train2014.zip'
if not os.path.exists(os.path.abspath('.') + '/' + name_of_zip):
  image_zip = tf.keras.utils.get_file(name_of_zip,
                                      cache_subdir=os.path.abspath('.'),
                                      origin = 'http://images.cocodataset.org/zips/trai
                                      extract = True)
  PATH = os.path.dirname(image_zip)+'/train2014/'
else:
  PATH = os.path.abspath('.')+'/train2014/'
```

```
Downloading data from http://images.cocodataset.org/annotations/annotations_trainval2
252878848/252872794 [==============================] - 7s 0us/step
Downloading data from http://images.cocodataset.org/zips/train2014.zip
13510574080/13510573713 [==============================] - 279s 0us/step
```

Optional: limit the size of the training set

To speed up training for this tutorial, you'll use a subset of 30,000 captions and their correspondin
more data would result in improved captioning quality.

```
# Read the json file
with open(annotation_file, 'r') as f:
    annotations = json.load(f)

# Store captions and image names in vectors
all_captions = []
all_img_name_vector = []

for annot in annotations['annotations']:
    caption = '<start> ' + annot['caption'] + ' <end>'
    image_id = annot['image_id']
    full_coco_image_path = PATH + 'COCO_train2014_' + '%012d.jpg' % (image_id)

    all_img_name_vector.append(full_coco_image_path)
    all_captions.append(caption)

# Shuffle captions and image_names together
# Set a random state
train_captions, img_name_vector = shuffle(all_captions,
                                          all_img_name_vector,
                                          random_state=1)

# Select the first 30000 captions from the shuffled set
num_examples = 30000
train_captions = train_captions[:num_examples]
img_name_vector = img_name_vector[:num_examples]
```

```
len(train_captions), len(all_captions)
```

```
(30000, 414113)
```

Preprocess the images using InceptionV3

Next, you will use InceptionV3 (which is pretrained on Imagenet) to classify each image. You will e
layer.

First, you will convert the images into InceptionV3's expected format by:

- Resizing the image to 299px by 299px

- Preprocess the images using the preprocess_input method to normalize the image so that it matches the format of the images used to train InceptionV3.

```
def load_image(image_path):
    img = tf.io.read_file(image_path)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, (299, 299))
    img = tf.keras.applications.inception_v3.preprocess_input(img)
    return img, image_path
```

Initialize InceptionV3 and load the pretrained Imagenet weights

Now you'll create a tf.keras model where the output layer is the last convolutional layer in the Inception of this layer is `8x8x2048`. You use the last convolutional layer because you are using attention in this initialization during training because it could become a bottleneck.

- You forward each image through the network and store the resulting vector in a dictionary (i
- After all the images are passed through the network, you pickle the dictionary and save it to

```
image_model = tf.keras.applications.InceptionV3(include_top=False,
                                                 weights='imagenet')
new_input = image_model.input
hidden_layer = image_model.layers[-1].output

image_features_extract_model = tf.keras.Model(new_input, hidden_layer)
```

```
Downloading data from https://github.com/fchollet/deep-learning-models/releases/downl
87916544/87910968 [==============================] - 2s 0us/step
```

Caching the features extracted from InceptionV3

You will pre-process each image with InceptionV3 and cache the output to disk. Caching the outp intensive, requiring 8 * 8 * 2048 floats per image. At the time of writing, this exceeds the memory memory).

Performance could be improved with a more sophisticated caching strategy (for example, by shar disk I/O), but that would require more code.

The caching will take about 10 minutes to run in Colab with a GPU. If you'd like to see a progress b

1. install tqdm:

   `!pip install tqdm`

2. Import tqdm:

   `from tqdm import tqdm`

3. Change the following line:

`for img, path in image_dataset:`

to:

`for img, path in tqdm(image_dataset):`

```
# Get unique images
encode_train = sorted(set(img_name_vector))

# Feel free to change batch_size according to your system configuration
image_dataset = tf.data.Dataset.from_tensor_slices(encode_train)
image_dataset = image_dataset.map(
  load_image, num_parallel_calls=tf.data.experimental.AUTOTUNE).batch(16)

for img, path in image_dataset:
  batch_features = image_features_extract_model(img)
  batch_features = tf.reshape(batch_features,
                              (batch_features.shape[0], -1, batch_features.shape[3]))

  for bf, p in zip(batch_features, path):
    path_of_feature = p.numpy().decode("utf-8")
    np.save(path_of_feature, bf.numpy())
```

Preprocess and tokenize the captions

- First, you'll tokenize the captions (for example, by splitting on spaces). This gives us a vocab (for example, "surfing", "football", and so on).
- Next, you'll limit the vocabulary size to the top 5,000 words (to save memory). You'll replace (unknown).
- You then create word-to-index and index-to-word mappings.
- Finally, you pad all sequences to be the same length as the longest one.

```
# Find the maximum length of any caption in our dataset
def calc_max_length(tensor):
    return max(len(t) for t in tensor)
```

```
# Choose the top 5000 words from the vocabulary
top_k = 5000
tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=top_k,
                                                  oov_token="<unk>",
                                                  filters='!"#$%&()*+.,-/:;=?@[\]^_`{|}
tokenizer.fit_on_texts(train_captions)
train_seqs = tokenizer.texts_to_sequences(train_captions)
```

```
tokenizer.word_index['<pad>'] = 0
tokenizer.index_word[0] = '<pad>'
```

```
# Create the tokenized vectors
train_seqs = tokenizer.texts_to_sequences(train_captions)
```

```
# Pad each vector to the max_length of the captions
# If you do not provide a max_length value, pad_sequences calculates it automatically
cap_vector = tf.keras.preprocessing.sequence.pad_sequences(train_seqs, padding='post')
```

```
# Calculates the max_length, which is used to store the attention weights
max_length = calc_max_length(train_seqs)
```

Split the data into training and testing

```
# Create training and validation sets using an 80-20 split
img_name_train, img_name_val, cap_train, cap_val = train_test_split(img_name_vector,
                                                                    cap_vector,
                                                                    test_size=0.2,
                                                                    random_state=0)
```

