VIDEO ACTIVITY RECOGNITION WITH TRANSFER LEARNING AND RECURRENT NEURAL NETWORKS USING TIME DISTRIBUTED LAYER IN KERAS

A Project Report submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

MANGALAPALLI KRISHNA SAMPATH, 121710317026

PABBA PAVAN KUMAR GOUD, 121710317036

VANAMA BHARGAVA SAI, 121710317062

AETUKURI SRIRAM, 121710317001

Under the esteemed guidance of

Dr. S. PRAVEEN KUMAR

Asst. Professor, CSE, GITAM

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GITAM

(Deemed to be University)

VISAKHAPATNAM

May 2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM INSTITUTE OF TECHNOLOGY

GITAM

(Deemed to be University)

CERTIFICATE

Certified that this project report titled "**VIDEO ACTIVITY RECOGNITION WITH TRANSFER LEARNING AND RECURRENT NEURAL NETWORKS USING TIME DISTRIBUTED LAYER IN KERAS**" is the bonafide record of work by **Mr. Mangalapalli Krishna Sampath (121710317026), Mr. Pabba Pavan Kumar Goud (121710317036), Mr. Vanama Bhargav Sai (121710317062), Mr. Aetukuri Sriram (121710317001)** students submitted in partial fulfillment of requirement for the award of degree of Bachelors of Technology in Computer Science and Engineering.

Project Guide

Dr. S. Praveen Kumar
Asst. Professor

Head of theDepartment

Dr. R. Sireesha
Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM INSTITUTE OF TECHNOLOGY

GITAM

(Deemed to be University)

DECLARATION

We, hereby declare that the project review entitled "**VIDEO ACTIVITY RECOGNITION WITH TRANSFER LEARNING AND RECURRENT NEURAL NETWORKS USING TIME DISTRIBUTED LAYER IN KERAS**" is an original work done in the Department of Computer Science and Engineering, GITAM Institute of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma**.**

Date:

Registration No.(s)	**Name(s)**
121710317026	**Mangalapalli Krishna Sampath**
121710317036	**Pabba Pavan Kumar Goud**
121710317062	**Vanama Bhargava Sai**
121710317001	**Aetukuri Sriram**

ABSTRACT

With the advancements in Deep Learning, many feats are achievable today that can be considered technologically appealing and also very useful in real life. Our project aims to recognize human activity in videos using Deep Learning and compare GRU efficiency with LSTM and various transfer learning models to determine an optimal and simplistic action recognition mode. Taking video as input, we convert it into sequences of frames, process each frame through a Convolutional Neural Network (CNN), and connect the entire sequence, using a time distributed layer, to LSTM or a GRU. To achieve this, we train our model on UCF101 and HMDB51, two large video activity datasets. UCF101 has 101 action classes that are widespread across various activities, and HMDB51 has 51 action classes that are more focused on humans' movement. Activity recognition is beneficial as it can help us refine many parts of society, especially post-pandemic. This project is our attempt at activity recognition and analysis on a couple of methods that can hopefully be helpful

ACKNOWLEDGEMENT

We seize this opportunity to convey our special thanks to our guide **Dr. S. Praveen Kumar**, who despite his busy schedule helped us with his insightful guidance throughout the development of the project and ensured that we were always on the right path.

We express our deepest thanks to **Dr. R. Sireesha**, HOD of C.S.E department, GIT for providing her constant support. We choose this moment to acknowledge her assistance gratefully.

We express our sincere thanks to **Dr. C. Dharma Raj**, I/c Principal & Professor, GITAM Institute of Technology, GITAM for encouraging us to explore new tools and work on new technology..

We express our sincere thanks to **Shri. A. Yashwanth**, our AMC and reviewers **Shri. S.V.G. Reddy** and **Shri. V. Praneel** who helped and supported us a lot in the successful completion of our project.

Sincerely,

Mangalapalli Krishna Sampath, 121710317026
Pabba Pavan Kumar Goud, 121710317036
Vanama Bhargav Sai, 121710317062
Aetukuri Sriram, 121710317001

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS, ABBREVIATIONS AND NOMENCLATURE

S. NO.	ACRONYM	ABBREVIATION
1	HMDB	Human Motion Database
2	ML	Machine Learning
3	DL	Deep Learning
4	TF	Tensorflow
5	LSTM	Long Short Term Memory
6	CNN	Convolutional Neural Network
7	RNN	Recurrent Neural Network
8	RGB	Red, Green & Blue
9	GRU	Gated Recurrent Unit

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION

Activity recognition in videos is identifying the action that is being performed in a video clip. We have seen many advancements in Deep Learning and Computer Vision in recent times, and many novel methods are available to process image data. However, when it comes to video data, there is still a lot of room for betterment.

Video activity recognition has never been straightforward, and that is because videos cannot be directly processed like images. First, the video data should be converted to a sequence of frames, and only by processing the individual frames and their arrangement can we work on video data.

In this project, we approach the conversion of video input to a sequence of frames using the video generators module provided by keras. The number of frames in a video can be very high. There is a significant possibility that consecutive frames are similar to one another and do not represent the movement that occurs in the video. For this reason, we skip consecutive frames and only select the frames that are apart from one another as it increases the possibility of capturing the frame sequence that shows more movement.

Next, we should process the series of frames through a classifier model. In deep learning, convolutional neural networks (CNN) generally provide excellent results when classifying image data. A traditional convolutional neural network cannot take a series of images, and it can only process a single frame at a time. A CNN can only accept a two-dimensional entity like an image as its input. However, in this case, the input has an additional dimension: the number of frames in the sequence. So we use the time distributed layer, which considers a temporal dimension.

The individual images are processed by the CNN according to the temporal dimension, and all the processed images connect to a recurrent neural network (RNN) in the respective sequence. We use either an LSTM layer and separately a GRU layer here and compare the results obtained. We operate CNN using transfer learning to repurpose pre-trained models to suit our use case. Transfer learning improves the accuracy of the model quite drastically and proves to be essential.

The proposed approach is simple to understand and implement. While still requiring some suitable hardware, considering modern computing standards, one does not need a laboratory with cutting-edge graphics cards to execute this method. Many solutions to video activity recognition take flow and RGB images of the datasets as inputs which require lots of pre-processing and larger disk spaces. In contrast, the proposed model takes videos as input directly and converts them into sequences of frames intrinsically while not compromising the model's final accuracy.

1.2 PROBLEM STATEMENT

In this project, we aim to build a deep learning model that predicts the activity in a given video . To achieve this we aim to use two datasets UCF101 and HMDB51 which have 101 and 51 action classes with 7GB and 2GB worth data of videos respectively. Our task is to build a classification model that has a CNN + LSTM / GRU by utilising multiple features in Tensorflow and Keras including the time distributed layer. We also compare multiple CNN algorithms like building a CNN from scratch and using transfer learning and find out what is the best model for the use case. We also aim to compare LSTM and GRU and how they perform in video classification problems by analysing the results from this project

1.3 SCOPE

There is a lot of scope in our project as we aim to recognize human activity in videos using Deep Learning. Activity recognition is very useful as it can help us in refining many parts of society, especially post-pandemic. For a high-concept society like having staff-less smart stores or high quality video monitoring, activity recognition is an essential fundamental. Generally, most of the research in activity recognition is based on data from sensors, but we aim to recognize activities by training and identifying from pictures and videos. Finally, our model will recognize various activities performed like walking, running, jumping etc, for a given video input.

1.4 TECHNOLOGIES AND DEVICES USED

The proposed system uses the following software and hardware:

- Python
- Tensorflow
- Open CV
- Keras
- Pandas
- Numpy
- Flask
- HTML
- CSS
- JavaScript
- Windows 10 Operating System
- High Performance NVIDIA Graphics Card (RTX 2070 Super is used)
- High Performance AMD Ryzen CPU (Ryzen 7 3700X is used)
- At Least 20 GB of free storage, preferable in SSD

CHAPTER 2

LITERATURE REVIEW

In [1] Khurram Soomro orchestrated a massive action recognition dataset called UCF101. This dataset contains 101 action classes with more than 13,000 videos that, combined, exceed 27 hours. The 101 action classes belong to various categories like human-human interactions, sports, playing musical instruments, body motion activities, and human-object interaction. Nearly 7 GB in size, training a model successfully on the UCF101 dataset can give us solid activity recognition results. Another large dataset is HMDB51 introduced by H. Kuehne in [2]. HMDB51 is a dataset on human motion only and spans 51 classes, and has over 7000 videos with a size of nearly 2GB. The contrast between the two datasets is that UCF101 is a significantly larger dataset and its classes are broader in range and category in comparison.

In [3] Rui Zhao pointed out how GRU can be faster, more efficient, and provide better accuracy compared to LSTM. LSTM and GRU are further tested and compared in this paper. [4] introduced a novel architecture called BERT using temporal modeling of 3D CNNs. This model holds the flagship results for the highest accuracy on UCF101 and HMDB51 datasets with 98.69% and 85.10% accuracy respectively. However, [4] used additional training data and pre-processed image datasets to achieve those results instead of using videos directly as in our approach. In [14] Amin Ullah introduced a bi-directional LSTM architecture which garnered a high accuracy of 91.21% and 87.64% on the two datasets. In this paper, we compare our results with these models to realize our models' efficiency.

In the paper [10] David Hubel introduced simple and complex cells for visual pattern recognition. Simple cells can recognize boundaries like edges of a particular orientation of the given input. The complex cells do this at a more holistic scale all over the provided input. This work laid the foundation for artificial neural networks. In [11] Kunihiko Fukushima designed an artificial neural network architecture where complex patterns are captured using lower-level complex cells and simple cells that detect small patterns. While this was a conceptual breakthrough, in [12] Yann LeCun from NYU's Courant Institute made the first modern use of artificial neural networks.

Yann LeCun and C. Cortes introduced the 'MNIST handwritten digit database,'[13] a dataset having 70,000 images of greyscale codes of handwritten digits (0 to 9). Based on Fukushima's artificial neural network accumulating from simple cells and smaller complex cells for processing complex patterns, LeCun devised a convolutional neural network and trained it on the MNIST dataset, which was the gold standard for image and computer vision applications for a long time.

In the more recent [5] Andrew G. Howard proposed a depthwise separable architecture for vision applications called MobileNets. The MobileNet model allowed in building light weight deep neural networks and provided good accuracy for image classification problems. In the paper, [6] Kaiming He proposed a residual learning framework architecture called ResNet to train substantially deep neural networks effectively. ResNets have up to 152 layers and perform very well on large image classification datasets like ImageNet or COCO. Kaiming He's work in 'Identity Mappings in Deep Residual Networks'[7] helped build improved ResNet versions that enhance performance.

In the paper 'Long Short-term Memory,'[8] back in 1997, Sepp Hochreiter presented a way to store information for extended sequences of time and called it long short-term memory (LSTM). LSTM has multiplicative gates that learn to keep data for a long time by highlighting only particular essential parts. LSTM structure is not affected by the reducing gradient descent problem present in vanilla recurrent neural networks and proved to be a go-to technique for deep learning problems, including sequences like text classification, speech recognition, and our use case, video activity recognition. In the more recent [9] Kyunghyun Cho introduced a gated recursive convolutional network with a focus on producing quick and efficient classification for short sequences. This method is known to us as Gated Recurrent Units(GRU), which contains a simpler two gate architecture and provides results faster than LSTM for short bursts or sequences.

Most methods for video action recognition use LSTMs, but they are slower when compared to GRU. There is a great scope for using GRUs, and in this project, we exploit the faster nature of GRUs and compare how they perform on large data with LSTM.

CHAPTER 3

PROBLEM IDENTIFICATION & OBJECTIVES

3.1 PROBLEM IDENTIFICATION

The problem here is to classify the activity performed in a given video by using deep learning and computer vision. We aim to use a classifier model that has a combined architecture of CNN+LSTM/GRU. First a video is taken as input and the input should be processed into a sequence of frames. Then this sequence should be processed by the model. The CNN part of the model should individually process each image in the generated sequence. This should be done in parallel by using the TimeDistributed layer in Keras. Then the entire processed sequence passes through LST/GRU which finally classifies the output. Here we use CNN so it is essential to analyze different CNN methods and find the optimal approach. Also, it is vital to compare the efficiency of LSTM with GRU and we can make important inferences from the results.

3.2 OBJECTIVES

The primary objectives of this project include the following:

- Applying Deep Learning algorithms to build models to predict the activity in a given video.

- Applying CNN algorithms to process the frames (images) in the video.

- Applying LSTM to process the sequence frames (images) and classify what action is performed in the video.

- Applying GRU to process the sequence frames (images) and classify what action is performed in the video.

- Compare the performance of the different CNN algorithms applied.

- Compare the performance and efficiency of LSTM and GRU.

- Ensure high-accuracy performance of the developed models

- Prepare a website using flask where uploading a video would provide the prediction.

- Using opencv and the developed models to make predictions and display them via command prompt.

CHAPTER 4

SYSTEM METHODOLOGY

4.1 SOFTWARE REQUIREMENTS

● Python is required as code will be written in python. Python Version 3.6 or newer is recommended.

● Any modern operating system like Windows 10/7 or Linux, 64 bit is preferable but not mandatory.

● Tensorflow, and Keras for Deep Learning. Many other packages like pandas are also recommended.

● OpenCV is required to perform computer vision operations.

● Flask is required to deploy the developed model.

● The UCF101 dataset is required as we need to train the model on it.

● The VideoFrameGenerator module from Keras is essential to process videos into a sequence of frames.

● The TimeDistributed layer is needed to parallelly process individual frames through a CNN and hold the sequence so that it can be provided as input to LSTM/GRU.

● An environment dedicated to this project on anaconda is helpful since installing packages in an environment gives us cohesive control.

● Tensorflow GPU and CUDA should be installed to use graphics card for training the model.

● The HMDB51 dataset is required as well to test the model on a different dataset and also draw important inferences from the conclusions..

● Anaconda Prompt and/or Command Prompt should be installed

4.2 HARDWARE REQUIREMENTS

● No specific hardware requirements are mandatory but at least 4GB RAM and 5GB storage space is recommended to run python smoothly.

● Since training models is a time taking task, the more cores a processor has, the faster it will take to train the models. An old dual core laptop will be much slower than say a PC with 8 core CPU. AMD Ryzen 7 3700x is used here.

● A discrete high end graphics card is essential, preferably a high performance NVIDIA graphics card which can take advantage of Tensorflow GPU resources.Nvidia RTX 2070 Super is used here.

● A minimum of 20 GB free space is required to carry out the project.

● Having SSD storage can help a lot. Here Gen4 PCIE NvME SSD is used.

