## Install & install all necessary packages

```
%matplotlib inline
from pycocotools.coco import COCO
import numpy as np
import skimage.io as io
import matplotlib.pyplot as plt
import pylab
pylab.rcParams['figure.figsize'] = (8.0, 10.0)
import pandas as pd
import os
import PIL
import zipfile
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
import tensorflow as tf
print(tf.__version__)

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.python.keras.layers import Dense

from tensorflow.keras.applications import ResNet50
from tensorflow.python.keras import optimizers
from tensorflow.python.keras.callbacks import EarlyStopping
from sklearn.metrics import accuracy_score
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, Attention
```

```
2.12.0
```

## Loading Dataset

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

```
Mounted at /content/drive
```

```
import zipfile

path_to_zip_file = '/content/drive/MyDrive/Colab Notebooks/annotations_trainval2014.zip'
output_folder = '/content/'

with zipfile.ZipFile(path_to_zip_file, 'r') as zip_ref:
    zip_ref.extractall(output_folder)
```

```
from pycocotools.coco import COCO

```

```
# Specify the correct absolute path to the COCO annotations file
annotations_path = 'annotations/instances_train2014.json'


```

```
import os

# Get the absolute path to the annotations file
annotations_path = os.path.abspath('annotations/instances_train2014.json')

```

```
!mkdir data
!cd data && wget http://images.cocodataset.org/zips/train2014.zip
!cd data && unzip train2014.zip


# Check if the file exists

import os


if os.path.exists(annotations_path):
    print("Annotations file exists.")
     # Continue with loading the COCO API
    coco = COCO(annotations_path)
else:
    print(f"Annotations file not found: {annotations_path}")
```

```
Streaming output truncated to the last 5000 lines.
 extracting: train2014/COCO_train2014_000000141015.jpg
 extracting: train2014/COCO_train2014_000000280923.jpg
 extracting: train2014/COCO_train2014_000000200024.jpg
 extracting: train2014/COCO_train2014_000000435713.jpg
 extracting: train2014/COCO_train2014_000000249993.jpg
 extracting: train2014/COCO_train2014_000000424160.jpg
 extracting: train2014/COCO_train2014_000000142761.jpg
 extracting: train2014/COCO_train2014_000000532668.jpg
 extracting: train2014/COCO_train2014_000000564904.jpg
 extracting: train2014/COCO_train2014_000000346384.jpg
 extracting: train2014/COCO_train2014_000000560934.jpg
 extracting: train2014/COCO_train2014_000000122356.jpg
 extracting: train2014/COCO_train2014_000000193042.jpg
 extracting: train2014/COCO_train2014_000000072702.jpg
 extracting: train2014/COCO_train2014_000000064795.jpg
 extracting: train2014/COCO_train2014_000000118089.jpg
 extracting: train2014/COCO_train2014_000000082228.jpg
 extracting: train2014/COCO_train2014_000000002148.jpg
 extracting: train2014/COCO_train2014_000000102210.jpg
 extracting: train2014/COCO_train2014_000000281355.jpg
 extracting: train2014/COCO_train2014_000000312076.jpg
 extracting: train2014/COCO_train2014_000000040008.jpg
 extracting: train2014/COCO_train2014_000000483458.jpg
 extracting: train2014/COCO_train2014_000000205131.jpg
 extracting: train2014/COCO_train2014_000000223534.jpg
 extracting: train2014/COCO_train2014_000000309137.jpg
 extracting: train2014/COCO_train2014_000000353559.jpg
 extracting: train2014/COCO_train2014_000000088136.jpg
 extracting: train2014/COCO_train2014_000000550649.jpg
 extracting: train2014/COCO_train2014_000000541343.jpg
 extracting: train2014/COCO_train2014_000000522992.jpg
 extracting: train2014/COCO_train2014_000000079107.jpg
 extracting: train2014/COCO_train2014_000000475185.jpg
 extracting: train2014/COCO_train2014_000000561339.jpg
 extracting: train2014/COCO_train2014_000000273503.jpg
 extracting: train2014/COCO_train2014_000000566439.jpg
 extracting: train2014/COCO_train2014_000000364457.jpg
 extracting: train2014/COCO_train2014_000000006355.jpg
 extracting: train2014/COCO_train2014_000000254078.jpg
 extracting: train2014/COCO_train2014_000000204867.jpg
 extracting: train2014/COCO_train2014_000000266496.jpg
 extracting: train2014/COCO_train2014_000000065712.jpg
 extracting: train2014/COCO_train2014_000000092731.jpg
 extracting: train2014/COCO_train2014_000000173474.jpg
 extracting: train2014/COCO_train2014_000000136043.jpg
 extracting: train2014/COCO_train2014_000000232919.jpg
 extracting: train2014/COCO_train2014_000000223526.jpg
 extracting: train2014/COCO_train2014_000000091973.jpg
 extracting: train2014/COCO_train2014_000000224407.jpg
 extracting: train2014/COCO_train2014_000000129753.jpg
 extracting: train2014/COCO_train2014_000000300454.jpg
 extracting: train2014/COCO_train2014_000000560388.jpg
 extracting: train2014/COCO_train2014_000000292146.jpg
 extracting: train2014/COCO_train2014_000000362839.jpg
 extracting: train2014/COCO_train2014_000000505418.jpg
 extracting: train2014/COCO_train2014_000000565858.jpg
 extracting: train2014/COCO_train2014_000000313971.jpg
```

```
import os
from pycocotools.coco import COCO

# Get the absolute path to the annotations file
annotations_path = os.path.abspath('annotations/instances_train2014.json')

# Create a COCO object
coco = COCO(annotations_path)