```
len(img_name_train), len(cap_train), len(img_name_val), len(cap_val)
```

```
(24000, 24000, 6000, 6000)
```

Create a tf.data dataset for training

Our images and captions are ready! Next, let's create a tf.data dataset to use for training our mode

```
# Feel free to change these parameters according to your system's configuration

BATCH_SIZE = 64
BUFFER_SIZE = 1000
embedding_dim = 256
units = 512
vocab_size = len(tokenizer.word_index) + 1
num_steps = len(img_name_train) // BATCH_SIZE
# Shape of the vector extracted from InceptionV3 is (64, 2048)
# These two variables represent that vector shape
features_shape = 2048
attention_features_shape = 64
```

```
# Load the numpy files
def map_func(img_name, cap):
  img_tensor = np.load(img_name.decode('utf-8')+'.npy')
  return img_tensor, cap
```

```
dataset = tf.data.Dataset.from_tensor_slices((img_name_train, cap_train))

# Use map to load the numpy files in parallel
dataset = dataset.map(lambda item1, item2: tf.numpy_function(
          map_func, [item1, item2], [tf.float32, tf.int32]),
```

```
          num_parallel_calls=tf.data.experimental.AUTOTUNE)

# Shuffle and batch
dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
dataset = dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

Model

Fun fact: the decoder below is identical to the one in the example for Neural Machine Translation

The model architecture is inspired by the Show, Attend and Tell paper.

- In this example, you extract the features from the lower convolutional layer of InceptionV3 g
- You squash that to a shape of (64, 2048).
- This vector is then passed through the CNN Encoder (which consists of a single Fully conne
- The RNN (here GRU) attends over the image to predict the next word.

```
class BahdanauAttention(tf.keras.Model):
  def __init__(self, units):
    super(BahdanauAttention, self).__init__()
    self.W1 = tf.keras.layers.Dense(units)
    self.W2 = tf.keras.layers.Dense(units)
    self.V = tf.keras.layers.Dense(1)

  def call(self, features, hidden):
    # features(CNN_encoder output) shape == (batch_size, 64, embedding_dim)

    # hidden shape == (batch_size, hidden_size)
    # hidden_with_time_axis shape == (batch_size, 1, hidden_size)
    hidden_with_time_axis = tf.expand_dims(hidden, 1)

    # score shape == (batch_size, 64, hidden_size)
    score = tf.nn.tanh(self.W1(features) + self.W2(hidden_with_time_axis))

    # attention_weights shape == (batch_size, 64, 1)
    # you get 1 at the last axis because you are applying score to self.V
    attention_weights = tf.nn.softmax(self.V(score), axis=1)

    # context_vector shape after sum == (batch_size, hidden_size)
    context_vector = attention_weights * features
    context_vector = tf.reduce_sum(context_vector, axis=1)

    return context_vector, attention_weights
```

```
class CNN_Encoder(tf.keras.Model):
    # Since you have already extracted the features and dumped it using pickle
    # This encoder passes those features through a Fully connected layer
    def __init__(self, embedding_dim):
        super(CNN_Encoder, self).__init__()
        # shape after fc == (batch_size, 64, embedding_dim)
        self.fc = tf.keras.layers.Dense(embedding_dim)

    def call(self, x):
```

```
        x = self.fc(x)
        x = tf.nn.relu(x)
        return x
```

```
class RNN_Decoder(tf.keras.Model):
  def __init__(self, embedding_dim, units, vocab_size):
    super(RNN_Decoder, self).__init__()
    self.units = units

    self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
    self.gru = tf.keras.layers.GRU(self.units,
                                   return_sequences=True,
                                   return_state=True,
                                   recurrent_initializer='glorot_uniform')
    self.fc1 = tf.keras.layers.Dense(self.units)
    self.fc2 = tf.keras.layers.Dense(vocab_size)

    self.attention = BahdanauAttention(self.units)

  def call(self, x, features, hidden):
    # defining attention as a separate model
    context_vector, attention_weights = self.attention(features, hidden)

    # x shape after passing through embedding == (batch_size, 1, embedding_dim)
    x = self.embedding(x)

    # x shape after concatenation == (batch_size, 1, embedding_dim + hidden_size)
    x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

    # passing the concatenated vector to the GRU
    output, state = self.gru(x)

    # shape == (batch_size, max_length, hidden_size)
    x = self.fc1(output)

    # x shape == (batch_size * max_length, hidden_size)
    x = tf.reshape(x, (-1, x.shape[2]))

    # output shape == (batch_size * max_length, vocab)
    x = self.fc2(x)

    return x, state, attention_weights

  def reset_state(self, batch_size):
    return tf.zeros((batch_size, self.units))
```

```
encoder = CNN_Encoder(embedding_dim)
decoder = RNN_Decoder(embedding_dim, units, vocab_size)
```

```
optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')
```

```
def loss_function(real, pred):
  mask = tf.math.logical_not(tf.math.equal(real, 0))
  loss_ = loss_object(real, pred)

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

  return tf.reduce_mean(loss_)
```

Checkpoint

```
checkpoint_path = "./checkpoints/train"
ckpt = tf.train.Checkpoint(encoder=encoder,
                           decoder=decoder,
                           optimizer = optimizer)
ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=5)
```

```
start_epoch = 0
if ckpt_manager.latest_checkpoint:
  start_epoch = int(ckpt_manager.latest_checkpoint.split('-')[-1])
```

Training

- You extract the features stored in the respective `.npy` files and then pass those features thr
- The encoder output, hidden state(initialized to 0) and the decoder input (which is the start to
- The decoder returns the predictions and the decoder hidden state.
- The decoder hidden state is then passed back into the model and the predictions are used t
- Use teacher forcing to decide the next input to the decoder.
- Teacher forcing is the technique where the target word is passed as the next input to the de
- The final step is to calculate the gradients and apply it to the optimizer and backpropagate.

```
# adding this in a separate cell because if you run the training cell
# many times, the loss_plot array will be reset
loss_plot = []
```

```
@tf.function
def train_step(img_tensor, target):
  loss = 0

  # initializing the hidden state for each batch
  # because the captions are not related from image to image
  hidden = decoder.reset_state(batch_size=target.shape[0])

  dec_input = tf.expand_dims([tokenizer.word_index['<start>']] * BATCH_SIZE, 1)

  with tf.GradientTape() as tape:
      features = encoder(img_tensor)

      for i in range(1, target.shape[1]):
```

```
      for i in range(1, target.shape[1]):
          # passing the features through the decoder
          predictions, hidden, _ = decoder(dec_input, features, hidden)

          loss += loss_function(target[:, i], predictions)

          # using teacher forcing
          dec_input = tf.expand_dims(target[:, i], 1)

  total_loss = (loss / int(target.shape[1]))

  trainable_variables = encoder.trainable_variables + decoder.trainable_variables

  gradients = tape.gradient(loss, trainable_variables)

  optimizer.apply_gradients(zip(gradients, trainable_variables))

  return loss, total_loss
```