● High amount of RAM is recommended. 16GB RAM of 3200MHz is used here.

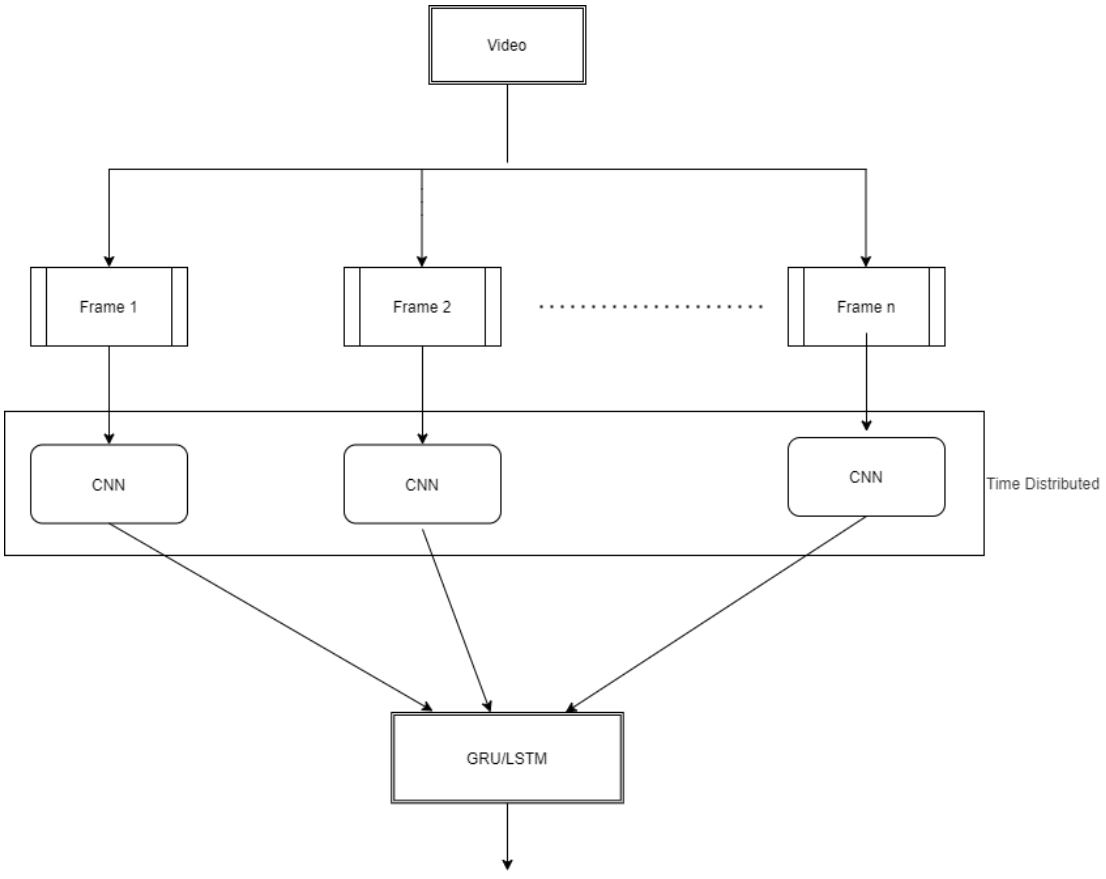
● Proper thermal management is essential as parts of the PC will be overheated during extensive training of the model.

● Internet connection is required to download datasets and also some models for transfer learning.

4.3 METHODOLOGY

We aim to provide a simple model to highlight the differences in performance between LSTM and GRU and provide efficient activity recognition. The first part of the model is converting video input to a sequence of frames. We do this with the help of the video generators module in keras. We can use the datasets directly without the need to use datasets having pre-processed frames and flow images. Such datasets provide a base for good results, and many state-of-the-art models like [4] use them, but those datasets take up a much larger space, and training them would require more hardware power. Instead, by using the videos from the datasets directly, we can focus more on how LSTM and GRU perform on them. Once we generate the sequence, we process each image from the series through a CNN. We do this by using the time-distributed layer in keras. The processed outputs then connect to LSTM/GRU. While this is a simple architecture, we can optimize the performance and get high accuracies on our datasets.

Figure 4.3: Basic architecture of the model



CHAPTER 5

OVERVIEW OF TECHNOLOGIES

5.1 PYTHON

The project is mainly implemented on python. A vast number of libraries present in python are used here. Firstly the library numpy is used which is of prime importance as we can not use images directly and have to convert them to numbers or values in matrix which is possible with the help of numpy. Many other libraries are also used, like os for accessing file paths in the system, tqdm to enquire about the completion of processes, matplotlib for visualizations and many more.

Python is an extremely flexible and easy to use language which is widely used for developing models involving machine learning and deep learning. In addition to these, python is very useful for deploying models using a framework known as Flask. Using Flask, we can code web deployment of ML/DL models in combination with HTML/CSS/JavaScript. This makes python the best language for our project, since we can build DL models with ease and deploy it using Flask.

5.2 MACHINE LEARNING

Machine Learning has gained widespread popularity in recent years as the machine learning algorithms have provided novel solutions to various problems and helped in providing a major increase in Artificial Intelligence. However the best solution lies by taking a step forward into a special area of Machine Learning called Deep Learning. Deep Learning is an integral part of Machine Learning, which is a part of Artificial Intelligence.

ML algorithms have grown to prove vital in most institutions around the globe. Data predictions and classification from ML models has brought forth novel solutions to highly complex problems.

5.3 DEEP LEARNING

The popularity of Deep Learning started not too long ago. In this project we used Deep Learning algorithms like CNN, LSTM and GRU. The CNN algorithm is a very effective and efficient algorithm when it comes to working on image data. This model is expected to give the best results in classifying image data. We approached CNN in two ways in this project. First we created a CNN model from scratch and then we used a transfer learning model where we use a pre-existing and pre-trained model and suit it to our case.

We use the Tensorflow library and the resources that come with it. Keras library is also used to make Deep Learning implementation with python much easier and smooth. We also used GRU and LSTM which are a form of Recurrent Neural Networks (RNN). They can process only important information in a sequence and discard the information deemed as not useful.

5.4 COMPUTER VISION

Computer Vision applications have been on a rise in recent years. Image and video processing is essential in today's world. In this project computer vision is essential since we are dealing with video data. OpenCV functions are vital in processing videos and formatting the frames. It is also essential in displaying outputs of the model.

Computer Vision is vital in problems related to image/video data. Various applications ranging from facial recognition, CCTV monitoring, staffless stores like Amazon Go depend on Computer Vision.

CHAPTER 6

ANALYSIS

6.1 TRANSFER LEARNING FOR CNN

When creating the CNN, we have two options: make one from scratch or use an existing model. The latter is called transfer learning. We used both methods to find out which is the better-suited technique for this case. We used a small part of the datasets, to be precise, four classes out of the 101 classes from the UCF101 dataset (classes archery, cricket shot, trampoline jumping, and walking a dog) and three classes out of the 51 classes from the HMDB51 dataset (classes run, jump and walk). The CNN model created from scratch gave an 80% accuracy for the four classes from UCF101 and 63% for the three classes from HMDB51. To improve this, we decided to use transfer learning instead of creating a CNN from scratch. [5] introduced a famous model called Mobilenet, which is available openly in TensorFlow Keras. Using transfer learning by changing the CNN to Mobilenet's CNN model boosted the accuracy to 96% for the four classes from UCF101 and 80% for the three classes from HMDB51. It also provided significantly better results in distinguishing between walking and running. [6] introduced ResNet, another revolutionary model for image classification. Using this model provided similar results for the smaller datasets but outperformed Mobilenet when trained on the entire datasets.

6.2 GRU VS. LSTM

GRU and LSTM are two widely used versions of Recurrent Neural Networks in Deep Learning. LSTM was first introduced in [8]. LSTM has a couple of gates for input and output respectively. However, it has another gate, the forget gate, which is the heart of the concept of LSTM as it decides what parts of the sequence can be forgotten so that only useful information is retained. GRU, a more recent concept, was introduced in [9]. GRU has a simpler architecture with only two gates: reset and update gates. Virtually, GRU is a simplified LSTM that combines the input and forget gates. In GRU, either the sequence is updated with new information or is reset. The general consensus is that due to its lesser parameters, GRU is quicker, but when dealing with large data, LSTM is more efficient. For highly intense tasks like training a model for activity recognition, LSTM is heavily preferred. However, in our findings, it is not as simple as that.

CHAPTER 7

IMPLEMENTATION

7.1 IMPORTING PACKAGES

Before beginning with everything else, we should first import all the necessary packages. Tensorflow and Keras are essential to building deep learning models, so we import them. The VideoFrameGenerator is imported from keras_video and this is used to process videos into a sequence of frames. We save the different classes in sorted order as well. Then we provide some standard values that we will use throughout the project. The size of every frame is fixed to 224*224 with 3 color channels which are RGB. We also fix the number of frames extracted from a video as 5 and give a batch size as 8 for the VideoFrameGenerator.

Figure 7.1: Code for importing packages

```
import os
import glob
import keras
import tensorflow as tf
from keras_video import VideoFrameGenerator

classes = [i.split(os.path.sep)[1] for i in glob.glob('vid/*')]
classes.sort()

SIZE = (224, 224)
CHANNELS = 3
NBFRAME = 5
BS = 8
```

7.2 PROCESSING INPUT VIDEO INTO A SEQUENCE OF FRAMES

The first part of implementation includes converting input videos to a sequence of frames. We do this by processing each video in the datasets through the VideoFrameGenerator module in Keras. This VideoFrameGenerator converts a given video into a sequence of images and also prepares the training and validation sets for the model. Figure 5.1 shows the code for this. The global pattern of videos is defined and each video that satisfies the pattern is processed through VideoFrameGenerator and added to the training or validation set. Features like number of frames, size of the frame, number of color channels and batch size are also fixed. In this project, we fix the frame size as 224*224 with 3 color channels (representing RGB).

Figure 7.2: Code for generating frames from input videos

```
glob_pattern='vid/{classname}/*.avi'

data_aug = keras.preprocessing.image.ImageDataGenerator(
    zoom_range=.1,
    horizontal_flip=True,
    rotation_range=8,
    width_shift_range=.2,
    height_shift_range=.2)

train = VideoFrameGenerator(
    classes=classes,
    glob_pattern=glob_pattern,
    nb_frames=NBFRAME,
    split=.33,
    shuffle=True,
    batch_size=BS,
    target_shape=SIZE,
    nb_channel=CHANNELS,
    transformation=data_aug,
    use_frame_cache=True)
```

7.3 IMPORTING KERAS LAYERS

Before we start with our model, we should import all the necessary modules from keras. Conv2D, BatchNormalizitaion, MaxPool2D and GlobalMaxPool2D are essential for building a CNN network. We also import GRU,LSTM along with Dense and Dropout layers. Optimizers like Adam, SGD are also imported. Only after importing these layers from Keras can we create the model.

Figure 7.3: Code for importing keras layers

```
from keras.layers import Conv2D, BatchNormalization, \
    MaxPool2D, GlobalMaxPool2D

from keras.layers import TimeDistributed, GRU, Dense, Dropout
from keras.layers import LSTM
from keras.optimizers import Adam
```

7.4 CONVOLUTIONAL NEURAL NETWORK

Once a sequence of frames is generated, we move to the next step. Each frame in the sequence must be individually processed through a CNN model. The following are the methods we used to do this.

7.4.1 Building A CNN Model From Scratch

First we tried to build our own CNN model. The input image is fixed for a size of 224*224 with 3 color channels as established earlier. This is a standard model with a few Conv2D layers with ReLu activation function, batch normalization and max pooling. This model gave decent results for a small part of the datasets but it is found to be inadequate for processing the entire dataset.

Figure 7.4.1: Code for CNN model

```
def build_convnet(shape=(224, 224, 3)):
    momentum = .9
    model = keras.Sequential()
    model.add(Conv2D(64, (3, 3), input_shape=shape,
                     padding='same', activation='relu'))
    model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
    model.add(BatchNormalization(momentum=momentum))

    model.add(MaxPool2D())

    model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
    model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
    model.add(BatchNormalization(momentum=momentum))

    model.add(MaxPool2D())

    model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
    model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
    model.add(BatchNormalization(momentum=momentum))

    model.add(MaxPool2D())

    model.add(Conv2D(512, (3, 3), padding='same', activation='relu'))
    model.add(Conv2D(512, (3, 3), padding='same', activation='relu'))
    model.add(BatchNormalization(momentum=momentum))


    model.add(GlobalMaxPool2D())
    return model
```

7.4.2 Building A MobileNet Model

Since the CNN model we built from the scratch is not providing high accuracy, we move on to a more sophisticated method, Transfer Learning. Using Transfer Learning, we can use a pre-existing model like MobileNet. We convert a few layers (9 in this case) as trainable to customize the outputs of the model to match our use case, as we intend to work this for different datasets.

Figure 7.4.2: Code for using MobileNet model via transfer learning

```
def build_mobilenet(shape=(224, 224, 3), nbout=3):
    model = keras.applications.mobilenet.MobileNet(
        include_top=False,
        input_shape=shape,
        weights='imagenet')

    trainable = 9
    for layer in model.layers[:-trainable]:
        layer.trainable = False
    for layer in model.layers[-trainable:]:
        layer.trainable = True
    output = keras.layers.GlobalMaxPool2D()
    return keras.Sequential([model, output])
```

7.4.3 Building A ResNet Model

The MobileNet model gave good accuracy for a part of the dataset but it's performance on the entire dataset is not high enough. So, we used another sophisticated model called ResNet instead of MobileNet. ResNet 152 has 152 layers, making it very comprehensive for processing images. A comparison between ResNet and MobileNet models' performance is highlighted in table 7.4. The code for the ResNet model is shown in Figure 7.4.3. The weights of this model are already defined on a massive image dataset called imagenet and this is very helpful for processing the individual images in our case. We also use the final 24 layers out of

the 152 layers by marking them as trainable, which makes us customize ResNet for our use case.

