

Human Action Recognition

Human action recognition is an important task in computer vision, with many potential applications in fields such as surveillance, robotics, and healthcare. In this project, we propose to develop a human action recognition model using the VGG16 convolutional neural network architecture. We will begin by collecting a dataset of videos depicting a variety of human actions, such as walking, running, and jumping. We will then pre-process the video data and extract features using the VGG16 model. We will train a classifier in the upcoming iterations (such as a support vector machine or a softmax regression model) on the extracted features to classify the videos into their corresponding action categories. We will evaluate the performance of the model on a held-out test set, measuring metrics such as accuracy, precision, recall, and F1 score. We will also compare the performance of the VGG16 model to other state-of-the-art models for human action recognition, such as ResNet and Inception. Finally, we will explore potential extensions to the model, such as incorporating temporal information across multiple video frames or fine-tuning the pre-trained VGG16 model on our specific dataset. Overall, this project will provide valuable insights into the effectiveness of the VGG16 model for human action recognition and may have important implications for a range of practical applications.

When deciding whether to use the VGG16 model for human action recognition or another model, it is important to consider a range of factors, such as model complexity, training time, and accuracy. One advantage of the VGG16 model is its simplicity and interpretability, which can make it easier to understand and debug the model. Additionally, the VGG16 model has been widely used and benchmarked on a range of image classification tasks, including human action recognition. However, it is important to note that the VGG16 model may not always be the best choice for human action recognition, depending on the specific requirements and characteristics of the dataset. Other models, such as ResNet and Inception, have been shown to achieve higher accuracy on certain benchmarks, and may be more suitable for specific types of actions or video data. Therefore, it is important to evaluate and compare the performance of multiple models on the specific dataset and application before selecting a final model for human action recognition. This may involve comparing metrics such as accuracy, precision, recall, and F1 score, as well as considering factors such as computational resources and training time.

Ultimately, the choice of model for human action recognition should be based on a careful consideration of the trade-offs between model complexity, accuracy, and practical considerations such as implementation and computational requirements.

Test and train datasets are commonly used in machine learning (ML) projects to evaluate the performance of a model on unseen data. The train dataset is used to train the model, while the test dataset is used to evaluate the model's performance and generalization ability on new data.

The train dataset is typically a large set of labeled data that is used to train the ML model. The model learns patterns and relationships in the data during training and is optimized to minimize the difference between its predicted output and the actual output. The train dataset is usually split into smaller subsets (such as a validation set) to monitor the model's performance during training and adjust its hyperparameters.

The test dataset is a separate set of data that is held out from the training process and used to evaluate the model's performance on new, unseen data. The test dataset should be representative of the distribution of the data that the model is expected to encounter in the real world. The test dataset should also be independent of the train dataset, meaning that it should not contain any data that was used to train the model.

In our project we have a train dataset - contains all the images that are to be used for training the model. In this folder, we will find 15 folders namely - 'calling', 'clapping', 'cycling', 'dancing', 'drinking', 'eating', 'fighting', 'hugging', 'laughing', 'listening_to_music', 'running', 'sitting', 'sleeping', 'texting', 'using_laptop' which contain the images of the respective human activities.

The training data has a CSV file which locates the filename present in the training directory and label associated with it.