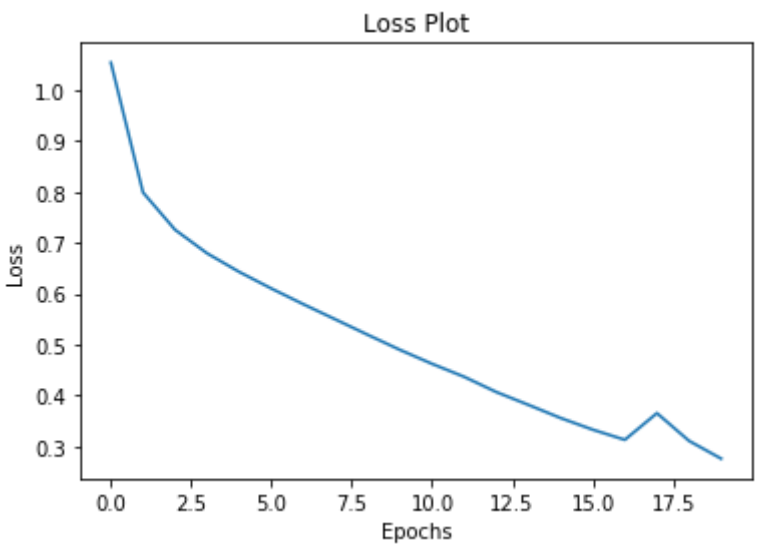
```
EPOCHS = 20

for epoch in range(start_epoch, EPOCHS):
    start = time.time()
    total_loss = 0

    for (batch, (img_tensor, target)) in enumerate(dataset):
        batch_loss, t_loss = train_step(img_tensor, target)
        total_loss += t_loss

        if batch % 100 == 0:
            print ('Epoch {} Batch {} Loss {:.4f}'.format(
              epoch + 1, batch, batch_loss.numpy() / int(target.shape[1])))
    # storing the epoch end loss value to plot later
    loss_plot.append(total_loss / num_steps)

    if epoch % 5 == 0:
      ckpt_manager.save()

    print ('Epoch {} Loss {:.6f}'.format(epoch + 1,
                                        total_loss/num_steps))
    print ('Time taken for 1 epoch {} sec\n'.format(time.time() - start))
```

```
Epoch 1 Batch 100 Loss 1.1190
Epoch 1 Batch 200 Loss 0.9899
Epoch 1 Batch 300 Loss 0.9442
Epoch 1 Loss 1.054041
Time taken for 1 epoch 192.8543345928192 sec

Epoch 2 Batch 0 Loss 0.8338
Epoch 2 Batch 100 Loss 0.8791
Epoch 2 Batch 200 Loss 0.7216
Epoch 2 Batch 300 Loss 0.7063
Epoch 2 Loss 0.798910
Time taken for 1 epoch 125.656747341156 sec

Epoch 3 Batch 0 Loss 0.7415
Epoch 3 Batch 100 Loss 0.8106
Epoch 3 Batch 200 Loss 0.6734
Epoch 3 Batch 300 Loss 0.6491
Epoch 3 Loss 0.725524
Time taken for 1 epoch 120.73184537887573 sec

Epoch 4 Batch 0 Loss 0.7349
Epoch 4 Batch 100 Loss 0.6838
Epoch 4 Batch 200 Loss 0.6813
Epoch 4 Batch 300 Loss 0.6364
Epoch 4 Loss 0.679165
Time taken for 1 epoch 122.10045790672302 sec

Epoch 5 Batch 0 Loss 0.6109
Epoch 5 Batch 100 Loss 0.6860
Epoch 5 Batch 200 Loss 0.6958
Epoch 5 Batch 300 Loss 0.6375
Epoch 5 Loss 0.642676
Time taken for 1 epoch 120.26841425895691 sec

Epoch 6 Batch 0 Loss 0.6268
Epoch 6 Batch 100 Loss 0.5890
Epoch 6 Batch 200 Loss 0.6445
Epoch 6 Batch 300 Loss 0.5690
Epoch 6 Loss 0.610092
Time taken for 1 epoch 119.7681314945221 sec

Epoch 7 Batch 0 Loss 0.6375
Epoch 7 Batch 100 Loss 0.6546
Epoch 7 Batch 200 Loss 0.5352
Epoch 7 Batch 300 Loss 0.5382
Epoch 7 Loss 0.579111
Time taken for 1 epoch 122.47705960273743 sec

Epoch 8 Batch 0 Loss 0.5969
Epoch 8 Batch 100 Loss 0.5345
Epoch 8 Batch 200 Loss 0.5162
Epoch 8 Batch 300 Loss 0.4649
Epoch 8 Loss 0.549228
Time taken for 1 epoch 122.76927876472473 sec

Epoch 9 Batch 0 Loss 0.5620
Epoch 9 Batch 100 Loss 0.5537
Epoch 9 Batch 200 Loss 0.4900
Epoch 9 Batch 300 Loss 0.5302
Epoch 9 Loss 0.519166
Time taken for 1 epoch 123.99425554275513 sec
```

```
Epoch 10 Batch 0 Loss 0.5277
Epoch 10 Batch 100 Loss 0.4870
Epoch 10 Batch 200 Loss 0.4874
Epoch 10 Batch 300 Loss 0.4939
Epoch 10 Loss 0.489652
Time taken for 1 epoch 124.35941863059998 sec

Epoch 11 Batch 0 Loss 0.4914
Epoch 11 Batch 100 Loss 0.4771
Epoch 11 Batch 200 Loss 0.5268
Epoch 11 Batch 300 Loss 0.4452
Epoch 11 Loss 0.462202
Time taken for 1 epoch 123.99740743637085 sec

Epoch 12 Batch 0 Loss 0.4475
Epoch 12 Batch 100 Loss 0.4462
Epoch 12 Batch 200 Loss 0.4191
Epoch 12 Batch 300 Loss 0.4433
Epoch 12 Loss 0.436695
Time taken for 1 epoch 124.58229184150696 sec

Epoch 13 Batch 0 Loss 0.4457
Epoch 13 Batch 100 Loss 0.3854
Epoch 13 Batch 200 Loss 0.4228
Epoch 13 Batch 300 Loss 0.3777
Epoch 13 Loss 0.406774
Time taken for 1 epoch 131.65870547294617 sec

Epoch 14 Batch 0 Loss 0.3600
Epoch 14 Batch 100 Loss 0.3559
Epoch 14 Batch 200 Loss 0.3925
Epoch 14 Batch 300 Loss 0.3848
Epoch 14 Loss 0.381558
Time taken for 1 epoch 126.39847564697266 sec

Epoch 15 Batch 0 Loss 0.4002
Epoch 15 Batch 100 Loss 0.3492
Epoch 15 Batch 200 Loss 0.3374
Epoch 15 Batch 300 Loss 0.3754
Epoch 15 Loss 0.355790
Time taken for 1 epoch 127.11393213272095 sec

Epoch 16 Batch 0 Loss 0.3441
Epoch 16 Batch 100 Loss 0.3307
Epoch 16 Batch 200 Loss 0.2964
Epoch 16 Batch 300 Loss 0.3048
Epoch 16 Loss 0.332936
Time taken for 1 epoch 126.98682117462158 sec

Epoch 17 Batch 0 Loss 0.3478
Epoch 17 Batch 100 Loss 0.3369
Epoch 17 Batch 200 Loss 0.2933
Epoch 17 Batch 300 Loss 0.3063
Epoch 17 Loss 0.312953
Time taken for 1 epoch 125.65758037567139 sec

Epoch 18 Batch 0 Loss 0.3403
Epoch 18 Batch 100 Loss 0.5351
Epoch 18 Batch 200 Loss 0.3742
Epoch 18 Batch 300 Loss 0.3004
Epoch 18 Loss 0.365327
```

```
Time taken for 1 epoch 126.11485457420349 sec

Epoch 19 Batch 0 Loss 0.3788
Epoch 19 Batch 100 Loss 0.3958
Epoch 19 Batch 200 Loss 0.3135
Epoch 19 Batch 300 Loss 0.2701
Epoch 19 Loss 0.310845
Time taken for 1 epoch 126.5215368270874 sec

Epoch 20 Batch 0 Loss 0.2974
Epoch 20 Batch 100 Loss 0.2842
Epoch 20 Batch 200 Loss 0.2918
Epoch 20 Batch 300 Loss 0.2699
Epoch 20 Loss 0.275944
Time taken for 1 epoch 125.78373312950134 sec
```

```
plt.plot(loss_plot)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Plot')
plt.show()
```