```

```
loading annotations into memory...
Done (t=20.83s)
creating index...
index created!
```

```
# Get all image IDs
image_ids = coco.getImgIds()
print(image_ids)
```

```
[57870, 384029, 222016, 520950, 69675, 547471, 122688, 392136, 398494, 90570, 504616, 161919, 457732, 44404, 4428, 170558, 405613, 28352
```

```
# Get information about a specific image
image_info = coco.loadImgs(image_ids[0])
print("Image Info:", image_info)
```

```
Image Info: [{'license': 5, 'file_name': 'COCO_train2014_000000057870.jpg', 'coco_url': 'http://images.cocodataset.org/train2014/COCO_tr
```

```
# Get annotations for a specific image
annotation_ids = coco.getAnnIds(imgIds=image_ids[0])
annotations = coco.loadAnns(annotation_ids)
print("Annotations:", annotations)
```

```
Annotations: [{'segmentation': [[233.29, 461.42, 235.35, 395.35, 240.52, 378.84, 245.68, 372.65, 250.84, 372.65, 283.87, 372.65, 294.19,
```

```
# Get all category IDs
category_ids = coco.getCatIds()

# Get information about a specific category
category_info = coco.loadCats(category_ids[0])
print("Category Info:", category_info)

```

```
Category Info: [{'supercategory': 'person', 'id': 1, 'name': 'person'}]
```

```
import matplotlib.pyplot as plt
from skimage.io import imread
```

```
# Get annotations for a specific image (e.g., the first image)
annotation_ids = coco.getAnnIds(imgIds=image_ids[0])
annotations = coco.loadAnns(annotation_ids)

# Print specific information about each annotation
for annotation in annotations:
    print("Annotation ID:", annotation['id'])
    print("Category ID:", annotation['category_id'])
    print("Segmentation:", annotation['segmentation'])
    print("Bounding Box:", annotation['bbox'])

    print("\n")

```

```
Annotation ID: 102924
Category ID: 62
Segmentation: [[233.29, 461.42, 235.35, 395.35, 240.52, 378.84, 245.68, 372.65, 250.84, 372.65, 283.87, 372.65, 294.19, 362.32, 298.3
Bounding Box: [233.29, 270.45, 170.32, 203.36]


Annotation ID: 118582
Category ID: 67
Segmentation: [[4.41, 338.69, 27.0, 342.08, 38.29, 361.28, 41.68, 378.22, 39.42, 480.0, 75.56, 479.87, 73.31, 357.89, 87.99, 359.02,
Bounding Box: [1.02, 279.96, 533.09, 200.04]
```

```
Annotation ID: 290250
Category ID: 62
Segmentation: [[515.6, 462.74, 515.6, 424.99, 478.92, 417.44, 476.76, 368.9, 496.18, 338.7, 515.6, 335.46, 526.38, 293.39, 560.9, 261
Bounding Box: [476.76, 261.03, 119.73, 201.71]


Annotation ID: 290659
Category ID: 62
Segmentation: [[10.79, 294.54, 15.1, 284.84, 17.26, 280.52, 25.89, 275.13, 37.75, 266.5, 46.38, 266.5, 63.64, 260.03, 72.27, 261.11,
Bounding Box: [10.79, 260.03, 114.33, 124.04]


Annotation ID: 290682
Category ID: 62
Segmentation: [[381.84, 382.92, 390.47, 365.66, 396.94, 358.11, 412.04, 358.11, 419.6, 350.56, 426.07, 299.87, 435.78, 279.37, 467.06
Bounding Box: [367.82, 264.27, 139.15, 215.73]


Annotation ID: 290718
Category ID: 62
Segmentation: [[563.06, 469.21, 539.33, 393.71, 551.19, 333.3, 577.08, 302.02, 608.36, 290.16, 634.25, 297.71, 637.48, 299.87, 640.0,
Bounding Box: [539.33, 290.16, 100.67, 179.05]


Annotation ID: 290760
Category ID: 62
Segmentation: [[364.4, 262.46, 369.36, 257.5, 380.82, 256.57, 392.59, 257.5, 409.62, 270.82, 417.06, 282.9, 396.92, 283.21, 384.84, 2
Bounding Box: [364.4, 256.57, 52.66, 26.64]


Annotation ID: 290772
Category ID: 62
Segmentation: [[152.64, 282.49, 173.3, 260.89, 190.67, 256.67, 212.27, 260.42, 226.35, 271.22, 230.58, 275.92, 207.1, 270.75, 189.73,
Bounding Box: [152.17, 256.67, 78.41, 29.11]


Annotation ID: 290776
Category ID: 62
Segmentation: [[32.36, 480.0, 32.36, 443.33, 28.04, 413.12, 22.65, 394.79, 1.08, 373.21, 2.16, 476.76, 30.2, 475.69]]
Bounding Box: [1.08, 373.21, 31.28, 106.79]


Annotation ID: 1154173
Category ID: 86
```

## Performing data cleaning

```
# Check for missing values in annotations
for annotation in annotations:
    if 'bbox' not in annotation:
        print(f"Error: Missing bounding box in annotation {annotation['id']}")


    if 'segmentation' not in annotation:
        print(f"Error: Missing segmentation in annotation {annotation['id']}")
    elif not annotation['segmentation']:
        print(f"Error: Empty segmentation in annotation {annotation['id']}")



    print("\n")


```

no missing bounding boxes or empty segmentations. The absence of output suggests that the conditions specified in the code were not met for any of the annotations in dataset.

```
# Simulate a case where 'bbox' is missing in one annotation
annotations[0].pop('bbox', None)

# Check for missing values in annotations
for annotation in annotations:
    if 'bbox' not in annotation:
        print(f"Error: Missing bounding box in annotation {annotation['id']}")


    if 'segmentation' not in annotation:
        print(f"Error: Missing segmentation in annotation {annotation['id']}")
    elif not annotation['segmentation']:
        print(f"Error: Empty segmentation in annotation {annotation['id']}")



    print("\n")

