Action recognition in KTH dataset using CNN Features

PREPARED BY

Prachi Khandelwal

From JK Lakshmipat University

FACULTY GUIDE

DR. SAKTHI BALAN

Dr. Arpan Gupta

Department of Computer Science Engineering
Institute of Engineering and Technology (IET)
LNM Institute of Information and Technology

CERTIFICATE

This is to certify that the Practice School-1 project work entitled "**Action recognition in KTH dataset using CNN Features**" Under LUSIP submitted by **Prachi Khandelwal** towards the partial fulfilment of the requirements for the degree of **Bachelor of Computer Application** of JK Lakshmipat University Jaipur is the record of work carried out by them under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for Practice School-II examination.

---------------------------------- --------------------------------------

Dr. Arpan Gupta	Dr. Sakthi Balan
Supervisor/Guide and Assistant Professor,	External Supervisor and Associate Professor
Department of CS Engineering	LNMIIT University Jaipur
Institute of Engineering & Technology (IET)	
JK Lakshmipat University Jaipur	

Date of Submission 02/08/2023

ACKNOWLEDGEMENTS

We would like to express my heartfelt gratitude to Dr. Sakthi Balan, Associate Professor, Department of Computer Science & Engineering for his valuable time and guidance that made the project work a success. I would also like to thank my mentor Dr. Arpan Gupta for his valuable input in the project work.

My heartfelt thanks to Dr. Arpan Gupta, Assistant Professor for his support in providing us the data for the project. Thanking all my friends and all those who have helped me in carrying out this work directly or indirectly without whom completion of this project work would not have been possible.

Sincerely yours,
Prachi Khandelwal

ABSTRACT

Human activity recognition is an important area of machine learning research as it has many utilizations in different areas such as sports training, security, entertainment, ambient-assisted living, and health monitoring and management. Studying human activity recognition shows that researchers are interested mostly in the daily activities of the human. Therefore, the general architecture of HAR system is presented in this paper, along with the description of its main components. The state of the art in human activity recognition based on accelerometer is surveyed. According to this survey, most of the research recently used deep learning for recognizing HAR, but they focused on CNN even though there are other deep learning types achieved a satisfied accuracy. The paper displays a two-level taxonomy in accordance with machine learning approach (either traditional or deep learning) and the processing mode (either online or offline). Forty-eight studies are compared in terms of recognition accuracy, classifier, activity types, and used devices. Finally, the paper concludes different challenges and issues online versus offline also using deep learning versus traditional machine learning for human activity recognition based on accelerometer sensors.

CONTENT

CHAPTER 1: INTRODUCTION

Human activity recognition using smartphone sensors like accelerometer is one of the hectic topics of research. HAR is one of the time series classification problems. In this project various machine learning and deep learning models have been worked out to get the best result. In the same sequence, we are using model of the Recurrent Neural Network (RNN) to recognize various activities of humans like standing, climbing upstairs and downstairs etc.

Human Activity Recognition dataset can be downloaded from the link given below: HAR dataset.

Activities:

-Walking

-Upstairs

-Downstairs

-Sitting

-Standing

Accelerometers detect magnitude and direction of the proper acceleration, as a vector quantity, and can be used to sense orientation (because direction of weight changes). Gyroscope maintains orientation along a axis so that the orientation is unaffected by tilting or rotation of the mounting, according to the conservation of angular momentum.

Understanding the dataset:

-Both the sensors generate data in 3D space over time.

-('XYZ' represents 3-axial signals in X, Y, and Z directions.)

The available data is pre-processed by applying noise filters and then sampled in fixed-width windows ie., each window has 128 readings.

Train and Test data were separated as

The readings from 80% of the volunteers were taken as training data and remaining 20% volunteer's records were taken for test data. All the data is present in the folder downloaded using the link provided above Phases.

-Choosing a dataset

-Uploading the dataset in the drive to work on google Collaboratory

-Dataset cleaning and data Preprocessing

-Choosing a model and building deep learned network model

-Exporting in Android Studio.

The IDE used for this project is Google Collaboratory which is the best of the times to deal with deep learning projects. Phase 1 was explained above as from where the dataset is downloaded. In this sequence to start with the project open a new notebook in Google Collaboratory first import all the necessary libraries.

Code: Importing Libraries

1.1 Objectives

Human activity recognition (HAR) aims to classify a person's actions from a series of measurements captured by sensors.
Nowadays, collecting this type of data is not an arduous task. With the growth of the Internet of Things, almost everyone has some gadget that monitors their movements. It can be a smartwatch, a pulsometer, or even a smartphone.
Usually, this is performed following a fixed length sliding window approach for feature extraction. Here two parameters need to be fixed: the window size and the shift.
These are some of the data you could use:
Body acceleration.
Gravity acceleration.
Body angular speed.
Body angular acceleration.
Etc.
The machine learning model used for activity recognition relies on top of the devices' available sensors.
However, analyzing this data can be a big challenge. Indeed, human activities are complex, and there are differences between individuals.

1.2 Benefits

Activity recognition is the basis for the development of many potential applications in health, wellness, or sports:

MONITOR HEALTH
Analyze the activity of a person from the information collected by different devices.

DISCOVER ACTIVITY PATTERNS
Discover which are the variables that determine which activity a person is doing.

DETECT ACTIVITY
Calculate a predictive model that can recognize a person's activity from the signals received by the sensors.

IMPROVE WELLBEING
Design individualized exercise tables to improve the health of a person.

1.3 Approach

Neural networks are the perfect algorithms to determine a person's physical activity. This is due to their ability to recognize the patterns behind the data.
The following graph illustrates a neural network that classifies different activities using smartphone data.

1.4 Conclusions

Human activity recognition has a wide range of uses because of its impact on wellbeing.
It is becoming a fundamental tool in healthcare solutions such as preventing obesity or caring for elderly persons.

CHAPTER 2: LITERATURE REVIEW

For many years human action recognition has been studied well. Most of the action recognition methods require to manually annotate the relevant portion of the action of interest in the video. In recent years it has been studied that the relevant portion of action of interest can be found out automatically and recognize the action. We can review the action recognition methods.

2.1 Action Recognition

For representing video, feature trajectories have shown efficiency. But the quality and quantity of these trajectories were not sufficient. As the use of dense sampling came popular for image classification Wang et al. [1] proposed to use dense trajectories for representing videos. Dense points from each frame are sampled and traced them based on displacement information. For improving the performance Wang et al. [2] considers the camera motion. The camera motion is estimated by matching feature points between the frames by using SURF descriptors and dense optical flow. Another approach [3] aimed at modeling the motion relationship. The approach operates on top of visual codewords derived from local patch trajectories, and therefore does not require accurate foreground-background separation.

. Dorr et al. [4] proposed another method for finding the informative regions. They used saliency mapping algorithms. As a new method this paper proposes using a joint learning framework for learning spatial and temporal extents of action of interest

2.2 Action Detection

Recognition was performed using the Mahala Nobis distance between the moment description of the input and each of the known actions. Recent popular methods which employ machine learning techniques such as SVMs and AdaBoost, provide one possibility for incorporating the information contained in a set of training examples.[4] introduces the Action MACH filter, a template-based method for action recognition which can capture intra- class variability by synthesizing a single Action.

Another method is proposed in [5], multiple-instance learning framework, named SMILE-SVM (Simulated annealing Multiple Instance Learning Support Vector Machines), is presented for learning human action detector based on imprecise action location. Wang et al. [6] used a figure-centric visual word representation. In that localization is treated as latent variable to recognize the action. A spatio-temporal model is learned. During the training [7] model parameters is estimated, and the relevant portion is identified.[8] proposed an independent motion evidence feature for distinguishing human actions from background motion.

Most of the methods require that the relevant portion of the video has to be annotated with bounding boxes. Human intervention was tedious. So, to overcome the bounding box Brendel et al. [9] divides the video into several subgroups and then a model was generated that identify the relevant subgroup.

This paper introduces a method that learns both spatial and temporal extents for detection improvement. Dense trajectory is used here as local features to represent the human action.