Caption!

- The evaluate function is similar to the training loop, except you don't use teacher forcing her is its previous predictions along with the hidden state and the encoder output.
- Stop predicting when the model predicts the end token.
- And store the attention weights for every time step.

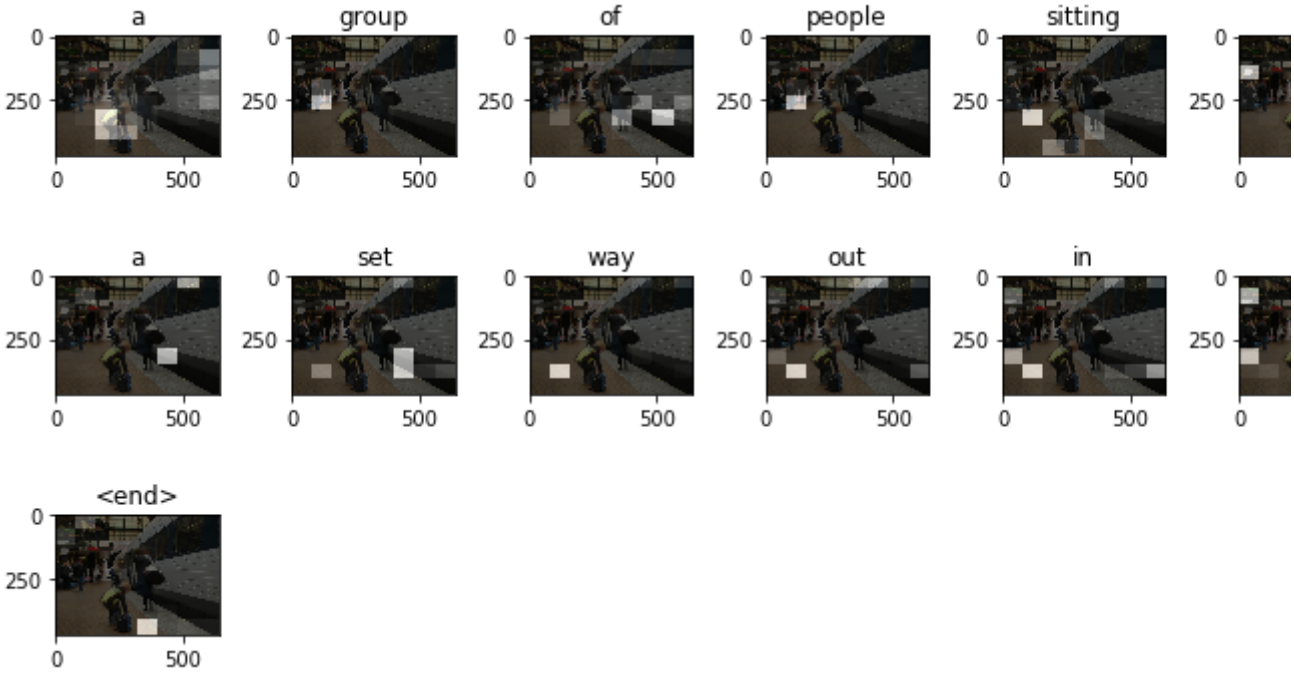
```
def evaluate(image):
    attention_plot = np.zeros((max_length, attention_features_shape))

    hidden = decoder.reset_state(batch_size=1)

    temp_input = tf.expand_dims(load_image(image)[0], 0)
    img_tensor_val = image_features_extract_model(temp_input)
    img_tensor_val = tf.reshape(img_tensor_val, (img_tensor_val.shape[0], -1, img_tenso
```

```
    features = encoder(img_tensor_val)

    dec_input = tf.expand_dims([tokenizer.word_index['<start>']], 0)
    result = []

    for i in range(max_length):
        predictions, hidden, attention_weights = decoder(dec_input, features, hidden)

        attention_plot[i] = tf.reshape(attention_weights, (-1, )).numpy()

        predicted_id = tf.argmax(predictions[0]).numpy()
        result.append(tokenizer.index_word[predicted_id])

        if tokenizer.index_word[predicted_id] == '<end>':
            return result, attention_plot

        dec_input = tf.expand_dims([predicted_id], 0)

    attention_plot = attention_plot[:len(result), :]
    return result, attention_plot
```

```
def plot_attention(image, result, attention_plot):
    temp_image = np.array(Image.open(image))

    fig = plt.figure(figsize=(10, 10))

    len_result = len(result)
    for l in range(len_result):
        temp_att = np.resize(attention_plot[l], (8, 8))
        ax = fig.add_subplot(len_result//2, len_result//2, l+1)
        ax.set_title(result[l])
        img = ax.imshow(temp_image)
        ax.imshow(temp_att, cmap='gray', alpha=0.6, extent=img.get_extent())

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