Figure 7.4.3: Code for using ResNet model via transfer learning

```
def build_resnet(shape=(224, 224, 3), nbout=3):
    model = tf.keras.applications.ResNet152V2(
    include_top=False,
    weights="imagenet",
    input_tensor=None,
    input_shape=None,
    pooling=None,
    classifier_activation="softmax",)

    trainable = 24

    for layer in model.layers[:-trainable]:
        layer.trainable = False
    for layer in model.layers[-trainable:]:
        layer.trainable = True
    output = keras.layers.GlobalMaxPool2D()
    return keras.Sequential([model, output])
```

Table 7.4 shows the results that ResNet and MobileNet models fetched (with LSTM) when trained on the entire UCF101 dataset. There is a massive difference between them in terms of accuracy and loss. The Mobilenet model gave an accuracy of 72% with a training loss of 0.76. While the ResNet scored an accuracy and loss of 96.06% and 0.0996, respectively. The Mobilenet model's best epoch is 192, while ResNet's is 110. The ResNet model trained significantly quicker than MobileNet. With these results, it is safe to assert the ResNet model is the better one for processing the frames, and we use this in the comparison between LSTM and GRU.

Table 7.4: Comparison of Resnet and Mobilenet with LSTM on UCF101 dataset

Model	Best Epoch	Accuracy (in %)	Loss
Mobilenet	192	72.00	0.7616
Resnet	110	96.06	0.0996

7.5 RECURRENT NEURAL NETWORK

Once we generate the sequence of images, we have our train and validation sets ready. We have to process them through our action model. We define our action model by first passing each image in the sequence through a CNN defined in the previous stem and then connect it via the TimeDistributed Layer in Keras. We can see this clearly in the code in the figure below. So, after CNN processes each image in the sequence, the entire sequence, which is held together by the TimeDistributed layer, is provided as input to a Recurrent Neural Network. In this project we tried both LSTM and GRU for this. The figure shows code for LSTM, but GRU is also defined the same way in Keras. The main difference is in their internal architecture and design rather than the code we need to call them from, which is simplified by Tensorflow and Keras. After the LSTM/GRU layer, a few dense layers and dropout layers are added.

Figure 7.5.1: Code for Action Model containing LSTM

```
def action_model(shape=(5, 224, 224, 3), nbout=101):

    convnet = build_resnet(shape[1:])

    model = keras.Sequential()
    model.add(TimeDistributed(convnet, input_shape=shape))

    model.add(LSTM(128))

    model.add(Dense(1024, activation='relu'))
    model.add(Dropout(.5))

    model.add(Dense(512, activation='relu'))
    model.add(Dropout(.5))

    model.add(Dense(128, activation='relu'))
    model.add(Dropout(.5))

    model.add(Dense(64, activation='relu'))
    model.add(Dense(nbout, activation='softmax'))

    return model
```

7.6 CREATING AND COMPILING THE MODEL

Now that we defined the model, we have to create it by calling the action model. We format the input shape and define the model and also define the optimizer. Stochastic Gradient Descent (SGD) optimizer is used instead of the popular Adam optimizer as it resulted in better training speeds and accuracy. Then we compile the model using model.compile function.

Figure 7.6: Code for creating and compiling the model

```
INSHAPE=(NBFRAME,) + SIZE + (CHANNELS,) # (5, 224, 224, 3)
model = action_model(INSHAPE, len(classes))
optimizer = keras.optimizers.SGD()


model.compile(
    optimizer,
    'categorical_crossentropy',
    metrics=['acc']
)
```

7.7 FITTING THE MODEL

Finally, we should fit the model and then begin training. We define callbacks before fitting the model. In callbacks, we include ReduceLOonPlateau which will automatically reduce the learning rate when necessary. Also we save the weights of the model with ModelCheckpoint after each epoch so that we can use the one with the best accuracy and loss among others. Then we fit the model using the model.fit function. We provide it with train and validation data along with callbacks and number of epochs.

Figure 7.7: Code for creating and compiling the model

```
callbacks = [
    keras.callbacks.ReduceLROnPlateau(verbose=1),
    keras.callbacks.ModelCheckpoint(
        'chkp_transfer_lstm_2/weights.{epoch:02d}-{val_loss:.2f}.hdf5',
        verbose=1),
]
model.fit(
    train,
    validation_data=valid,
    verbose=1,
    epochs=EPOCHS,
    callbacks=callbacks
)
```

7.8 LOADING THE MODEL AND PROCESSING INPUT VIDEO

First we have to import load_model from keras.models. Then with the help of the load_model function, we can load the model with the best results from the training period. After loading the model, we have to prepare the input for the model. The input to our program is a video, which is given in the command prompt line argument. So we store the name of the video as our pattern. Then we process the input video into a sequence of frames with the help of VideoFrameGenerator. The variable test2 then will contain the formatted sequence of frames that will be given to the model as input.

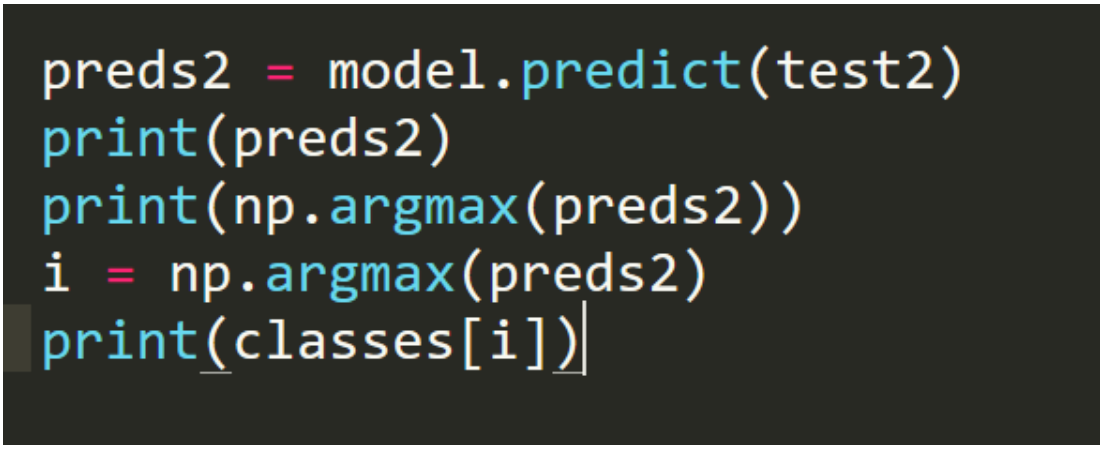
Figure 7.8: Code for loading the model and processing a given video

```
from keras.models import load_model
model=
load_model('chkp_resnet_full/weights.87-0.554-0.80-2.36-0.58.hdf5')

pattern2 = str(sys.argv[1])
test2 = VideoFrameGenerator(
        glob_pattern=pattern2,
        nb_frames=NBFRAME,
        batch_size=1,
        target_shape=SIZE,
        nb_channel=CHANNELS,
        transformation=data_aug,
        use_frame_cache=True)
```

7.9 MAKING THE PREDICTION

The input video is converted to a sequence of images in the previous step and it is stored in test2. We now have to make a prediction for this input, so we call model.predict and give test2 as input to the model. The result will be a list of predictions for the confidence scores for all the classes in the dataset. We use argmax from numpy to identify which argument has the highest confidence score. Then we use this argument number as an index to find out which class has the highest score and print that class as the predicted class.

Figure 7.9: Code for loading the model and processing a given video



7.10 READING THE INPUT VIDEO USING COMPUTER VISION

Firstly, we have to import cv2 to use Computer Vision functions. We capture and read the input video by sending its pattern as input to cv2.VideoCapture and store it in cap. We define the width and height of the frames with the help of get function. Then, we use cv2.VideoWriter to create and save an output video defined by the same frame size as the input video.

Figure 7.10: Code for reading the input video using computer vision and creating a new output video with the prediction

```
import cv2


cap = cv2.VideoCapture(pattern)

frame_width = int(cap.get(3))
frame_height = int(cap.get(4))
out = cv2.VideoWriter('Predictions/outpy.avi',
      cv2.VideoWriter_fourcc('M','J','P','G'),
       10,
       (frame_width,frame_height))
```

7.11 DISPLAYING THE OUTPUT VIDEO WITH PREDICTION

We now display the video with the prediction written on it and save each frame to the output file. We write the prediction onto the video by using the cv2.putText method. For this first we define the font style, for which we chose the Hershey Simplex font. Then we provide the value of the prediction i.e. the action class predicted by the model for the given input video. Then we provide the X and Y coordinates at which the text on video would begin (we can consider it as the origin point of the text. In addition to that, we should also mention the color of the text, for which we chose red (RGB coordinates (0,0,255)) and the thickness of the line which we chose as four. Then we write the updated frame to the output video using the write method and display it using the imshow method in cv2. Finally we release the cv2 instances and destroy all windows before finishing the program.

Figure 7.11: Code for displaying the output video with the prediction

```
while(True):
  ret, frame = cap.read()

  if ret == True:

    # Write the frame into the file 'output.avi'
    font = cv2.FONT_HERSHEY_SIMPLEX

    cv2.putText(frame,
                str(classes[i]),
                (0, 50),
                font, 1,
                (0, 0, 255),
                4,
                cv2.LINE_4)

    out.write(frame)

    # Display the resulting frame
    cv2.imshow('frame',frame)

    # Press Q on keyboard to stop recording
    if cv2.waitKey(1) & 0xFF == ord('q'):
      break

  # Break the loop
  else:
    break

cap.release()
out.release()

cv2.destroyAllWindows()
```

7.12 FINDING THE ACCURACIES ACHIEVED BY THE MODEL FOR DIFFERENT TYPES OF ACTIVITIES IN THE UCF101 DATASET

Once again, first we load the model and then process the input videos into sequences of frames. We store the pattern of the input files for each type of activity in the variable pattern. Then we process all the videos satisfying the pattern with the VideoFrameGenerator, similar to earlier. We then use model.compile and use the metric TopKCategoricalAccuracy and set it to one so that we can evaluate for the best prediction only. Then we use model.evaluate to find out the accuracy and loss for that particular set of activities.

Figure 7.12.1: Code for finding the accuracy of Body Motion classes

```
pattern = 'Action_Type/1_Body_Motion/{classname}/*.avi'


test2 = VideoFrameGenerator(
        glob_pattern=pattern,
        nb_frames=NBFRAME,
        batch_size=1,
        # split=1,
        target_shape=SIZE,
        nb_channel=CHANNELS,
        transformation=data_aug,
        use_frame_cache=True)

model.compile(metrics= [tf.keras.metrics.TopKCategoricalAccuracy(k=1)])
loss,acc = model.evaluate(test2)
print('\nAccuracy for Body Motion Only : {}\n'.format(acc))
```

Figure 7.12.2: Code for finding the accuracy of Human-Human Interaction classes

```
pattern = 'Action_Type/2_Human_human/{classname}/*.avi'

test2 = VideoFrameGenerator(
        glob_pattern=pattern,
        nb_frames=NBFRAME,
        batch_size=1,
        # split=1,
        target_shape=SIZE,
        nb_channel=CHANNELS,
        transformation=data_aug,
        use_frame_cache=True)

model.compile(metrics= [tf.keras.metrics.TopKCategoricalAccuracy(k=1)])
loss,acc = model.evaluate(test2)
print('\nAccuracy for Human_Human_Interaction : {}\n'.format(acc))
```

Figure 7.12.3: Code for finding the accuracy of Human Object Interaction classes

```
pattern = 'Action_Type/3_Human_object/{classname}/*.avi'


test2 = VideoFrameGenerator(
        glob_pattern=pattern,
        nb_frames=NBFRAME,
        batch_size=1,
        target_shape=SIZE,
        nb_channel=CHANNELS,
        transformation=data_aug,
        use_frame_cache=True)


model.compile(metrics= [tf.keras.metrics.TopKCategoricalAccuracy(k=1)])
loss,acc = model.evaluate(test2)
print('\nAccuracy for Human_Object_Interaction : {}\n'.format(acc))
```

Figure 7.12.4: Code for finding the accuracy of Playing Musical Instruments classes

```
pattern = 'Action_Type/4_music/{classname}/*.avi'


test2 = VideoFrameGenerator(
        glob_pattern=pattern,
        nb_frames=NBFRAME,
        batch_size=1,
        target_shape=SIZE,
        nb_channel=CHANNELS,
        transformation=data_aug,
        use_frame_cache=True)


model.compile(metrics= [tf.keras.metrics.TopKCategoricalAccuracy(k=1)])
loss,acc = model.evaluate(test2)
print('\nAccuracy for Playing_Muscial_Instruments : {}\n'.format(acc))
```

Figure 7.12.5: Code for finding the accuracy of Sports classes

```
test2 = VideoFrameGenerator(
        glob_pattern=pattern,
        nb_frames=NBFRAME,
        batch_size=1,
        # split=1,
        target_shape=SIZE,
        nb_channel=CHANNELS,
        transformation=data_aug,
        use_frame_cache=True)


model.compile(metrics= [tf.keras.metrics.TopKCategoricalAccuracy(k=1)])
loss,acc = model.evaluate(test2)
print('\nAccuracy for Sports : {}\n'.format(acc))
```

7.13 IMPORTING PACKAGES FOR DEPLOYMENT OF MODEL USING FLASK

We import all the necessary modules from Flask including request, url-for, redirect and render_template. We also import secure_filename from werkzeug.utils to obtain the file name of the dynamically selected video from the deployed website. We also import WSGIServer from gevent.pywsgi to utilize its high speed and thread pooling.

Figure 7.13: Code for importing necessary packages for Flask

```
# Flask utils
from flask import Flask, redirect, url_for, request, render_template
from werkzeug.utils import secure_filename
from gevent.pywsgi import WSGIServer
```

7.14 DEFINING THE FLASK APPLICATION

We define the flask application in the following way. We also define all the necessary parameters like the size of frames (which is 224*224), number of color channels (which is 3), the number of frames and others. We also define and sort the classes in the dataset in order. We also define the pre-processing for frames or images using the ImageDataGenerator in keras.preprocessing.

Figure 7.14: Code for defining the Flask application

```
# Define a flask app
app = Flask(__name__)


classes = [i.split(os.path.sep)[1] for i in glob.glob('test/*')]
classes.sort()

SIZE = (224, 224)
CHANNELS = 3
NBFRAME = 5
BS = 8

data_aug = keras.preprocessing.image.ImageDataGenerator(
    zoom_range=.1,
    horizontal_flip=True,
    rotation_range=8,
    width_shift_range=.2,
    height_shift_range=.2)
```

7.15 DEFINING THE PREDICTION FUNCTION FOR FLASK APPLICATION

The function defined below makes predictions for the given video and returns the argument with the highest confidence score from the list generated by the predict function. First the path of the video is given as an argument, so from this path we navigate to the video and provide it as a pattern. This pattern is processed through the VideoFrameGenerator and the video is converted to a sequence of frames that can now be given as input to the model. Then we load the model that we saved using load_model. Then we perform model.predict which generates a list of confidence scores for each of the available classes. We then use the argmax function from numpy to identify the highest confidence score and return that index.

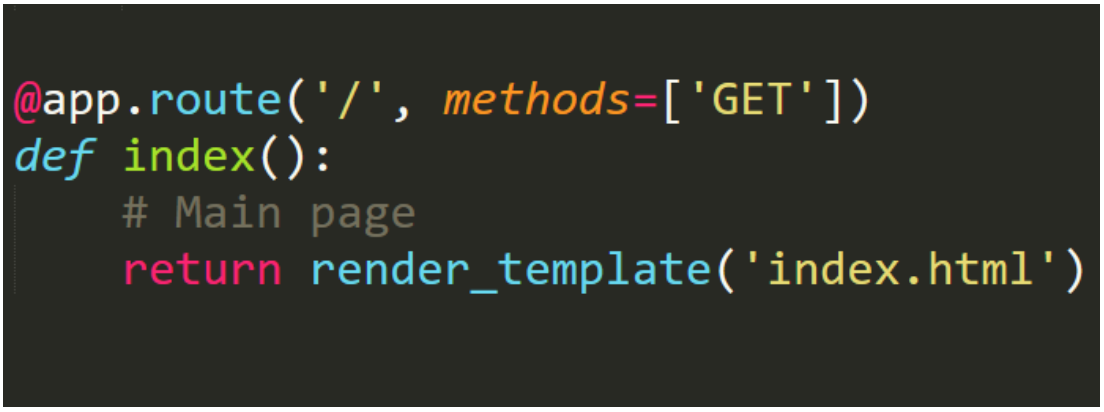
Figure 7.15: Code for defining the prediction function Flask application

```
def model_predict(path):

        pattern = path
        pattern2 = pattern[74:]
        pattern3 = "uploads/"+pattern2

        print('**********************')
        print(pattern3)
        print('**********************')

        test = VideoFrameGenerator(
        glob_pattern=pattern3,
        nb_frames=NBFRAME,
        batch_size=1,
        target_shape=SIZE,
        nb_channel=CHANNELS,
        transformation=data_aug,
        use_frame_cache=True)

        graph = tf.compat.v1.get_default_graph()

        model1 =  load_model('model_best.h5')

        print(model1.predict(test))
        preds = model1.predict(test)
        print(preds)
        print(np.argmax(preds))
        return np.argmax(preds)
        return None
```

7.16 DEFINING THE INDEX PAGE IN THE FLASK APPLICATION

By using app.route, we define the index page of the flask application. We use the GET method to achieve this. In the definition of the function, we use render_template that we impoirted earlier to set up the index.html webpage.

Figure 7.16: Code for defining the index page in the Flask application



7.17 DEFINING THE UPLOAD METHOD IN THE FLASK APPLICATION

Now we define the upload method which is used to upload the selected video to the uploads folder and secure its path and call the prediction function that we defined earlier. We use both GET and POST methods here because with GET we are obtaining the selected video file and with POST we are posting the prediction for the activity being performed in the video. We utilize the os module in python to navigate through the paths to process the file path. We use os.path.dirname to secure the local file path and then using os.path.join, we upload the file to the Uploads folder where we save the dynamically selected videos. We call the model_predict method and it returns the class index of the highest score. With that we navigate to that index in the classes list to find out the predicted class. Then we return this class name as string to the web page.

Figure 7.17: Code for defining the upload method in the Flask application

```
@app.route('/', methods=['GET'])
def index():
    # Main page
    return render_template('index.html')


@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        # Get the file from post request
        f = request.files['file']

        # Save the file to ./uploads
        basepath = os.path.dirname(__file__)
        file_path = os.path.join(
            basepath, 'uploads', secure_filename(f.filename))
        f.save(file_path)

        # # Make prediction
        preds = model_predict(file_path)

        result = classes[preds]

        return result
    return None


if __name__ == '__main__':
    app.run(debug=True)
```

7.18 INDEX.HTML PAGE

We use the form tag to create a form from which we can access the files in the computer and then select and upload the avi files with the help of the input tag. Then we define a button for Predict on clicking which our flask application will return the predicted class which will be displayed on the page.

Figure 7.18: Code for index.html page

```
{% extends "base.html" %} {% block content %}

<h2>Video Activity recognition</h2>

<div>
    <form id="upload-file"
        method="post"
        enctype="multipart/form-data">

        <label for="imageUpload"
            class="upload-label">

            Choose...

        </label>
        <input type="file" name="file" id="imageUpload" accept=".avi">
    </form>
            <button type="button"
                class="btn btn-primary btn-lg
                "id="btn-predict">

                Predict!

            </button>

    </div>

    <div class="loader" style="display:none;"></div>

    <h3 id="result">
        <span> </span>
    </h3>

</div>

{% endblock %}
```

7.19 UPLOAD.JS

With the help of this JavaScript code, we carry out the uploading of the video from the background.

Figure 7.19: Code for upload.js

```
function readURL(input) {
if (input.files && input.files[0]) {
    var reader = new FileReader();

    reader.onload = function (e) {
        $('#blah')
            .attr('src', e.target.result)
            .width(150)
            .height(200);
    };

    reader.readAsDataURL(input.files[0]);
}
}
```

7.20 DATA ANALYSIS BY COMPARING ACCURACIES OF LSTM AND GRU PERFORMANCE

Once we obtain the accuracies of models with GRU and LSTM for THE UCF101 and HMDB51 datasets, we can compare them and plot them in graphical form to perform data analysis. We import pyplot from matplotlib for this. Here we plotted bar graphs for two entities for each combination - accuracy and loss. With the help of this graph we can make important inferences.

Figure 7.20: Code for comparing LSTM and GRU accuracies

```
[ ]  import numpy as np
     import matplotlib.pyplot as plt
```

```
     data = [[96.06/100, 97.03/100,79.65/100,70.03/100],
       [0.0996, 0.0883 , 0.5540, 0.8350]]

     algo = ['LSTM (UCF101)', 'GRU (UCF101)', 'LSTM (HMDB51)', 'GRU (HMDB51)']
```

```
     X = np.arange(4)
     plt.figure(figsize=(5, 4.25))
     plt.plot(algo,data[0], linewidth=0)
     plt.bar(X + 0.00, data[0], color = 'b', width = 0.25, label = "Accuracy")
     plt.bar(X + 0.25, data[1], color = 'r', width = 0.25, label = "Loss")
     # plt.xlabel('x - axis')
     # plt.ylabel('y - axis')
     plt.legend()

     plt.show()
```

7.21 DATA ANALYSIS BY COMPARING THE ACCURACIES FOR DIFFERENT TYPES OF ACTIVITIES IN THE UCF101 DATASET

Similar to the last segment, we can plot a bar graph with the available data on accuracies for the different types of activity classes in the dataset. We store the values of those accuracies in a list and plot a bar graph against the names of the types of activities (which is stored in another list).

Figure 7.21: Code for comparing the accuracies for different types of activities in the ucf101 dataset

```
[ ]  data2 = [97.94, 95.05,98.29,99.37,94.91]
     algo2 = ['Body Motion Only',
              'Human-Human Interaction',
              'Human-Object Interaction',
              'Playing Musical Instruments',
              'Sports']
     plt.figure(figsize=(15, 8))
     plt.ylim(ymin=90)
     plt.ylim(ymax=100)
     plt.bar(algo2,data2)
```

CHAPTER 8

TESTING

8.1 COMPARING GRU AND LSTM

After training the model(s) by fitting them on the train sets, their loss and accuracies are tested through the validation sets. Table 6.1 and Figure 6.1.1 disclose the results for training ResNet+LSTM/GRU on the entire datasets of UCF101 and HMDB51. For the UCF101 dataset, the accuracy and loss are pretty close for both LSTM and GRU. GRU has accuracy and loss of 97.03 and 0.0883, respectively, while LSTM fetched 96.06 and a loss of 0.0996. While they are almost identical, it is essential to note that GRU's best epoch is 102, and LSTM is 110. GRU is quicker while providing slightly better results when training on nearly 7GB worth of data. However, it is a different scenario for HMDB51. LSTM gave an accuracy of nearly 80% with a 0.554 training loss. In contrast, GRU scored an accuracy of only 70% and a loss of 0.835. The gap is narrow if you consider validation accuracy and loss, but LSTM still performs better. GRU proved to be quicker, with its best epoch being 73, while LSTM's best epoch is 87. Though quicker, GRU's results are not as good considering the superior performance of LSTM.

Figure 8.1: Graph plotting accuracy and loss for LSTM/GRU+ResNet model

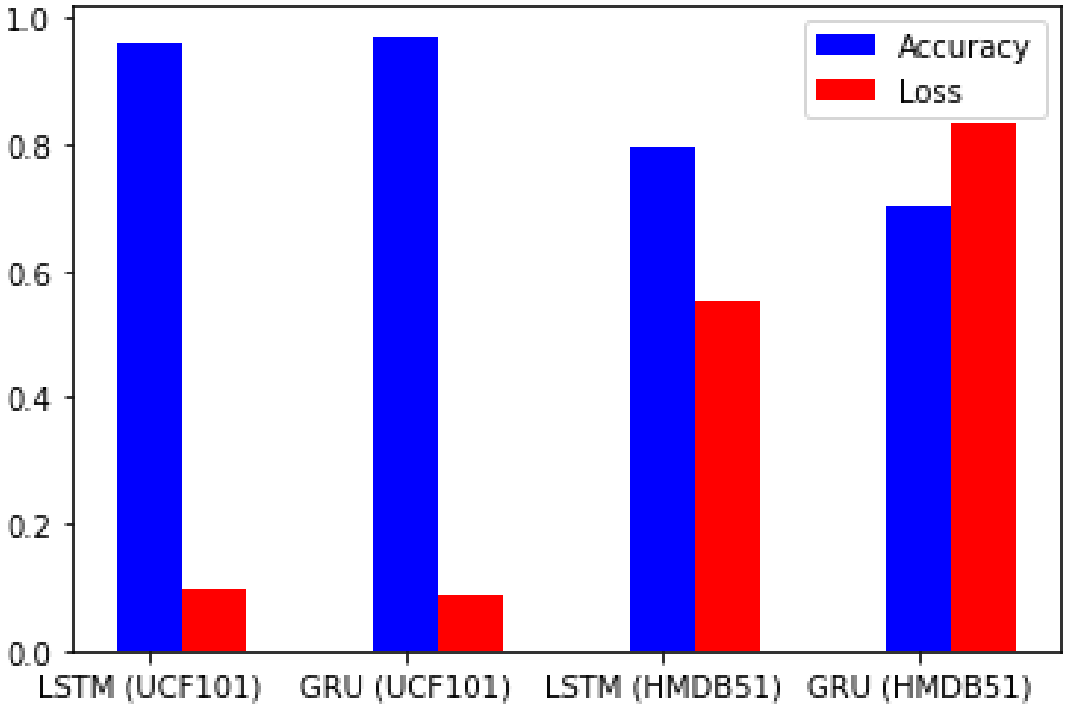


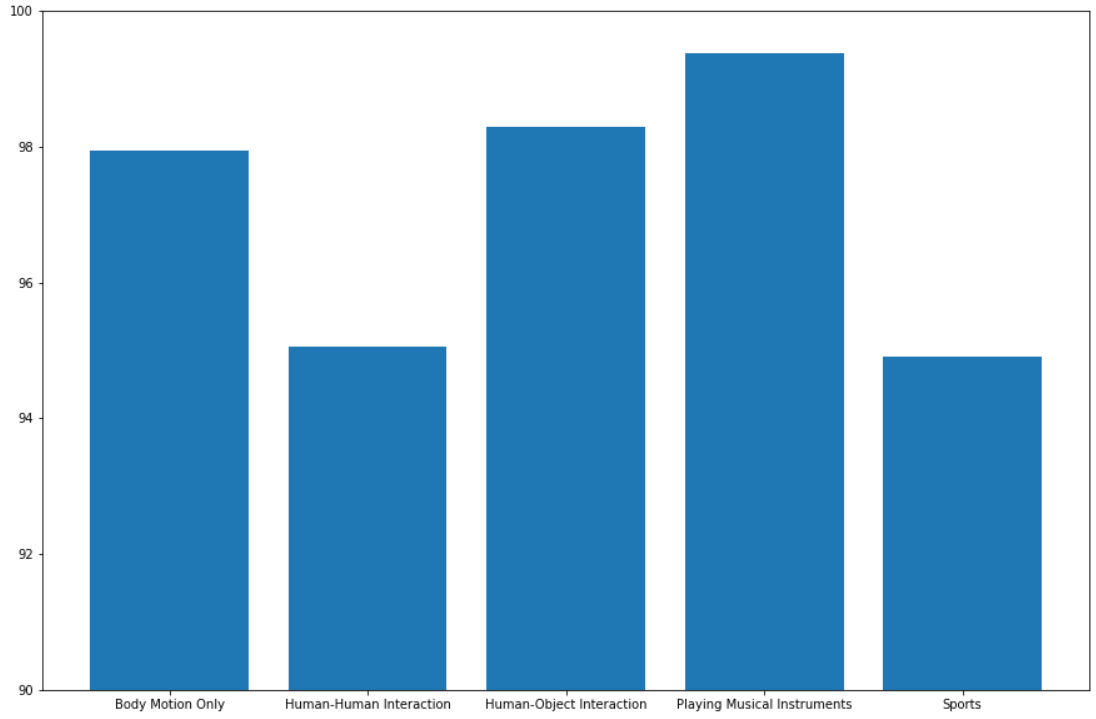
Table 8.1.1: Comparison of LSTM and GRU with Resnet on UCF101 and HMDB51 datasets

Model	Best Epoch	Accuracy (in %)	Loss	Validation Accuracy (in%)	Validation Loss
LSTM (UCF101)	110	96.06	0.0996	93.93	0.3750
GRU (UCF101)	102	97.03	0.0883	95.09	0.3364
LSTM (HMDB51)	87	79.65	0.5540	58.24	2.36
GRU (HMDB51)	73	70.03	0.8350	54.02	2.25

8.2 COMPARING THE ACCURACIES FOR DIFFERENT TYPES OF ACTIVITIES IN THE UCF101 DATASET

Figure 8.2 shows the accuracy of each of the five kinds of activities in the USF101 dataset. All the different breadths of activities achieved high accuracy. The lowest accuracy achieved is 94.91% for sports, and the highest achieved is 99.37% for playing musical instruments.

Figure 8.1.2: Graph comparing the accuracy achieved on the different groups of activities for LSTM+ResNet model on UCF101 dataset



In addition to that, the accuracies achieved are not far from [4] 's flagship action recognition model, which has an accuracy of 98.69 for the UCF101 dataset and 85.10 for the HMDB51 dataset. Our model fares better than [14] 's accuracy of 91.21% on the UCF101 dataset, but it is interesting to note that [14] 's bi-directional LSTM structure fared a much better accuracy of 87.64% on the HMDB51.

8.3 SCREENSHOTS OF THE PREDICTIONS MADE BY THE MODEL ON GIVEN INPUT VIDEOS

Figures 8..3.1 - 8.3.7 shows a few screenshots of the model's prediction on some videos. It is important to note that these are screenshots of videos and the actual output is for the entire video and not just the provided screenshot.

Figure 8.3.1: Model's prediction for a video on Archery

Figure 8.3.2: Model's prediction for a video on Trampoline Jumping

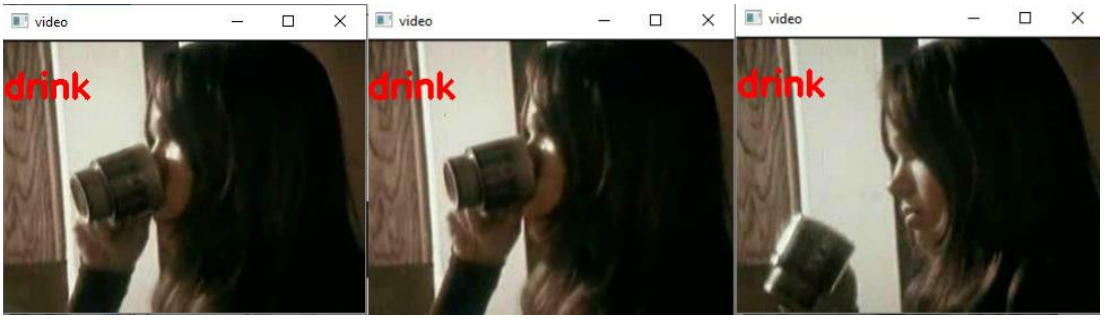
Figure 8.3.3: Model's prediction for a video on Walking with Dog

Figure 8.3.4: Model's prediction for a video on Cricket Bowling

Figure 8.3.5: Model's prediction for a video on Playing Guitar

Figure 8.3.6: Model's prediction for a video on Running

Figure 8.3.7: Model's prediction for a video on Drinking



8.4 SCREENSHOTS OF THE FLASK APPLICATION

The following figures are screenshots of the flask application built for this project. It includes a basic webpage, an upload screen and finally the webpage with the output i.e with the predicted class.

Figure 8.4.1: Screenshot of the index page of the flask application

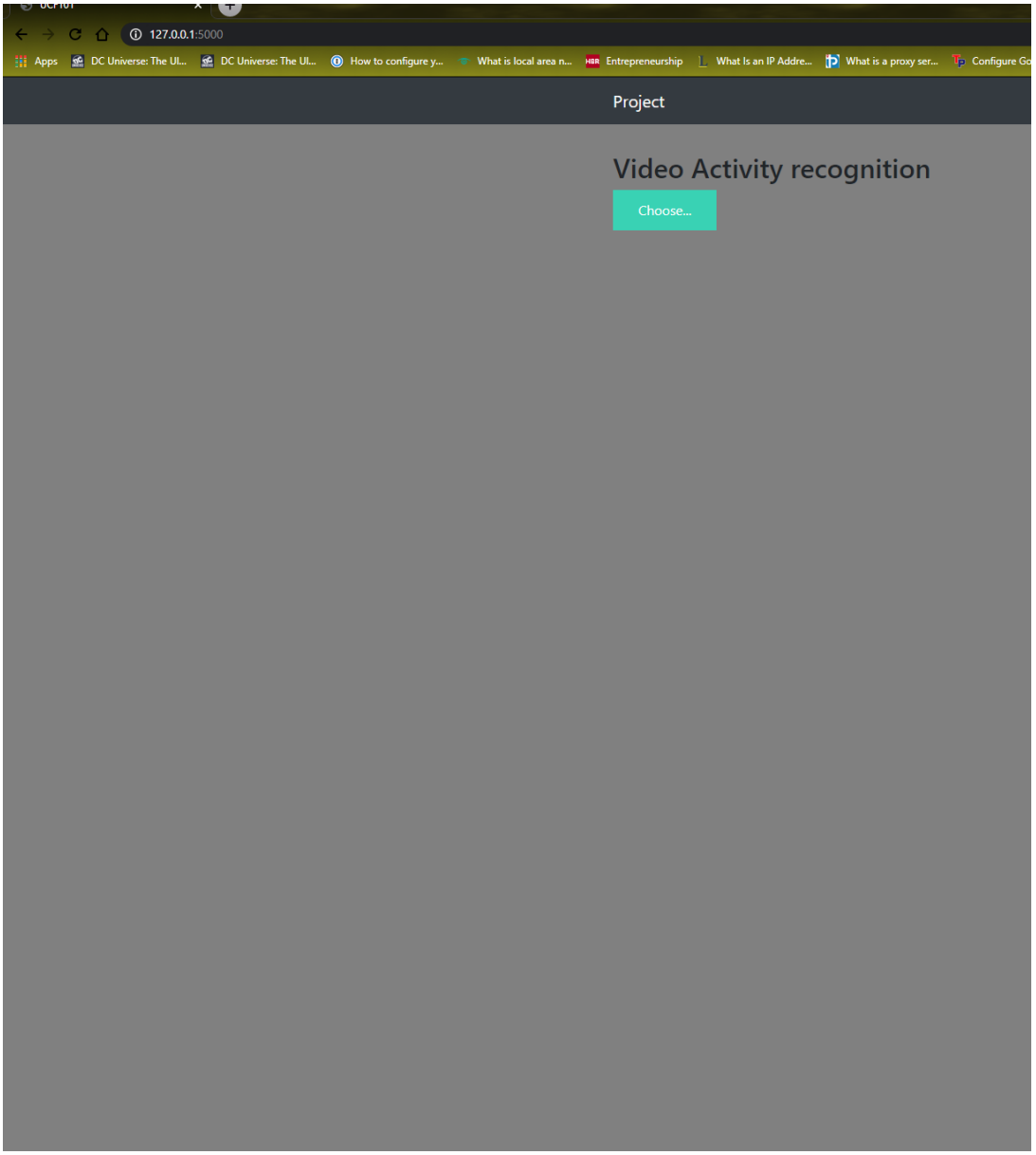


Figure 8.4.2: Screenshot of the upload page where an archery video is being uploaded

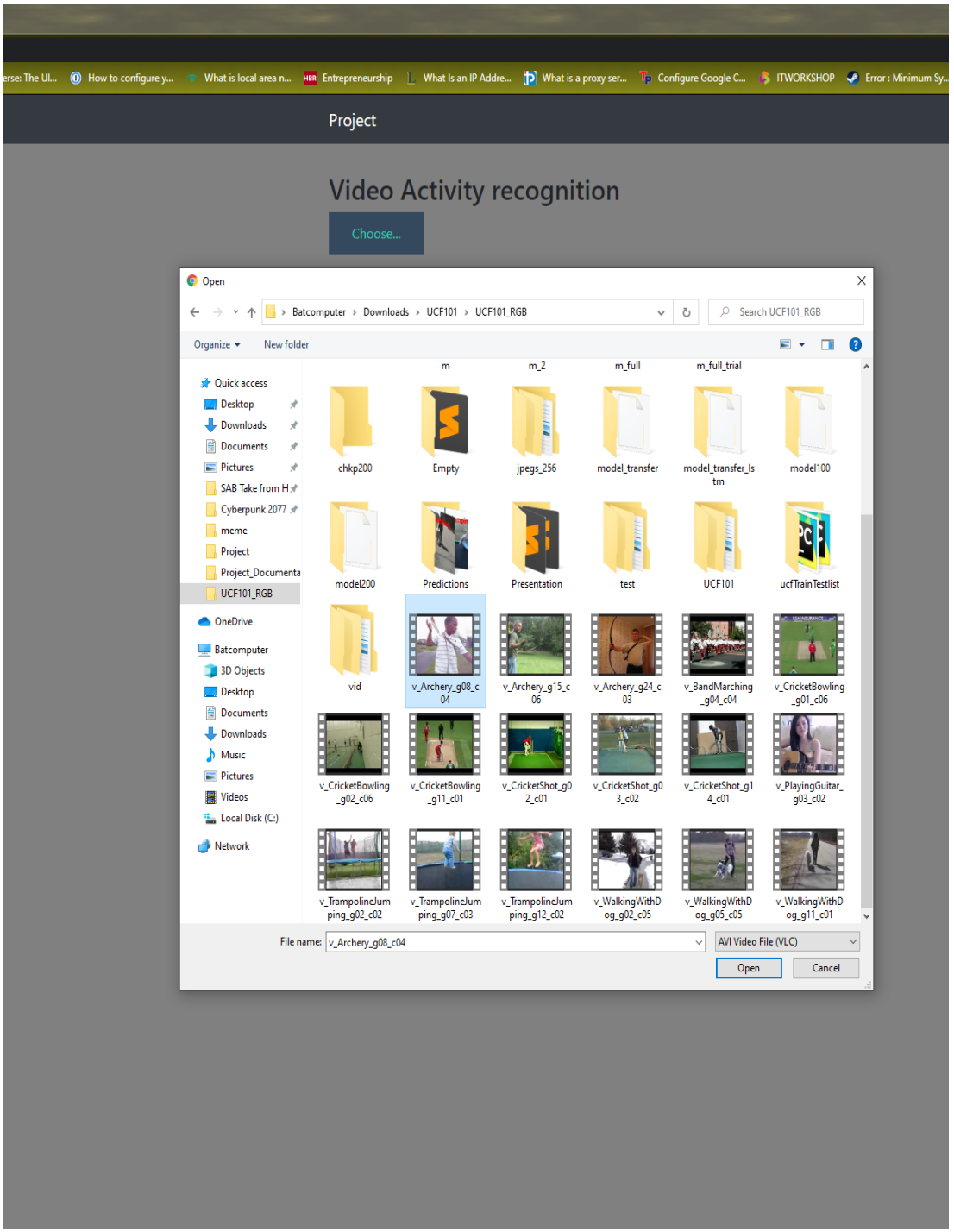


Figure 8.4.3: Screenshot of the page after loading video with the predict button

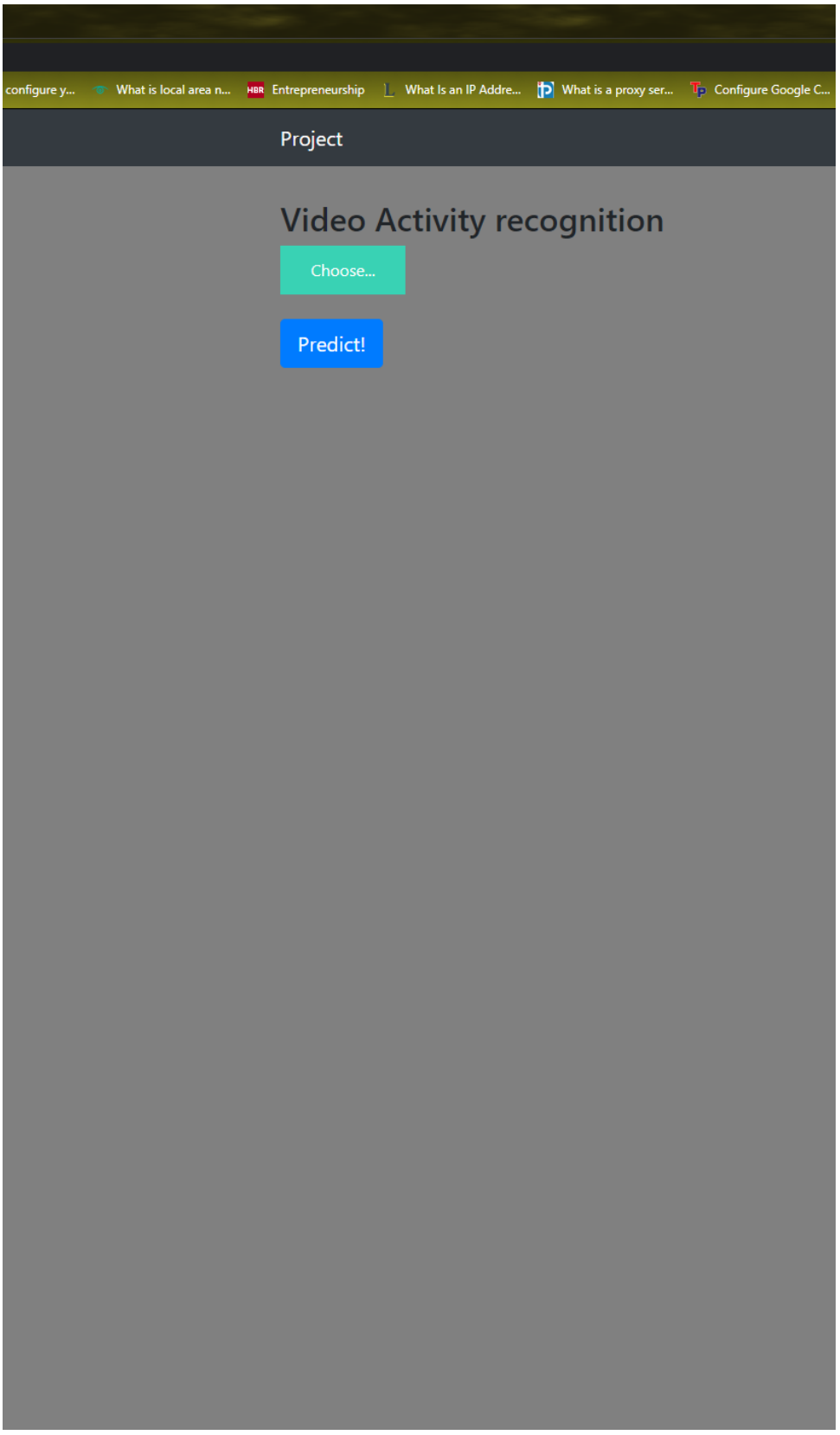


Figure 8.4.4: Screenshot of the page when model is loading the answer

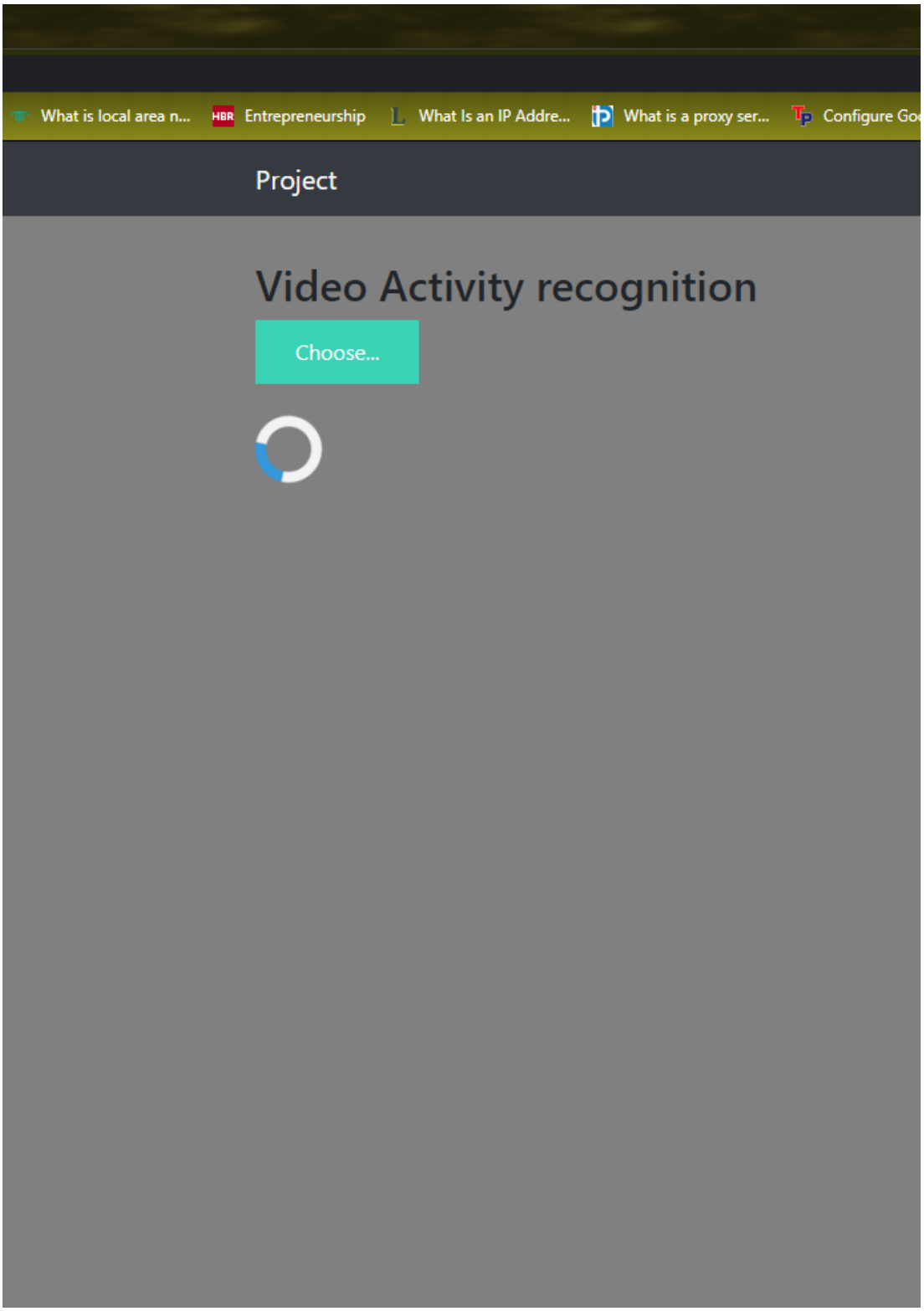


Figure 8.4.5: Screenshot of the page when model predicts archery

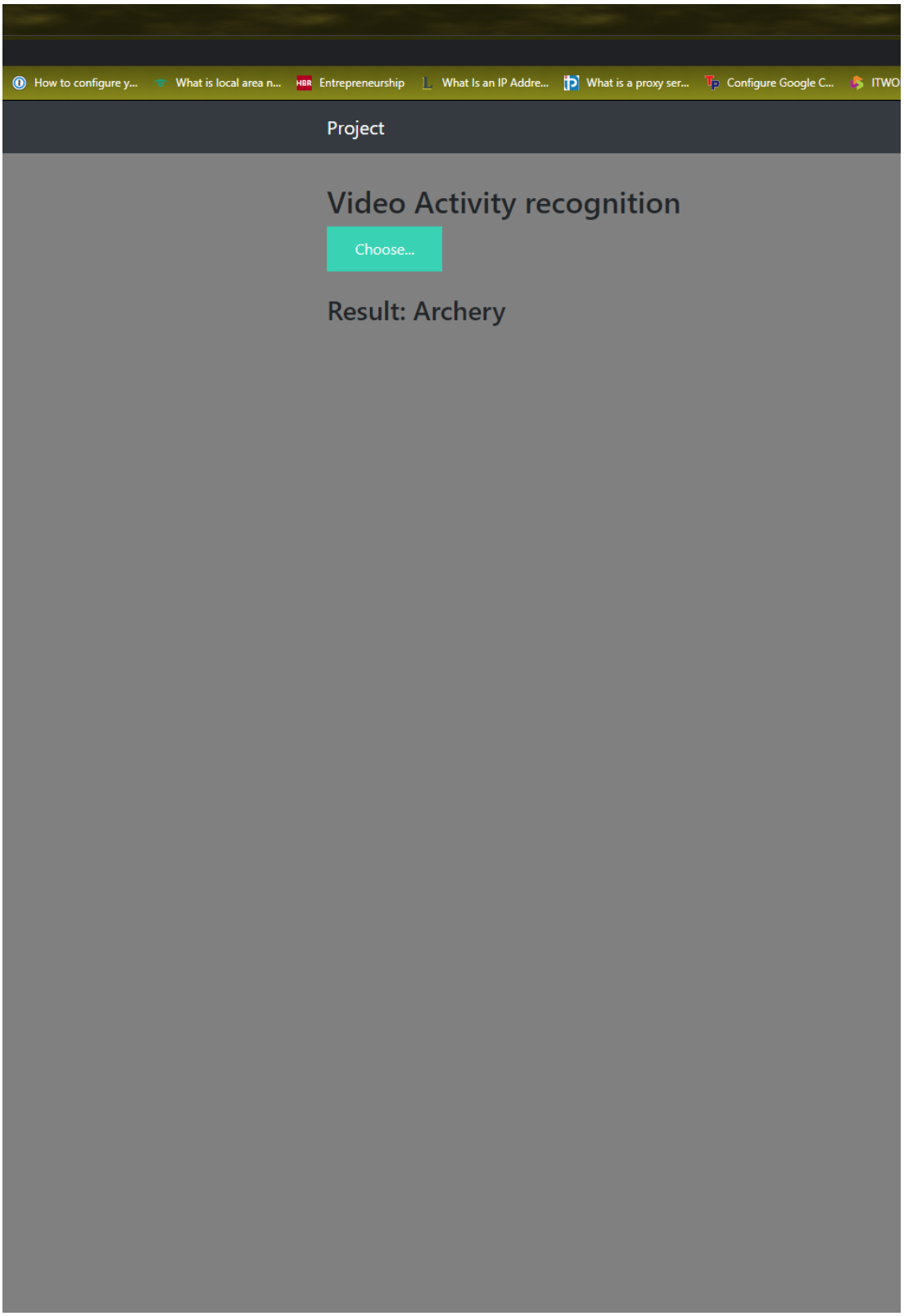


Figure 8.4.6: Screenshot of the upload page where an guitar playing video is being uploaded

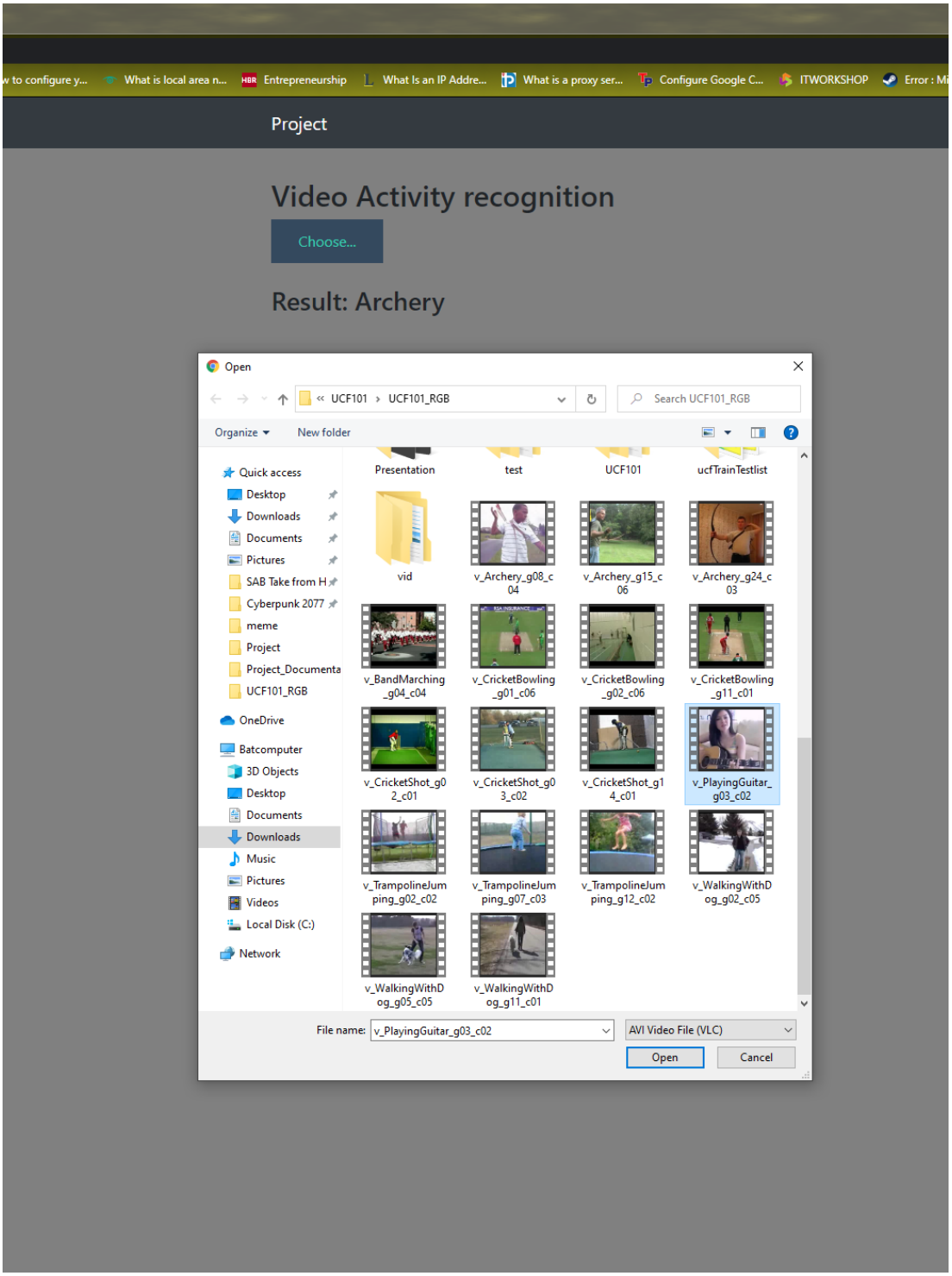


Figure 8.4.7: Screenshot of the page after loading video with the predict button

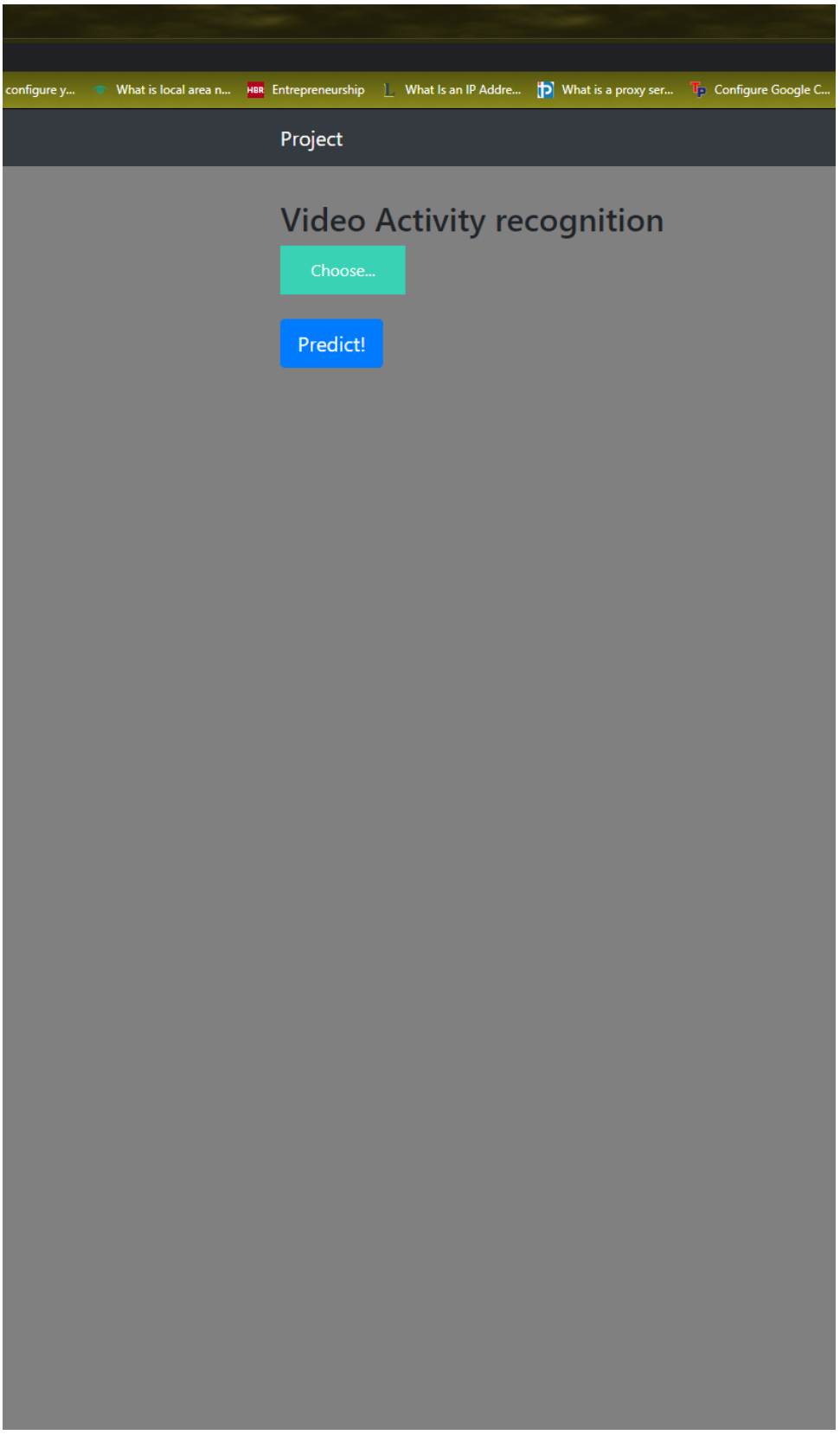


Figure 8.4.8: Screenshot of the page when model predicts PlayingGuitar

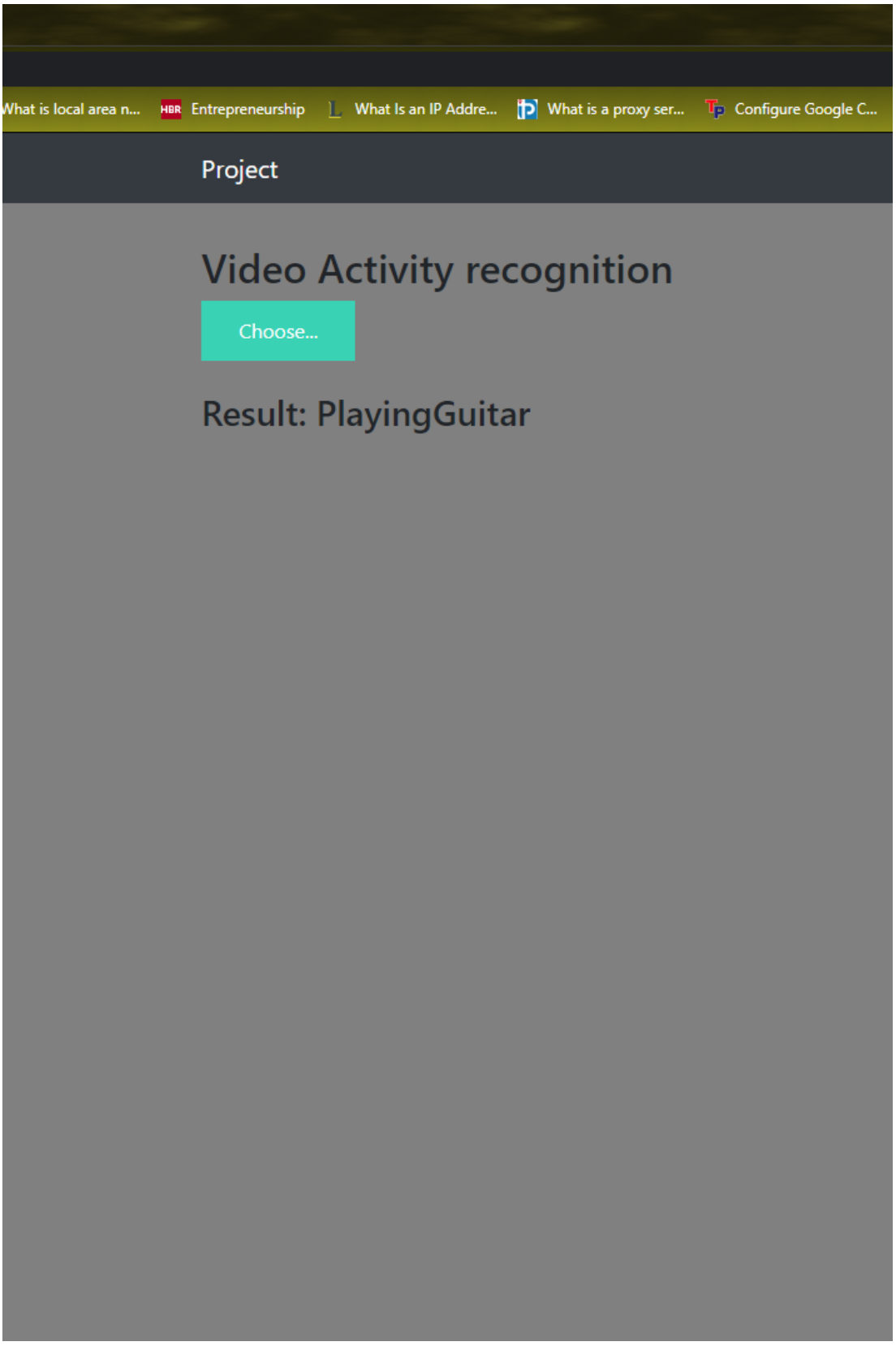


Figure 8.4.9: Screenshot of the upload page where a cricket shot video is being uploaded

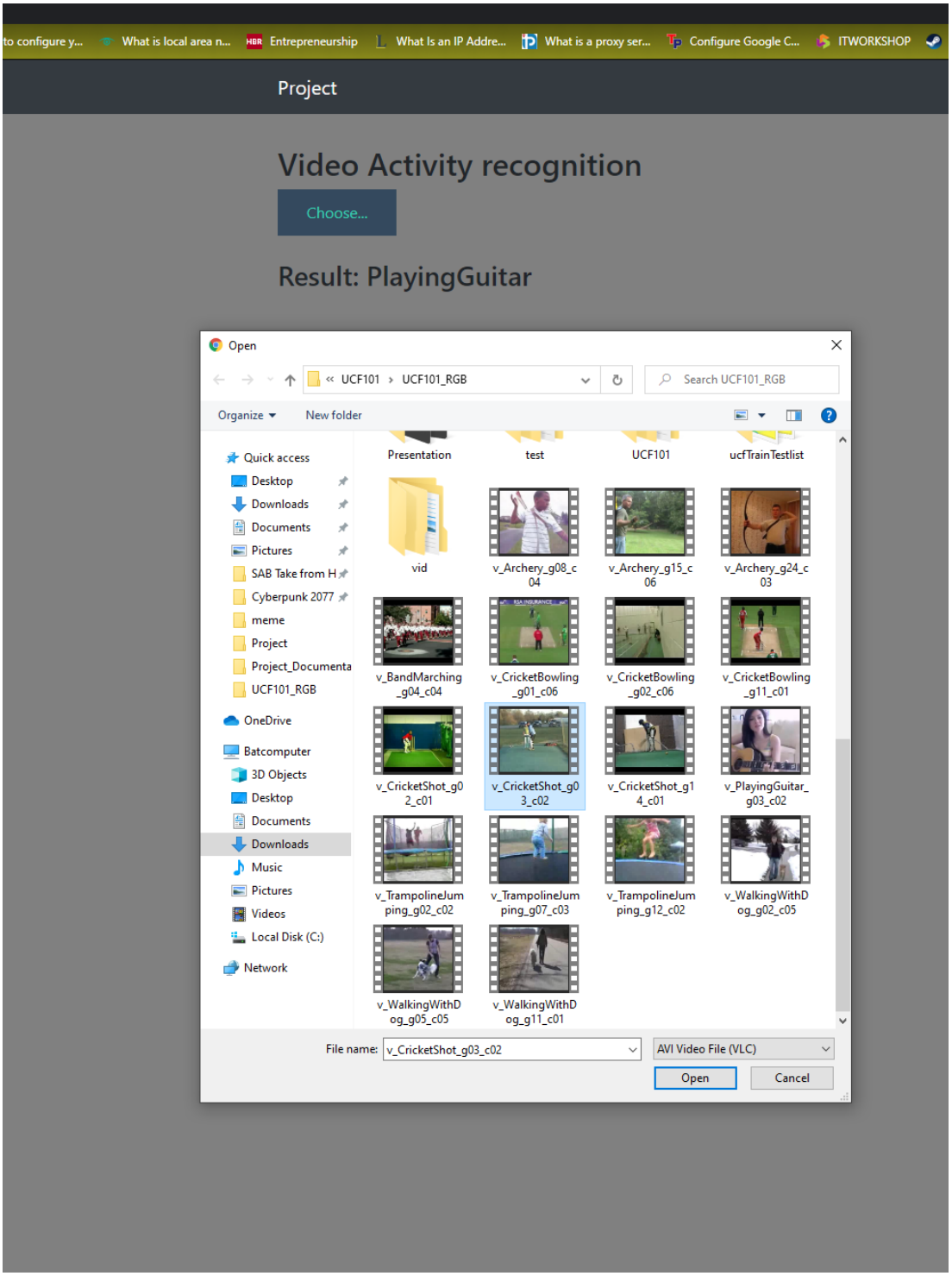


Figure 8.4.10: Screenshot of the page after loading video with the predict button

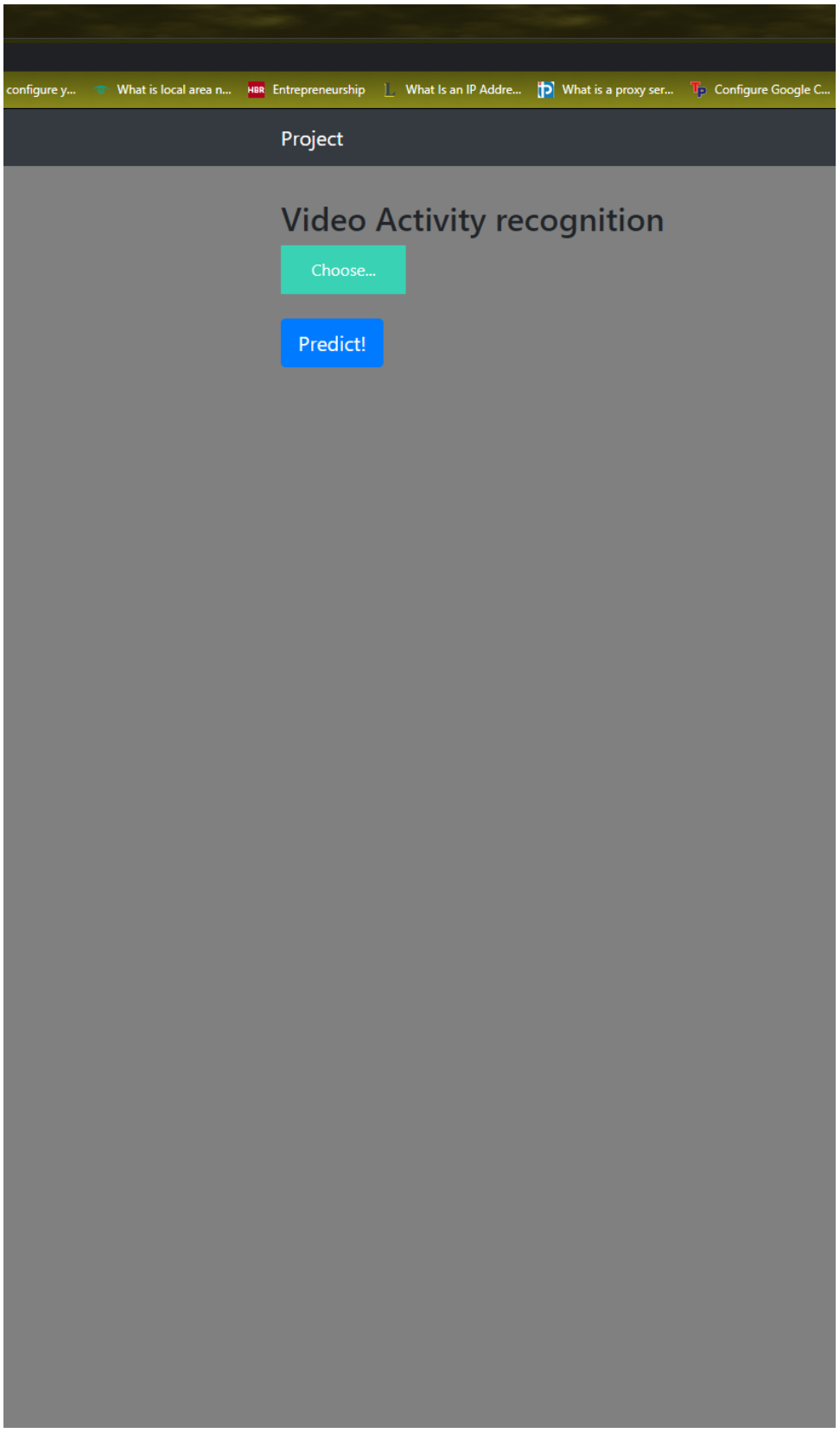


Figure 8.4.11: Screenshot of the page when model predicts CricketShot

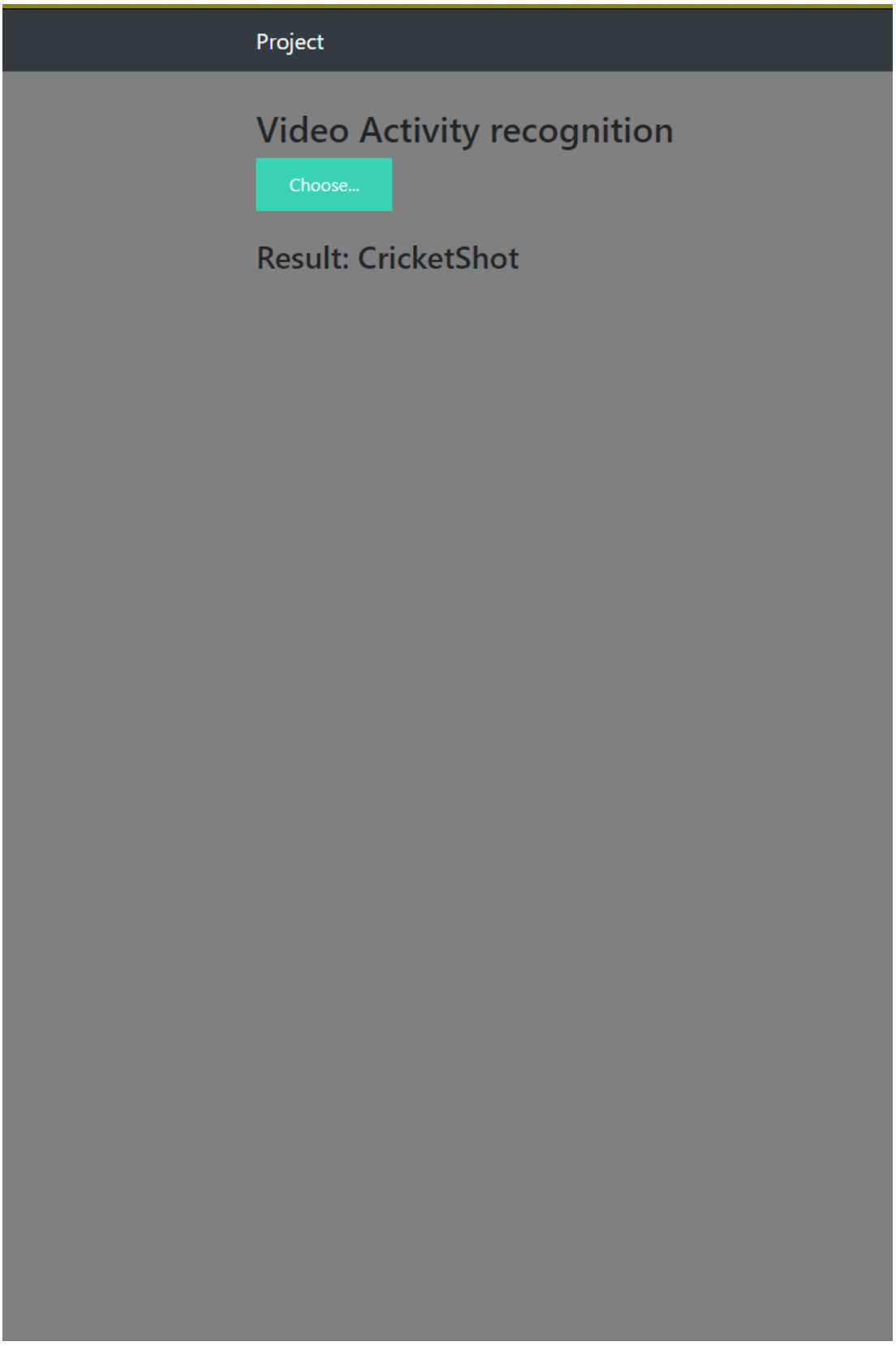


Figure 8.4.12: Screenshot of the command prompt in the background where it prints all predictions before passing the index of the predicted class to the web page

```
127.0.0.1 - - [12/May/2021 23:12:23] "←[37mPOST /predict HTTP/1.1←[0m" 200 -
**********
uploads/v_CricketShot_g03_c02.avi
**********
Total data: 1 classes for 1 files for train
[[9.1726458e-14 6.3364608e-10 2.5124223e-09 1.9143120e-17 1.2608406e-17
  3.1742511e-19 1.0694556e-02 6.0581549e-08 3.8415755e-12 3.3134330e-16
  2.3945375e-16 1.8206545e-15 9.4578963e-18 1.2991786e-16 1.6367323e-14
  5.3813763e-22 9.1449478e-16 1.7040368e-14 1.4092224e-17 1.5442403e-07
  1.8348413e-17 3.4931189e-17 1.1744164e-03 9.7623402e-01 2.9562128e-18
  9.5984384e-20 1.7470825e-16 4.4200457e-10 1.0309837e-02 8.8132301e-13
  3.4602490e-07 3.0788774e-20 1.3613098e-04 6.1361089e-18 8.4337780e-11
  1.2280730e-16 3.7398516e-20 2.8856776e-15 2.1051992e-15 4.3108602e-13
  1.4785692e-13 8.6949033e-16 1.2398582e-11 2.2631362e-16 2.9170121e-06
  2.5968452e-10 1.1690676e-08 5.3427599e-14 1.0696446e-18 8.6464667e-19
  6.4485988e-07 5.0387434e-13 4.1056848e-20 1.6864184e-14 1.6979740e-13
  4.1424405e-06 1.7511089e-20 3.9209400e-12 3.2788979e-13 1.6982329e-17
  4.9433537e-15 1.1619167e-11 1.3645389e-11 9.9344072e-07 8.3012945e-12
  1.1872255e-13 9.0817066e-17 4.2177091e-14 2.3707364e-21 5.7877161e-19
  7.8928086e-13 9.1354945e-17 2.2865466e-18 1.7834835e-18 1.9214600e-17
  3.5608051e-17 5.4337815e-11 2.8362143e-10 5.9693085e-04 5.0544392e-18
  1.2456897e-18 2.2556527e-18 1.1503257e-20 5.1009694e-09 6.9623649e-09
  4.1281286e-19 8.5916137e-14 4.7184314e-18 4.6834887e-15 9.0362363e-08
  1.2827610e-07 8.0539833e-04 2.2979418e-09 4.0056135e-17 1.2615857e-18
  8.4245431e-20 2.3166152e-12 3.5989451e-14 1.3015069e-13 9.3916919e-10
  3.9017730e-05]]
[[1.32840647e-18 1.13839900e-13 1.07469790e-10 1.10630346e-24
  2.31902519e-26 7.54365634e-29 2.09389655e-05 4.95613099e-13
  1.77202379e-20 1.11928918e-23 1.36380610e-24 1.12807286e-22
  6.59000950e-24 3.28751598e-23 1.88850007e-18 5.02463356e-32
  3.26296882e-21 9.66462577e-20 2.80129315e-26 1.32385791e-08
  1.84965893e-24 2.98324860e-25 1.38738533e-05 9.99950767e-01
  5.67608532e-26 1.45484838e-29 1.30258894e-25 2.36761035e-16
  4.06265826e-06 3.07973842e-21 4.52501995e-13 1.07231235e-29
  9.37549339e-06 1.79564621e-24 6.50312722e-17 1.38890360e-22
  8.33053173e-29 2.39692970e-21 8.80874263e-20 1.03284324e-21
  1.15759847e-22 7.60249741e-23 8.92583940e-14 3.08652005e-24
  9.53184255e-13 5.41993276e-14 2.42967313e-11 2.28944039e-17
  2.81949576e-27 8.01586603e-27 1.06273097e-13 3.54837529e-18
  3.29815616e-30 1.01656717e-20 3.53431011e-19 1.15261493e-07
  1.88363942e-30 1.43394261e-16 1.29661345e-20 3.13354410e-26
  7.13243460e-24 3.06997876e-14 2.49302608e-17 7.16229576e-11
  8.32315432e-16 1.33745670e-19 1.01836332e-22 8.75349217e-23
  1.32362274e-31 8.66890712e-26 2.92548773e-20 7.38062651e-25
  2.04818844e-26 1.21832603e-26 5.05304163e-25 2.41160695e-26
  2.39185045e-14 4.74907819e-13 1.69916134e-07 1.48498961e-25
  9.80435660e-28 1.08865992e-27 2.85069410e-29 2.61232702e-12
  1.15490384e-14 3.01966460e-28 4.27613603e-21 4.60091259e-27
  1.79964674e-21 1.01533896e-12 6.54349838e-11 9.63262092e-09
  4.38693314e-15 2.08283389e-25 2.47464125e-27 6.05906776e-30
  1.80984456e-19 6.98321706e-20 5.07593127e-19 7.38215980e-14
  8.02815748e-07]]
23
127.0.0.1 - - [12/May/2021 23:13:31] "←[37mPOST /predict HTTP/1.1←[0m" 200 -
```

CONCLUSION AND FUTURE WORK

When comparing LSTM and GRU in activity recognition, the results are nothing short of surprising. While the preconceived consensus is that GRU is preferable for smaller datasets and LSTM for the larger ones, this project's findings contrast that assumption. GRU performed slightly better and faster than LSTM when training on UCF101, a large dataset that is nearly 7GB in size. However, LSTM is significantly superior when dealing with HMDB51, a 2GB dataset. Another essential factor we should consider is the differences between the two datasets. UCF101 has a wide range of classes focusing on various activities, while HMDB51 focuses more on human movement. It is not unusual that GRU didn't perform better on the HMDB51 dataset, as it has a simpler architecture, and LSTM has the internal structure to deal with the complexity of the similarity of the actions.

GRU's performance on the UCF101 dataset proves that it can handle large amounts of data. However, that depends on the data itself. HMDB51, though a smaller dataset, has classes like run and walk, which, when stripped down to frames, look similar. There is massive scope for improvement for GRU in distinguishing such complex data. A customized and capable GRU model can potentially provide equivalent or better results than LSTMs while being faster.

REFERENCES

1. Khurram Soomro, Amir Roshan Zamir and Mubarak Shah, UCF101: A Dataset of 101 Human Action Classes From Videos in The Wild, CRCV-TR-12-01, November, 2012.
2. H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: A Large Video Database for Human Motion Recognition. ICCV, 2011
3. Zhao, Rui & Ali, Haider & van der Smagt, Patrick. (2017). Two-stream RNN/CNN for action recognition in 3D videos. 4260-4267. 10.1109/IROS.2017.8206288.
4. Kalfaoglu, Esat & Kalkan, Sinan & Alatan, A.. (2020). Late Temporal Modeling in 3D CNN Architectures with BERT for Action Recognition. 10.1007/978-3-030-68238-5_48.
5. Howard, Andrew & Zhu, Menglong & Chen, Bo & Kalenichenko, Dmitry & Wang, Weijun & Weyand, Tobias & Andreetto, Marco & Adam, Hartwig. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.
6. He, Kaiming & Zhang, Xiangyu & Ren, Shaoqing & Sun, Jian. (2016). Deep Residual Learning for Image Recognition. 770-778. 10.1109/CVPR.2016.90.
7. He, Kaiming & Zhang, Xiangyu & Ren, Shaoqing & Sun, Jian. (2016). Identity Mappings in Deep Residual Networks. 9908. 630-645. 10.1007/978-3-319-46493-0_38.
8. Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.
9. Cho, Kyunghyun & van Merriënboer, Bart & Bahdanau, Dzmitry & Bengio, Y.. (2014). On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. 10.3115/v1/W14-4012.
10. Hubel, D.H. & Wiesel, T.N.. (1959). Receptive Fields of Single Neurons in the Cat's Striate Cortex. JP. 148. 574-591.
11. Fukushima, Kunihiko. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological cybernetics. 36. 193-202. 10.1007/BF00344251.
12. Lecun, Yann & Bottou, Leon & Bengio, Y. & Haffner, Patrick. (1998). Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE. 86. 2278 - 2324. 10.1109/5.726791.
13. LeCun, Y. & Cortes, C. (2010), 'MNIST handwritten digit database', .
14. Ullah, Amin & Ahmad, Jamil & Muhammad, Khan & Sajjad, Muhammad & Baik, Sung. (2017). Action Recognition in Video Sequences using Deep Bi-directional LSTM with CNN Features. IEEE Access. PP. 1-1. 10.1109/ACCESS.2017.2778011.
15. P.Raja Mani, Dr.S.Praveen Kumar, P.Surya Chandra,G.Santoshi Kumari. A Robust Copy Move Tamper Detection and Localization Method for Image Forensics . JCR. 2020; 7(17): 1522-1530. doi:10.31838/jcr.07.17.192

16. Dr. S.Praveen Kumar , Dr. K.Naveen Kumar, Dr. Y.Srinivas, Dr. G.V.S Rajkumar 2020. Defect Detection On Manufacturing Product Images by Applying Weekly Detector Algorithm using Deep Learning Technology. International Journal of Advanced Science and Technology. 29, 7 (May 2020), 186 - 194.
17. Praveen Kumar S, Srinivas Y, Bhargav K, “An n-gram analysis approach for sharing authentication of data using model based techniques”, Test Engineering and Management, 2020.
18. Praveen Kumar S, Sahithi Choudary A, “An Innovative ModelBased Approach for CreditCard Fraud Detection Using Predictive Analysis and Logical Regression.”, International Journal of Innovative Technology and Exploring Engineering (IJITEE), Scopus, 2019, 8 , 1683-1688
19. Praveen Kumar S, Srinivas Y, Vamsi Krishna M, “Latent Semantic Indexing based Approach for Identification of Frequent Itemset.”,Jour of Adv Research in Dynamical & Control Systems, Scopus, 2018, Vol. 10, 686-690.
20. Praveen Kumar S, Srinivas Y, Vamsi Krishna M, “A mechanism for identifying the guilt agent in a network using vector quantization and skew Gaussian distribution.” International Journal of Engineering & Technology, Scopus, 2018, 7, 149-151
21. Praveen Kumar S, Srinivas Y, Vamsi Krishna M, “A Data Leakage Identification System Based on Truncated Skew Symmetric Gaussian Mixture Model.”, International Journal of Recent Technology and Engineering (IJRTE), Scopus, 2018, 7, 111-113
22. S. Praveen Kumar, Gandham Bharathi, Chintala Neeraj, Killamsetty Akilesh, Surendranath, "An Efficient Key Transmission Model in Attribute Based Encryption Scheme in Cloud Computing", International Journal of Creative Research Thoughts (IJCRT), ISSN:2320-2882, Volume.6, Issue 1, pp.1134-1137, March 2018, Available at :http://www.ijcrt.org/papers/IJCRT1872344.pdf
23. Kumar, S. (2018). IoT-based Temperature and Humidity Monitoring System using Raspberry pi. International Journal for Research in Applied Science and Engineering Technology. 6. 288-291. 10.22214/ijraset.2018.4052.
24. S Kumar, Y Srinivas, D Suba Rao, Ashish Kumar A Novel Model for Data Leakage Detection and Prevention in Distributed Environment International Journal of Engineering and Technical Research Posted: 2016
25. S. Praveen Kumar, Y. Srinivas, M. Ranjan Senapat and A. Kumar, "An enhanced model for preventing guilt agents and providing data security in distributed environment," 2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES), Paralakhemundi, India, 2016, pp. 337-339, doi: 10.1109/SCOPES.2016.7955847.
26. A Complete Introspection on Big Data and Apache Spark – S.Praveen Kumar, Dr. Y .Srinivas, Dr. D.Subbarao ,Ashish Kumar—http://www.ijsdr.org/papers/IJSDR1604006.pdf
27. D. R. Giri, S. P. Kumar, L. Prasannakumar and R. N. V. V. Murthy, "Object oriented approach to SQL injection preventer," 2012 Third International Conference on Computing, Communication and Networking Technologies (ICCCNT'12), Coimbatore, India, 2012, pp. 1-7, doi: 10.1109/ICCCNT.2012.6395979.

28. L. P. Kumar, S. P. Kumar, D. R. Giri and V. Jayavani, "Object oriented approach to prefix based fast mining of closed sequential patterns," 2012 Third International Conference on Computing, Communication and Networking Technologies (ICCCNT'12), Coimbatore, India, 2012, pp. 1-6, doi: 10.1109/ICCCNT.2012.6395980.
29. Kumar S P , K. Anusha, R.Venkata Ramana, “A Novel Approach to Enhance Robustness in Steganography Using Multiple Watermark Embedding Algorithm”. International Journal of Soft Computing and Engineering (IJSCE), ISSN: 2231-2307 (Online), Volume-1, Issue-1, March 2011
30. S. P. Kumar, K. N. Kumar, S. Sreenadh, B. Aravind, K. H. Kumar, Novel Advent for Add-On Security by Magic Square Intrication, Global Journal of Computer Science and Technology,Vol. 11, No. 21, December 2011.