CHAPTER 3: DATASET

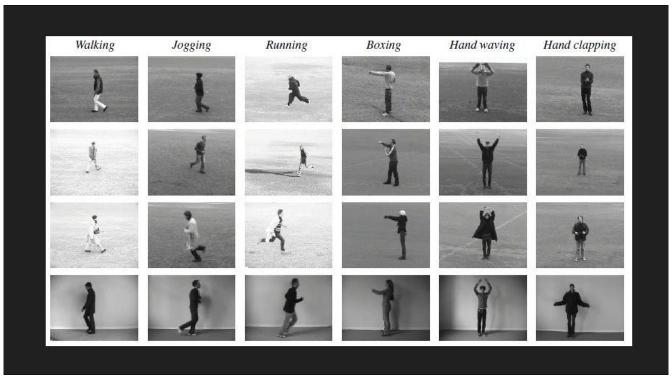
The KTH Actions dataset contains six actions (walk, jog, run, box, hand-wave, and handclap). It is one of the most standard datasets. Each action in the dataset is performed by 25 different individuals, and the setting is systematically altered for each action per actor. Setting variations include outdoor, outdoor with scale variation, outdoor with different clothes, and indoor (s4). These variations test the ability of algorithms to identify actions independent of the background, the appearance of the actors, and the scale of the actors.

3.1 PROBLEM STATEMENT

- **Human Action Recognition (HAR)** aims to understand human behaviour and assign a label to each action. It has a wide range of applications, and therefore has been attracting increasing attention in the field of computer vision. Human actions can be represented using various data modalities, such as RGB, skeleton, depth, infrared, point cloud, event stream, audio, acceleration, radar, and Wi-Fi signal, which encode different sources of useful yet distinct information and have various advantages depending on the application scenarios.
- Consequently, lots of existing works have attempted to investigate different types of approaches for HAR using various modalities.
- Your Task is to build an Image Classification Model using CNN that classifies to which class of activity a human is performing.

3.2 Data Overview

In the current video database, 25 subjects repeatedly performed six different human actions (walking, jogging, running, boxing, hand waving, and hand clapping) in four different scenarios: outdoors (s1), outdoors with scale variation (s2), outdoors with different clothes (s3), and indoors (s4), as shown below. There are now 2391 sequences in the database. With a static camera and a frame rate of 25 frames per second, all sequences were recorded over uniform backgrounds. The sequences, which were down sampled to a spatial resolution of 160x120 pixels, have an average runtime of four seconds.



CHAPTER 4: METHODOLOGY

4.1 OpenCV

OpenCV is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today’s systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as NumPy, python can process the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features.

The first OpenCV version was 1.0. OpenCV is released under a BSD license and hence it’s free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS, and Android. When OpenCV was designed the focus was real-time applications for computational efficiency. All things are written in optimized C/C++ to take advantage of multi-core processing.

4.2 OpenCV Functionality

- Image/video I/O, processing, display (core, imgproc, highgui)

-Object/feature detection (objdetect, features2d, nonfree)

-Geometry-based monocular or stereo computer vision (calib3d, stitching, videostab)

-Computational photography (photo, video, superres)

-Machine learning & clustering (ml, flann)

-CUDA acceleration (gpu)

4.3 Feature Extraction

The technique of automatically choosing or identifying the most significant and pertinent data or features from a dataset is known as feature extraction. The process of extracting patterns or features from photos or videos that can be used to identify or categorize objects, persons, or activities is known as feature extraction in the field of computer vision and image processing.

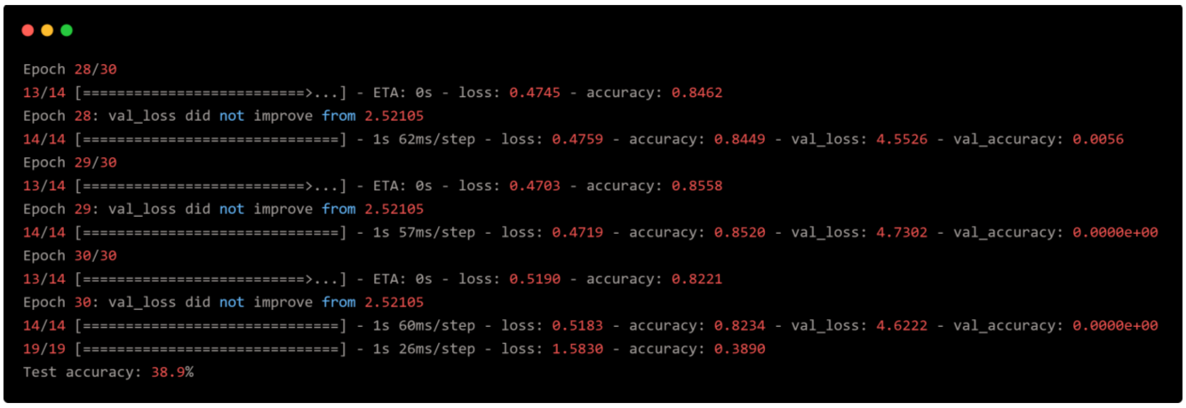
To find and extract the most pertinent information, feature extraction approaches often require processing the input data using various mathematical operations or algorithms. Edge detection, texture analysis, and color histograms, for instance, are frequently used feature extraction techniques in image processing.

In many applications of computer vision and machine learning, feature extraction is a crucial step. It can do this by making the supplied data's dimensions smaller.

CHAPTER 5: RESULTS

After using the KTH dataset to train the Conv CNN model and assessing its performance, we were able to achieve an accuracy of 38.9% on the test set. To evaluate the model, a set of previously unviewed video clips were used to gauge the model's accuracy in predicting the action categories.

The model's moderate performance in identifying human behaviors in videos was indicated by the accuracy of 38.9% that was ultimately reached. Although the accuracy is superior to guessing at random, it also emphasizes the complexity and difficulties that come with Human Activity Recognition (HAR) tasks. It is challenging to effectively capture the complex temporal patterns of actions across many movies due to the dataset's dynamic and diversified character, which includes a variety of human behaviors carried out by distinct subjects. Additionally, the 38.9% accuracy shows that the model has space for improvement.



```
Epoch 28/30
13/14 [==========================>...] - ETA: 0s - loss: 0.4745 - accuracy: 0.8462
Epoch 28: val_loss did not improve from 2.52105
14/14 [==============================] - 1s 62ms/step - loss: 0.4759 - accuracy: 0.8449 - val_loss: 4.5526 - val_accuracy: 0.0056
Epoch 29/30
13/14 [==========================>...] - ETA: 0s - loss: 0.4703 - accuracy: 0.8558
Epoch 29: val_loss did not improve from 2.52105
14/14 [==============================] - 1s 57ms/step - loss: 0.4719 - accuracy: 0.8520 - val_loss: 4.7302 - val_accuracy: 0.0000e+00
Epoch 30/30
13/14 [==========================>...] - ETA: 0s - loss: 0.5190 - accuracy: 0.8221
Epoch 30: val_loss did not improve from 2.52105
14/14 [==============================] - 1s 60ms/step - loss: 0.5183 - accuracy: 0.8234 - val_loss: 4.6222 - val_accuracy: 0.0000e+00
19/19 [==============================] - 1s 26ms/step - loss: 1.5830 - accuracy: 0.3890
Test accuracy: 38.9%
```