```
# captions on the validation set
rid = np.random.randint(0, len(img_name_val))
image = img_name_val[rid]
real_caption = ' '.join([tokenizer.index_word[i] for i in cap_val[rid] if i not in [0]]
result, attention_plot = evaluate(image)

print ('Real Caption:', real_caption)
print ('Prediction Caption:', ' '.join(result))
plot_attention(image, result, attention_plot)
# opening the image
Image.open(img_name_val[rid])
```

```
Real Caption: <start> the little boy is picking up his suitcases <end>
Prediction Caption: a group of people sitting on a set way out in front <end>
```

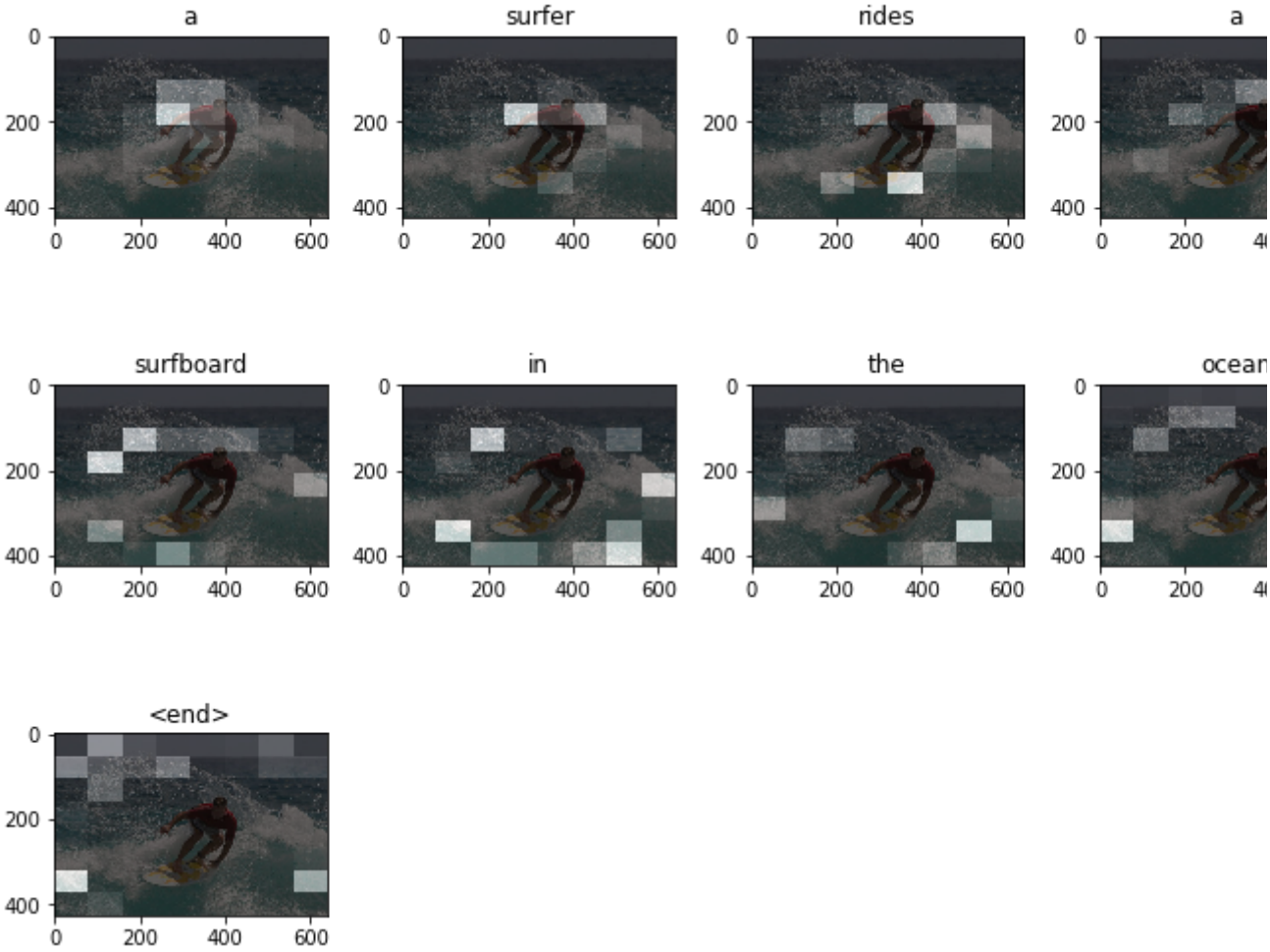


Try it on your own images

For fun, below we've provided a method you can use to caption your own images with the model w
on a relatively small amount of data, and your images may be different from the training data (so