```

```
Error: Missing bounding box in annotation 102924
```

```
import os
import zipfile

# Path to the ZIP file
zip_file_path = '/content/data/train2014.zip'

# Directory where you want to extract the contents
extracted_dir = '/content/data/'

# Extract the ZIP file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extracted_dir)

# Check the extracted directory
extracted_train_dir = os.path.join(extracted_dir, 'train2014')

# List the contents of the extracted directory
extracted_files = os.listdir(extracted_train_dir)
print("Extracted Files:", extracted_files)
```

```
Extracted Files: ['COCO_train2014_000000298031.jpg', 'COCO_train2014_000000278776.jpg', 'COCO_train2014_000000019964.jpg', 'COCO_train20
```

```
import tensorflow as tf
import os

# Define parameters
batch_size = 32
img_height = 180
img_width = 180
train_dir = '/content/data/train2014'

```

```
import zipfile
import os
import tensorflow as tf

# Path to the ZIP file
zip_file_path = '/content/drive/MyDrive/test2014.zip'

# Directory where you want to extract the contents
extracted_dir = '/content/data/test2014/'

# Extract the ZIP file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extracted_dir)

# Check the extracted directory
extracted_test_dir = os.path.join(extracted_dir, 'test2014')

# Create a dataset using list_files
test_file_pattern = os.path.join(extracted_test_dir, '*.jpg')
test_file_list_ds = tf.data.Dataset.list_files(test_file_pattern, shuffle=False)

# Print the first few elements of the dataset
for file_path in test_file_list_ds.take(5):
    print(f"Test File: {file_path.numpy().decode('utf-8')}")

```

```
Test File: /content/data/test2014/test2014/COCO_test2014_000000000001.jpg
Test File: /content/data/test2014/test2014/COCO_test2014_000000000014.jpg
Test File: /content/data/test2014/test2014/COCO_test2014_000000000016.jpg
Test File: /content/data/test2014/test2014/COCO_test2014_000000000027.jpg
Test File: /content/data/test2014/test2014/COCO_test2014_000000000057.jpg
```

```
# List the contents of the directory
files = os.listdir(train_dir)

# Print the first few files for inspection
print("Files in the directory:")
print(files[:5])

# Count the total number of files
num_files = len(files)
print(f"Total number of files: {num_files}")
```

```
Files in the directory:
['COCO_train2014_000000298031.jpg', 'COCO_train2014_000000278776.jpg', 'COCO_train2014_000000019964.jpg', 'COCO_train2014_000000578819.j
Total number of files: 82783
```

```
import os

# Specify the path to the training directory
train_dir = '/content/data/train2014'

# List a few files in the directory
sample_files = os.listdir(train_dir)[:5]

# Print the names of the sample files
print("Sample files:")
for file_name in sample_files:
    print(file_name)

```

```
Sample files:
COCO_train2014_000000298031.jpg
COCO_train2014_000000278776.jpg
COCO_train2014_000000019964.jpg
COCO_train2014_000000578819.jpg
COCO_train2014_000000184718.jpg
```

```
import tensorflow as tf

# Define parameters
batch_size = 32
img_height = 180
img_width = 180
train_dir = '/content/data/train2014'

```

```
import os
# Create a dataset using list_files
file_pattern = os.path.join(train_dir, '*.jpg')
file_list_ds = tf.data.Dataset.list_files(file_pattern, shuffle=False)

```

```

# Create a dataset using list_files
file_pattern = os.path.join(train_dir, '*.jpg')
file_list_ds = tf.data.Dataset.list_files(file_pattern, shuffle=False)

# Print the first few elements of the dataset
for file_path in file_list_ds.take(5):
    print(f"File: {file_path.numpy().decode('utf-8')}")

# Print the total number of files in the dataset
num_files = tf.data.experimental.cardinality(file_list_ds).numpy()
print(f"Total number of files: {num_files}")
```

```
File: /content/data/train2014/COCO_train2014_000000000009.jpg
File: /content/data/train2014/COCO_train2014_000000000025.jpg
File: /content/data/train2014/COCO_train2014_000000000030.jpg
File: /content/data/train2014/COCO_train2014_000000000034.jpg
File: /content/data/train2014/COCO_train2014_000000000036.jpg
Total number of files: 82783
```

```
# Get the list of image file paths
file_paths = [os.path.join(train_dir, file) for file in os.listdir(train_dir) if file.endswith('.jpg')]

```

```
import os

# Define the directory path
train_dir = '/content/data/train2014'  # Update this with the actual path to your training data

# List a few file paths
for root, dirs, files in os.walk(train_dir):
    for file in files[:5]:  # Print the first 5 files for inspection
        file_path = os.path.join(root, file)
        print("File:", file_path)

```

```
File: /content/data/train2014/COCO_train2014_000000298031.jpg
File: /content/data/train2014/COCO_train2014_000000278776.jpg
File: /content/data/train2014/COCO_train2014_000000019964.jpg
File: /content/data/train2014/COCO_train2014_000000578819.jpg
File: /content/data/train2014/COCO_train2014_000000184718.jpg
```

```
import tensorflow as tf
import os

# Define parameters
batch_size = 32
img_height = 180
img_width = 180
train_dir = '/content/data/train2014'  # Update this with the actual path to your training data

# List file paths and corresponding labels
file_paths = []
labels = []

for root, dirs, files in os.walk(train_dir):
    for file in files:
        file_paths.append(os.path.join(root, file))
        labels.append(os.path.basename(root))

# Create a dataset from the file paths and labels
dataset = tf.data.Dataset.from_tensor_slices((file_paths, labels))

# Print a few file paths and labels
for file_path, label in dataset.take(5):
    print("File Path:", file_path.numpy().decode('utf-8'))
    print("Label:", label.numpy().decode('utf-8'))
    print("\n")

# Note: You may need to customize the parsing of labels based on your directory structure