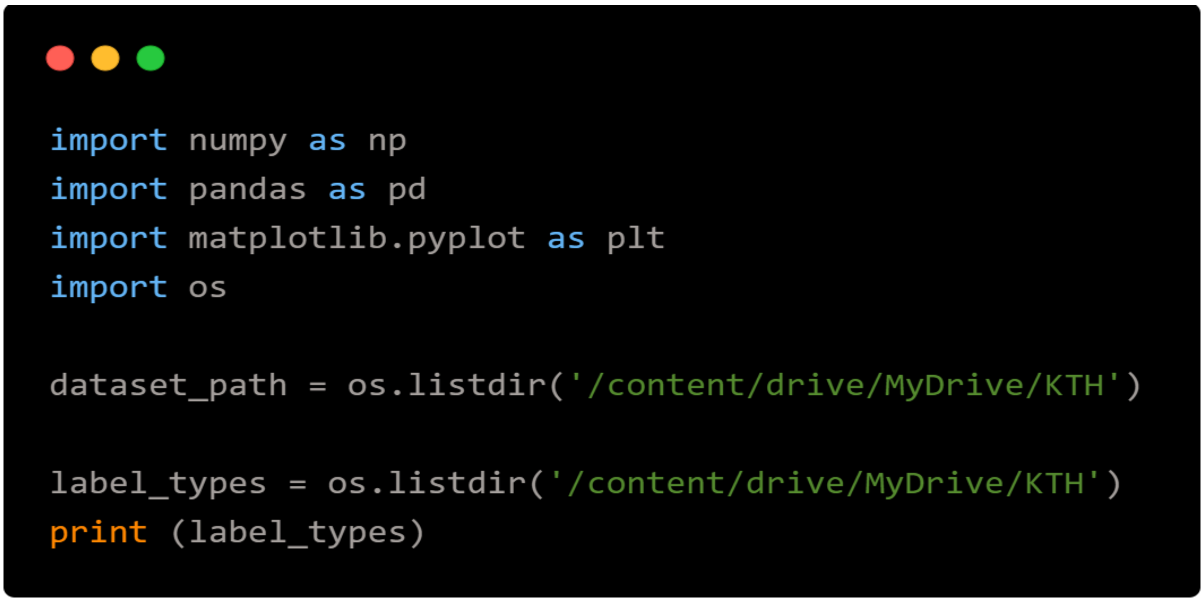
CHAPTER 6: Future Work

Several tactics can be investigated in future study to improve the performance of the model, including:
Data Augmentation: Using techniques like temporal jittering, flipping, and scaling, the dataset can be artificially increased, giving the model access to a wider range of action representations.
Transfer Learning: Improving feature extraction and representation for the ConvLSTM model may be accomplished by using pre-trained models on larger action recognition datasets (such as Kinetics, UCF101) as a starting point.
Model designs: Experimenting with more sophisticated model designs, such as attention-based mechanisms or 3D CNNs, can improve the model's capacity to detect intricate spatiotemporal patterns.
Hyperparameter Tuning: The performance of the model's convergence and generalization could be enhanced by carefully adjusting hyperparameters such learning rate, batch size, and dropout rate.

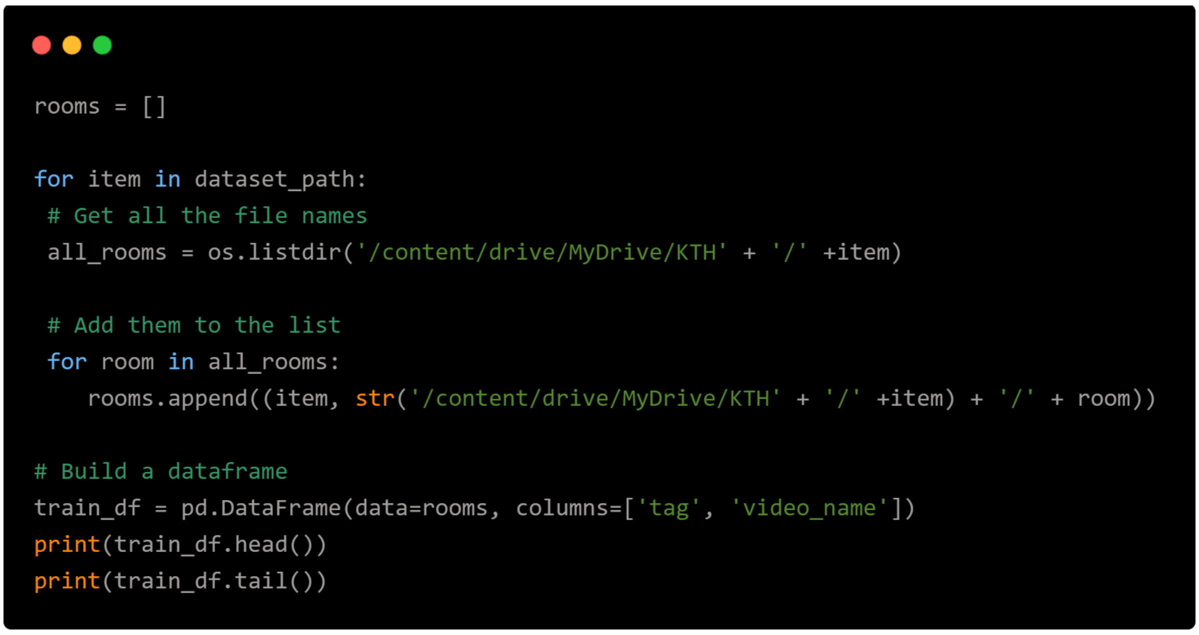
CHAPTER 7: REFERENCES

1. -OpenCV - Overview - GeeksforGeeks
2. -KTH Actions Dataset - Machine Learning Datasets (activeloop.ai)
3. -Human Action Recognition: A Literature Review – IJERT
4. -14) Learn to Extract Images/ Frames from Any Video in 6 minutes| Complete OpenCV Tutorial in Python - YouTube
5. -Recognition of human actions (kth.se)
6. -2: KTH action dataset [Schueldt et al., 2004] | Download Scientific Diagram (researchgate.net)