```
image_url = 'https://tensorflow.org/images/surf.jpg'
image_extension = image_url[-4:]
image_path = tf.keras.utils.get_file('image'+image_extension,
                                     origin=image_url)

result, attention_plot = evaluate(image_path)
print ('Prediction Caption:', ' '.join(result))
plot_attention(image_path, result, attention_plot)
# opening the image
Image.open(image_path)
```

```
Downloading data from https://tensorflow.org/images/surf.jpg
65536/64400 [==============================] - 0s 0us/step
Prediction Caption: a surfer rides a surfboard in the ocean <end>
```



Chapter 3 : Number Plate Detection

```
Import numpy as np
import cv2
from copy import deepcopy
from PIL import Image
import pytesseract as tess


def preprocess(img):
	cv2.imshow("Input",img)
	imgBlurred = cv2.GaussianBlur(img, (5,5), 0)
	gray = cv2.cvtColor(imgBlurred, cv2.COLOR_BGR2GRAY)


	sobelx = cv2.Sobel(gray,cv2.CV_8U,1,0,ksize=3)
	#cv2.imshow("Sobel",sobelx)
	#cv2.waitKey(0)
	ret2,threshold_img =
cv2.threshold(sobelx,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
	#cv2.imshow("Threshold",threshold_img)
	#cv2.waitKey(0)
	return threshold_img


def cleanPlate(plate):
	print("CLEANING PLATE. . .")
	gray = cv2.cvtColor(plate, cv2.COLOR_BGR2GRAY)
	#kernel = cv2.getStructuringElement(cv2.MORPH_CROSS, (3, 3))
	#thresh= cv2.dilate(gray, kernel, iterations=1)


	_, thresh = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY)
```

```
	im1,contours,hierarchy =
cv2.findContours(thresh.copy(),cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)


	if contours:
		areas = [cv2.contourArea(c) for c in contours]
		max_index = np.argmax(areas)


		max_cnt = contours[max_index]
		max_cntArea = areas[max_index]
		x,y,w,h = cv2.boundingRect(max_cnt)


		if not ratioCheck(max_cntArea,w,h):
			return plate,None


		cleaned_final = thresh[y:y+h, x:x+w]
		#cv2.imshow("Function Test",cleaned_final)
		return cleaned_final,[x,y,w,h]


	else:
		return plate,None




def extract_contours(threshold_img):
	element = cv2.getStructuringElement(shape=cv2.MORPH_RECT,
ksize=(17, 3))
	morph_img_threshold = threshold_img.copy()
	cv2.morphologyEx(src=threshold_img, op=cv2.MORPH_CLOSE,
kernel=element, dst=morph_img_threshold)
```

```
        cv2.imshow("Morphed",morph_img_threshold)
        cv2.waitKey(0)


        im2,contours, hierarchy=
cv2.findContours(morph_img_threshold,mode=cv2.RETR_EXTERNAL,method=cv2.
CHAIN_APPROX_NONE)
        return contours




def ratioCheck(area, width, height):
        ratio = float(width) / float(height)
        if ratio < 1:
                ratio = 1 / ratio


        aspect = 4.7272
        min = 15*aspect*15  # minimum area
        max = 125*aspect*125  # maximum area


        rmin = 3
        rmax = 6


        if (area < min or area > max) or (ratio < rmin or ratio > rmax):
                return False
        return True


def isMaxWhite(plate):
        avg = np.mean(plate)
        if(avg>=115):
```

```
                return True
        else:
                return False


def validateRotationAndRatio(rect):
        (x, y), (width, height), rect_angle = rect


        if(width>height):
                angle = -rect_angle
        else:
                angle = 90 + rect_angle


        if angle>15:
                return False


        if height == 0 or width == 0:
                return False


        area = height*width
        if not ratioCheck(area,width,height):
                return False
        else:
                return True




def cleanAndRead(img,contours):
```

```
	#count=0
	for i,cnt in enumerate(contours):
		min_rect = cv2.minAreaRect(cnt)


		if validateRotationAndRatio(min_rect):


			x,y,w,h = cv2.boundingRect(cnt)
			plate_img = img[y:y+h,x:x+w]




			if(isMaxWhite(plate_img)):
				#count+=1
				clean_plate, rect = cleanPlate(plate_img)


				if rect:
					x1,y1,w1,h1 = rect
					x,y,w,h = x+x1,y+y1,w1,h1
					cv2.imshow("Cleaned
Plate",clean_plate)
					cv2.waitKey(0)
					plate_im =
Image.fromarray(clean_plate)
					text = tess.image_to_string(plate_im,
lang='eng')
					print("Detected Text : ",text)
					img =
cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
					cv2.imshow("Detected Plate",img)
					cv2.waitKey(0)
```

```
        #print "No. of final cont : " , count




if __name__ == '__main__':
        print( "DETECTING PLATE . . .")



        img = cv2.imread("test.jpeg")


        threshold_img = preprocess(img)
        contours= extract_contours(threshold_img)


        #if len(contours)!=0:
                #print len(contours) #Test
                # cv2.drawContours(img, contours, -1, (0,255,0), 1)
                # cv2.imshow("Contours",img)
                # cv2.waitKey(0)




        cleanAndRead(img,contours)
```


E 7215 BGN

Chapter 4 : Human Posture Detection

```
# To use Inference Engine backend, specify location of plugins:
# export LD_LIBRARY_PATH=/opt/intel/deeplearning_deploymenttoolkit/
deployment_tools/external/mklml_lnx/lib:$LD_LIBRARY_PATH
import cv2 as cv
import numpy as np
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('--input', help='image.jpg')
parser.add_argument('--thr', default=0.2, type=float, help='Threshold value for
pose parts heat map')
parser.add_argument('--width', default=368, type=int, help='Resize input to
specific width.')
parser.add_argument('--height', default=368, type=int, help='Resize input to
specific height.')