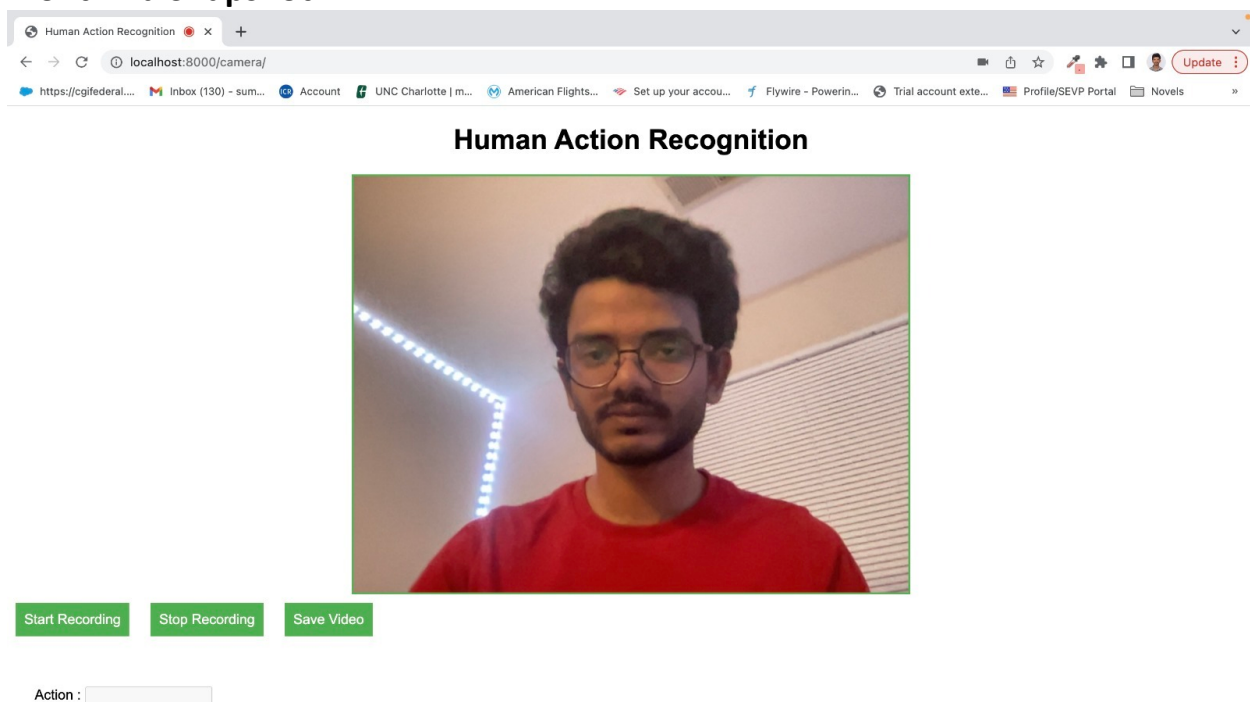
For the purpose of testing, a front-end web application was developed. The application allows users to input actions by clicking a button, which triggers the camera to capture a video of the user's actions.

The front-end application was developed using modern web development technologies, such as HTML5, CSS, and JavaScript, and utilizes the latest browser APIs for video capture and transmission. The application code was committed to a

Git repository for collaboration and can be accessed by authorized team members for testing and evaluation purposes.

We chose this dataset to train the model because of the diverse actions it must train a model.

Front-End Snapshot:

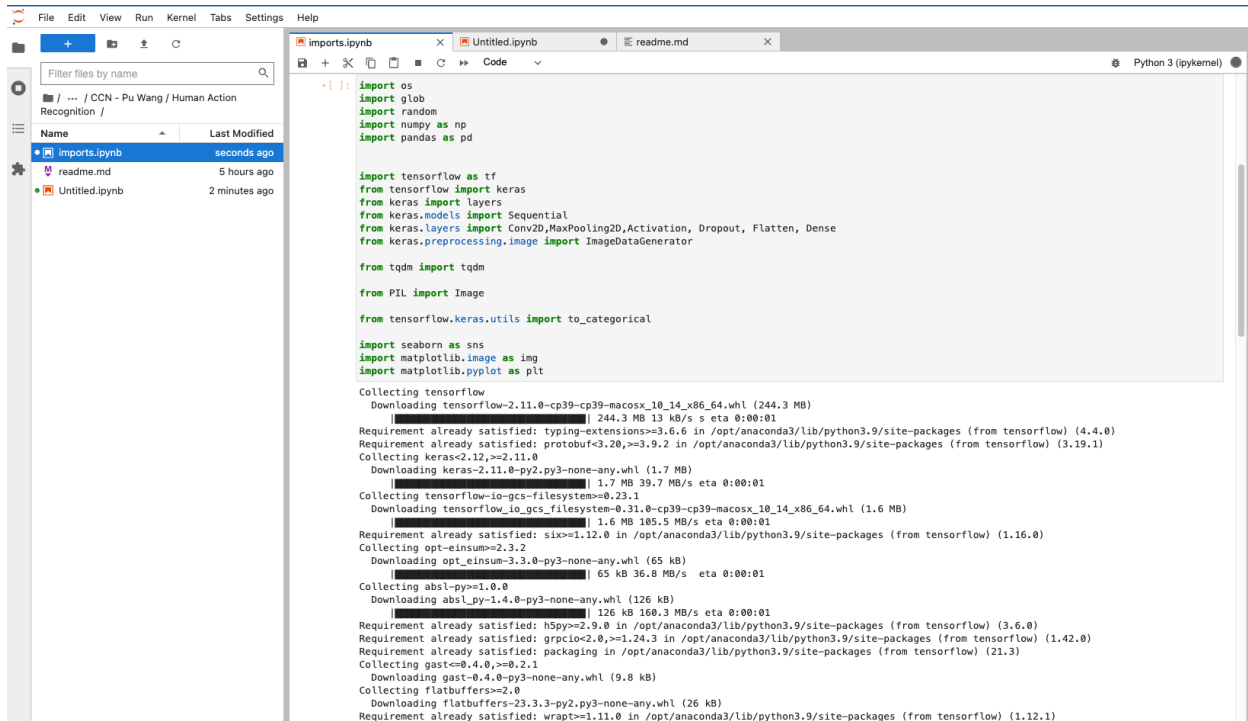


This web page contains a video element, three buttons, and a text input field. We have created JavaScript code event listeners for the "Start Recording", "Stop Recording", and "Save Video" buttons. When the "Start Recording" button is clicked, the code requests permission to access the user's camera and microphone, and then starts displaying the live video stream in the video element. It also creates a MediaRecorder object that records the video stream and saves it in an array of chunks. When the "Stop Recording" button is clicked, the MediaRecorder object is

stopped, and the recorded video is stored in a Blob object. Finally, when the "Save Video" button is clicked, the recorded video is downloaded by the user as a WebM file. This video data will be transmitted to the server for further processing.

Python Scripting:

We have used JupyterLab for python scripting.



The screenshot shows the JupyterLab interface. On the left is a file browser showing files like 'imports.ipynb', 'readme.md', and 'Untitled.ipynb'. The main area displays a Python script in a code editor. The script imports various modules including os, glob, random, numpy, pandas, tensorflow, keras, and matplotlib. Below the code editor, a terminal window shows the output of the script, which includes the installation progress for several packages like tensorflow, keras, and tensorflow-io-gcs-filesystem.

```
import os
import glob
import random
import numpy as np
import pandas as pd

import tensorflow as tf
from tensorflow import keras
from keras import layers
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense
from keras.preprocessing.image import ImageDataGenerator

from tqdm import tqdm

from PIL import Image

from tensorflow.keras.utils import to_categorical

import seaborn as sns
import matplotlib.image as img
import matplotlib.pyplot as plt

Collecting tensorflow
Downloading tensorflow-2.11.0-cp39-cp39-macosx_10_14_x86_64.whl (244.3 MB)
Requirement already satisfied: typing-extensions>=3.6.6 in /opt/anaconda3/lib/python3.9/site-packages (from tensorflow) (4.4.0)
Requirement already satisfied: protobuf<3.20,>=3.9.2 in /opt/anaconda3/lib/python3.9/site-packages (from tensorflow) (3.19.1)
Collecting keras-2.12.0-py2.py3-none-any.whl (1.7 MB)
Requirement already satisfied: tensorflow>=2.11.0 in /opt/anaconda3/lib/python3.9/site-packages (from tensorflow) (2.11.0)
Collecting tensorflow-io-gcs-filesystem==0.23.1
Requirement already satisfied: six>=1.12.0 in /opt/anaconda3/lib/python3.9/site-packages (from tensorflow) (1.16.0)
Collecting opt-einsum==2.3.2
Requirement already satisfied: numpy>=1.19.0 in /opt/anaconda3/lib/python3.9/site-packages (from tensorflow) (1.24.0)
Collecting absl-py==1.0.0
Requirement already satisfied: h5py>=2.9.0 in /opt/anaconda3/lib/python3.9/site-packages (from tensorflow) (3.6.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /opt/anaconda3/lib/python3.9/site-packages (from tensorflow) (1.42.0)
Requirement already satisfied: packaging in /opt/anaconda3/lib/python3.9/site-packages (from tensorflow) (21.3)
Collecting gast==0.4.0-py3-none-any.whl (9.8 kB)
Collecting flatbuffers==23.3.3-py2.py3-none-any.whl (26 kB)
Requirement already satisfied: wrapt>=1.11.0 in /opt/anaconda3/lib/python3.9/site-packages (from tensorflow) (1.12.1)
```

Libraries Used:

1. **os**: This module provides a way of using operating system dependent functionality like reading or writing to the file system.
2. **glob**: This module provides a way to retrieve files/pathnames matching a specified pattern. It is often used for batch processing of files.
3. **random**: This module implements pseudo-random number generators for various uses.
4. **numpy**: This is a fundamental package for scientific computing with Python. It provides support for arrays and matrices, as well as mathematical functions to operate on them.
5. **pandas**: This is a library used for data manipulation and analysis. It offers data structures and functions needed to manipulate numerical tables and time series.

6. **tensorflow_addons**: This is a library built on top of TensorFlow that contains a collection of custom TensorFlow operators and high-level APIs for building deep learning models.
7. **tensorflow**: This is a popular machine learning library that is widely used for building deep learning models.
8. **keras**: This is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano.

In the further iterations, we will try to enhance the front end by including new components and will start working on server-side architecture implementation.