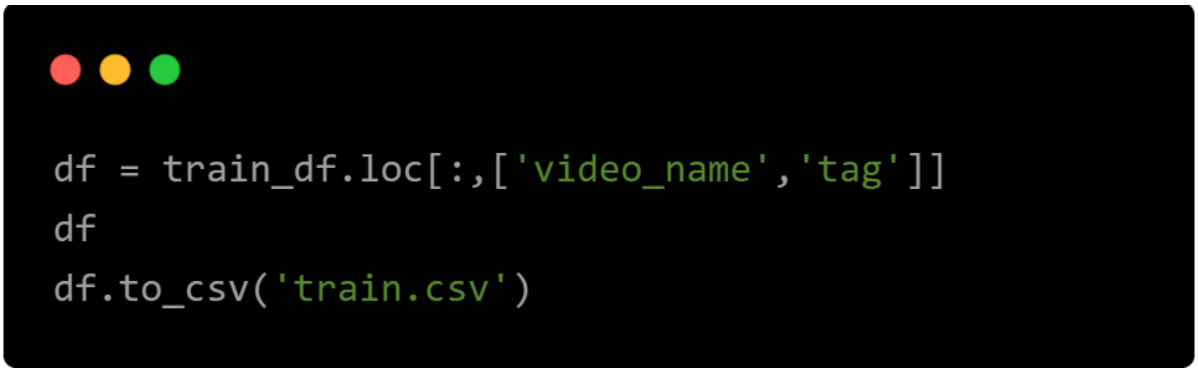
CHAPTER 8: APPENDIX



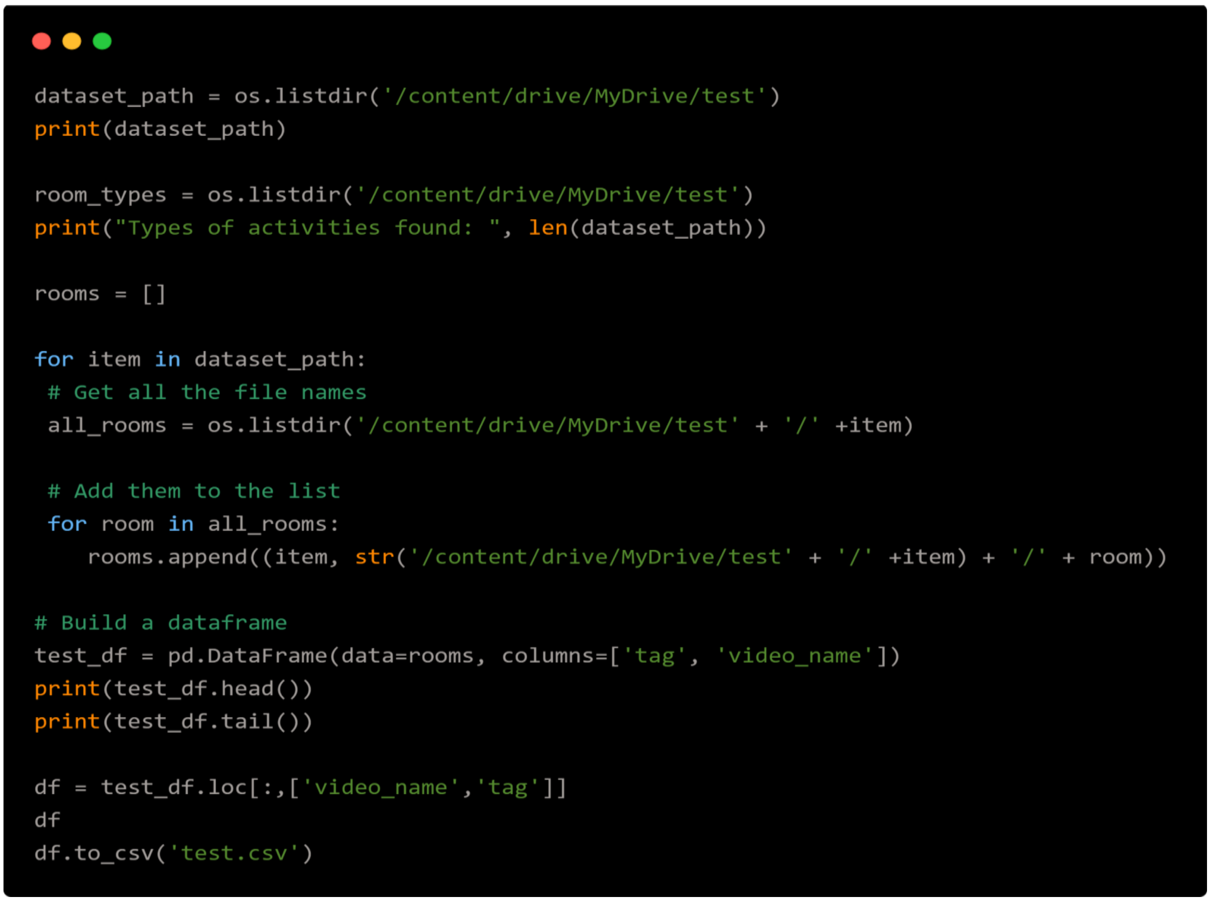
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os

dataset_path = os.listdir('/content/drive/MyDrive/KTH')

label_types = os.listdir('/content/drive/MyDrive/KTH')
print (label_types)
```



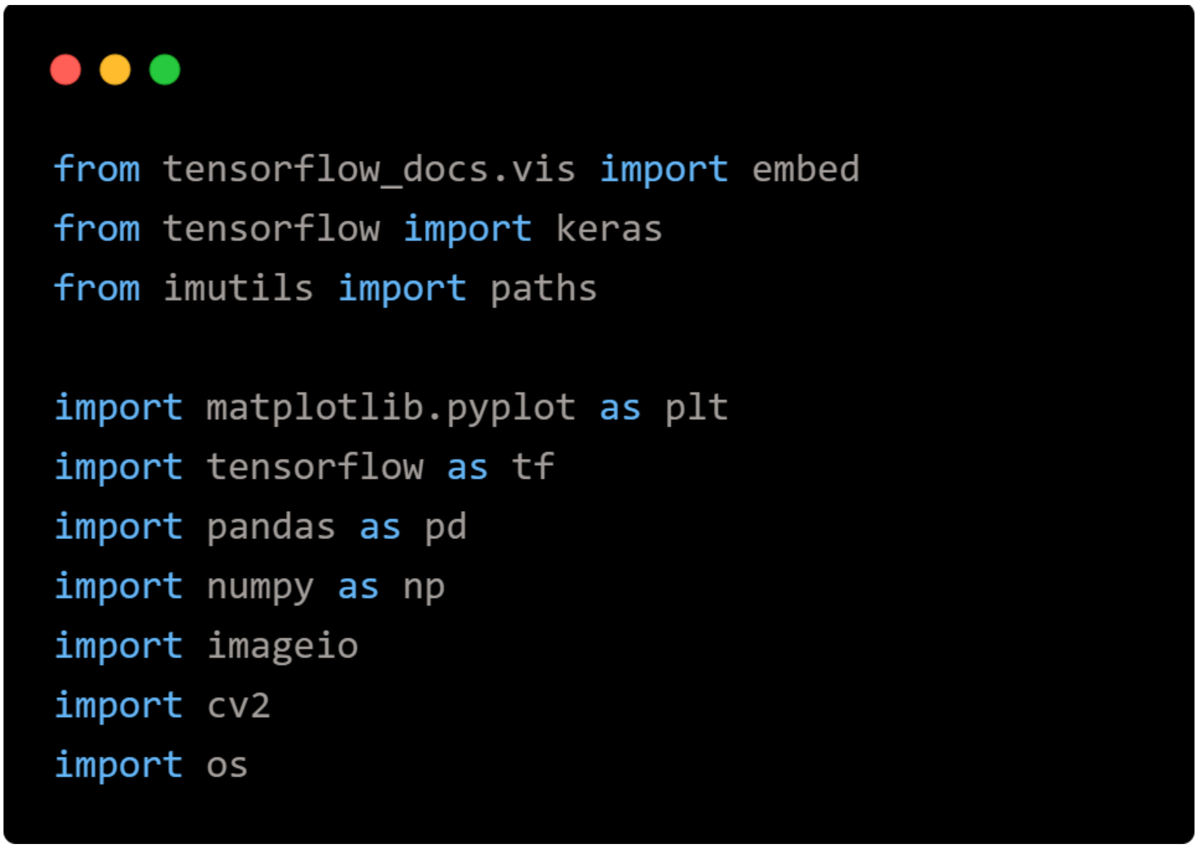
```python
rooms = []

for item in dataset_path:
 # Get all the file names
 all_rooms = os.listdir('/content/drive/MyDrive/KTH' + '/' +item)

 # Add them to the list
 for room in all_rooms:
    rooms.append((item, str('/content/drive/MyDrive/KTH' + '/' +item) + '/' + room))

# Build a dataframe
train_df = pd.DataFrame(data=rooms, columns=['tag', 'video_name'])
print(train_df.head())
print(train_df.tail())
```



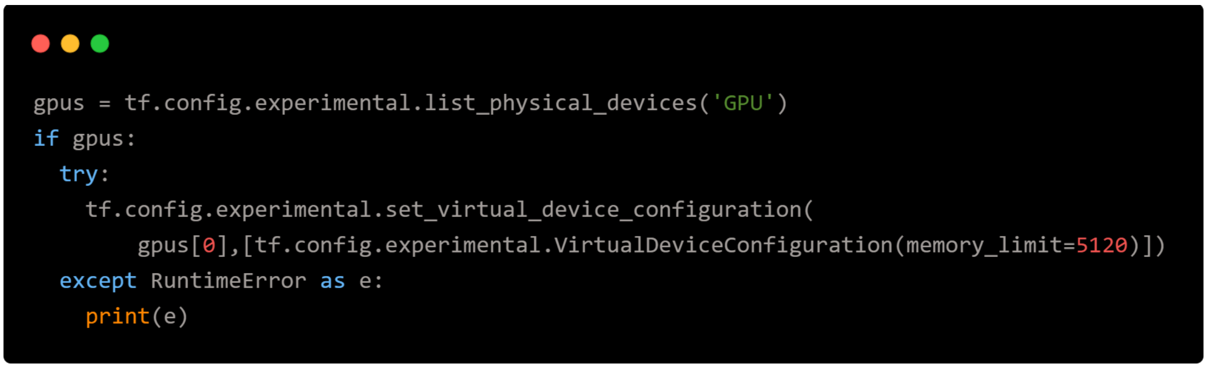
```python
df = train_df.loc[:,['video_name','tag']]
df
df.to_csv('train.csv')
```



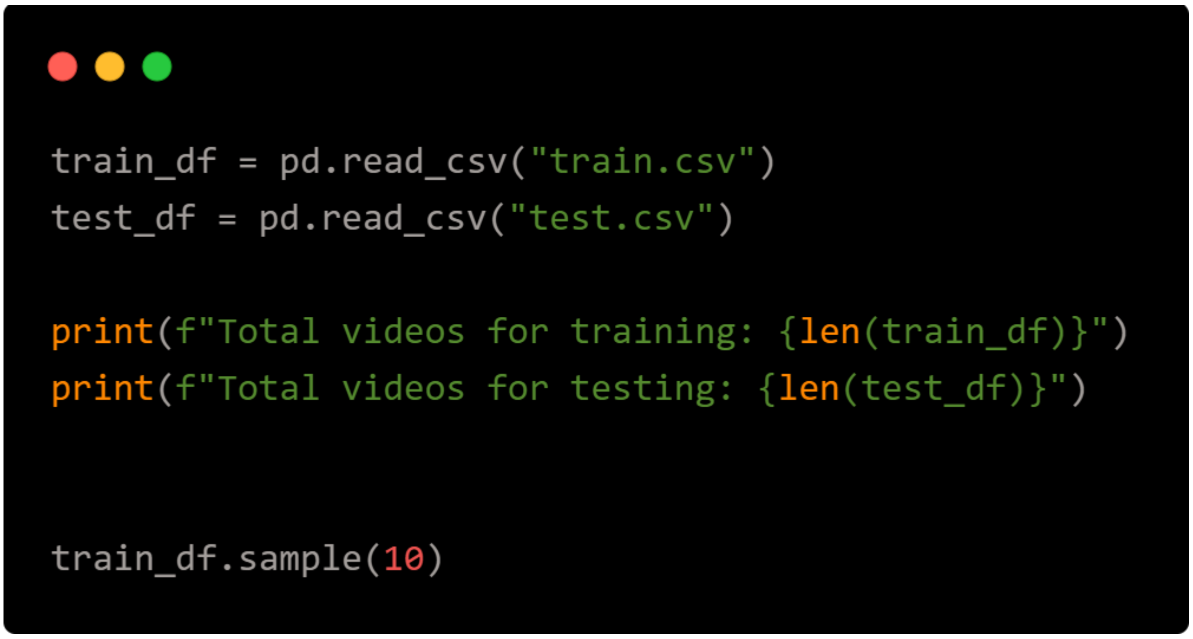
```python
dataset_path = os.listdir('/content/drive/MyDrive/test')
print(dataset_path)

room_types = os.listdir('/content/drive/MyDrive/test')
print("Types of activities found: ", len(dataset_path))

rooms = []

for item in dataset_path:
 # Get all the file names
 all_rooms = os.listdir('/content/drive/MyDrive/test' + '/' +item)

 # Add them to the list
 for room in all_rooms:
    rooms.append((item, str('/content/drive/MyDrive/test' + '/' +item) + '/' + room))

# Build a dataframe
test_df = pd.DataFrame(data=rooms, columns=['tag', 'video_name'])
print(test_df.head())
print(test_df.tail())

df = test_df.loc[:,['video_name','tag']]
df
df.to_csv('test.csv')
```



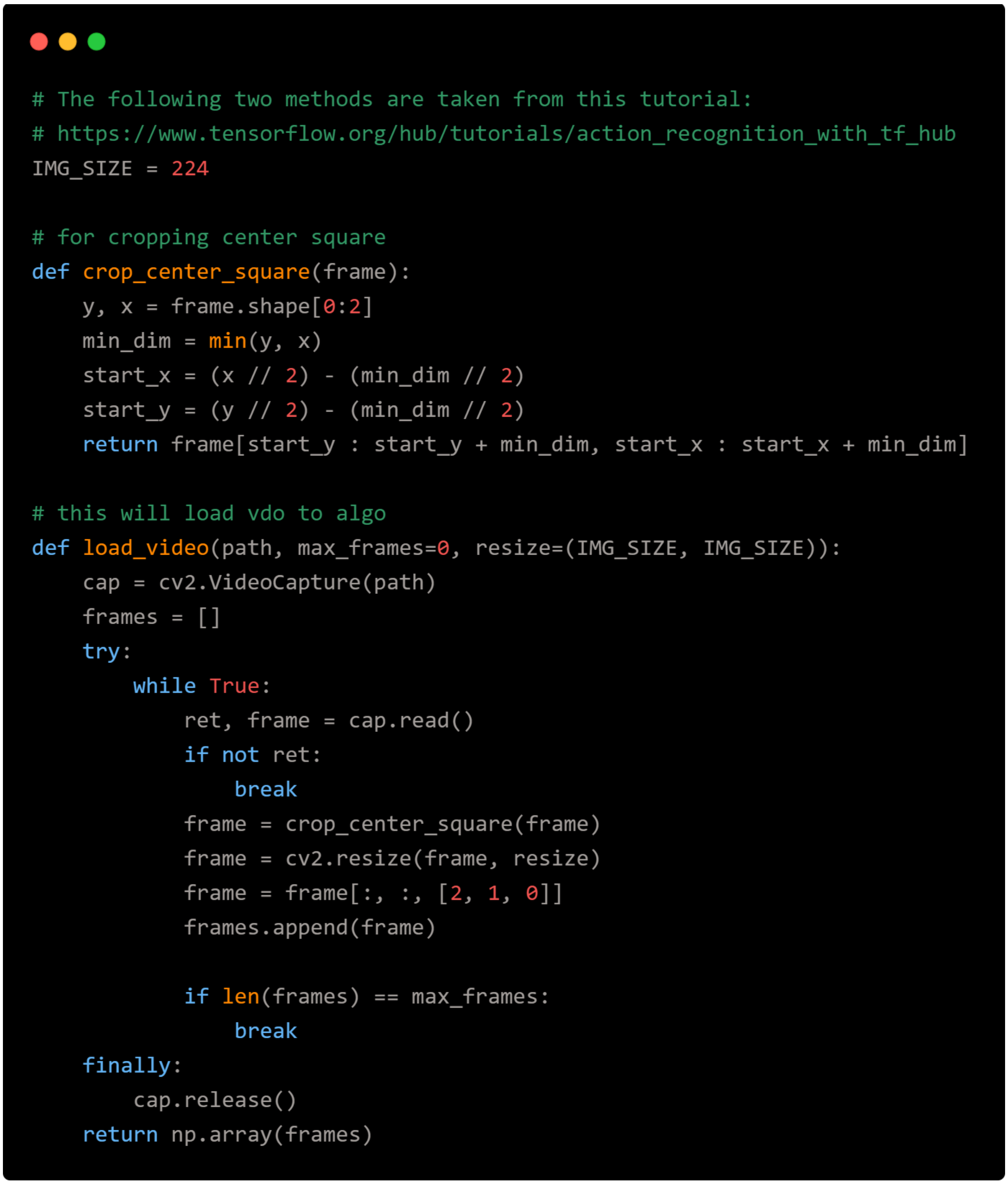
```python
from tensorflow_docs.vis import embed
from tensorflow import keras
from imutils import paths