args = parser.parse_args()

BODY_PARTS = { "Nose": 0, "Neck": 1, "RShoulder": 2, "RElbow": 3, "RWrist": 4,
               "LShoulder": 5, "LElbow": 6, "LWrist": 7, "RHip": 8, "RKnee": 9,
               "RAnkle": 10, "LHip": 11, "LKnee": 12, "LAnkle": 13, "REye": 14,
               "LEye": 15, "REar": 16, "LEar": 17, "Background": 18 }

POSE_PAIRS = [ ["Neck", "RShoulder"], ["Neck", "LShoulder"], ["RShoulder",
"RElbow"],
               ["RElbow", "RWrist"], ["LShoulder", "LElbow"], ["LElbow", "LWrist"],
               ["Neck", "RHip"], ["RHip", "RKnee"], ["RKnee", "RAnkle"], ["Neck",
"LHip"],
               ["LHip", "LKnee"], ["LKnee", "LAnkle"], ["Neck", "Nose"], ["Nose",
"REye"],
               ["REye", "REar"], ["Nose", "LEye"], ["LEye", "LEar"] ]

inWidth = args.width
inHeight = args.height

net = cv.dnn.readNetFromTensorflow("graph_opt.pb")

cap = cv.VideoCapture(args.input if args.input else 0)

while cv.waitKey(1) < 0:
    hasFrame, frame = cap.read()
    if not hasFrame:
        cv.waitKey()
        break

    frameWidth = frame.shape[1]
    frameHeight = frame.shape[0]

    net.setInput(cv.dnn.blobFromImage(frame, 1.0, (inWidth, inHeight), (127.5,
127.5, 127.5), swapRB=True, crop=False))
    out = net.forward()
    out = out[:, :19, :, :]  # MobileNet output [1, 57, -1, -1], we only need the
first 19 elements

    assert(len(BODY_PARTS) == out.shape[1])

    points = []
    for i in range(len(BODY_PARTS)):
        # Slice heatmap of corresponging body's part.
        heatMap = out[0, i, :, :]

        # Originally, we try to find all the local maximums. To simplify a sample
        # we just find a global one. However only a single pose at the same time
        # could be detected this way.
        _, conf, _, point = cv.minMaxLoc(heatMap)
        x = (frameWidth * point[0]) / out.shape[3]
```

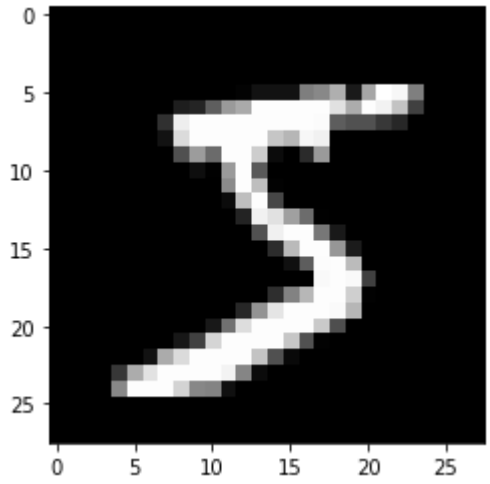
```
            y = (frameHeight * point[1]) / out.shape[2]
            # Add a point if it's confidence is higher than threshold.
            points.append((int(x), int(y)) if conf > args.thr else None)

        for pair in POSE_PAIRS:
            partFrom = pair[0]
            partTo = pair[1]
            assert(partFrom in BODY_PARTS)
            assert(partTo in BODY_PARTS)

            idFrom = BODY_PARTS[partFrom]
            idTo = BODY_PARTS[partTo]

            if points[idFrom] and points[idTo]:
                cv.line(frame, points[idFrom], points[idTo], (0, 255, 0), 3)
                cv.ellipse(frame, points[idFrom], (3, 3), 0, 0, 360, (0, 0, 255),
cv.FILLED)
                cv.ellipse(frame, points[idTo], (3, 3), 0, 0, 360, (0, 0, 255),
cv.FILLED)

        t, _ = net.getPerfProfile()
        freq = cv.getTickFrequency() / 1000
        cv.putText(frame, '%.2fms' % (t / freq), (10, 20), cv.FONT_HERSHEY_SIMPLEX,
0.5, (0, 0, 0))

        cv.imshow('OpenPose using OpenCV', frame)
```

Chapter 5 : Handwritten Digit Recognition

```
!pip install numpy
import numpy
```

```
Requirement already satisfied: numpy in /tensorflow-2.0.0/python3.6 (1.17.4)
```

```
try:
  # %tensorflow_version only exists in Colab.
  %tensorflow_version 2.x
except Exception:
  pass

import tensorflow as tf
import matplotlib.pyplot as plt # Import matplotlib library

mnist = tf.keras.datasets.mnist # Object of the MNIST dataset
(x_train, y_train),(x_test, y_test) = mnist.load_data() # Load data


plt.imshow(x_train[0], cmap="gray") # Import the image
plt.show() # Plot the image
```




```
# Normalize the train dataset
x_train = tf.keras.utils.normalize(x_train, axis=1)
# Normalize the test dataset
x_test = tf.keras.utils.normalize(x_test, axis=1)


#Build the model object
model = tf.keras.models.Sequential()
# Add the Flatten Layer
model.add(tf.keras.layers.Flatten())
# Build the input and the hidden layers
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
# Build the output layer
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
```

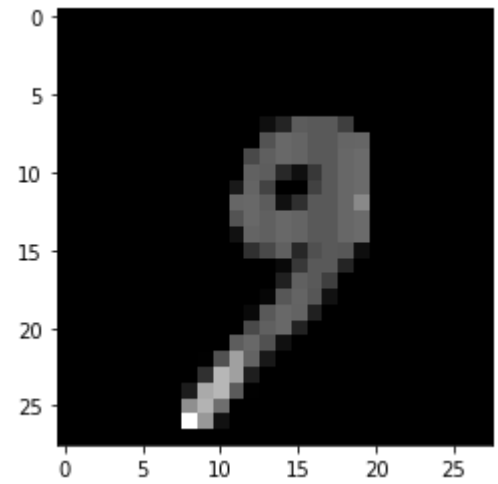
```
# Compile the model
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])


model.fit(x=x_train, y=y_train, epochs=10) # Start training process
```

```
Train on 60000 samples
Epoch 1/10
60000/60000 [==============================] - 6s 102us/sample - loss: 0.2656 - accuracy
Epoch 2/10
60000/60000 [==============================] - 5s 92us/sample - loss: 0.1058 - accuracy:
Epoch 3/10
60000/60000 [==============================] - 6s 93us/sample - loss: 0.0727 - accuracy:
Epoch 4/10
60000/60000 [==============================] - 6s 96us/sample - loss: 0.0538 - accuracy:
Epoch 5/10
60000/60000 [==============================] - 6s 95us/sample - loss: 0.0403 - accuracy:
Epoch 6/10
60000/60000 [==============================] - 5s 91us/sample - loss: 0.0324 - accuracy:
Epoch 7/10
60000/60000 [==============================] - 6s 97us/sample - loss: 0.0260 - accuracy:
Epoch 8/10
60000/60000 [==============================] - 5s 92us/sample - loss: 0.0217 - accuracy:
Epoch 9/10
60000/60000 [==============================] - 6s 95us/sample - loss: 0.0161 - accuracy:
Epoch 10/10
60000/60000 [==============================] - 6s 95us/sample - loss: 0.0162 - accuracy:
<tensorflow.python.keras.callbacks.History at 0x7f1697771c50>
```

```
predictions = model.predict([x_test]) # Make prediction


print(numpy.argmax(predictions[1000])) # Print out the number
```

```
9
```

```
plt.imshow(x_test[1000], cmap="gray") # Import the image
plt.show() # Show the image
```



Chapter 6: Face Matching

```
!pip3 install face_recognition
```

```
Collecting face_recognition
  Downloading https://files.pythonhosted.org/packages/3f/ed/ad9a28042f373d4633
Requirement already satisfied: Click>=6.0 in /usr/local/lib/python3.6/dist-pac
Collecting face-recognition-models>=0.3.0
  Downloading https://files.pythonhosted.org/packages/cf/3b/4fd8c534f6c0d1b80c
     |████████████████████████████████| 100.2MB 102kB/s
Requirement already satisfied: Pillow in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: dlib>=19.7 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: olefile in /usr/local/lib/python3.6/dist-packag
Building wheels for collected packages: face-recognition-models
  Building wheel for face-recognition-models (setup.py) ... done
  Created wheel for face-recognition-models: filename=face_recognition_models-
  Stored in directory: /root/.cache/pip/wheels/d2/99/18/59c6c8f01e39810415c0e6
Successfully built face-recognition-models
Installing collected packages: face-recognition-models, face-recognition
Successfully installed face-recognition-1.2.3 face-recognition-models-0.3.0
```

```
import face_recognition
known_image = face_recognition.load_image_file("obama.jpg")
unknown_image = face_recognition.load_image_file("unknown.jpg")

biden_encoding = face_recognition.face_encodings(known_image)[0]
unknown_encoding = face_recognition.face_encodings(unknown_image)[0]

results = face_recognition.compare_faces([biden_encoding], unknown_encoding)


print(results)
```

```
[True]
```

Chapter 7 : Image Classification

```
import cv2
import argparse
import numpy as np

ap = argparse.ArgumentParser()
ap.add_argument('-i', '--image', required=True,
                help='C:/Users/Juhi/Downloads/faces.jpg')
ap.add_argument('-c', '--config', required=True,
                help='C:/Users/Juhi/Downloads/yolov3Config.cfg')
ap.add_argument('-w', '--weights', required=True,
                help='C:/Users/Juhi/Downloads/yolov3we.weights')
ap.add_argument('-cl', '--classes', required=True,
                help='C:/Users/Juhi/Downloads/yolov3.txt')
args = ap.parse_args()
```

Importing the libraries and passing the argument.

```
def get_output_layers(net):
    layer_names = net.getLayerNames()

    output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]

    return output_layers
```

Create layers

```
def draw_prediction(img, class_id, confidence, x, y, x_plus_w, y_plus_h):
    label = str(classes[class_id])

    color = COLORS[class_id]

    cv2.rectangle(img, (x, y), (x_plus_w, y_plus_h), color, 2)

    cv2.putText(img, label, (x - 10, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
```

Setting the frame which would appear on the object.

```
image = cv2.imread(args.image)

Width = image.shape[1]
Height = image.shape[0]
scale = 0.00392

classes = None

with open(args.classes, 'r') as f:
    classes = [line.strip() for line in f.readlines()]

COLORS = np.random.uniform(0, 255, size=(len(classes), 3))

net = cv2.dnn.readNet(args.weights, args.config)
```

Classes are used to classify the objects into a particular set. (List of classes are mention in yolov3.txt file)

```
blob = cv2.dnn.blobFromImage(image, scale, (416, 416), (0, 0, 0), True, crop=False)

net.setInput(blob)

outs = net.forward(get_output_layers(net))

class_ids = []
confidences = []
boxes = []
conf_threshold = 0.5
nms_threshold = 0.4
```

Setting the thresholds.

```
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:
            center_x = int(detection[0] * Width)
            center_y = int(detection[1] * Height)
            w = int(detection[2] * Width)
            h = int(detection[3] * Height)
            x = center_x - w / 2
            y = center_y - h / 2
            class_ids.append(class_id)
            confidences.append(float(confidence))
            boxes.append([x, y, w, h])
```

Detection using random anchor box and setting confidence. If confidence is greater than 0.5 than it is an object else discard.

```
indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold, nms_threshold)

for i in indices:
    i = i[0]
    box = boxes[i]
    x = box[0]
    y = box[1]
    w = box[2]
    h = box[3]
    draw_prediction(image, class_ids[i], confidences[i], round(x), round(y), round(x + w), round(y + h))

cv2.imshow("object detection", image)
cv2.waitKey()

cv2.imwrite("object-detection.jpg", image)
cv2.destroyAllWindows()
```

Objects are detected from the image and now predicted according to classes. Finally the file is saved.

Output: