HUMAN ACTION RECOGNITION IN THE DARK: A SIMPLE EXPLORATION WITH LATE FUSION AND IMAGE ENHANCEMENT

Joshua Robert John Kennedy
Department of Electrical and Electronics Engineering
Nanyang Technological University
50 Nanyang Avenue, Singapore 639798
joshuaro001@e.ntu.edu.sg

ABSTRACT

Human Action Recognition (HAR) has been a fundamental challenge in the field of computer vision for over a decade, with diverse applications ranging from intelligent surveillance to human-computer interaction. However, recognizing actions in real-world scenarios, especially under low-light conditions, presents unique challenges characterized by reduced visibility and information loss. In this project, I propose a simple yet effective approach for human action recognition in low-light environments. Firstly, I address the treatment of low-light videos to enhance information extraction. Subsequently, I employ various machine learning and deep learning methods to develop robust models. This includes leveraging the power of Vision Transformers for end-to-end training, along with traditional machine learning techniques such as Extreme Learning Machines (ELM), Naive Bayes, and Logistic Regression models. Furthermore, I conduct a comparative analysis of the performance of these different models using metrics tailored to the given dataset. This comparative evaluation reveals the most effective model for performing HAR in low-light conditions, offering valuable insights into the efficacy of different methodologies.

1 INTRODUCTION

Human Action Recognition (HAR) has been a perennially captivating topic in the realm of computer vision, owing to its wide-ranging applications. In its early stages, researchers faced limitations such as sparse data availability, low-quality videos, and computational inefficiencies, which constrained their exploration. However, the landscape has evolved significantly with the proliferation of smartphones equipped with high-resolution cameras, coupled with the advent of powerful GPUs and efficient CPUs. These advancements have revolutionized research capabilities, providing unprecedented opportunities to tackle more complex challenges in the HAR domain.

One such formidable challenge is the recognition of human actions in low-light conditions, a scenario often encountered in real-world settings where perfect footage is not always attainable due to various environmental factors. In this project, we delve into the intricacies of processing low-light videos, extracting meaningful features from them, and employing diverse learning methods for action classification. The ultimate goal is to discern the most effective model for this task based on the provided dataset. For this endeavor, we utilize a custom dataset comprising six action classes, each containing 25 videos, resulting in a total of 150 training videos. Additionally, the validation set comprises 96 videos encompassing all six classes.

So this project contain five major section. Section 2.1 is about frame sampling, Section 2.2 is about feature extraction and, Section 2.3 is about classification using the extracted, Section 2.4 is about does Image enhancement has any impact on, and Section 2.5 is about how to end-end training and. Based on this we will conclude which model is doing well in this dataset and why it is performing well compare to other models.

2 METHODOLOGY

2.1 FRAME SAMPLING

The first step in classification the video is, to do frame sampling. As video are the has 3 dimension which is height, width and time. To process this video, we sample some frames from the video and use that frames to process and train our model. In simple term these frames will represent the whole video. As because the video has approximately 60 frames per second now a days, and if we use that video to train our model it will be computationally expensive. So there are two main sampling method which are commonly used by everyone, those are uniform frame sampling and random frame sampling. As name suggest in uniform frame sample we will sample frame from uniform distance across the whole video As shown in Fig 1, but in random sampling we will randomly choose the frames As shown in Fig 2.

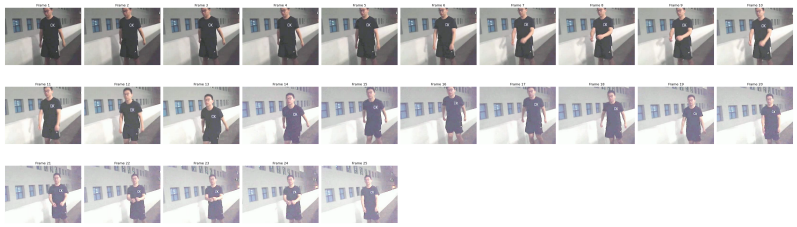


Figure 1: Uniform Sampling

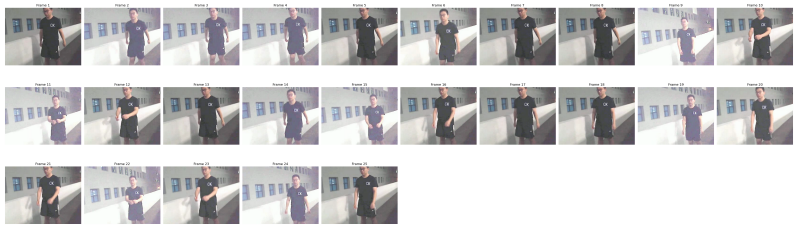


Figure 2: Random Sampling

In this project we are gonna to use uniform sampling with the Max Frame Length of 50. As because as we working with much smaller dataset to attain maximum accuracy. Adding to that in the dataset, videos given for the training has a maximum length of 2 second. But in the validation set the video or much bigger compare to the training dataset. Hence to if we choose 25 frame for training the model, then for prediction we have to give same size as a input for our model. By doing so we will lose some data, so i am trying to give much features to the model to workup with. Adding to that most of the video doesn't capture full action done by the human, in the sense most of the video shown liked cropped. The reason i didn't use the random sampling is because of its randomness hence we can't predict which frame are going to separate from the video. Adding to that as most of the action in this dataset only last for few frames, so there is high possibility that this method wont pickup that frames dues it randomness. So, The model wont detect the action. Hence to maximise the efficiency of the model i am going with Uniform frame sampling and also with 50 frame per video. The formula for do that is shown in the Eq 1. After collection every frames from the videos , I will save it as a list for further use.

$$\text{Sampling Interval} = \frac{\text{Total Number of Frames}}{\text{Desired Number of Sampled Frames}} \tag{1}$$

2.2 FEATURE EXTRACTION

The second and the most important step in this project is to extract the features from the frame. As because we will use only those features to train our models. Adding to that in this project we are

trying to classify the low light video, hence it is very important to give the model some very useful information about the video so that it can efficiently classify our videos.

2.2.1 NORMALIZATION

First we will start with normalization, as because we have to normalize our frame so that the feature extractor will extract much information from the frames. The video contains frames with pixel values ranging from 0 to 255 (8-bit depth) in each channel (Red, Green, Blue). Pixel brightness is represented by these numbers, where 255 indicates brightness and 0 indicates darkness. Pixel values, on the other hand, can differ greatly across frames and have a broad range. We normalise the pixel values in order to get them ready for training a learning model. In particular, we set the numbers so that each channel has a zero-mean and a unit standard deviation. We utilise a reference mean for the dark video frames of [0.07, 0.07, 0.07] and a standard deviation of [0.1, 0.09, 0.08] for this project. Normalisation modifies the pixel value scale, which impacts colour saturation and the relative brightness of various regions in the video. Normalisation can increase or decrease colour saturation by scaling the results to a standard range. As a result, our film looks better visually and has more information than the low-light, raw footage. The normalisation procedure also guarantees that the input data is reliable and appropriate for deep learning model training. Training stability promotes improved performance and generalisation.

Figure 3: Unprocessed Frame

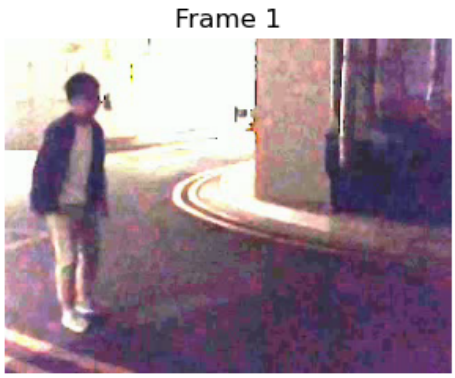


Figure 4: Normalized Frame

The following images show the importance of normalizing especially for the dark images. Because as you see the unprocessed frame as shown in Fig 3 is more dark and doesn't hold much information. After done normalization on the same dark image, now the image get more brighter, rich in contrast and also more visible as shown in Fig 4 compare to the unprocessed frame. Adding to that the normalized frame hold more information.

2.2.2 RESIZING

After normalization of frame we have to resize the frame the frame. As because as now a days most of the video are capture in high resolution which mean 1920 x 1080 pixels per frames on average. Hence to process these data need high computational cost and take more time to train.So, we have to resize the frame which is computational not expensive and also hold enough information to process that frame. Hence Resizing the frame is the process of altering an image's or video's spatial dimensions. In order to accomplish the following goals, we will resize each frame in this project to attain: standardisation, which will guarantee that every frame has the exact same dimensions; model compatibility, since some machine learning models or algorithms need certain dimensions for input data; and performance enhancement, which will involve resizing frames to smaller dimensions in order to speed up processing, particularly in applications that run in real time or scenarios where prompt response times are essential. Hence in this project we will resize the frame into 299 x 299 or 224 x 224 according to the model needs.

2.2.3 FEATURE EXTRACTION

Final and last step in this section is to extract the features from the resized frame. For that we are using some pre-trained models that are trained on large datasets and possess good generalizability so that we are using EfficientNet B7 in this project for feature extraction. There are several reasons why the pre-trained EfficientNet B7 model is used. First off, its performance on benchmark datasets has proven to be state-of-the-art, outperforming earlier designs in terms of accuracy and efficiency as shown in Fig 5. With millions of photos and dozens of classes, this model has been trained on massive datasets like ImageNet. We may take use of learnt representations and save time by using pre-trained models instead of requiring in-depth training on our particular dataset. When compared to other designs with comparable performance levels, EfficientNet B7 maintains its efficiency in terms of computing resources despite its size and complexity, which makes it adaptable to a range of applications and situations.

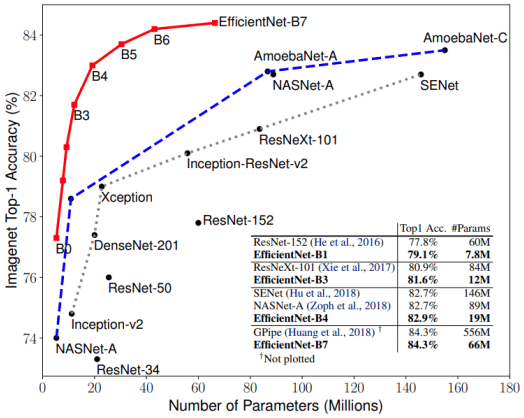


Figure 5: Accuracy Comparison of Different Model

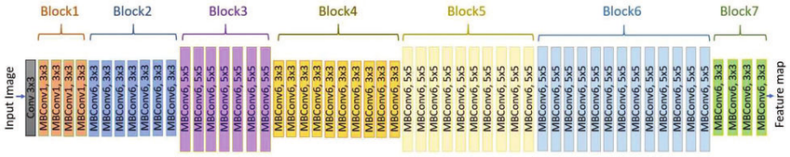


Figure 6: EfficientNet B7's Architecture

So the architecture of the EfficientNet B7 as follow as show in Fig 6: In this project, input photos with a resolution of 299x299 are accepted by the Input Layer. Convolutional Blocks, which are made up of many blocks with a number of convolutional layers within them, come next. These blocks use a mix of 3x3 and 1x1 convolutions to learn hierarchical characteristics from input photos. Subsequently, each convolutional block employs Bottleneck Blocks to minimise computational expenses while preserving representational capacity. Usually, 1x1, 3x3, and 1x1 convolutional layers are included, after which batch normalisation and activation functions such as ReLU are applied. Squeeze-and-Excitation (SE) Blocks adaptively re-calibrate feature maps depending on channel-wise relationships, assisting the model in focusing on relevant features. These blocks are inserted into certain convolutional blocks. The final step in the convolutional backbone process is Global Average Pooling (GAP), which reduces the spatial dimensions of the feature maps to a single vector. The process generates a fixed-size feature vector for every input picture by averaging the values of each feature map over all of its spatial dimensions. The number of channels in the final feature maps following global average pooling is represented by the feature extractor's output dimension, which normally ranges from 2560 to 4096. In our instance, we use global average pooling to produce final feature vectors with a size of 2048, which is the global average pooling layer in EfficientNet B7's default output dimensionality.

2.3 CLASSIFIER TRAINING AND EVALUATION

The final step in this process involves classification and evaluation of our models. Before that, we have to ensure that our data satisfies the model's criteria, we must first preprocess it. Following feature extraction, we are left with vectors of size (50, 2048), where 2048 is the number of features and 50 is the batch size (i.e., 50 frames were used for feature extraction). Nevertheless, a lot of machine learning functions, libraries, and algorithms demand that input data come in a certain format or structure. Consequently, we may preprocess our data into a format that is compatible with these techniques by flattening and reshaping arrays. By arranging every element of the input array into a single linear sequence, the technique known as "flattening" can convert a multi-dimensional array into a 1-dimensional array. Consequently, if we flatten a 2-dimensional array (50, 2048), the resultant size is 25x2048, or 51200 items.

2.3.1 MODEL SELECTION AND TRAINING

For classification, in this project we are using some machine leaning algorithm such as Naive Bayes, Logistic Regression and Extreme Learning Machine. For identity the models performance, based on this dataset.

Naive Bayes Model

Naive Bayes is a simple yet powerful probabilistic classifier based on Bayes' theorem with the "naive" assumption of feature independence. It calculates the probability of a given sample belonging to each class based on the features observed in the sample. Despite its simplicity, Naive Bayes often performs well in practice, particularly for sequential data's. It's computationally efficient and works well with high-dimensional data. It is also Assumes feature independence, which may not always hold true in practice.

Logistic Regression Model

Logistic Regression is a linear model used for binary classification tasks. It predicts the probability of a binary outcome using a logistic function, mapping input features to a range between 0 and 1. Despite its name, it's not a regression model but rather a classification method. Logistic Regression is interpretable, efficient, and extendable to multi-class problems. However, it assumes a linear relationship between input features and the log-odds of the outcome, which may not always be accurate in practice.

Extreme Learning Machine Model

Extreme Learning Machine (ELM) is a simple and efficient machine learning algorithm used for supervised learning tasks like classification and regression. Unlike traditional neural networks, ELM randomly generates and fixes weights connecting the input to hidden layer, avoiding iterative optimization during training. This approach makes ELM computationally efficient and fast-learning. While it may not perform as well as deep learning models on complex tasks, ELM is favored for its simplicity and ability to handle large datasets effectively. The cons of the model is ELM may not capture the optimal representation of the data, leading to suboptimal performance due to its fixed random weights.

Using the Scikit-learn library, we trained our data with classifiers such as Bayes, Logistic Regression, and ELM models mentioned above. Scikit-learn is a popular machine learning library in Python, offering simple and efficient tools for data analysis and modeling. With this library, we trained our data using above mentioned models, using features as input and corresponding labels (actions) as output.

2.3.2 EVALUATION

Evaluation is one the most important step, as because based on this step only we can come to know how well our model perform the classification task, and also we can also know the capability of different models in our dataset. As because most of model doesn't grantee any result we want to find the best model only by testing that model in our dataset. So we have trained the models using our dataset, and we will compare and evaluate the model performance using some metrics. They are

accuracy of the model calculated using the Eq 2, precision and recall of the model calculated using the Eq 3 , Eq 4 respectively, F1 scores of the model calculated using the Eq5. Using these metric we will know how well the model performed the classification task.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \times 100\% \tag{2}$$

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{3}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{4}$$

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5}$$

Logistic Regression Model

Logistic Regression models give the accuracy of 21.8 percentage with the F1 micro score as 21 percentage, F1 macro score as 18 percentage and precision and recall value as 5 percentage each. It also gives True negative as 7 percentage, False positive 5 percentage, False negative 3 percentage, True positive 5 percentage as shown in Fig 7.

Logistic Regression achieved the highest accuracy among the models, with relatively balanced precision and recall across all classes. It demonstrated better overall performance and balanced classification accuracy across different actions.

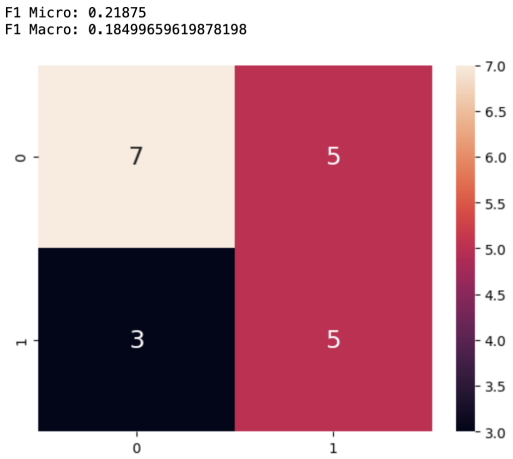


Figure 7: Logistic Regression Metrics

Naive Bayes Model

Naive Bayes models give the accuracy of 18.7 percentage with the F1 micro score as 18 percentage, F1 macro score as 13 percentage and precision and recall value as 5 percentage each. It also gives True negative as 8 percentage, False positive 5 percentage, False negative 2 percentage, True positive 5 percentage as shown in Fig 8.

Naive Bayes achieved a moderate accuracy, with a slightly lower F1 score compared to Logistic Regression. It exhibited comparable precision and recall to Logistic Regression but with slightly lower performance across all classes.

Extreme Learning Machine Model

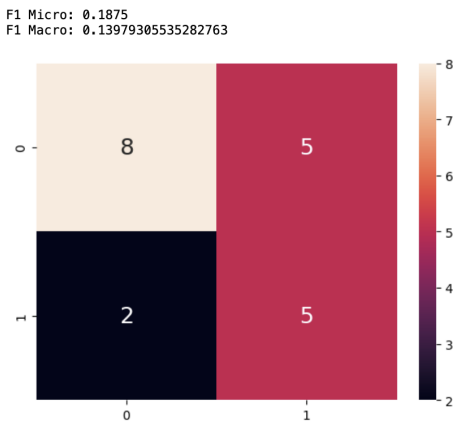


Figure 8: Naive Bayes Metrics

Extreme Learning Machine Model give the accuracy of 17.7 percentage with the F1 micro score as 17.7 percentage, F1 macro score as 16.4 percentage and precision and recall value as 4 percentage each. It also gives True negative as 2 percentage, False positive 1 percentage, False negative 4 percentage, True positive 1 percentage as shown in Fig 9.

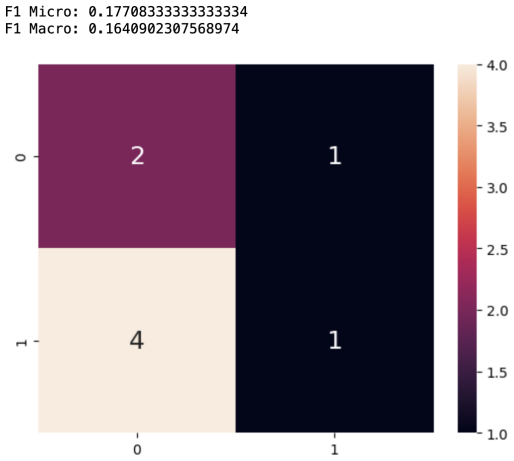


Figure 9: Extreme Learning Machine Metrics

ELM achieved the lowest accuracy among the models, with lower precision and recall rates compared to Logistic Regression and Naive Bayes. It struggled to effectively classify actions in low-light videos, possibly due to its simplicity and limited adaptability to the dataset's complexities.

Table 1: Evaluation Metrics for Different Models

Metric	**Logistic Regression**	**Naive Bayes**	**ELM**
Accuracy (%)	21.8	18.7	17.7
Precision (%)	5	5	4
Recall (%)	5	5	4
F1 Micro (%)	21.8	18.7	17.7
F1 Macro (%)	18.5	17.4	16.4

In summary, Logistic Regression demonstrated the best overall performance as shown in table 1, followed by Naive Bayes, while ELM lagged behind in accuracy and classification quality. The

precision and recall metrics provided insights into the models' strengths and weaknesses, highlighting the importance of selecting appropriate algorithms based on the specific characteristics of the dataset and task requirements. Adding to that its is evident that some of the action are not classified properly by the classifier which intern gives less F1 macro score, which calculate the precision of the each class. Also as we have only 150 videos in total for training the model with more constraints like 6 classes which mean every classes only has 25 video to train and also these are low light videos there is very less information compared to day light video. Adding to that most of the video are didn't capture the action of human with full body covered, mean most of the video are viewed as a cropped video hence we loss some valuable information's. It is worth noticing that in training video the action is performed only once and in evaluation video the action are done continuously. In the sense if we take a jumping action training video the person is only jumping one time, but in evaluation video person is jumping and also moving forward hence as we have less video, the model may found those videos as different, this may also cause in getting less accuracy.