import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd
import numpy as np
import imageio
import cv2
import os
```



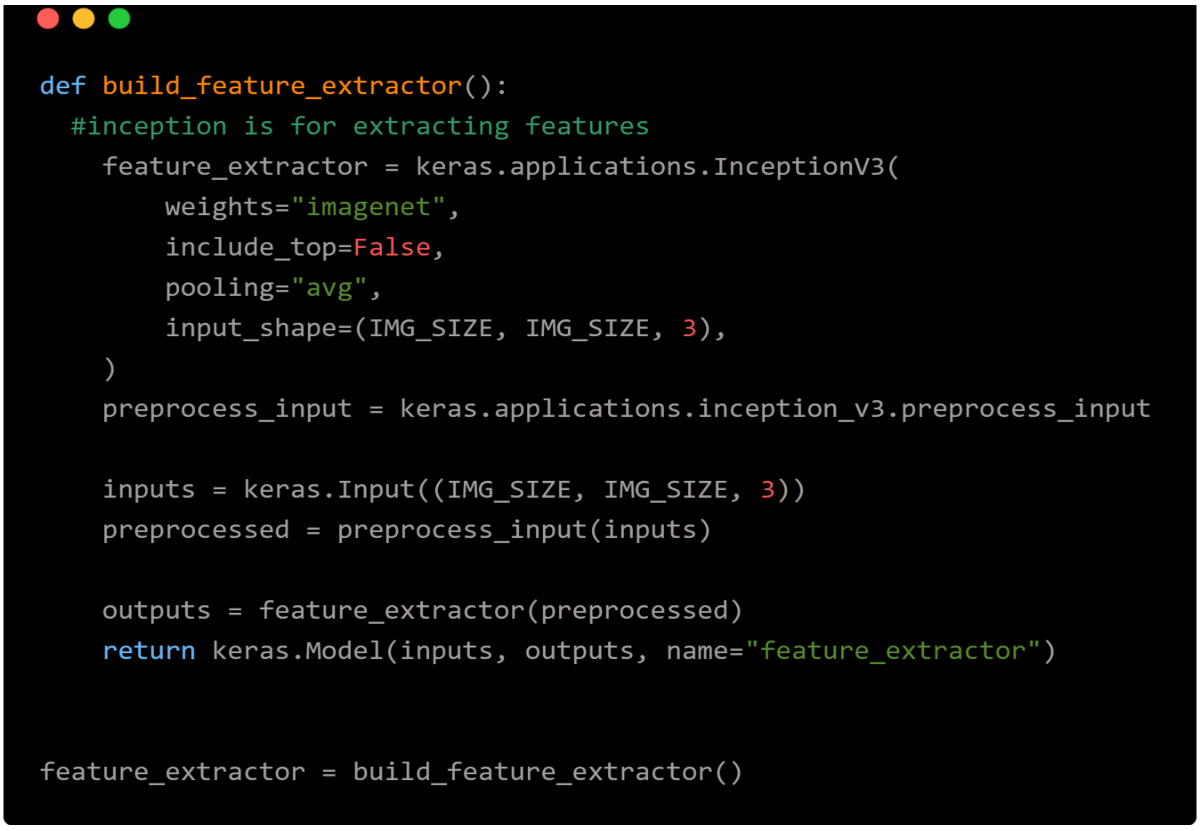
```
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
  try:
    tf.config.experimental.set_virtual_device_configuration(
        gpus[0],[tf.config.experimental.VirtualDeviceConfiguration(memory_limit=5120)])
  except RuntimeError as e:
    print(e)
```



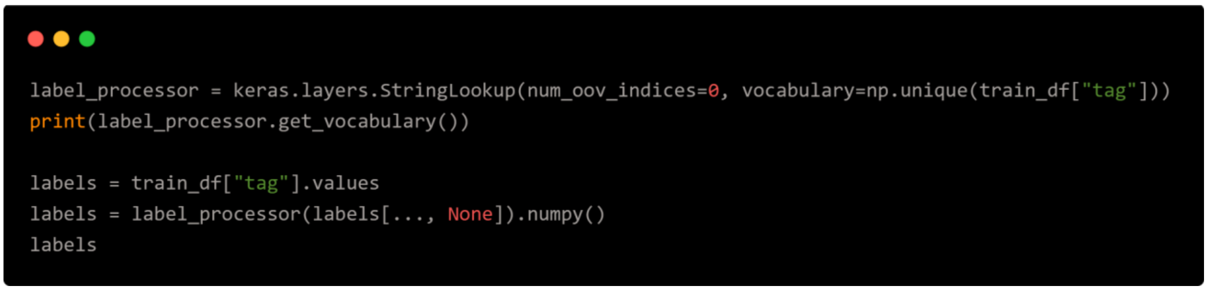
```
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")

print(f"Total videos for training: {len(train_df)}")
print(f"Total videos for testing: {len(test_df)}")


train_df.sample(10)
```



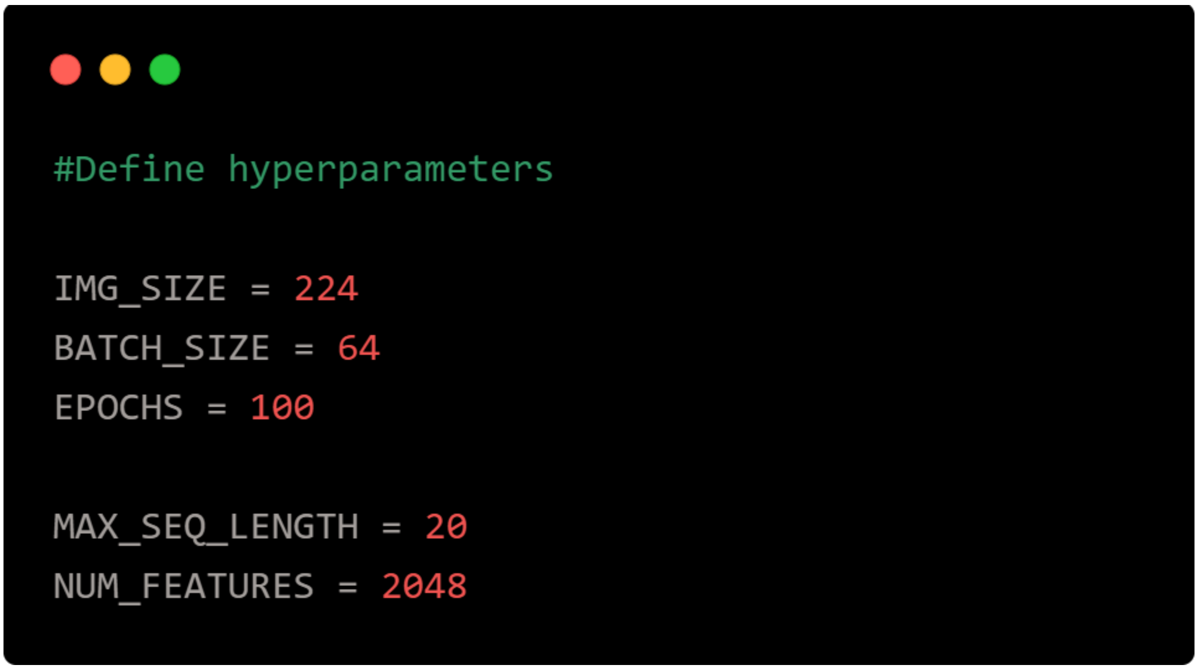
```python
# The following two methods are taken from this tutorial:
# https://www.tensorflow.org/hub/tutorials/action_recognition_with_tf_hub
IMG_SIZE = 224

# for cropping center square
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]

# this will load vdo to algo
def load_video(path, max_frames=0, resize=(IMG_SIZE, IMG_SIZE)):
    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)
```



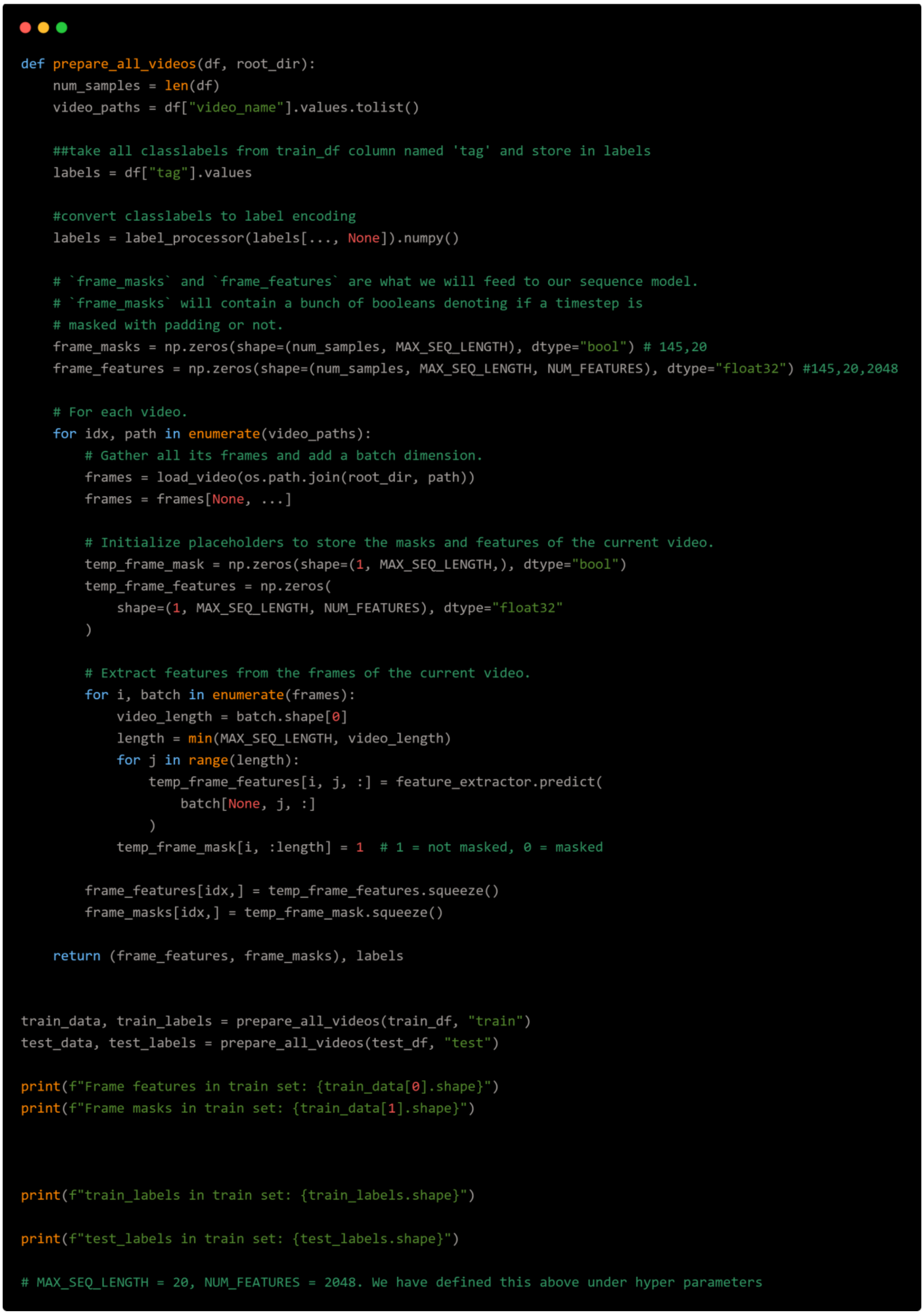
```python
def build_feature_extractor():
  #inception is for extracting features
    feature_extractor = keras.applications.InceptionV3(
        weights="imagenet",
        include_top=False,
        pooling="avg",
        input_shape=(IMG_SIZE, IMG_SIZE, 3),
    )
    preprocess_input = keras.applications.inception_v3.preprocess_input

    inputs = keras.Input((IMG_SIZE, IMG_SIZE, 3))
    preprocessed = preprocess_input(inputs)

    outputs = feature_extractor(preprocessed)
    return keras.Model(inputs, outputs, name="feature_extractor")


feature_extractor = build_feature_extractor()
```



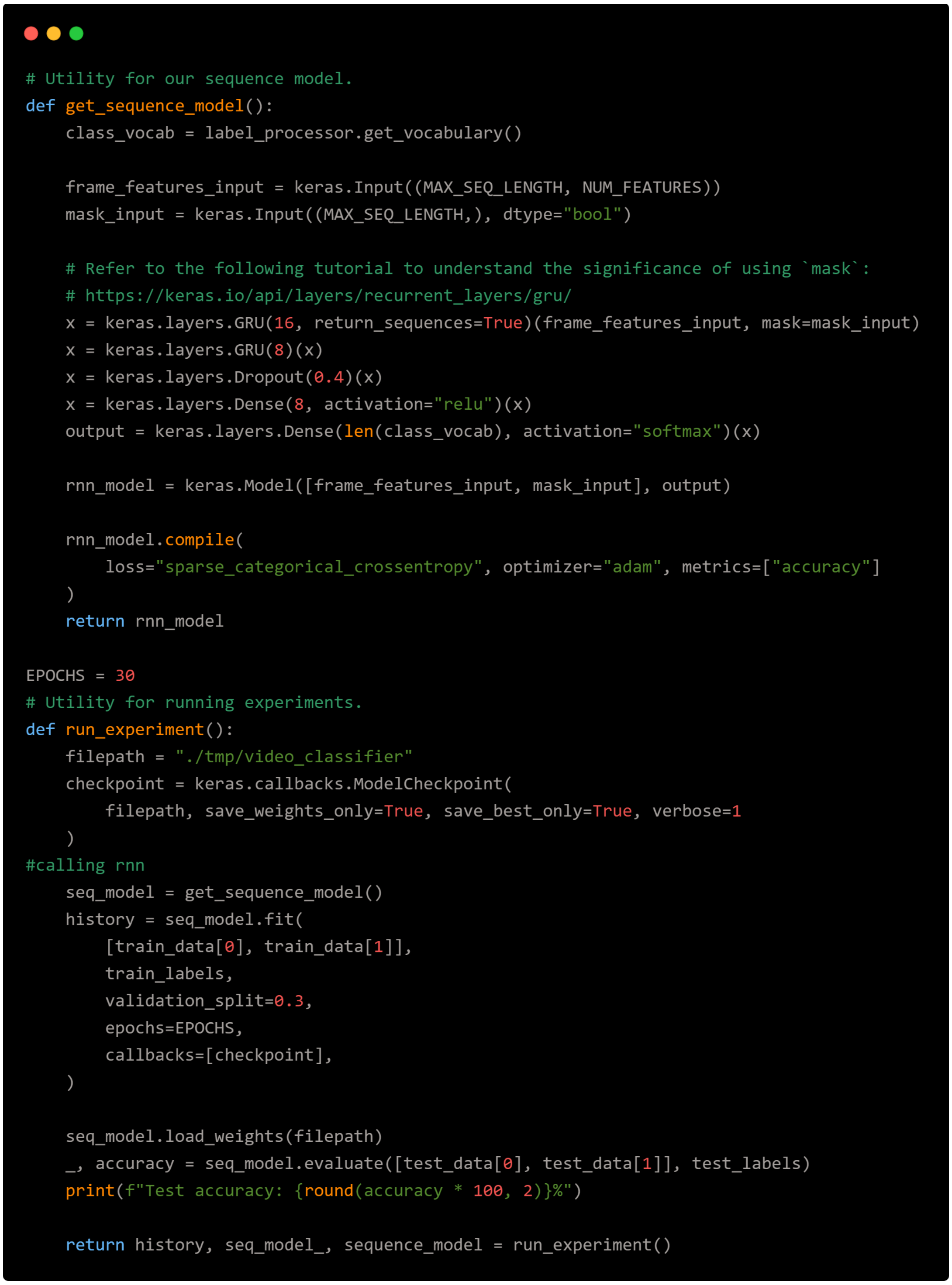
```python
label_processor = keras.layers.StringLookup(num_oov_indices=0, vocabulary=np.unique(train_df["tag"]))
print(label_processor.get_vocabulary())

labels = train_df["tag"].values
labels = label_processor(labels[..., None]).numpy()
labels
```



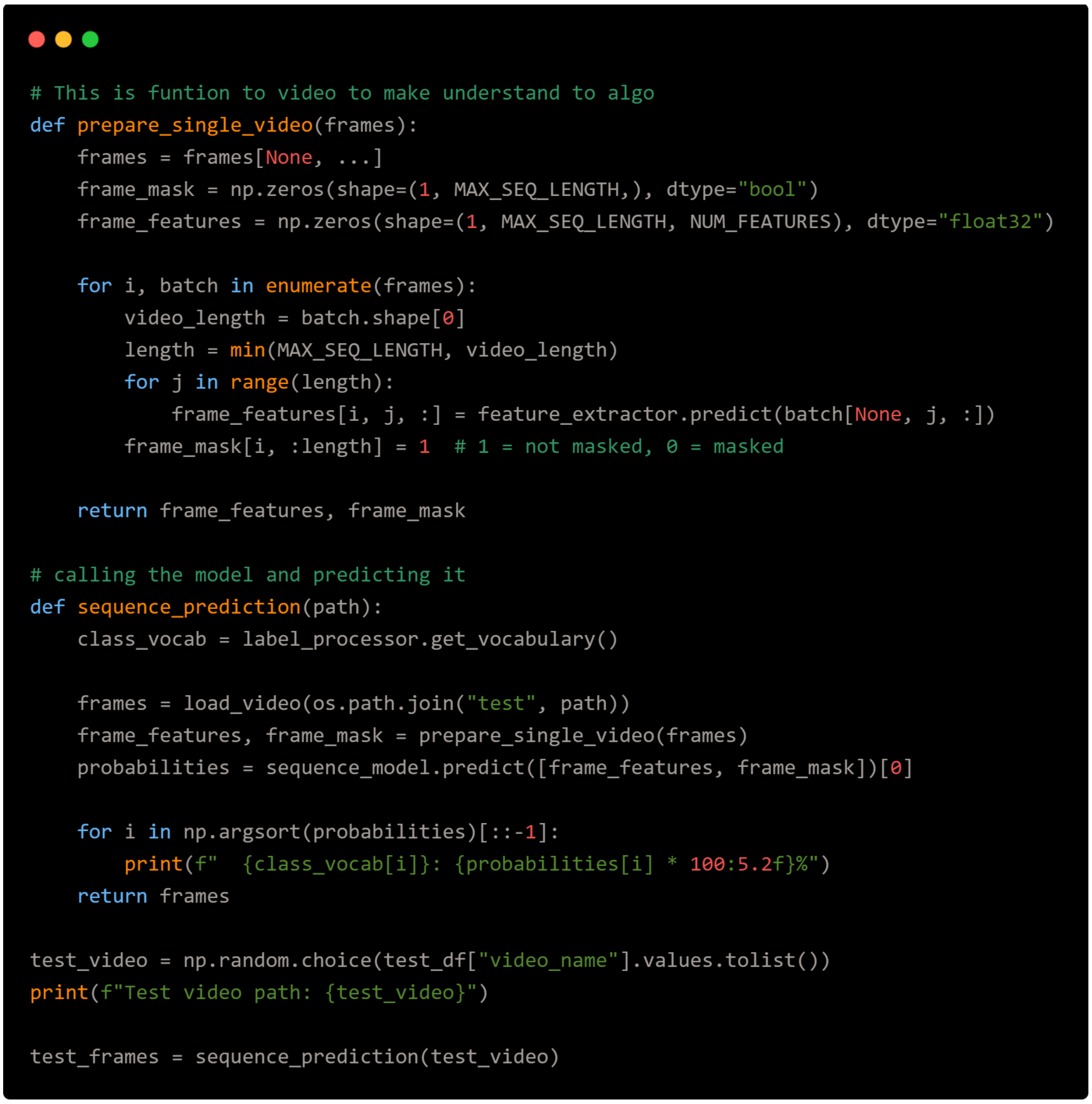
```python
#Define hyperparameters

IMG_SIZE = 224
BATCH_SIZE = 64
EPOCHS = 100

MAX_SEQ_LENGTH = 20
NUM_FEATURES = 2048
```



```python
def prepare_all_videos(df, root_dir):
    num_samples = len(df)
    video_paths = df["video_name"].values.tolist()

    ##take all classlabels from train_df column named 'tag' and store in labels
    labels = df["tag"].values

    #convert classlabels to label encoding
    labels = label_processor(labels[..., None]).numpy()

    # `frame_masks` and `frame_features` are what we will feed to our sequence model.
    # `frame_masks` will contain a bunch of booleans denoting if a timestep is
    # masked with padding or not.
    frame_masks = np.zeros(shape=(num_samples, MAX_SEQ_LENGTH), dtype="bool") # 145,20
    frame_features = np.zeros(shape=(num_samples, MAX_SEQ_LENGTH, NUM_FEATURES), dtype="float32") #145,20,2048

    # For each video.
    for idx, path in enumerate(video_paths):
        # Gather all its frames and add a batch dimension.
        frames = load_video(os.path.join(root_dir, path))
        frames = frames[None, ...]

        # Initialize placeholders to store the masks and features of the current video.
        temp_frame_mask = np.zeros(shape=(1, MAX_SEQ_LENGTH,), dtype="bool")
        temp_frame_features = np.zeros(
            shape=(1, MAX_SEQ_LENGTH, NUM_FEATURES), dtype="float32"
        )

        # Extract features from the frames of the current video.
        for i, batch in enumerate(frames):
            video_length = batch.shape[0]
            length = min(MAX_SEQ_LENGTH, video_length)
            for j in range(length):
                temp_frame_features[i, j, :] = feature_extractor.predict(
                    batch[None, j, :]
                )
            temp_frame_mask[i, :length] = 1  # 1 = not masked, 0 = masked

        frame_features[idx,] = temp_frame_features.squeeze()
        frame_masks[idx,] = temp_frame_mask.squeeze()

    return (frame_features, frame_masks), labels


train_data, train_labels = prepare_all_videos(train_df, "train")
test_data, test_labels = prepare_all_videos(test_df, "test")

print(f"Frame features in train set: {train_data[0].shape}")
print(f"Frame masks in train set: {train_data[1].shape}")



print(f"train_labels in train set: {train_labels.shape}")

print(f"test_labels in train set: {test_labels.shape}")

# MAX_SEQ_LENGTH = 20, NUM_FEATURES = 2048. We have defined this above under hyper parameters
```



```python
# Utility for our sequence model.
def get_sequence_model():
    class_vocab = label_processor.get_vocabulary()

    frame_features_input = keras.Input((MAX_SEQ_LENGTH, NUM_FEATURES))
    mask_input = keras.Input((MAX_SEQ_LENGTH,), dtype="bool")

    # Refer to the following tutorial to understand the significance of using `mask`:
    # https://keras.io/api/layers/recurrent_layers/gru/
    x = keras.layers.GRU(16, return_sequences=True)(frame_features_input, mask=mask_input)
    x = keras.layers.GRU(8)(x)
    x = keras.layers.Dropout(0.4)(x)
    x = keras.layers.Dense(8, activation="relu")(x)
    output = keras.layers.Dense(len(class_vocab), activation="softmax")(x)

    rnn_model = keras.Model([frame_features_input, mask_input], output)

    rnn_model.compile(
        loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["accuracy"]
    )
    return rnn_model

EPOCHS = 30
# Utility for running experiments.
def run_experiment():
    filepath = "./tmp/video_classifier"
    checkpoint = keras.callbacks.ModelCheckpoint(
        filepath, save_weights_only=True, save_best_only=True, verbose=1
    )
#calling rnn
    seq_model = get_sequence_model()
    history = seq_model.fit(
        [train_data[0], train_data[1]],
        train_labels,
        validation_split=0.3,
        epochs=EPOCHS,
        callbacks=[checkpoint],
    )

    seq_model.load_weights(filepath)
    _, accuracy = seq_model.evaluate([test_data[0], test_data[1]], test_labels)
    print(f"Test accuracy: {round(accuracy * 100, 2)}%")

    return history, seq_model_, sequence_model = run_experiment()
```



```python
# This is funtion to video to make understand to algo
def prepare_single_video(frames):
    frames = frames[None, ...]
    frame_mask = np.zeros(shape=(1, MAX_SEQ_LENGTH,), dtype="bool")
    frame_features = np.zeros(shape=(1, MAX_SEQ_LENGTH, NUM_FEATURES), dtype="float32")

    for i, batch in enumerate(frames):
        video_length = batch.shape[0]
        length = min(MAX_SEQ_LENGTH, video_length)
        for j in range(length):
            frame_features[i, j, :] = feature_extractor.predict(batch[None, j, :])
        frame_mask[i, :length] = 1  # 1 = not masked, 0 = masked

    return frame_features, frame_mask

# calling the model and predicting it
def sequence_prediction(path):
    class_vocab = label_processor.get_vocabulary()

    frames = load_video(os.path.join("test", path))
    frame_features, frame_mask = prepare_single_video(frames)
    probabilities = sequence_model.predict([frame_features, frame_mask])[0]

    for i in np.argsort(probabilities)[::-1]:
        print(f"  {class_vocab[i]}: {probabilities[i] * 100:5.2f}%")
    return frames

test_video = np.random.choice(test_df["video_name"].values.tolist())
print(f"Test video path: {test_video}")

test_frames = sequence_prediction(test_video)
```