```

```
File Path: /content/data/train2014/COCO_train2014_000000298031.jpg
Label: train2014


File Path: /content/data/train2014/COCO_train2014_000000278776.jpg
Label: train2014


File Path: /content/data/train2014/COCO_train2014_000000019964.jpg
Label: train2014


File Path: /content/data/train2014/COCO_train2014_000000578819.jpg
Label: train2014


File Path: /content/data/train2014/COCO_train2014_000000184718.jpg
Label: train2014
```

```
def load_and_preprocess_image(file_path, label):
    # Load and decode the image
    img = tf.io.read_file(file_path)
    img = tf.image.decode_jpeg(img, channels=3)

    # Resize the image
    img = tf.image.resize(img, [img_height, img_width])

    # Normalize the pixel values to be in the range [0, 1]
    img = img / 255.0

    return img, label

# Apply the load_and_preprocess_image function to the dataset
dataset = dataset.map(load_and_preprocess_image)

# Example: Print the first 5 images and labels
for image, label in dataset.take(5):
    print("Image Shape:", image.shape)
    print("Label:", label.numpy().decode('utf-8'))
    print("\n")

```

```
Image Shape: (180, 180, 3)
Label: train2014


Image Shape: (180, 180, 3)
Label: train2014


Image Shape: (180, 180, 3)
Label: train2014


Image Shape: (180, 180, 3)
Label: train2014


Image Shape: (180, 180, 3)
Label: train2014
```

```
# Shuffle and split the dataset into training and validation sets
dataset_size = len(file_paths)
train_size = int(0.8 * dataset_size)
val_size = dataset_size - train_size

train_dataset = dataset.take(train_size)
val_dataset = dataset.skip(train_size)

# Batch the datasets
train_dataset = train_dataset.batch(batch_size)
val_dataset = val_dataset.batch(batch_size)

# Prefetch the datasets for better performance
train_dataset = train_dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
val_dataset = val_dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)

# Example: Print the first batch of images and labels from the training set
for images, labels in train_dataset.take(1):
    print("Batch Shape:", images.shape)
    print("Labels:", labels.numpy())

```

```
Batch Shape: (32, 180, 180, 3)
Labels: [b'train2014' b'train2014' b'train2014' b'train2014' b'train2014'
 b'train2014' b'train2014' b'train2014' b'train2014' b'train2014'
 b'train2014' b'train2014' b'train2014' b'train2014' b'train2014'
 b'train2014' b'train2014' b'train2014' b'train2014' b'train2014'
 b'train2014' b'train2014' b'train2014' b'train2014' b'train2014'
 b'train2014' b'train2014' b'train2014' b'train2014' b'train2014'
 b'train2014' b'train2014']
```

```
import pandas as pd
```

```
data_dir = '/content/data/train2014'
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

```

```
image_size = (224, 224)

```

```
# Create a list of all image files in the directory
file_list = [os.path.join(data_dir, f) for f in os.listdir(data_dir) if f.endswith(('.jpg', '.jpeg', '.png'))]

# Create a DataFrame with the file paths
df = pd.DataFrame({'file_path': file_list})

# Create a custom generator using flow_from_dataframe
datagen = ImageDataGenerator(rescale=1./255, rotation_range=20)
generator = datagen.flow_from_dataframe(
    df,
    x_col='file_path',
    target_size=image_size,
    batch_size=batch_size,
    class_mode=None  # No labels, as there are no classes
)

# Check if images are found
if len(generator) > 0:
    print(f"Found {len(generator)} images.")
else:
    print("No images found.")
```

```
Found 82783 validated image filenames.
Found 2587 images.
```

Double-click (or enter) to edit

```
generator = datagen.flow_from_dataframe(
    df,
    x_col='file_path',
    target_size=image_size,
    batch_size=batch_size,
    class_mode=None,  # No labels, as there are no classes
    validation_split=0.2  # Specify the validation split
)

```

```
Found 82783 validated image filenames.
```

```
import os

# Directory containing training images
train_dir = '/content/data/train2014'

# List the contents of the directory
files = os.listdir(train_dir)

# Print the first few files for inspection
print("Files in the directory:")
print(files[:5])

# Count the total number of files
num_files = len(files)
print(f"Total number of files: {num_files}")

```

```
Files in the directory:
['COCO_train2014_000000298031.jpg', 'COCO_train2014_000000278776.jpg', 'COCO_train2014_000000019964.jpg', 'COCO_train2014_000000578819.j
Total number of files: 82783
```

```
import os

# Specify the directory path
train_dir = '/content/data/train2014'

# List a few files in the directory
files = os.listdir(train_dir)
print("Files in the directory:")
print(files[:5])

```

```
Files in the directory:
['COCO_train2014_000000298031.jpg', 'COCO_train2014_000000278776.jpg', 'COCO_train2014_000000019964.jpg', 'COCO_train2014_000000578819.j
```

```
import os

# Specify the directory path
train_dir = '/content/data/train2014'

# Get the actual class names from subdirectories
class_names = os.listdir(train_dir)
print("Actual Class Names:", class_names)

```

```
Actual Class Names: ['COCO_train2014_000000298031.jpg', 'COCO_train2014_000000278776.jpg', 'COCO_train2014_000000019964.jpg', 'COCO_trai
```

```
import os

train_dir = '/content/data/train2014/'
files = os.listdir(train_dir)
print("Files in the directory:")
print(files[:5])

```

```
Files in the directory:
['COCO_train2014_000000298031.jpg', 'COCO_train2014_000000278776.jpg', 'COCO_train2014_000000019964.jpg', 'COCO_train2014_000000578819.j
```

## Data Augmentation

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define data augmentation parameters
datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Create a custom generator using flow_from_directory
train_datagen = datagen.flow_from_directory(
    train_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',  # Change to 'categorical' if you have multiple classes
    shuffle=True,
    seed=123
)


```

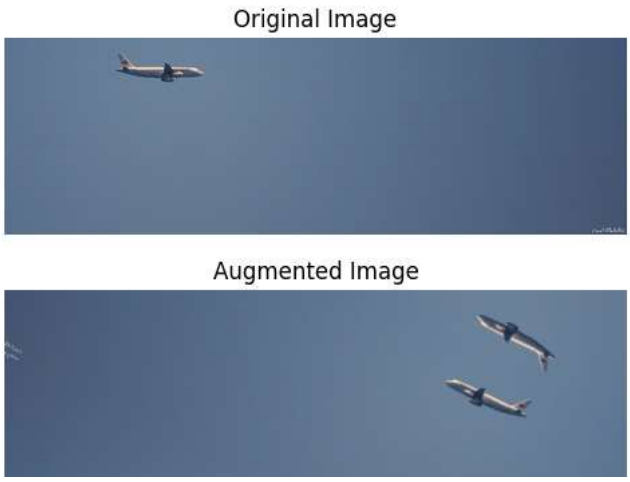
```
Found 0 images belonging to 0 classes.
```

```
import tensorflow as tf
```

```
import tensorflow as tf

# Define data augmentation parameters
def augment_image(img):
    img = tf.image.random_flip_left_right(img)
    img = tf.image.random_flip_up_down(img)
    img = tf.image.rot90(img, tf.random.uniform(shape=[], minval=0, maxval=4, dtype=tf.int32))  # Random rotation

    # Random zoom
    scale = tf.random.uniform(shape=[], minval=0.9, maxval=1.1, dtype=tf.float32)
    img_shape = tf.shape(img)
    new_height = tf.cast(tf.cast(img_shape[0], dtype=tf.float32) * scale, dtype=tf.int32)
    new_width = tf.cast(tf.cast(img_shape[1], dtype=tf.float32) * scale, dtype=tf.int32)
    img = tf.image.resize(img, [new_height, new_width])

    img = tf.image.random_brightness(img, 0.1)
    img = tf.image.random_contrast(img, 0.8, 1.2)
    return img

# Create a dataset using list_files
train_file_pattern = '/content/data/train2014/*.jpg'  # Adjust the path pattern as needed
file_list_ds = tf.data.Dataset.list_files(train_file_pattern, shuffle=False)

# Define a function to load and augment images
def process_path(file_path):
    img = tf.io.read_file(file_path)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, [img_height, img_width])
    img = augment_image(img)
    img = img / 255.0  # Rescale to [0,1]
    return img

# Map the process_path function to the dataset
train_ds = file_list_ds.map(process_path)

# Display the augmented images
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for i in train_ds.take(9):
    # Clip pixel values to the valid range [0, 1]
    images = tf.clip_by_value(images, 0, 1)
    plt.subplot(3, 3, i + 1)
    plt.imshow(images.numpy())
    plt.axis("off")

plt.show()

```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-41-622a2505932b> in <cell line: 40>()
     41     # Clip pixel values to the valid range [0, 1]
     42     images = tf.clip_by_value(images, 0, 1)
---> 43     plt.subplot(3, 3, i + 1)
     44     plt.imshow(images.numpy())
     45     plt.axis("off")

                                  1 frames
/usr/local/lib/python3.10/dist-packages/matplotlib/gridspec.py in _from_subplot_args(figure, args)
    596         else:
    597             if not isinstance(num, Integral) or num < 1 or num > rows*cols:
--> 598                 raise ValueError(
    599                     f"num must be an integer with 1 <= num <= {rows*cols}, "
    600                     f"not {num!r}"

ValueError: num must be an integer with 1 <= num <= 9, not <tf.Tensor: shape=(175, 175, 3), dtype=float32, numpy=
array([[[0.9290544 , 1.0468547 , 1.4414327 ],
        [0.92647123, 1.0421069 , 1.4487634 ],
        [0.9267137 , 1.050411  , 1.4653924 ],
        ...,
        [0.96778655, 0.9619597 , 0.99505687],
        [0.95310915, 0.9503184 , 0.98406684],
        [0.94000185, 0.94984716, 0.9752774 ]],

       [[0.9270664 , 1.0448173 , 1.4585187 ],
        [0.9265282 , 1.0548129 , 1.4665492 ],
        [0.9278304 , 1.0575358 , 1.4692636 ],
        ...,
        [1.0016737 , 1.0187608 , 0.9948099 ],
        [0.99817264, 1.0055786 , 1.0150816 ],
        [0.98217905, 0.97331876, 1.0157716 ]],

       [[0.9268832 , 1.0469283 , 1.4720718 ],
        [0.92667776, 1.0536456 , 1.4672486 ],
        [0.9278695 , 1.067265  , 1.4837327 ],
        ...,
        [1.0530446 , 1.0135752 , 0.98700446],
        [1.0490112 , 1.0287253 , 1.001697  ],
```

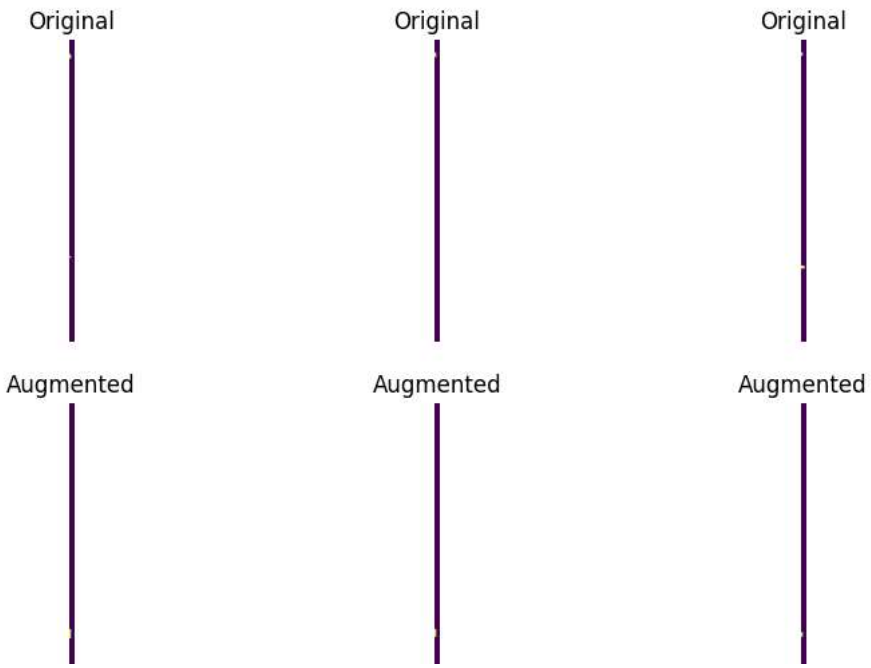
```
 1 import tensorflow as tf
 2 from tensorflow import keras
 3 from tensorflow.keras.layers.experimental import preprocessing
 4 import matplotlib.pyplot as plt
 5
 6 # Load an example image from the dataset
 7 image_path = "/content/data/train2014/COCO_train2014_000000007489.jpg"  # Replace with the path to your image
 8 img = tf.keras.preprocessing.image.load_img(image_path)
 9 img_array = tf.keras.preprocessing.image.img_to_array(img)
10 img_array = tf.expand_dims(img_array, 0)  # Create batch dimension
11
12 # Display the original image
13 plt.figure(figsize=(6, 6))
14 plt.imshow(img_array[0] / 255.0)  # Normalize pixel values to [0, 1]
15 plt.title("Original Image")
16 plt.axis("off")
17 plt.show()
18
19 # Define data augmentation layers
20 data_augmentation = keras.Sequential(
21     [
22         preprocessing.RandomFlip("horizontal"),
23         preprocessing.RandomRotation(0.1),
24         preprocessing.RandomZoom(0.1),
25     ]
26 )
27
28 # Apply data augmentation to the image
29 augmented_img = data_augmentation(img_array)
30
31 # Display the augmented image
32 plt.figure(figsize=(6, 6))
33 plt.imshow(augmented_img[0] / 255.0)  # Normalize pixel values to [0, 1]
34 plt.title("Augmented Image")
35 plt.axis("off")
36 plt.show()
37
```




```
# input image loaded with TensorFlow
img_rescaled = tf.image.convert_image_dtype(img, dtype=tf.float32) / 255.0

```

```
import tensorflow as tf

# images is your batch of images
images_rescaled = images / 255.0


```

```
import matplotlib.pyplot as plt

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

# Iterate over the dataset
for images in train_ds.take(1):
    # Apply data augmentation to the images
    augmented_images = data_augmentation(images)

    # Display the original and augmented images side by side
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)

        # Display the original image
        if i < 3:
            plt.imshow(images[i].numpy().astype("uint8"))
            plt.title(f"Original")

        # Display the augmented image
        else:
            plt.imshow(augmented_images[i - 3].numpy().astype("uint8"))
            plt.title("Augmented")

        plt.axis("off")

plt.show()

```




```
import tensorflow as tf
from tensorflow.keras.preprocessing import image_dataset_from_directory

batch_size = 32
img_height = 180
img_width = 180

# Update with the correct path in Colab
train_dir = '/content/'

train_ds = image_dataset_from_directory(
    train_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size
)

val_ds = image_dataset_from_directory(
    train_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size
)

```

```
Found 124134 files belonging to 5 classes.
Using 99308 files for training.
Found 124178 files belonging to 5 classes.
Using 24835 files for validation.
```

```
# Load COCO dataset annotations
captions_train_path = "/content/annotations/captions_train2014.json"
captions_val_path = "/content/annotations/captions_val2014.json"
instances_train_path = "/content/annotations/instances_train2014.json"
instances_val_path = "/content/annotations/instances_val2014.json"
keypoints_train_path = "/content/annotations/person_keypoints_train2014.json"
keypoints_val_path = "/content/annotations/person_keypoints_val2014.json"
test_data="/content/test2014"
```

```
import json
from pprint import pprint

# Load COCO dataset annotations
def load_coco_annotations(file_path):
    with open(file_path, 'r') as file:
        annotations = json.load(file)
    return annotations

captions_train_annotations = load_coco_annotations(captions_train_path)
captions_val_annotations = load_coco_annotations(captions_val_path)
instances_train_annotations = load_coco_annotations(instances_train_path)
instances_val_annotations = load_coco_annotations(instances_val_path)
keypoints_train_annotations = load_coco_annotations(keypoints_train_path)
keypoints_val_annotations = load_coco_annotations(keypoints_val_path)

# Print some information about the loaded annotations
pprint(captions_train_annotations['info'])
pprint(captions_train_annotations['licenses'])
pprint(captions_train_annotations['images'][:5])  # Print information about the first 5 images
pprint(captions_train_annotations['annotations'][:5])  # Print information about the first 5 annotations

```

```
{'contributor': 'COCO Consortium',
 'date_created': '2017/09/01',
 'description': 'COCO 2014 Dataset',
 'url': 'http://cocodataset.org',
 'version': '1.0',
 'year': 2014}
[{'id': 1,
  'name': 'Attribution-NonCommercial-ShareAlike License',
  'url': 'http://creativecommons.org/licenses/by-nc-sa/2.0/'},
 {'id': 2,
  'name': 'Attribution-NonCommercial License',
  'url': 'http://creativecommons.org/licenses/by-nc/2.0/'},
 {'id': 3,
  'name': 'Attribution-NonCommercial-NoDerivs License',
  'url': 'http://creativecommons.org/licenses/by-nc-nd/2.0/'},
 {'id': 4,
  'name': 'Attribution License',
  'url': 'http://creativecommons.org/licenses/by/2.0/'},
 {'id': 5,
  'name': 'Attribution-ShareAlike License',
  'url': 'http://creativecommons.org/licenses/by-sa/2.0/'},
 {'id': 6,
  'name': 'Attribution-NoDerivs License',
  'url': 'http://creativecommons.org/licenses/by-nd/2.0/'},
 {'id': 7,
  'name': 'No known copyright restrictions',
  'url': 'http://flickr.com/commons/usage/'},
 {'id': 8,
  'name': 'United States Government Work',
  'url': 'http://www.usa.gov/copyright.shtml'}]
[{'coco_url': 'http://images.cocodataset.org/train2014/COCO_train2014_000000057870.jpg',
  'date_captured': '2013-11-14 16:28:13',
  'file_name': 'COCO_train2014_000000057870.jpg',
  'flickr_url': 'http://farm4.staticflickr.com/3153/2970773875_164f0c0b83_z.jpg',
  'height': 480,
  'id': 57870,
  'license': 5,
  'width': 640},
 {'coco_url': 'http://images.cocodataset.org/train2014/COCO_train2014_000000384029.jpg',
  'date_captured': '2013-11-14 16:29:45',
  'file_name': 'COCO_train2014_000000384029.jpg',
  'flickr_url': 'http://farm3.staticflickr.com/2422/3577229611_3a3235458a_z.jpg',
  'height': 429,
  'id': 384029,
  'license': 5,
  'width': 640},
 {'coco_url': 'http://images.cocodataset.org/train2014/COCO_train2014_000000222016.jpg',
  'date_captured': '2013-11-14 16:37:59',
  'file_name': 'COCO_train2014_000000222016.jpg',
  'flickr_url': 'http://farm2.staticflickr.com/1431/1118526611_09172475e5_z.jpg',
  'height': 640,
  'id': 222016,
  'license': 1,
  'width': 480},
 {'coco_url': 'http://images.cocodataset.org/train2014/COCO_train2014_000000520950.jpg',
  'date_captured': '2013-11-14 16:44:40',
  'file_name': 'COCO_train2014_000000520950.jpg',
  'flickr_url': 'http://farm8.staticflickr.com/7007/6413705793_1c391cd697_z.jpg',
```

```
import matplotlib.pyplot as plt

# Assuming class_names represent the COCO captions
class_names = ['A very clean and well decorated empty bathroom',
               'A panoramic view of a kitchen and all of its appliances.',
               'A blue and white bathroom with butterfly themed wall tiles.',
               'A panoramic photo of a kitchen and dining room',
               'A graffiti-ed stop sign across the street from a red car']

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
plt.show()

```

```
import tensorflow as tf
from tensorflow.keras import layers

# Assuming img_height and img_width are defined
img_height = 180
img_width = 180

data_augmentation = tf.keras.Sequential(
    [
        layers.experimental.preprocessing.RandomFlip("horizontal", input_shape=(img_height, img_width, 3)),
        layers.experimental.preprocessing.RandomRotation(0.1),
        layers.experimental.preprocessing.RandomZoom(0.1),
    ]
)
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```

```
normalization_layer = layers.experimental.preprocessing.Rescaling(1./255)
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
```

```
1 labels_batch
2
```

```
<tf.Tensor: shape=(32,), dtype=int32, numpy=
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2], dtype=int32)>
```

```
1 image_batch
```

```
<tf.Tensor: shape=(32, 180, 180, 3), dtype=float32, numpy=
array([[[[4.24107373e-01, 2.82930911e-01, 2.35872075e-01],
         [4.29911077e-01, 2.88734615e-01, 2.41675764e-01],
         [4.38722491e-01, 2.97546029e-01, 2.44175151e-01],
         ...,
         [5.57690203e-01, 5.24295747e-01, 5.61662793e-01],
         [6.77975953e-01, 6.38760209e-01, 6.78139269e-01],
         [3.09503466e-01, 3.26660544e-01, 3.21616143e-01]],

        [[4.25708085e-01, 2.84531623e-01, 2.37472787e-01],
         [4.43137288e-01, 3.01960796e-01, 2.54901975e-01],
         [4.48484063e-01, 3.07307571e-01, 2.52405614e-01],
         ...,
         [4.39167589e-01, 4.06959653e-01, 4.85228032e-01],
         [4.03131634e-01, 3.67837518e-01, 4.43926990e-01],
         [1.83105856e-01, 2.06000030e-01, 2.27241635e-01]],

        [[4.23478007e-01, 2.82301515e-01, 2.35242695e-01],
         [4.39215720e-01, 2.98039228e-01, 2.50980407e-01],
         [4.50163424e-01, 3.08986932e-01, 2.56015509e-01],
         ...,
         [5.24977922e-01, 5.04568577e-01, 5.87629437e-01],
         [5.23447096e-01, 4.92337883e-01, 5.73900938e-01],
         [3.21302176e-01, 3.55725050e-01, 3.46580505e-01]],

        ...,

        [[4.08657163e-01, 3.55056375e-01, 4.05386180e-01],
         [4.21151161e-01, 3.70933354e-01, 4.15668160e-01],
         [3.99945468e-01, 3.46980155e-01, 3.85729820e-01],
         ...,
         [2.98653185e-01, 2.55515903e-01, 3.10417891e-01],
         [3.05110455e-01, 2.67882854e-01, 3.28658074e-01],
         [3.09531569e-01, 2.75000006e-01, 3.19438398e-01]],

        [[3.80282998e-01, 3.33224177e-01, 3.72439861e-01],
         [4.21949834e-01, 3.82407427e-01, 4.14106697e-01],
         [3.80501032e-01, 3.32570761e-01, 3.66339833e-01],
         ...,
         [3.09640586e-01, 2.86111176e-01, 3.33169997e-01],
         [3.38126183e-01, 2.97358334e-01, 3.59313637e-01],
         [3.31962734e-01, 3.00590187e-01, 3.51570576e-01]],

        [[3.62850994e-01, 3.12285215e-01, 3.58069986e-01],
         [3.64143431e-01, 3.24927747e-01, 3.60221863e-01],
         [3.65898162e-01, 3.17967832e-01, 3.51736933e-01],
         ...,
         [2.86598444e-01, 2.55225867e-01, 3.04675013e-01],
         [2.99700499e-01, 2.58524269e-01, 3.12772542e-01],
         [3.09692234e-01, 2.80129254e-01, 3.30204844e-01]]],


       [[[5.68845332e-01, 3.72585356e-01, 1.78467691e-01],
         [5.69063246e-01, 3.38888913e-01, 2.97167778e-01],
         [5.54575205e-01, 3.36601347e-01, 3.01234573e-01],
         ...,
         [1.19316280e-01, 1.37690887e-01, 7.99913183e-02],
         [1.80501103e-01, 1.18191727e-01, 6.33986965e-02],
```

## Extracting the feature vector from all images

Load Pre-trained Model:

```
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.models import Model

# Load the MobileNetV2 model pre-trained on ImageNet data
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(img_height, img_width, 3))

# Choose the layer from which you want to extract features
feature_extractor_layer = base_model.get_layer('out_relu')

# Create a new model with the desired output layer
feature_extractor = Model(inputs=base_model.input, outputs=feature_extractor_layer.output)

```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input shape (224,
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_
9406464/9406464 [==============================] - 0s 0us/step
```

## Preprocess and Extract Features:

```
def extract_features(image_path):
    # Load and preprocess the image
    img = tf.keras.preprocessing.image.load_img(image_path, target_size=(img_height, img_width))
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0)
    img_array = preprocess_input(img_array)

    # Extract features using the pre-trained model
    features = feature_extractor.predict(img_array)

    return features.flatten()

# usage with COCO dataset paths
image_paths = [
    "/content/data/train2014/COCO_train2014_000000057870.jpg",
    "/content/data/train2014/COCO_train2014_000000384029.jpg",
    "/content/data/train2014/COCO_train2014_000000222016.jpg",

]

for image_path in image_paths:
    features = extract_features(image_path)
    print(f"Image: {image_path}, Feature vector shape: {features.shape}")
```

```
1/1 [==============================] - 2s 2s/step
Image: /content/data/train2014/COCO_train2014_000000057870.jpg, Feature vector shape: (46080,)
1/1 [==============================] - 0s 98ms/step
Image: /content/data/train2014/COCO_train2014_000000384029.jpg, Feature vector shape: (46080,)
1/1 [==============================] - 0s 88ms/step
Image: /content/data/train2014/COCO_train2014_000000222016.jpg, Feature vector shape: (46080,)
```

# Loading dataset for Training the model

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
# Define parameters for data augmentation and normalization
batch_size = 32
img_height = 180
img_width = 180
```

```
# Create an ImageDataGenerator for data augmentation and normalization
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Create the training dataset
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical'  # Assuming you have multiple classes
)
```

```
Found 124134 images belonging to 5 classes.
```

```
# Create the validation dataset
val_dir = '/content/data/test2014'

val_datagen = ImageDataGenerator(rescale=1./255)  # No augmentation for validation data
val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical'
)
```

```
Found 40775 images belonging to 1 classes.
```

```
# training dataset is in the 'train2014' directory
train_dir = '/content/data/train2014'


# test dataset is in the 'test2014' directory
test_dir = '/content/data/test2014'
```

```
image_size = (180, 180)  # Adjust according to your model's input size

```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers

```

```
model = Sequential([
    data_augmentation,
    layers.experimental.preprocessing.Rescaling(1./255),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(6, activation='softmax')
])

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

```

## Inspecting the Model

```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_2 (Sequential)   (None, 180, 180, 3)       0

 rescaling_1 (Rescaling)     (None, 180, 180, 3)       0

 conv2d (Conv2D)             (None, 180, 180, 16)      448

 max_pooling2d (MaxPooling2D  (None, 90, 90, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 45, 45, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 22, 22, 64)       0
 2D)

 dropout (Dropout)           (None, 22, 22, 64)        0

 flatten (Flatten)           (None, 30976)             0

 dense (Dense)               (None, 128)               3965056

 dense_1 (Dense)             (None, 6)                 774

=================================================================
Total params: 3,989,414
Trainable params: 3,989,414
Non-trainable params: 0
_________________________________________________________________
```

## Tokenizing the vocabulary

```
import json

# Load annotations from captions validation file
captions_val_path = "/content/annotations/captions_val2014.json"
with open(captions_val_path, "r") as captions_val_file:
    captions_val_data = json.load(captions_val_file)

```

```

# Load annotations from instances validation file
instances_val_path = "/content/annotations/instances_val2014.json"
with open(instances_val_path, "r") as instances_val_file:
    instances_val_data = json.load(instances_val_file)

```

```
# Load annotations from keypoints validation file
keypoints_val_path = "/content/annotations/person_keypoints_val2014.json"
with open(keypoints_val_path, "r") as keypoints_val_file:
    keypoints_val_data = json.load(keypoints_val_file)
```

```
# Merge the validation data as needed
val_data = {
    "captions": captions_val_data,
    "instances": instances_val_data,
    "keypoints": keypoints_val_data
}
```

```
val_data
```

```
import json

# Load annotations from captions file
with open("/content/annotations/captions_train2014.json", "r") as captions_file:
    captions_data = json.load(captions_file)

# Load annotations from instances file
with open("/content/annotations/instances_train2014.json", "r") as instances_file:
    instances_data = json.load(instances_file)

# Load annotations from keypoints file
with open("/content/annotations/person_keypoints_train2014.json", "r") as keypoints_file:
    keypoints_data = json.load(keypoints_file)

```

```
# Access the merged data
train_data = {
    "captions": captions_data,
    "instances": instances_data,
    "keypoints": keypoints_data
}
```