2.4 Effects of Leveraging Image Enhancements

In this section we going to use some image enhancement techniques, on the same dataset. To do that we are using gamma correction. Gamma correction is an image processing method that modifies an image's contrast and brightness. It entails performing a nonlinear operation on the intensity values of the picture pixels, by Eq 6. Here G denotes output pixel value, and f denotes input pixel value. Here if the gamma ¿ 1, then it will darken the video by increasing contrast and decrease the brightness of the video, if the gamma ¡ 1 then it will increase the brightness and decrease the contrast the video. Hence, as our video are darker we used gamma value as 0.33.

$$G = f^{\gamma} \tag{6}$$

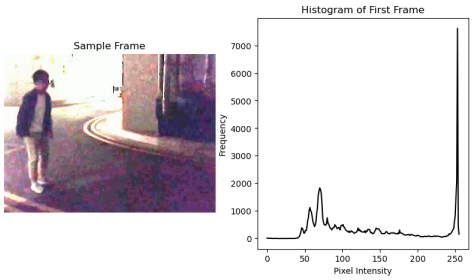


Figure 10: Histogram of Normalized Image

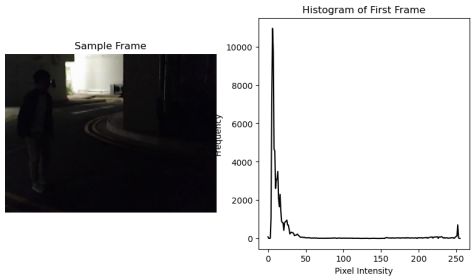


Figure 11: Histogram of Unprocessed Image

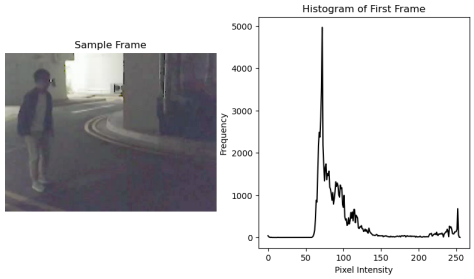


Figure 12: Histogram of Processed Image

From these Fig 10, Fig 12, Fig 11, we can clearly the processed image look good for visible eye with less noise and natural colors. In normalized image it look visible but it has more noise and has added artificial color and some areas are over exposed. It is evident by seeing the histogram map we can say gamma correction pushed the intensity values of the pixels.

Then i applied this filter to every video in the dataset to find out whether this changes has bring any changes in the performance of the model. Hence i just repeated first three step again and got the following results.

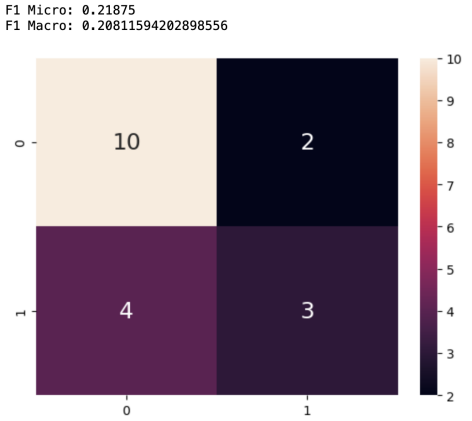


Figure 13: Extreme Learning Machine Metrics for Processed Image

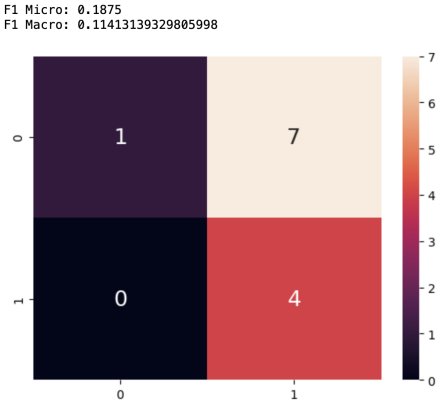


Figure 14: Naive Bayes Metrics for Processed Image

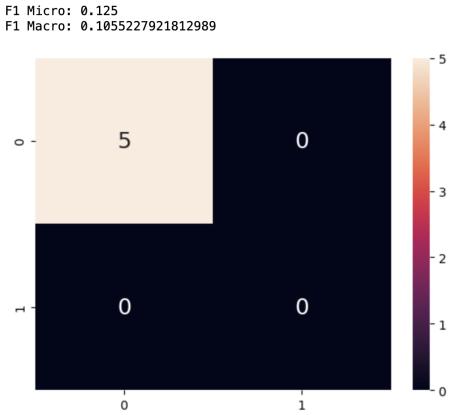


Figure 15: Logistic Regression Metrics for Processed Image

Upon analysis, we observe minimal variations in the outcomes. The Extreme Learning Machine (ELM) model exhibits commendable performance on the processed images, yielding an accuracy of 21.8, which matches the highest accuracy achieved with normalized images by the logistic regression model. Interestingly, the naive Bayes model yields identical results for both processed and unprocessed images. However, logistic regression under performs on processed images.

From this observation, we deduce that image enhancement has negligible impact on the efficacy of learning models. This conclusion holds true particularly in our case, suggesting that image enhancement fails to provide any additional efficiency to the model. This phenomenon may be attributed to the dataset's size. As modern learning methods increasingly utilize sophisticated filters to extract information from frames, the influence of image enhancement on model efficiency diminishes.

2.5 End-to-end Model

In real world scenario we don't get a luxury of storing every data and then processes that data and then classify the data as we done above. Because it time consuming and also it is not cost efficient method. Hence in this section we will build that model which classify our dataset in real time and also give better efficiency than the previous model. Hence i am going to implement vision transformer for this task of classify the human action.

Transformer is a deep learning architecture created for dealing the sequential data like NLP. As is it performed well in the sequential data, researchers extend that thing to image and video classification purpose and these transformer are called as vision transformers.

2.5.1 TRANSFORMER MODEL ARCHITECTURE

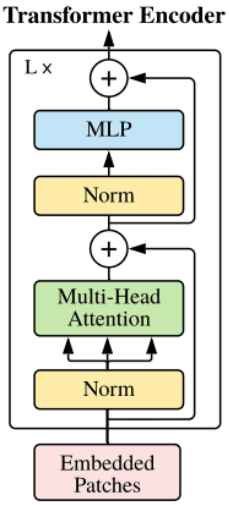


Figure 16: Vanilla Transformer Architecture

- In this project i have created a vanilla transformer model with two layers. To train this model i have take the same dataset as input and output as action classification.
- First i just sampled the frames using the uniform frame sampling with the number of frames is equal to 50. i also added padding to this step, padding make sure that we got a desired frames length from the videos by adding the extra frame when video doesn't have enough frames to samples and avoid the frames if the video has more frames. By this i make sure that the model will have a equal number of frames to process.
- Then to decrease the complexity of the model. i just cropped the frames to have frames with same dimension of 224x224. This helps the model to focus on the important features. But by doing so, sometimes we will loss some information also, this a optional step. i done this because to decrease the model complexity.
- Then i extracted the features from the frames using the DenseNet which a CCN based pretrained model used to extract the features from the frames. I am using this feature extractor in this model because feature extractor can captures spatial information within each frame, while temporal dependencies between frames can capture using self-attention mechanisms in transformer.
- After the feature extraction, those vectors of the frames are then organized into chunks suitable for processing by the Transformer. Then position of those vector is embedded using Positional Embedding. this allow models to understand the sequential order of the frames.
- Then comes Transformer Encoder layer. This layer consists of self-attention mechanisms and feed forward neural networks (FFNNs). It processes the embedded frames, capturing temporal dependencies between them. The self-attention mechanism helps the model focus on relevant frames while considering the entire sequence.
- Then Multi-Head Attention layer. Within the Transformer Encoder, multiple parallel attention calculations are performed, enhancing the model's ability to capture complex patterns in the video sequences. In this project i used 2 layers, but we can increase the layers, but it will also increase the complexity of the model.
- Then Layer Normalization layer. Layer normalization stabilizes the training process by normalizing the activation's of each layer.
- Then Global Max Pooling and Dropout layer. this layer has a global max pooling layer aggregates the output features across all frames into a single vector. Dropout helps prevent over fitting by randomly setting some activation's to zero during training.

- Final layer is the dense layer with softmax activation produces the classification probabilities for each class.
- The basic architecture of the vision based transformer is shown in Fig 16

Then The model is trained using labeled video data. During training, the model learns to predict the correct class label for each input video. The training process involves optimizing the model's parameters using a training dataset. The loss function used is sparse categorical cross-entropy, and the Adam optimizer is employed. The ModelCheckpoint callback saves the weights of the model whenever there is an improvement on the validation set.

2.5.2 METRIC EVALUATION

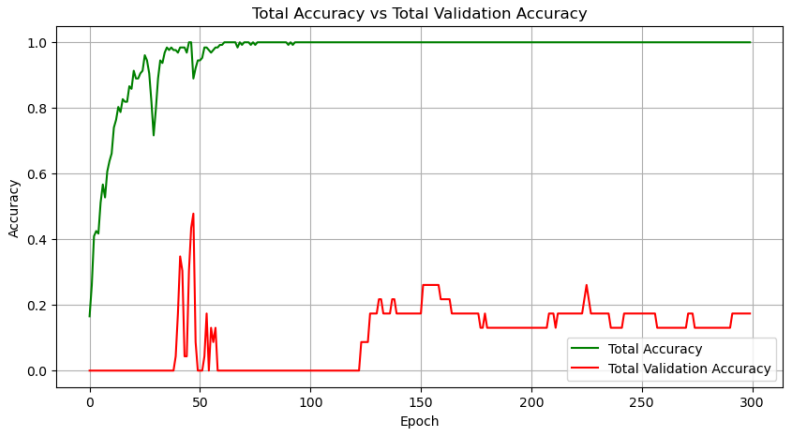


Figure 17: Accuracy vs Validation Accuracy

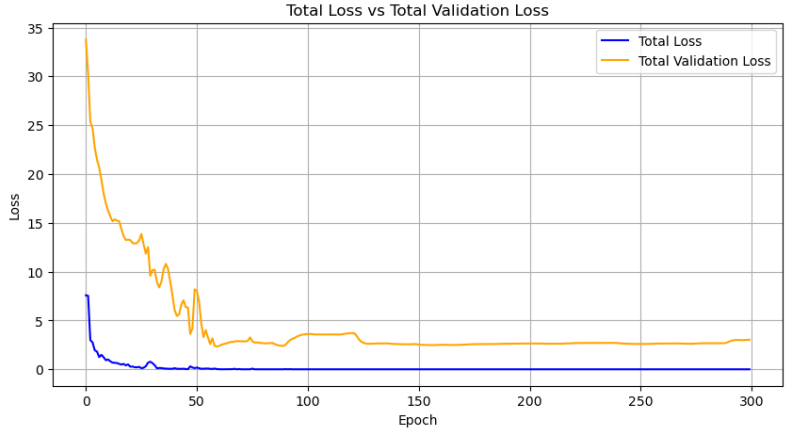


Figure 18: Loss vs Validation Loss

After analyzing Fig. 17, it becomes evident that while the training accuracy reaches 100, the validation accuracy peaks at around 45, with an overall validation accuracy of 25. This indicates a clear case of overfitting to the training data, wherein the model's performance on unseen validation data is significantly lower. This discrepancy can be attributed to several factors: Firstly, the disparity between training and testing data characteristics. In the training data, actions are executed singularly, whereas in testing data, actions occur multiple times in quick succession, often accompanied by additional movements such as continuous forward motion while jumping. This discrepancy is visually evident in the validation loss graph depicted in Fig. 18, where the model struggles to maintain accuracy. Moreover, the limited availability of training data, with only 25 videos per action, combined with the use of a vanilla transformer architecture, inevitably impacts the model's accuracy.

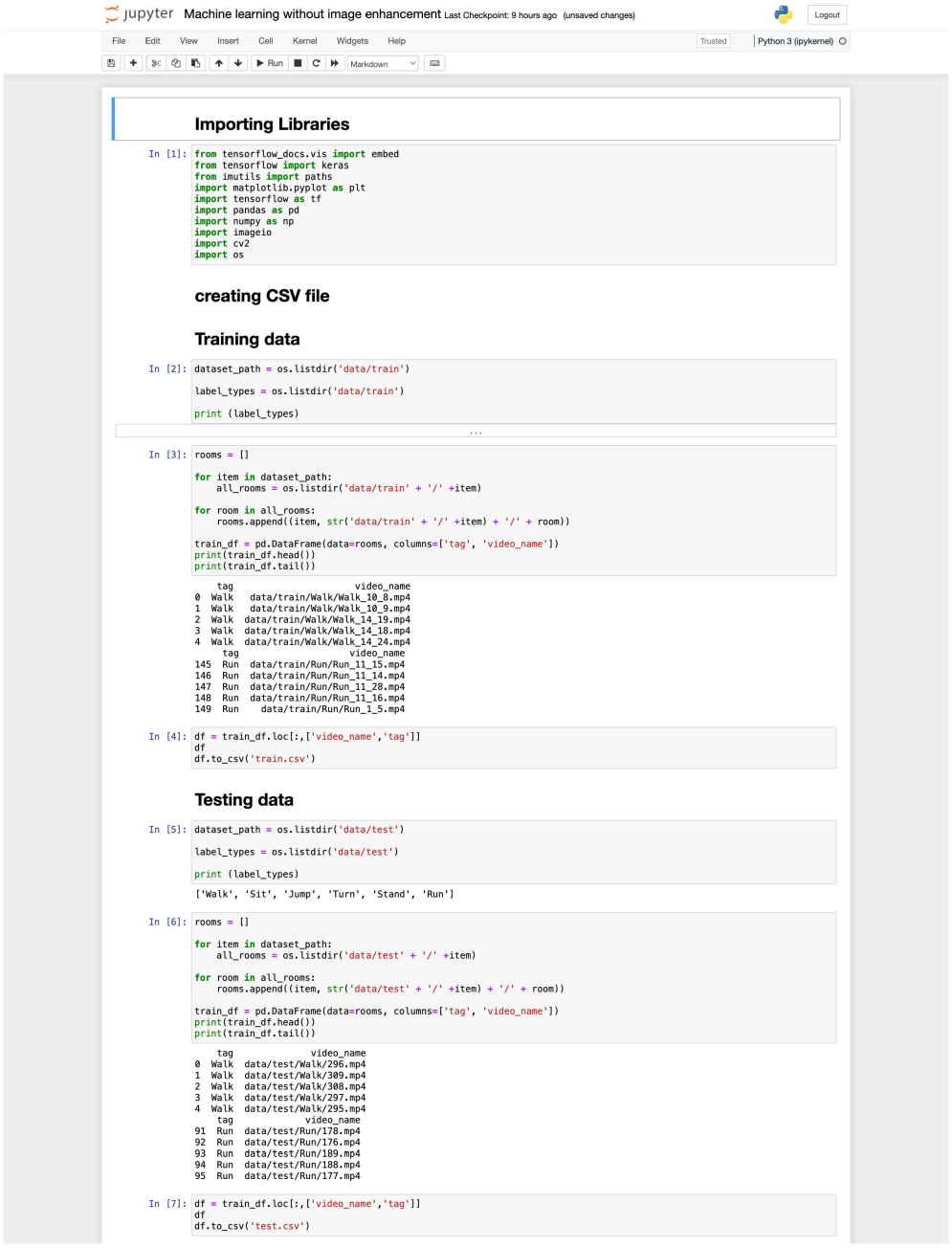
Following this analysis, the weights associated with higher accuracy and lower validation loss are saved to the computer with the extension '.h5'. These weights can then be utilized for real-time video classification. Upon inputting a video into the model, the system automatically preprocesses the video according to the model's requirements, facilitating real-time action prediction.

3 CONCLUSION

In this project, I successfully classified a provided low-light videos dataset, achieving a maximum accuracy of 22.8 percentage using various learning methods including ELM, Naive Bayes, and Logistic Regression. I conducted a thorough comparison of the results obtained from each model, providing valuable insights into their efficiency within the given dataset. Additionally, I explored preprocessing techniques tailored for low-light videos and trained the models accordingly. Furthermore, I investigated the impact of image enhancement on the model's performance, shedding light on its significance. As a culmination of my efforts, I developed an end-to-end model integrating the vision transformer architecture, accompanied by a detailed explanation of its workings. I also delved into the model's performance using evaluation metrics, providing a comprehensive analysis. Moreover, I elucidated on the practical implementation of the model in real-world scenarios. Despite achieving an accuracy of approximately 25 percentage with only 25 videos per class and employing a vanilla vision transformer, I highlighted the potential for enhanced accuracy in the future. This could be accomplished through expanding the dataset and increasing the complexity of the model, paving the way for improved performance in subsequent iterations.

A APPENDIX

A.1 CODES



jupyter Machine learning without image enhancement Last Checkpoint: 9 hours ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Markdown

Importing Libraries

In [1]:
```
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
```

creating CSV file

Training data

In [2]:
```
dataset_path = os.listdir('data/train')

label_types = os.listdir('data/train')

print (label_types)
```

...

In [3]:
```
rooms = []

for item in dataset_path:
    all_rooms = os.listdir('data/train' + '/' +item)

for room in all_rooms:
    rooms.append((item, str('data/train' + '/' +item) + '/' + room))

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

```
    tag                        video_name
0  Walk   data/train/Walk/Walk_10_8.mp4
1  Walk   data/train/Walk/Walk_10_9.mp4
2  Walk  data/train/Walk/Walk_14_19.mp4
3  Walk  data/train/Walk/Walk_14_18.mp4
4  Walk  data/train/Walk/Walk_14_24.mp4
     tag                       video_name
145  Run  data/train/Run/Run_11_15.mp4
146  Run  data/train/Run/Run_11_14.mp4
147  Run  data/train/Run/Run_11_28.mp4
148  Run  data/train/Run/Run_11_16.mp4
149  Run    data/train/Run/Run_1_5.mp4
```

In [4]:
```
df = train_df.loc[:,['video_name','tag']]
df
df.to_csv('train.csv')
```

Testing data

In [5]:
```
dataset_path = os.listdir('data/test')

label_types = os.listdir('data/test')

print (label_types)
```

```
['Walk', 'Sit', 'Jump', 'Turn', 'Stand', 'Run']
```

In [6]:
```
rooms = []

for item in dataset_path:
    all_rooms = os.listdir('data/test' + '/' +item)

for room in all_rooms:
    rooms.append((item, str('data/test' + '/' +item) + '/' + room))

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

```
    tag             video_name
0  Walk  data/test/Walk/296.mp4
1  Walk  data/test/Walk/309.mp4
2  Walk  data/test/Walk/308.mp4
3  Walk  data/test/Walk/297.mp4
4  Walk  data/test/Walk/295.mp4
    tag             video_name
91  Run  data/test/Run/178.mp4
92  Run  data/test/Run/176.mp4
93  Run  data/test/Run/189.mp4
94  Run  data/test/Run/188.mp4
95  Run  data/test/Run/177.mp4
```

In [7]:
```
df = train_df.loc[:,['video_name','tag']]
df
df.to_csv('test.csv')
```

import csv

In [8]:
```
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)}")

test_df.head(10)
```

```
Total videos for training: 150
Total videos for testing: 96
```

Out[8]:

	Unnamed: 0	video_name	tag
0	0	data/test/Walk/296.mp4	Walk
1	1	data/test/Walk/309.mp4	Walk
2	2	data/test/Walk/308.mp4	Walk
3	3	data/test/Walk/297.mp4	Walk
4	4	data/test/Walk/295.mp4	Walk
5	5	data/test/Walk/300.mp4	Walk
6	6	data/test/Walk/301.mp4	Walk
7	7	data/test/Walk/303.mp4	Walk
8	8	data/test/Walk/302.mp4	Walk
9	9	data/test/Walk/299.mp4	Walk

Apply normalization to every videos

In [12]:
```
import cv2
import numpy as np
import os
import shutil

def normalize_video(video_path, out_path, reference_mean, reference_std):
    cap = cv2.VideoCapture(video_path)
    fps = cap.get(cv2.CAP_PROP_FPS)
    frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(out_path, fourcc, fps, (frame_width, frame_height))

    while True:
        ret, frame = cap.read()
        if not ret:
            break
        normalized_frame = ((frame - reference_mean) / reference_std).astype(np.float32)
        normalized_frame = np.clip(normalized_frame, 0, 255).astype(np.uint8)
        out.write(normalized_frame)

    cap.release()
    out.release()

def process_vid(video_input_path):
    temp_output_path = video_input_path + ".temp.mp4"

    try:
        normalize_video(video_input_path, temp_output_path, reference_mean=np.array([0.07, 0.07, 0.07]), reference_s
        shutil.move(temp_output_path, video_input_path)

        print("Processed:", video_input_path)
    except Exception as e:
        print("Error processing:", video_input_path)
        print(e)
for video_name in train_df['video_name']:
    process_vid(video_name)

for video_name in test_df['video_name']:
    process_vid(video_name)
```

...

Preprocessing Technique's

In [11]:
```
# Train
features = []
labels = []
# Test
features_1 = []
labels_1 = []
```

In [13]:
```
import cv2
import numpy as np
from keras.applications import InceptionV3
from keras.applications.inception_v3 import preprocess_input

def extract_features(frames):
    base_model = InceptionV3(weights='imagenet', include_top=False, pooling='avg')
    preprocessed_frames = preprocess_input(np.array(frames))
    features = base_model.predict(preprocessed_frames)
    return features


def process_video(video_path, max_frames=25):
    cap = cv2.VideoCapture(video_path)
    frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    sampling_interval = max(1, frame_count // max_frames)
    frames = []
    while True:
```

```
        ret, frame = cap.read()
        if not ret:
            break
        frame = cv2.resize(frame, (299, 299))
        frames.append(frame)
        for _ in range(sampling_interval - 1):
            cap.grab()

        if len(frames) == max_frames:
            break

    cap.release()

    if len(frames) < max_frames:
        frames.extend([frames[-1]] * (max_frames - len(frames)))

    features = extract_features(frames)

    return features
```

Training video preprocessing

In [14]:
```
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
```

...

In [15]:
```
from collections import Counter
my_list_tuples = [tuple(x) for x in labels]
counter = Counter(my_list_tuples)
unique_count = len(counter)
print("Number of unique values:", unique_count)

print("Frequency of each unique value:")
for value, frequency in counter.items():
    print(f"{value}: {frequency}")
```

```
Number of unique values: 6
Frequency of each unique value:
(5,): 25
(2,): 25
(0,): 25
(4,): 25
(3,): 25
(1,): 25
```

In [16]:
```
for videos in train_df['video_name']:
    video_path = videos
    video_features = process_video(video_path)
    features.append(video_features)
```

...

Testing video preprocessing

In [17]:
```
label_processor = keras.layers.StringLookup(num_oov_indices=0, vocabulary=np.unique(test_df["tag"]))
print(label_processor.get_vocabulary())

labels_1 = test_df["tag"].values
labels_1 = label_processor(labels_1[..., None]).numpy()
labels_1
```

...

In [18]:
```
from collections import Counter
my_list_tuples = [tuple(x) for x in labels_1]
counter = Counter(my_list_tuples)
unique_count = len(counter)
print("Number of unique values:", unique_count)

print("Frequency of each unique value:")
for value, frequency in counter.items():
    print(f"{value}: {frequency}")
```

```
Number of unique values: 6
Frequency of each unique value:
(5,): 16
(2,): 15
(0,): 17
(4,): 17
(3,): 16
(1,): 15
```

In [19]:
```
for videos in test_df['video_name']:
    video_path = videos
    video_features = process_video(video_path)
    features_1.append(video_features)
```

...

In [20]:
```
print('train features:')
print(len(features))
print('train labels:')
print(len(labels))

print('test features:')
print(len(features_1))
print('test labels:')
print(len(labels_1))
```

```
train features:
```

```
150
train labels:
150
test features:
96
test labels:
96
```

In [21]:
```
X_train = features

y_train = labels
X_test = features_1
y_test = labels_1
```

In [22]:
```
for i, x_train_element in enumerate(X_train):
    print(f"Shape of X_train[{i}]:", x_train_element.shape)

for i, x_test_element in enumerate(X_test):
    print(f"Shape of X_test[{i}]:", x_test_element.shape)
```

```
Shape of X_train[0]: (25, 2048)
Shape of X_train[1]: (25, 2048)
Shape of X_train[2]: (25, 2048)
Shape of X_train[3]: (25, 2048)
Shape of X_train[4]: (25, 2048)
Shape of X_train[5]: (25, 2048)
Shape of X_train[6]: (25, 2048)
Shape of X_train[7]: (25, 2048)
Shape of X_train[8]: (25, 2048)
Shape of X_train[9]: (25, 2048)
Shape of X_train[10]: (25, 2048)
Shape of X_train[11]: (25, 2048)
Shape of X_train[12]: (25, 2048)
Shape of X_train[13]: (25, 2048)
Shape of X_train[14]: (25, 2048)
Shape of X_train[15]: (25, 2048)
Shape of X_train[16]: (25, 2048)
Shape of X_train[17]: (25, 2048)
Shape of X_train[18]: (25, 2048)
```

flatten

In [23]:
```
X_train_array = np.array(X_train)
X_test_array = np.array(X_test)

# Reshape the arrays
X_train_flattened = X_train_array.reshape(X_train_array.shape[0], -1)
X_test_flattened = X_test_array.reshape(X_test_array.shape[0], -1)
```

In [24]:
```
for i, x_train_element in enumerate(X_train_flattened):
    print(f"Shape of X_train[{i}]:", x_train_element.shape)

for i, x_test_element in enumerate(X_test_flattened):
    print(f"Shape of X_test[{i}]:", x_test_element.shape)
```

```
Shape of X_train[0]: (51200,)
Shape of X_train[1]: (51200,)
Shape of X_train[2]: (51200,)
Shape of X_train[3]: (51200,)
Shape of X_train[4]: (51200,)
Shape of X_train[5]: (51200,)
Shape of X_train[6]: (51200,)
Shape of X_train[7]: (51200,)
Shape of X_train[8]: (51200,)
Shape of X_train[9]: (51200,)
Shape of X_train[10]: (51200,)
Shape of X_train[11]: (51200,)
Shape of X_train[12]: (51200,)
Shape of X_train[13]: (51200,)
Shape of X_train[14]: (51200,)
Shape of X_train[15]: (51200,)
Shape of X_train[16]: (51200,)
Shape of X_train[17]: (51200,)
Shape of X_train[18]: (51200,)
```

Metric

In [25]:
```
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
import seaborn as sns
def getConfMatrix(pred_data, actual):
    conf_mat = confusion_matrix(actual, pred_data, labels=[0,1])
    micro = f1_score(actual, pred_data, average='micro')
    macro = f1_score(actual,pred_data, average='macro')
    sns.heatmap(conf_mat, annot = True, fmt=".0f", annot_kws={"size": 18})
    print('F1 Micro: '+ str(micro))
    print('F1 Macro: '+ str(macro))
```

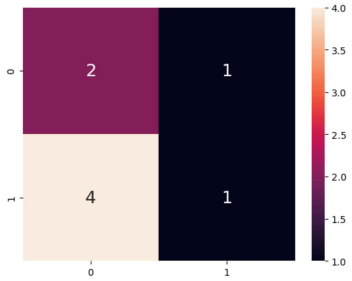
ELM

In [26]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from skelm import ELMClassifier

elm = ELMClassifier()

elm.fit(X_train_flattened, y_train)

y_pred = elm.predict(X_test_flattened)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.17708333333333334
```

In [27]:
```
getConfMatrix(y_pred,y_test)
```

```
F1 Micro: 0.17708333333333334
F1 Macro: 0.1640902307568974
```



Bayes

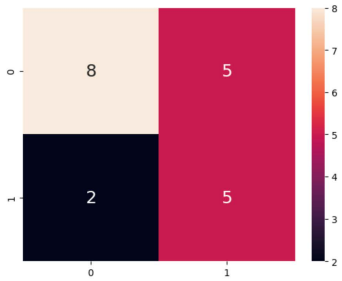
In [28]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

clf = GaussianNB()

clf.fit(X_train_flattened, y_train)

y_pred = clf.predict(X_test_flattened)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.1875
```

```
/Users/joshua/anaconda3/lib/python3.11/site-packages/sklearn/utils/validation.py:1184: DataConversionWarning: A col
umn-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  y = column_or_1d(y, warn=True)
```

In [29]:
```
getConfMatrix(y_pred,y_test)
```

```
F1 Micro: 0.1875
F1 Macro: 0.13979305535282763
```



LogisticRegression

In [30]:
```
import numpy as np
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(solver='lbfgs', multi_class='multinomial')

classifier.fit(X_train_flattened, y_train)

y_pred = classifier.predict(X_test_flattened)

print("Classification complete!")
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
/Users/joshua/anaconda3/lib/python3.11/site-packages/sklearn/utils/validation.py:1184: DataConversionWarning: A col
umn-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  y = column_or_1d(y, warn=True)
```

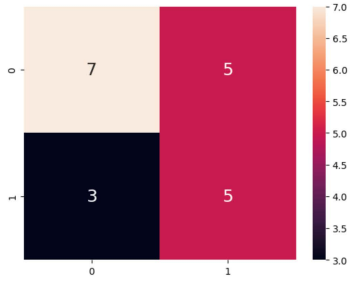
```
Classification complete!
```

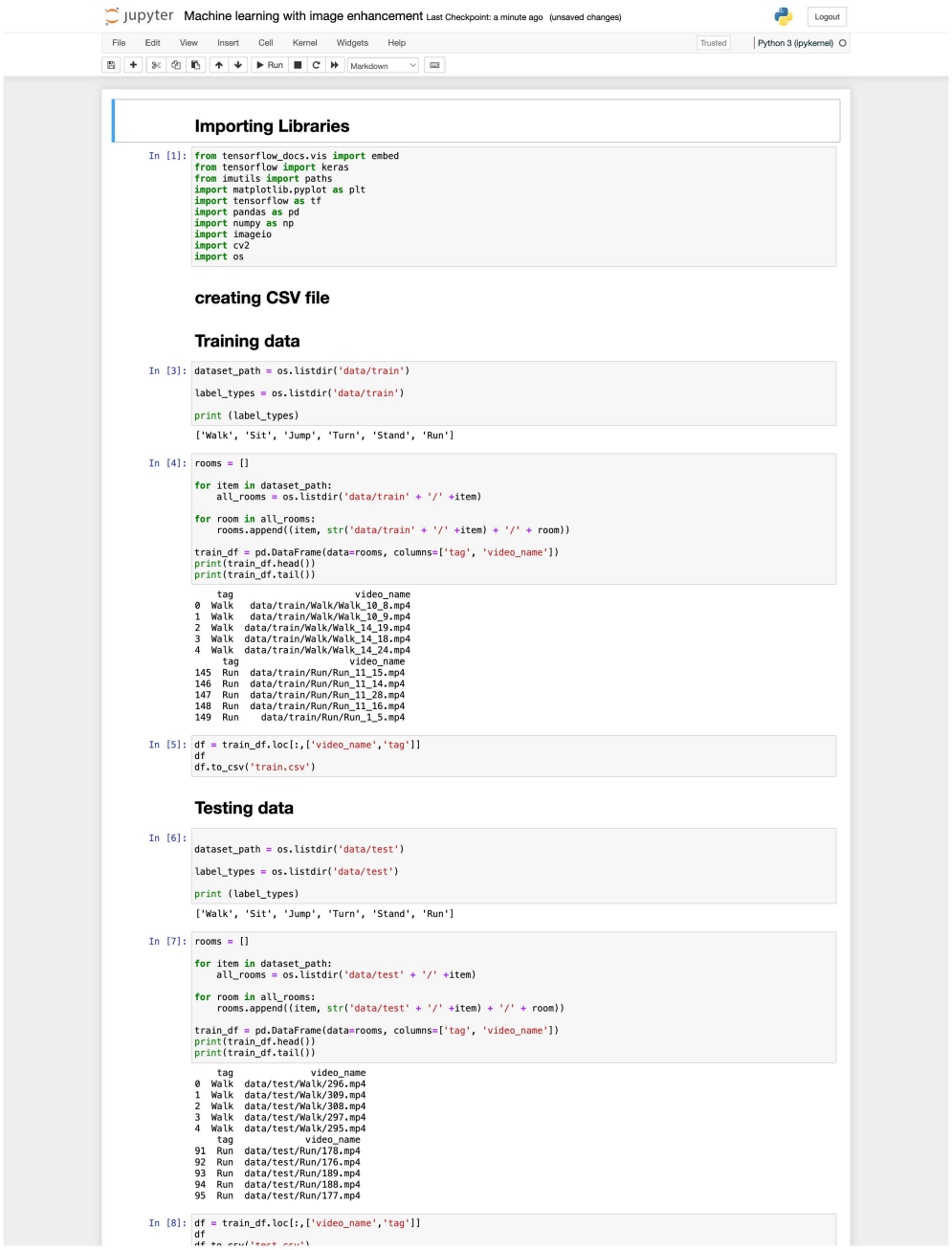
Accuracy: 0.21875

```
/Users/joshua/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [31]:
```
getConfMatrix(y_pred,y_test)
```

```
F1 Micro: 0.21875
F1 Macro: 0.18499659619878198
```





jupyter Machine learning with image enhancement Last Checkpoint: a minute ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Markdown

Importing Libraries

In [1]:
```
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
```

creating CSV file

Training data

In [3]:
```
dataset_path = os.listdir('data/train')

label_types = os.listdir('data/train')

print (label_types)
```

```
['Walk', 'Sit', 'Jump', 'Turn', 'Stand', 'Run']
```

In [4]:
```
rooms = []

for item in dataset_path:
    all_rooms = os.listdir('data/train' + '/' +item)

for room in all_rooms:
    rooms.append((item, str('data/train' + '/' +item) + '/' + room))

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

```
    tag                      video_name
0  Walk   data/train/Walk/Walk_10_8.mp4
1  Walk   data/train/Walk/Walk_10_9.mp4
2  Walk  data/train/Walk/Walk_14_19.mp4
3  Walk  data/train/Walk/Walk_14_18.mp4
4  Walk  data/train/Walk/Walk_14_24.mp4
     tag                video_name
145  Run  data/train/Run/Run_11_15.mp4
146  Run  data/train/Run/Run_11_14.mp4
147  Run  data/train/Run/Run_11_28.mp4
148  Run  data/train/Run/Run_11_16.mp4
149  Run    data/train/Run/Run_1_5.mp4
```

In [5]:
```
df = train_df.loc[:,['video_name','tag']]
df
df.to_csv('train.csv')
```

Testing data

In [6]:
```
dataset_path = os.listdir('data/test')

label_types = os.listdir('data/test')

print (label_types)
```

```
['Walk', 'Sit', 'Jump', 'Turn', 'Stand', 'Run']
```

In [7]:
```
rooms = []

for item in dataset_path:
    all_rooms = os.listdir('data/test' + '/' +item)

for room in all_rooms:
    rooms.append((item, str('data/test' + '/' +item) + '/' + room))

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

```
    tag             video_name
0  Walk  data/test/Walk/296.mp4
1  Walk  data/test/Walk/309.mp4
2  Walk  data/test/Walk/308.mp4
3  Walk  data/test/Walk/297.mp4
4  Walk  data/test/Walk/295.mp4
    tag             video_name
91  Run  data/test/Run/178.mp4
92  Run  data/test/Run/176.mp4
93  Run  data/test/Run/189.mp4
94  Run  data/test/Run/188.mp4
95  Run  data/test/Run/177.mp4
```

In [8]:
```
df = train_df.loc[:,['video_name','tag']]
df
df.to_csv('test.csv')
```

import csv

In [9]:
```
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)}")


test_df.head(10)
```

```
Total videos for training: 150
Total videos for testing: 96
```

Out[9]:

	Unnamed: 0	video_name	tag
0	0	data/test/Walk/296.mp4	Walk
1	1	data/test/Walk/309.mp4	Walk
2	2	data/test/Walk/308.mp4	Walk
3	3	data/test/Walk/297.mp4	Walk
4	4	data/test/Walk/295.mp4	Walk
5	5	data/test/Walk/300.mp4	Walk
6	6	data/test/Walk/301.mp4	Walk
7	7	data/test/Walk/303.mp4	Walk
8	8	data/test/Walk/302.mp4	Walk
9	9	data/test/Walk/299.mp4	Walk

Image enhancement

In [10]:
```
import cv2
import numpy as np
import os
import shutil

def gamma_correction(image, gamma):
    inv_gamma = 1.0 / gamma
    table = np.array([((i / 255.0) ** inv_gamma) * 255 for i in np.arange(0, 256)]).astype("uint8")
    return cv2.LUT(image, table)

def process_video(video_input_path):
    temp_output_path = video_input_path + ".temp.mp4"

    try:
        cap = cv2.VideoCapture(video_input_path)

        if not cap.isOpened():
            print("Error: Could not open video.")
            return

        fps = cap.get(cv2.CAP_PROP_FPS)
        width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

        out = cv2.VideoWriter(temp_output_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (width, height))

        gamma_value = 3

        while True:
            ret, frame = cap.read()
            if not ret:
                break

            corrected_frame = gamma_correction(frame, gamma_value)

            out.write(corrected_frame)

        cap.release()
        out.release()

        shutil.move(temp_output_path, video_input_path)

        print("Processed:", video_input_path)
    except Exception as e:
        print("Error processing:", video_input_path)
        print(e)

for video_name in test_df['video_name']:
    process_video(video_name)

for video_name in train_df['video_name']:
    process_video(video_name)
```

Preprocessing Technique's

In [11]:
```
# Train
features = []
labels = []
# Test
features_1 = []
labels_1 = []
```

In [12]:
```
import cv2
import numpy as np
from keras.applications import InceptionV3
from keras.applications.inception_v3 import preprocess_input
```

```
def extract_features(frames):
    base_model = InceptionV3(weights='imagenet', include_top=False, pooling='avg')
    preprocessed_frames = preprocess_input(np.array(frames))
    features = base_model.predict(preprocessed_frames)
    return features

def process_video(video_path, max_frames=25):
    cap = cv2.VideoCapture(video_path)
    frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    sampling_interval = max(1, frame_count // max_frames)
    frames = []
    while True:
        ret, frame = cap.read()
        if not ret:
            break

        frame = cv2.resize(frame, (299, 299))

        frames.append(frame)
        for _ in range(sampling_interval - 1):
            cap.grab()
        if len(frames) == max_frames:
            break

    cap.release()

    if len(frames) < max_frames:
        frames.extend([frames[-1]] * (max_frames - len(frames)))


    features = extract_features(frames)

    return features
```

Training video preprocessing

In [13]:
```
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
```

...

In [14]:
```
from collections import Counter
my_list_tuples = [tuple(x) for x in labels]
counter = Counter(my_list_tuples)
unique_count = len(counter)

print("Number of unique values:", unique_count)

print("Frequency of each unique value:")
for value, frequency in counter.items():
    print(f"{value}: {frequency}")
```

```
Number of unique values: 6
Frequency of each unique value:
(5,): 25
(2,): 25
(0,): 25
(4,): 25
(3,): 25
(1,): 25
```

In [15]:
```
# Train

for videos in train_df['video_name']:
    video_path = videos
    video_features = process_video(video_path)
    features.append(video_features)
```

...

Testing video preprocessing

In [16]:
```
label_processor = keras.layers.StringLookup(num_oov_indices=0, vocabulary=np.unique(test_df["tag"]))
print(label_processor.get_vocabulary())

labels_1 = test_df["tag"].values
labels_1 = label_processor(labels_1[..., None]).numpy()
labels_1
```

...

In [17]:
```
from collections import Counter
my_list_tuples = [tuple(x) for x in labels_1]
counter = Counter(my_list_tuples)
unique_count = len(counter)

print("Number of unique values:", unique_count)

print("Frequency of each unique value:")
for value, frequency in counter.items():
    print(f"{value}: {frequency}")
```

```
Number of unique values: 6
Frequency of each unique value:
(5,): 16
(2,): 15
(0,): 17
(4,): 17
(3,): 16
(1,): 15
```

In [18]:
```
# Test
for videos in test_df['video_name']:
    video_path = videos
    video_features = process_video(video_path)
    features_1.append(video_features)
```

...

In [19]:
```
print('train features:')
print(len(features))
print('train labels:')
print(len(labels))

print('test features:')
print(len(features_1))
print('test labels:')
print(len(labels_1))
```

```
train features:
150
train labels:
150
test features:
96
test labels:
96
```

In [20]:
```
X_train = features

y_train = labels
X_test = features_1
y_test = labels_1
```

In [21]:
```
# Check shapes of X_train
for i, x_train_element in enumerate(X_train):
    print(f"Shape of X_train[{i}]:", x_train_element.shape)

# Check shapes of X_test
for i, x_test_element in enumerate(X_test):
    print(f"Shape of X_test[{i}]:", x_test_element.shape)
```

```
Shape of X_train[0]: (25, 2048)
Shape of X_train[1]: (25, 2048)
Shape of X_train[2]: (25, 2048)
Shape of X_train[3]: (25, 2048)
Shape of X_train[4]: (25, 2048)
Shape of X_train[5]: (25, 2048)
Shape of X_train[6]: (25, 2048)
Shape of X_train[7]: (25, 2048)
Shape of X_train[8]: (25, 2048)
Shape of X_train[9]: (25, 2048)
Shape of X_train[10]: (25, 2048)
Shape of X_train[11]: (25, 2048)
Shape of X_train[12]: (25, 2048)
Shape of X_train[13]: (25, 2048)
Shape of X_train[14]: (25, 2048)
Shape of X_train[15]: (25, 2048)
Shape of X_train[16]: (25, 2048)
Shape of X_train[17]: (25, 2048)
Shape of X_train[18]: (25, 2048)
```

flatten

In [22]:
```
X_train_array = np.array(X_train)
X_test_array = np.array(X_test)

# Reshape the arrays
X_train_flattened = X_train_array.reshape(X_train_array.shape[0], -1)
X_test_flattened = X_test_array.reshape(X_test_array.shape[0], -1)
```

In [23]:
```
for i, x_train_element in enumerate(X_train_flattened):
    print(f"Shape of X_train[{i}]:", x_train_element.shape)

for i, x_test_element in enumerate(X_test_flattened):
    print(f"Shape of X_test[{i}]:", x_test_element.shape)
```

```
Shape of X_train[0]: (51200,)
Shape of X_train[1]: (51200,)
Shape of X_train[2]: (51200,)
Shape of X_train[3]: (51200,)
Shape of X_train[4]: (51200,)
Shape of X_train[5]: (51200,)
Shape of X_train[6]: (51200,)
Shape of X_train[7]: (51200,)
Shape of X_train[8]: (51200,)
Shape of X_train[9]: (51200,)
Shape of X_train[10]: (51200,)
Shape of X_train[11]: (51200,)
Shape of X_train[12]: (51200,)
Shape of X_train[13]: (51200,)
Shape of X_train[14]: (51200,)
Shape of X_train[15]: (51200,)
Shape of X_train[16]: (51200,)
Shape of X_train[17]: (51200,)
Shape of X_train[18]: (51200,)
```

Metric

In [24]:
```
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
import seaborn as sns
def getConfMatrix(pred_data, actual):
    conf_mat = confusion_matrix(actual, pred_data, labels=[0,1])
    micro = f1_score(actual, pred_data, average='micro')
    macro = f1_score(actual,pred_data, average='macro')
```

```
    print('F1 Micro: '+ str(micro))
    print('F1 Macro: '+ str(macro))
```

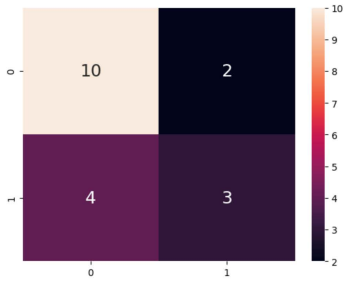
ELM

In [32]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from skelm import ELMClassifier

elm = ELMClassifier()

elm.fit(X_train_flattened, y_train)

y_pred = elm.predict(X_test_flattened)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.21875
```

In [33]:
```
getConfMatrix(y_pred,y_test)
```

```
F1 Micro: 0.21875
F1 Macro: 0.20811594202898556
```



Bayes

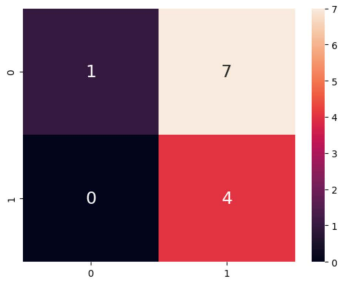
In [27]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

clf = GaussianNB()

clf.fit(X_train_flattened, y_train)

y_pred = clf.predict(X_test_flattened)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.1875
```

```
/Users/joshua/anaconda3/lib/python3.11/site-packages/sklearn/utils/validation.py:1184: DataConversionWarning: A col
umn-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  y = column_or_1d(y, warn=True)
```

In [28]:
```
getConfMatrix(y_pred,y_test)
```

```
F1 Micro: 0.1875
F1 Macro: 0.11413139329805998
```



LogisticRegression

In [38]:
```
import numpy as np
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(solver='lbfgs', multi_class='multinomial')

classifier.fit(X_train_flattened, y_train)

y_pred = classifier.predict(X_test_flattened)

print("Classification complete!")
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

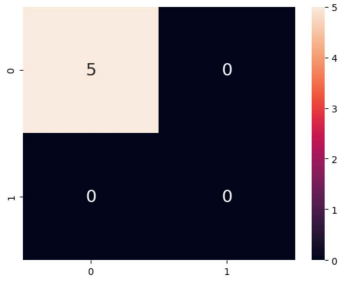
```
/Users/joshua/anaconda3/lib/python3.11/site-packages/sklearn/utils/validation.py:1184: DataConversionWarning: A col
umn-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example us
ing ravel().
  y = column_or_1d(y, warn=True)
```

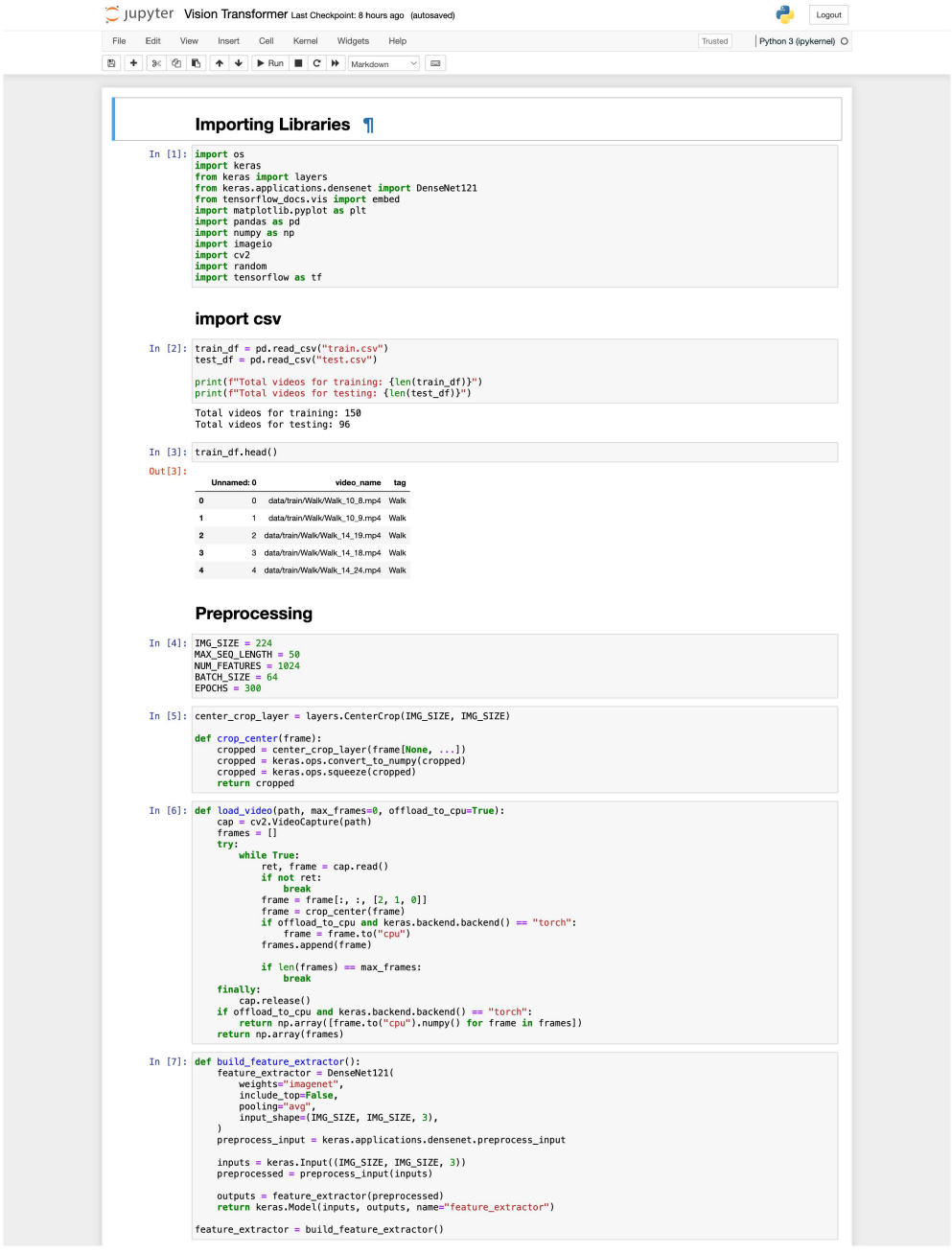
```
Classification complete!
Accuracy: 0.125
```

```
/Users/joshua/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [39]:
```
getConfMatrix(y_pred,y_test)
```

```
F1 Micro: 0.125
F1 Macro: 0.1055227921812989
```





jupyter Vision Transformer Last Checkpoint: 8 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Markdown

Importing Libraries ¶

In [1]:
```
import os
import keras
from keras import layers
from keras.applications.densenet import DenseNet121
from tensorflow_docs.vis import embed
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import imageio
import cv2
import random
import tensorflow as tf
```

import csv

In [2]:
```
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)}")
```

```
Total videos for training: 150
Total videos for testing: 96
```

In [3]:
```
train_df.head()
```

Out[3]:

	Unnamed: 0	video_name	tag
0	0	data/train/Walk/Walk_10_8.mp4	Walk
1	1	data/train/Walk/Walk_10_9.mp4	Walk
2	2	data/train/Walk/Walk_14_19.mp4	Walk
3	3	data/train/Walk/Walk_14_18.mp4	Walk
4	4	data/train/Walk/Walk_14_24.mp4	Walk

Preprocessing

In [4]:
```
IMG_SIZE = 224
MAX_SEQ_LENGTH = 50
NUM_FEATURES = 1024
BATCH_SIZE = 64
EPOCHS = 300
```

In [5]:
```
center_crop_layer = layers.CenterCrop(IMG_SIZE, IMG_SIZE)

def crop_center(frame):
    cropped = center_crop_layer(frame[None, ...])
    cropped = keras.ops.convert_to_numpy(cropped)
    cropped = keras.ops.squeeze(cropped)
    return cropped
```

In [6]:
```
def load_video(path, max_frames=0, offload_to_cpu=True):
    cap = cv2.VideoCapture(path)
    frames = []
    try:
        while True:
            ret, frame = cap.read()
            if not ret:
                break
            frame = frame[:, :, [2, 1, 0]]
            frame = crop_center(frame)
            if offload_to_cpu and keras.backend.backend() == "torch":
                frame = frame.to("cpu")
            frames.append(frame)

            if len(frames) == max_frames:
                break
    finally:
        cap.release()
    if offload_to_cpu and keras.backend.backend() == "torch":
        return np.array([frame.to("cpu").numpy() for frame in frames])
    return np.array(frames)
```

In [7]:
```
def build_feature_extractor():
    feature_extractor = DenseNet121(
        weights="imagenet",
        include_top=False,
        pooling="avg",
        input_shape=(IMG_SIZE, IMG_SIZE, 3),
    )
    preprocess_input = keras.applications.densenet.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()
```

In [8]:
```
# Label preprocessing
label_processor = keras.layers.StringLookup(
    num_oov_indices=0, vocabulary=np.unique(train_df["tag"]), mask_token=None
)
print(label_processor.get_vocabulary())

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

...

In [9]:
```
def prepare_all_videos(df, root_dir):
    num_samples = len(df)
    video_paths = df["video_name"].values.tolist()

    labels = df["tag"].values
    labels = label_processor(labels[..., None]).numpy()
    frame_features = np.zeros(
        shape=(num_samples, MAX_SEQ_LENGTH, NUM_FEATURES), dtype="float32"
    )

    for idx, path in enumerate(video_paths):
        frames = load_video(path)
        # Pad shorter videos.
        if len(frames) < MAX_SEQ_LENGTH:
            diff = MAX_SEQ_LENGTH - len(frames)
            padding = np.zeros((diff, IMG_SIZE, IMG_SIZE, 3))
            frames = np.concatenate([frames, padding])

        frames = frames[None, ...]

        temp_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):
                if np.mean(batch[j, :]) > 0.0:
                    temp_frame_features[i, j, :] = feature_extractor.predict(
                        batch[None, j, :]
                    )

                else:
                    temp_frame_features[i, j, :] = 0.0

        frame_features[idx,] = temp_frame_features.squeeze()

    return frame_features, labels
```

apply

In [10]:
```
train_data, train_labels = prepare_all_videos(train_df, "train")
test_data, test_labels = prepare_all_videos(test_df, "test")
```

...

In [11]:
```
print(f"Frame features in train set: {train_data.shape}")
print(f"Frame features in test set: {test_data.shape}")



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

```
Frame features in train set: (150, 50, 1024)
Frame features in test set: (96, 50, 1024)
train_labels in train set: (150, 1)
test_labels in test set: (96, 1)
```

Transformer model

In [12]:
```
class PositionalEmbedding(layers.Layer):
    def __init__(self, sequence_length, output_dim, **kwargs):
        super().__init__(**kwargs)
        self.position_embeddings = layers.Embedding(
            input_dim=sequence_length, output_dim=output_dim
        )
        self.sequence_length = sequence_length
        self.output_dim = output_dim

    def build(self, input_shape):
        self.position_embeddings.build(input_shape)

    def call(self, inputs):
        inputs = tf.cast(inputs, self.compute_dtype)
        length = tf.shape(inputs)[1]
        positions = tf.range(start=0, limit=length, delta=1)
        embedded_positions = self.position_embeddings(positions)
        return inputs + embedded_positions
```

In [13]:
```
class TransformerEncoder(layers.Layer):
    def __init__(self, embed_dim, dense_dim, num_heads, **kwargs):
        super().__init__(**kwargs)
        self.embed_dim = embed_dim
        self.dense_dim = dense_dim
        self.num_heads = num_heads
        self.attention = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim, dropout=0.3
        )
        self.dense_proj = keras.Sequential(
            [
                layers.Dense(dense_dim, activation=keras.activations.gelu),
                layers.Dense(embed_dim),
```

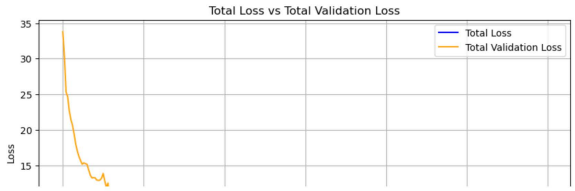
```
                layers.Dense(embed_dim),
            ]
        )
        self.layernorm_1 = layers.LayerNormalization()
        self.layernorm_2 = layers.LayerNormalization()

    def call(self, inputs, mask=None):
        attention_output = self.attention(inputs, inputs, attention_mask=mask)
        proj_input = self.layernorm_1(inputs + attention_output)
        proj_output = self.dense_proj(proj_input)
        return self.layernorm_2(proj_input + proj_output)
```

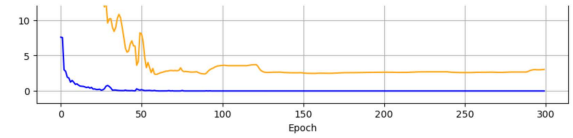
double layer

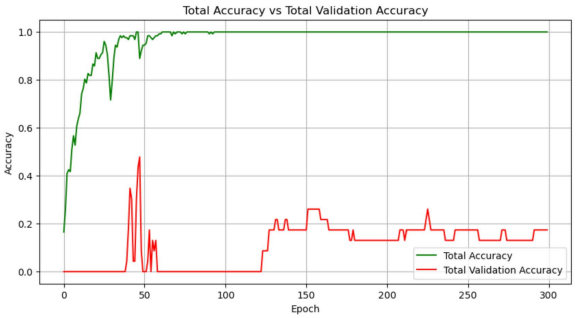
In [21]:
```
def get_compiled_model(shape):
    sequence_length = MAX_SEQ_LENGTH
    embed_dim = NUM_FEATURES
    dense_dim = 4
    num_heads = 1
    classes = len(label_processor.get_vocabulary())

    inputs = keras.Input(shape=shape)
    x = PositionalEmbedding(
        sequence_length, embed_dim, name="frame_position_embedding"
    )(inputs)
    # Stack two TransformerEncoder layers
    x = TransformerEncoder(embed_dim, dense_dim, num_heads, name="transformer_layer_1")(x)
    x = TransformerEncoder(embed_dim, dense_dim, num_heads, name="transformer_layer_2")(x)
    x = layers.GlobalMaxPooling1D()(x)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(classes, activation="softmax")(x)
    model = keras.Model(inputs, outputs)

    model.compile(
        optimizer="adam",
        loss="sparse_categorical_crossentropy",
        metrics=["accuracy"],
    )
    model.summary()
    return model
model = get_compiled_model(shape=(MAX_SEQ_LENGTH, NUM_FEATURES))
model.summary()

import matplotlib.pyplot as plt

def run_experiment():
    filepath = "video_classifier.weights.h5"
    checkpoint = keras.callbacks.ModelCheckpoint(
        filepath, save_weights_only=True, save_best_only=True, verbose=1
    )

    model = get_compiled_model(train_data.shape[1:])
    history = model.fit(
        train_data,
        train_labels,
        validation_split=0.15,
        epochs=EPOCHS,
        callbacks=[checkpoint],
    )

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

    return model, history

# trained model and history
trained_model, history = run_experiment()
```

...

In [22]:
```
# Plot Total Loss vs Total Validation Loss
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Total Loss', color='blue')
plt.plot(history.history['val_loss'], label='Total Validation Loss', color='orange')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Total Loss vs Total Validation Loss')
plt.legend()
plt.grid(True)
plt.show()

# Plot Total Accuracy vs Total Validation Accuracy
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], label='Total Accuracy', color='green')
plt.plot(history.history['val_accuracy'], label='Total Validation Accuracy', color='red')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Total Accuracy vs Total Validation Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```







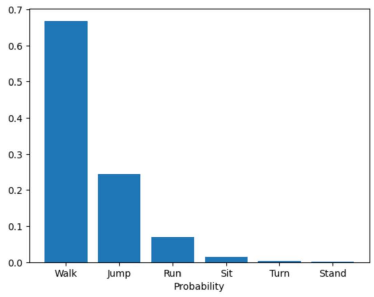
Inference

In [23]:
```
import matplotlib.pyplot as plt
import numpy as np

def plot_probabilities(probabilities, class_vocab):
    plot_x_axis, plot_y_axis = [], []

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

    plt.bar(plot_x_axis, plot_y_axis, label=plot_x_axis)
    plt.xlabel("class_label")
    plt.xlabel("Probability")
    plt.show()
```

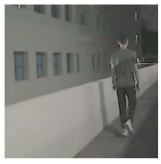
In [26]:
```
def prepare_single_video(frames, max_seq_length):
    num_frames = len(frames)
    frame_features = np.zeros(shape=(1, max_seq_length, NUM_FEATURES), dtype="float32")

    # Pad shorter videos
    if num_frames <= max_seq_length:
        diff = max_seq_length - num_frames
        padding = np.zeros((diff, IMG_SIZE, IMG_SIZE, 3))
        frames = np.concatenate((frames, padding), axis=0)
    else:
        # Truncate longer videos
        frames = frames[:max_seq_length]

    for i, frame in enumerate(frames):
        if np.mean(frame) > 0.0:
            frame_features[0, i, :] = feature_extractor.predict(frame[None, ...])
        else:
            frame_features[0, i, :] = 0.0

    return frame_features

def predict_action(path, trained_model, max_seq_length=MAX_SEQ_LENGTH):
    class_vocab = label_processor.get_vocabulary()
    frames = load_video(path)
    frame_features = prepare_single_video(frames, max_seq_length)
    probabilities = trained_model.predict(frame_features)[0]
    plot_probabilities(probabilities, class_vocab)
    return frames


def to_gif(images, duration=20):
    converted_images = images.astype(np.uint8)
    imageio.mimsave("animation.gif", converted_images, duration=duration)
    return embed.embed_file("animation.gif")


test_video = np.random.choice(test_df["video_name"].values.tolist())
print(f"Test video path: {test_video}")
test_frames = predict_action(test_video, trained_model,50)

to_gif(test_frames[:MAX_SEQ_LENGTH])
```

```
Test video path: data/test/Walk/310.mp4
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 92ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 92ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 101ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 130ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 113ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 103ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 85ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
  Walk: 66.86%
  Jump: 24.39%
  Run:  6.89%
  Sit:  1.41%
  Turn:  0.37%
  Stand:  0.09%
```



Out[26]:



In [ ]: