Department of Computer Science

Submitted in part fulfilment for the degree of BEng

Human Activity Recognition: Video Classification

Abiram Panchalingam

16 May 2021

Supervisor: Dr Adrian Bors

Acknowledgements

Thank you to my Supervisor, Dr Adrian Bors for his guidance and supervision throughout this project.

STATEMENT OF ETHICS

This Project has no concerns on the ethics of its data. All data used in this paper was from public datasets that have permission granted for educational/research use.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1 Executive Summary

Human Activity Recognition is a branch of study that looks at reading different types of data such as video, images, and sensor data to detect what the action or activity that is being performed in the data is. Analysing and determining such features and actions will form the basis of this project as I look to compare and contrast different methods. Recent studies have achieved state of the art results on various public datasets and I will look to replicate such results with less computationally expensive methods on smaller datasets. This project will settle on the best method in regard to computation costs, time needed, complexity and most importantly the reliability and precision indicated by the method.

The purpose of this report is to compare two deep learning models that will be performed and tested on a variety of datasets. We will analyse not only the performance of each model but also its usability and effectiveness on different datasets. Each model will also be easier/harder to tune than the other. Tuning hyperparameters is an important aspect of deep learning as it allows the model to train on the given dataset more quickly and also allows us to reach higher levels of accuracy.

By settling on a model/algorithm that will perform the best in these conditions, we will be able to recommend this to small businesses or individual entrepreneurs. It can be doubted that the models trained from scratch will be able to achieve state of the art results but for a smaller business efficiency will be important as they need to keep overheads low. So, such algorithms will be targeted for use with these types of users in mind. These businesses will theoretically use the model in applications such as automated security systems, labelling big data to learn useful patterns to help with marketing or perhaps detecting specific objects in videos etc.

There are a variety of methods available for my use and testing. Deep learning is the main approach currently being used to detecting actions within video datasets. Convolutional neural networks and Recurrent neural networks are already popular methods used to classify images. Images contain only spatial features so are easier to classify than videos. CNNs have proven to be adept on image and image-based data. A hybrid implementation with convolutional, recurrent, and other layers can be used also to solve our problem. Some state-of-the-art models use this method. [4]. Because videos can be perceived to be a sequence of frames (images); we can use methods used for images on videos as well.

Convolutional neural networks are the product of advancements within computer vision and deep learning [5]. They use convolutional kernels to extract features from images. Weights and bias are assigned to each feature in the image during training. This way the high-level features are extracted from the edge and given more/less importance. This all happens in the convolutional layer. The more of these layers there are in a model, the more precise the model adapts to the features of the images inputted. However, this would also increase the likelihood of overfitting.

Recurrent neural networks address the issue of traditional neural networks (Feed Forward Networks for example) being unable to use reasoning about previous inputs (beyond weight changes) [1]. RNNs are essentially normal multilayer perceptrons but with loops which allow information from pervious computations to persist and not be vanished. RNNs have been used successfully in a variety of applications such as speech recognition, language modelling and time series models. However, they struggle with long term dependencies. This is when information from a computation a few time-steps ago is needed in the current time-step. This is where **LSTMs** come in. They are a special type of RNN [1]. Designed specifically to handle long term dependencies using a special repeating module.

So, for my problem I will be using a combination of these neural network algorithms/models whilst analysing their particular variations. And deciding on the “best” model for use on a small scale. I have achieved high accuracy levels in Conv2D and Conv-LSTM models which work well on video datasets to recognise human actions.

There were no legal, social, ethical, professional or commercial issues concerning this work. All references and datasets used are publicly available for academic purposes such as this. However, data issues should not be looked over in this project. Especially in the field of video surveillance [6], businesses are taking more and more advantage of such footage to learn more about customer habits.

Results for the Conv2D model shows ~99.83% accuracy on test sets and ConvLSTM2D shows ~80.64% accuracy on test sets. These are generally good results but this does not determine the success of this project. This project can only be considered a success if we are able to choose one algorithm that not only performs better on a variety of video sources but also is robust and is able to run on a low computational cost budget.

2 Introduction

2.1 Applications of Human Activity Recognition

Computer Vision has seen a lot of growth over the last few decades, allowing the automatic classification and recognition of things in the real world. These things can be people, objects, actions or anything that can be recorded. The field has developed using a variety of techniques and methods, producing different algorithms for computer vision. This report will compare and contrast two algorithms in the field of human activity recognition, and work out which parameters improve or limit classification accuracy as well as which of the algorithms are less resource heavy. HAR has many applications. Self-driving cars require intricate analysis of pedestrians and their movement [26]; classifying a pedestrian's movement can help a system understand their direction and speed which is useful for the calculation of potential collisions, ultimately increasing its reaction ability. Surveillance systems can also use HAR, for example; detection of pedestrians waiting by traffic light crossings, detection of people committing crimes or detection of anything that is unusual. Movement classification can also be useful for human-computer interaction, allowing computer systems to be controlled by real life actions through a camera - such systems have no need for physical controls, increasing their compactness and accessibility for people that are unable to use standard physical controls.

An additional application of HAR is for use in hospitals. A video capturing system can be produced where patients are monitored and a doctor/nurse is notified as soon as a patient on a bed is suffering from a seizure etc. This is perhaps one of the most important use cases so it requires the best performance from these algorithms. As a false call could waste the valuable time of doctors who could be saving another patient's life elsewhere.

2.2 Project Aim

The aim of this project is to compare two modern relevant methods for Human Activity Recognition models in videos that can prove to be useful for a wide range of applications and organizations. I will be trying to find the model with the best performance in terms of accuracy, loss, speed, and usability in a professional environment. As mentioned previously, two models will be compared that will be able to recognise human actions/activities from video datasets. Two well-known models have been settled on: **2D convolutional networks** and **Conv-LSTM** networks. These are two small but efficient well-known networks today. Justification for these choices will follow whilst reviewing other methods also.

3 Background Review

3.1 History of Human Activity Recognition

Vision-based Human activity recognition is becoming a trending area of research due to its broad application such as security and surveillance, human–computer interactions, patients monitoring system, and robotics. These variety of applications mean that we need more and more reliable methods to ensure that a variety of requirements can be met.

Non-deep-learning methods for Human Activity Recognition include Optical flow, Kalman filtering and Hidden Markov models [12]. They work under single camera, stereo, and infrared modalities. These are seen as “Space-Time based” approaches to Human Activity Recognition. The main difference between Deep learning methods and these methods is that feature extraction has to be done manually in the older Space-time based approaches. In other words, deep learning methods are able to process the data in raw form [13] unlike the other methods which have to go through a manual feature extraction process.

3.2 Current Methods

Deep learning has emerged as the most popular computer vision method for recognising complex human activities from video. They have achieved high rates of accuracy on the two state of the art datasets HMDB51 and UCF101. These two datasets are the standard in this research area. And all high performing models on these datasets use deep learning techniques and nearly all use some form of Convolutional layers. CNN’s have become the standard, especially 3D CNNs which are able to learn spatial and temporal features quite easily.

However, all these methods have been developed with only high performance in mind but not quite usability in the professional environment. I will be trying to compare two models with usability in mind as well as performance. I will test the speed on execution, memory/data/CPU usage, and analyse how well it can be integrated in real world live systems. Of course, performance will be the biggest priority of this paper but disadvantages such as deep learning requiring large datasets will also be considered.

3.3 2D Convolutional Neural Networks

Convolutional networks can be 1D, 2D or 3D (rarely 4D). 1D convolutional networks use kernels that only move in one direction. So, it is only useful for time-series data and is not compatible with image data (and therefore videos). 1D networks would perhaps work better on sensor data so it is still relevant to human activity recognition but having to wear sensors requires a controlled environment. I am more interested in in data examples where we can recognise natural human actions taken from natural videos. Thus, because 1D networks are more suitable for time series, audio, and text data I will not consider it for this implementation.

Both 2D and 3D networks can be used on video data. 2D will look at frames independently and take the average prediction to be correct and it becomes the final prediction. 3D works more naturally on video datasets as the convolutional kernel moves in 3 dimensions as shown in Appendix A. However, its memory consumption is high and wouldn't be suitable for use in small businesses so I will use the Conv2D architecture for my first model as it also has cases of high performance. It works by predicting each frame then finally using the probabilities for each prediction to select the final prediction class which would be the action class with the highest probability. Because this model extracts frames from individual frames it relies on environmental context more rather than the difference in positioning of the human in the video. So only spatial features are considered and not temporal.

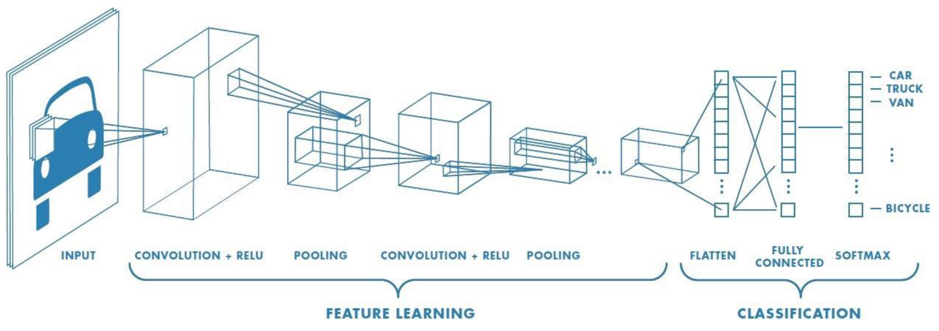


Figure 3:1: An example 2D convolutional network [5]

In Figure 3:1 is an example convolutional network from image input to eventual classification. The convolutional layer will always be the input layer and additional convolutional layers can be used for added precision. After moving through pooling layers and flatten layer we move to the fully connected layers which act as the classifier as it chooses what was detected in the image. In our case this would be from a range of human actions or activities depending on the dataset used.

3.4 ConvLSTM2D Networks

A Conv-LSTM is a network designed to exploit features of both Convolutional and Recurrent LSTM networks. It was introduced as an approach for "Precipitation Nowcasting" by combining LSTM gates with 2D convolutions [8]. Their “Experiments show that our ConvLSTM network captures spatiotemporal correlations better and consistently outperforms FC-LSTM and the state-of-the-art operational ROVER algorithm for precipitation nowcasting”. Videos have spatiotemporal features as they contain images distributed in terms of time. So hypothetically this type of model should work well with videos and be able to extract human actions quite easily given a competent and balanced dataset. Below is the layout of a ConvLSTM cell [9].

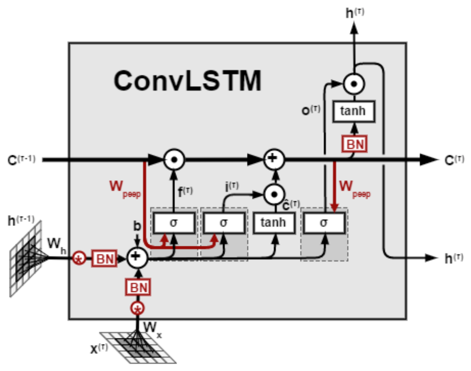


Figure 3:2: ConvLSTM cell

ConvLSTM is primarily of recurrent architecture. Work’s like normal LSTM’s but matrix multiplications are exchanged with convolutional operatiuons [9]. Data that flows through ConvLSTM cells maintains its input shape.

3.5 Datasets

There are many different types of datasets. The first ones were recorded in controlled situations such as Weizmann (2001/2005) and KTH (2004) [3]. The problem with these datasets however is that they are not representative of human actions in the real world. Actions are performed in identical fashion in each class with a stable background. So, it is quite unrealistic when compared to a real-life use case which makes using these types of datasets pointless. Models could easily train on these datasets but when then being used with natural video datasets they have the potential to heavily underperform.

Also, earlier datasets focussed on recognising simple human actions such as walking, handwaving and running. However, these models will not be useful either in the real world. For example, a surveillance camera needs to be able to detect people fighting or perhaps criminal activity. This would require us to train our model on datasets that show humans performing activities rather than simple quick actions.

Therefore, for these two reasons I have decided to use the datasets **HMDB51** and **UCF50**. Both have a variety of different classes and some overlapping classes which is vital for me as I will be testing the models on both datasets. They have heterogeneous actions and activities that are popular. Also, importantly they both contain a very high number of videos with HMDB51 having 6849 clips and UCF50 with 6676. This means that when we split the datasets, we should be able to acquire reliable train, validation and test sets that are representative of the whole dataset and balanced. And the fact that both datasets have videos pulled from YouTube means we can expect the models to perform similarly well (or not well) on other natural YouTube videos and use of the model's on YouTube videos definitely has a business use case such as reading user behaviours.

3.6 Project Objectives

- Implement a 2D convolutional network model on more than one video dataset
- Implement a Conv-LSTM network model on more than one video dataset
- Tune hyperparameters and structural parameters of each model to maximise accuracy and minimize loss
- Evaluate models using different evaluation metrics and learn behavioural traits of each model
- Beyond testing on test sets, test both models on external data such as a different dataset or on my own set of YouTube videos
- Use scores, metrics, and behavioural traits to compare and contrast models
- Compare both models to a state-of-the-art pre-trained model fairly and comment on any shortcomings of the models in comparison
- Identify possible future work and what can be done further on this topic

4 Methodology

Python will be used alongside the deep-learning library Keras [7] in the solutions. Python is one of the most popular programming languages so it will fit into most business environments and Keras was chosen as it is a high-level API that is easy to use than many other libraries. This will make it easier to be integrated into business environments where minimizing employee learning curves will be key to their success. The project code will be implemented in the Google Colab web IDE so as to make up for the limtations of the author’s current hardware and to take advantage of Colab Pro’s dedicated high-end GPU offerings.

Figure 4:1 shows the basic flowchart of a typical implementation of Human Activity Recognition. The major difference is that we are not using K means clustered learning or SVM models but instead pre-dominantly Convolutional Neural Networks. So, the main processes I will be using is described below as well as the two NN architectures.

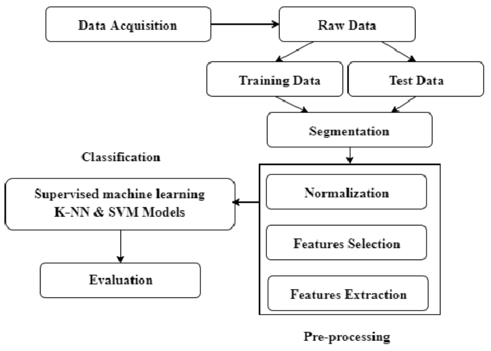


Figure 4:1: Typical flowchart for a HAR process implemented through deep learning [25]

4.1 Data Pre-Processing

The first step is to transform the videos into formats that can be fed into the neural networks. This involves extract individual frames but not all the frames as this would require too much memory in terms of RAM and disk. And more frames do not necessarily equate to higher performance. OpenCV is a computer vision library that can be used to store images/videos in numerical matrices. It supports many programming languages including Python which is what I am using. A module called cv2 from OpenCV can be used to extract images from videos [14].

The overall process for reading the videos for the two networks differs slightly and is described below.

Conv2D

1. Mount Google Drive
2. Locate correct dataset folder
3. Set reasonable image height and width

4. Specify chosen classes (labels) in dataset to pull videos from. This can be the whole dataset if enough system computation power is available
5. Iterate over list of chosen dataset classes
6. Get list of videos in each dataset
7. Iterate through each video file
    a. Iterate through each frame
    b. Check if successfully read
    c. Resize frame to previously defined pixel size
    d. Normalize frames so that each pixel has a value between 0 and 1
    e. Add to list of frames for current video
8. Add random number of frames to the final features list
9. Label all the frames with correct class number
10. Repeat steps 7 to 9 for every single video
11. Convert features (x) and labels (y) to numpy arrays
12. One Hot Encode labels or use dummy variables
13. Split both into train/test sets. Ratio of split is to be tested with different values

ConvLSTM2D

For ConvLSTM2D most of the steps are the same but there will be some differences. Steps 8 and 9 differ:

8. Add a specified number of frames for each video in sequential order

9. Label all the videos with correct class label

So, the difference is inputs are separated by videos as opposed to just frames. And for each video we are choosing a specific number of frames and not in random order as for this model the order of frames matter as temporal differences are learnt as well as spatial.

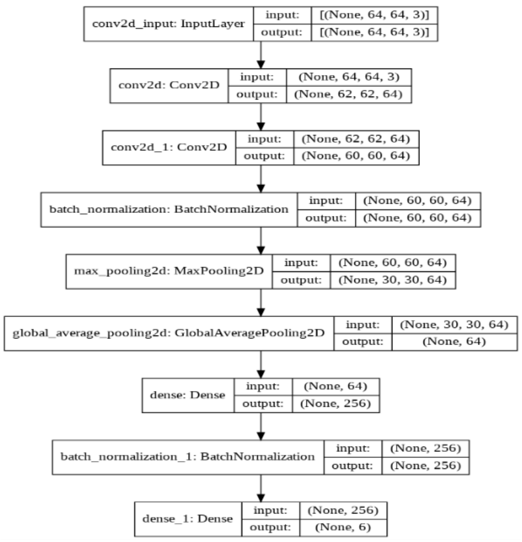
Normalization

Normalization is important as it is used to narrow the range of input values. In this instance this would be the pixel values. We could want to narrow them down from a range of 0-255 to 0-1. This can be achieved by simply dividing the numerical representation of each frame by 255. This would mean no information is lost. We will try normalization with both networks and analyse the results to see if it adds any benefits. This is because normalization doesn’t always help networks.

However, the proposed benefits of normalization are faster convergence to optimal accuracy levels/loss levels [15]. It also helps the neural network process information more accurately than without normalisation of data. Unscaled data can sometimes lead to a slow/unstable learning process and can cause the exploding gradient problem [16]. So, I will make sure the pixel values are tested normalised to a range between 0 and 1. Then depending on how the models perform I will decide whether to keep it or not.

4.2 Conv2D Architecture - Initial

The Conv2D based model has many layers including the convolutional layer as well as other layers which include the input layer, Batch Normalization layer, Max Pooling layer, Global Average Pooling layer and dense layer. Below is a list describing all the layers and justifying the inclusion of each layer.


conv2d_input: InputLayer
input: [(None, 64, 64, 3)]
output: [(None, 64, 64, 3)]
conv2d: Conv2D
input: (None, 64, 64, 3)
output: (None, 62, 62, 64)
conv2d_1: Conv2D
input: (None, 62, 62, 64)
output: (None, 60, 60, 64)
batch_normalization: BatchNormalization
input: (None, 60, 60, 64)
output: (None, 60, 60, 64)
max_pooling2d: MaxPooling2D
input: (None, 60, 60, 64)
output: (None, 30, 30, 64)
global_average_pooling2d: GlobalAveragePooling2D
input: (None, 30, 30, 64)
output: (None, 64)
dense: Dense
input: (None, 64)
output: (None, 256)
batch_normalization_1: BatchNormalization
input: (None, 256)
output: (None, 256)
dense_1: Dense
input: (None, 256)
output: (None, 6)
Figure 4:2: Conv2D initial layout

Input: (64,64,3) – 64 refers to the height and width of each frame from the videos. And the number 3 represents the colour space RGB. This allows colour images to be recognised as opposed to just grayscale. Reads the input video frames and passes them in the correct format to the Conv2D layer. The specified input shape is then reflected in the shape of weights.

Conv2D: (64,64,3) – Same meaning as previous layer. Creates a convolutional kernel that is convolved with the 2D input image to create an output. The kernel can detect edges and features in each frame. The output is then passed on to the next layer.

Batch Normalization: Keras Default: 0.99. Applies a transformation to the outputs of the previous layer before feeding into the next layer. This helps deep networks like this train faster as there will be less difference between each of the input values. This prevents issues such as the exploding gradient problem and also allows the model to fit faster to the dataset.

Max Pooling 2D: Down sampling feature used as an essential tool in convolutional neural networks. It reduces the feature map by two and allows fine detail not needed to be not included from memory.

Global Average Pooling 2D: It is also a pooling operation and is used before sending values to the output layer and also helps combat overfitting.

Dense: Extra fully connected layers for more weight changes and therefore precision. Final number of neurons will be the same as the number of classes used from the chosen dataset.

4.3 ConvLSTM2D Architecture - Initial

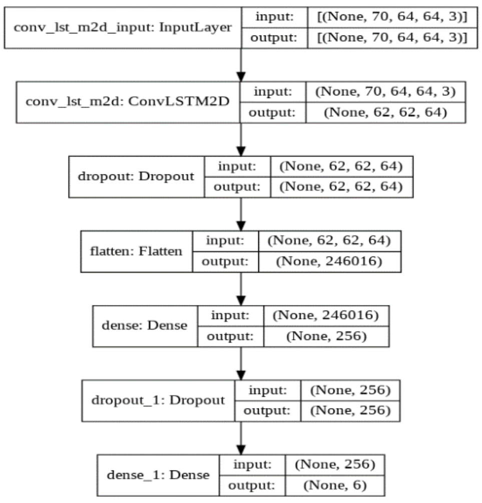


Figure 4:3: ConvLSTM2D Initial architecture

This is an architecture that is primarily of recurrent type so it cannot be used with the same type of layers as a CNN can. However, it has its own set of layers that could be used to do a similar job such as helping achieve regularization and generalisation.

Input Layer: (70,64,64,3) represents (70 frames per video, 64x64 pixels frame and 3 colour space (RGB).

ConvLSTM2D Layer: Perhaps the most important layer. It is able to learn from spatiotemporal features automatically. So spatial features are extracted as well as the correlation in time. It contains primarily an LSTM gate but does convolutional operations instead of matrix multiplications.

Dropout Layer: Some random neurons are switched off/ignored so all of the neurons are not used in the forward pass and therefore the weights of these selected neurons are not updated. This means that the model is restricted from learning and changing weights too radically at each pass.   Dropout is a form of regularization so should allow us to prevent overfitting. This allows the model to generalize and not overfit on the training data which large neural networks are vulnerable for.

Flatten Layer: Flattens the data into a 1-dimensional array. This can then be passed to the fully connected layers before classification. So, data is converted from a feature map that has been pooled to a 1-dimensional array of numbers. This is now in a format that can be handled by fully connected layers.

Dense Layer: Extra fully connected layers for more weight changes and therefore precision. Final number of neurons will be the same as the number of classes used from the chosen dataset.

4.4 Tuning Parameters

Neural networks generally need to be tuned once they have been developed and is working. Even though it may seem like they are already performing well, it is good practise to assume it can do even better. Especially since there are a lot of parameters that can be tweaked. Here is a list of all parameters and how they can affect the performance of my two models.

Number of layers: This refers to how deep the network is, more layers means more weight changes. Generally more complex representations can be learnt with more layers. **Number of neurons:** The number of neurons effects how many levels of precision the machine is able to reach as more neurons mean more computations. Using too little amount of neurons can cause underfitting and too many neurons can cause overfitting (however this is easier to control). More neurons also mean a longer training time for the model. **Activation functions:** Adds non-linearity to the model and keeps neuron's outputs limited to the limit the model needs to classify classes. Activation functions work differently with different model types. **Training Algorithm:** This carries out the learning process; how and when the model changes weights and based on what criteria. Need to select the right training algorithm for the right architecture, can help maximise accuracy. **Learning Rate:** Learning rate refers to how much change is made to weights based on what the machine has learnt at a certain timestep. Higher learning rate means an unstable model as it may struggle to fit. Low learning rate could mean the machine learns too slowly so need to find a balance. **Epochs:** The amount of times the model traverses through the dataset. Any possible configuration of the model would have its optimum epochs range so need to make sure this is chosen so as to maximise that configuration's perfornance. **Batch Size:** Refers to how much of the data is fed into the model at any one time. Higher batch size uses up a lot of RAM so need to find a balance with this as low batch size means longer training time. **Train/Test/Validation data split:** How the overall dataset is split. Training data will be used to make weight changes as the model fits to the data. Validation data is for us to see how the data might perform on unseen data and this will give us an idea of what parameters to tune. Test data is what we will not use in the training process, instead it will be held back for the final evaluation. Need to make sure all the splits are representative of the whole dataset as a whole; only then would the final results be valid. One factor that could affect this is the percentage of the spilts for each set.

4.5 Comparison and Evaluation techniques

The first evaluation steps that would be taken is based on the models' performances on the validation set as they train and the final evaluation on the test set. I will use various metrics to see how the models perform on the test set such as categorical accuracy and categorical cross entropy loss for the overall performance.

And, to see which specific action classes are underperforming for each model I will analyse the Precision, Recall and F1 scores for each class when evaluating on the test set. Doing this should allow me to learn about each models' features such as what type of videos specifically does each model struggle or perform well on. For example, one model could be better at using background information than the other when deciding upon the action class. If one action class was "playing golf" (UCF 50) then using the green scenery in the training process is important however the models' shouldn't rely on scenery alone.

Also evaluating the two models on the test set is important because it is an unbiased evaluation of the final model fit on the training set and the closest indication of how the model could perform in real world applications on different video data.

And for even further completeness we will test the two models on external video datasets different to the dataset they were trained on. These other datasets will have to have overlapping action classes. Evaluating the model on these videos would further validate the performance of the two models and it gives them a stronger case for real world use. So, of the two datasets chosen, one will be used for training and testing and the other for further testing using the same models' prediction abilities.

4.6 Pre-trained model/Transfer learning comparison

The two models will be compared to a pre-trained network example from the Keras Applications that are built in [10]. These models have already been trained on many images, so have been tested to a very high accuracy. They come with pre-loaded weights, so they just have to be fine-tuned to train on our chosen dataset using transfer learning. This is when the final few layers are trained on a dataset.

VGG16 is one of the available pre-trained models available for our use. It is a convolutional neural network with 13 convolutional layers, 5 max pooling layers, 3 fully connected layers and 1 output SoftMax layer.

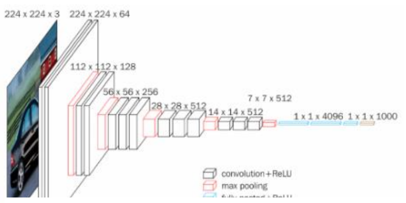


Figure 4:4: VGG16 Architecture [11]

This model was trained on the ImageNet dataset which consists of 15 million images that are in 22,000 different categories and 1000 classes. VGG16 was trained for weeks and achieved a 92.7% top-5 accuracy in the Image-Net dataset. So, using these pre-trained weights will help us when trying to achieve high accuracy using transfer learning on our dataset.

Comparing our two models to a pre-trained network is vital as we can assess the potential advantages and disadvantages such as training time, difficulty in tuning hyper-parameters and performance overall.

4.7 Explanation of metrics

Categorical cross-entropy loss will be used to calculate 'loss'. This is a standard in the Computer Vision/Machine learning community for multi-class classification. It trains a CNN to output a probability over the C classes for each image. The loss is the negative log-likelihood for each sample. It increases as the probability distribution predicted for that sample diverges away from the actual label. Equation below shows how to calculate loss at each stage [21]. Using the loss, the model is able to backpropagate and amend weights accordingly.

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\\text{size}}} y_i \cdot \log \hat{y}_i$$

Figure 4:5: Categorical Loss equation

The **Keras Accuracy class** [20] will be used to calculate the 'accuracy' of the model. It works by calculating how often the model's maximum probability prediction is the correct class for each input. So, it can be seen as a % with 100% (1.00) meaning the model is able to predict every single input in that particular set of data correctly. Categorical accuracy can also be used which is recommended when labels are used in a one-hot encoded format such as in this project. However, it didn't produce any known advantages in terms of performance or speed so it was neglected.

F1 Score/Precision/Recall can be used to analyse the performance of each class from the dataset. Recall is the ratio of correctly predicted positive observations against all predicted observations in that class. An example: for a class "walk", the Recall would be a ratio of how many times "Walk" was predicted against how many times other classes were predicted instead of "Walk". Recall helps when the cost of false negatives is high. Precision is how many times a class is predicted correctly as a ratio of how many times the same class was predicted instead of a different class. So, it scrutinizes False Positives. F1 score is the weighted average of Recall and Precision [22]. It is more useful than ordinary accuracy if we have an uneven class distribution, i.e. there are more videos in some classes than others.

5 Results and Further analysis

In this section I will be comparing the two models completely and how they perform when training and on the test set. A short analysis will be performed on the individual classes in the ‘HMDB51’ dataset to see which classes perform well or underperform and find out why in relation to each model. This should tell us more about the characteristics of each model.

5.1 Conv-2D Tuning

I needed an initial model to first test the model. It is extremely unlikely that I will choose the most optimal hyper parameters at this stage. But by being reasonable with my choices I can get a good enough start and then change parameters accordingly in an attempt to increase accuracy and minimize loss.

Table 5.1: Conv2D initial parameters

Number of filters	**Number of Conv2D layers**	**Kernel Size**	**Hidden activation function**	**Optimization Algorithm**	**Learning Rate**	**Batch Size**
64	2	(3,3)	ReLu	SGD	0.001 (Default)	4

I will test on the HMDB51 dataset first using a subset of the classes due to computation power limitations. I will use the 6 classes: “golf”, “eat”, “dribble”, “climb”, “clap” and “run”. 8000 frames will be chosen per class and that means 48000 frames (from around 700 videos) will be trained on altogether as we are using 6 classes. This will be then split into Train, Validation and Test sets in the ratio 64:16:20. So in fact 30,720 frames will be used for the training process. This will include a “fairly” equal combination from all 6 classes as the order of frames will have been randomly shuffled before the split into individual sets.

Right now, the model is using purely a frame-based training model as explained previously. However, this also means that testing would show results in terms of accuracy on individual frames rather than videos. However later on in the report I will be evaluating further using the “rolling average” technique which would allow me to input frames of individual videos together and choose the most popular prediction as the final prediction for a certain video. But because the test set is shuffled and split with the training/validation set, this is not possible yet.

I will also be using a particular **callback** method called early stopping as it will allow me to save time. I just need to specify a maximum epoch and then when the model stops improving after a certain specified number of epochs in terms of validation loss it will stop training automatically and use the best set of weights computed. So, I can save time this way and also because I do not need to test the model with different epochs ranges.

The “Patience” value of 15 denotes that the model will train for 15 epochs of non-improvement on validation loss before stopping. After stopping the model will revert to the epochs number before the model stopped improving.

Initial Results – 3 Runs

Table 5.2: Conv2D Initial results

Optimal Epochs	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
23	0.2133	0.9287	0.1984	0.9520
34	0.1746	0.9450	0.2637	0.9535
17	0.2460	0.9179	0.1795	0.9533

As you can see the scores are already quite good for this model. There can be a lot of reasons for this. First of all, I am only using 6 action classes which is making the possibility of the model predicting classes lower as there is less classes to choose from. Also, the validation data may not be enough to be representative of the whole dataset in terms of variation between videos and the number of videos in each class.

However, we can still work to improve our scores as much as possible and see if we can get better results before testing this model (and the other) on external datasets. Here are loss and accuracy graphs for the optimal scored model:

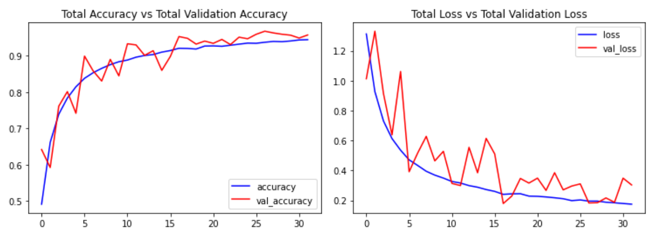


Figure 5:1: Conv2D initial run example validation accuracy and loss graphs

As you can see the model is converging quite steadily and is not taking too long to do so. The validation loss and accuracy are important scores because they are more reflective of how the model could perform on unseen data such as test data/different datasets. In turn this shows how the model could perform in real world use.

The model seems to not be overfitting either. This is because there are no signs of the validation loss increasing as the training loss gets lower and lower. So, it is clear to see that the model generalizes well so far otherwise it would not have performed as well on the validation dataset.

Hyperparameters – Next Steps

There are **structural parameters** to look at such as number of Conv2D, fully connected layers and number of filters/neurons in such layers. This will determine how many features the model is able extract but will also have an effect on training time and computational power needed.

Other parameters include training algorithm, its learning rate and also activation functions.

I will test some of these parameters manually and then for the others use a Random Search. This is so as to reduce the search space of the Random Search. As a very large Random Search will limit its ability to find the optimal configuration and also will take a very long time to compute.

Optimization algorithm and Activation Functions

The **optimization** algorithm (or training algorithm as it is sometime known) is an important part of any Neural Network. It affects how the weights are changed at each layer. For example, the algorithm “Gradient Descent” works by calculating the loss at each layer then backpropagating the loss from one layer to another and in turn modifying the weights depending on the losses so as to minimize the loss [17]. Different optimization algorithms can work better with different model architectures and layer types, so it is good practise to test a variety. For example, SGD is known to work better with shallower networks (so less layers). Accuracy, loss, epochs taken to fit are all variables that can change with different optimization algorithms.

Activation functions and Training algorithm

They are mathematical functions that determine the output of a neural network. It determines whether each neuron should be fired or not based on relevancy of each neurons input to the model's prediction [18]. The main purpose of activation functions is to introduce non-linearity into a neural network. So, the output of neurons isn't just the sum of inputs multiplied by weight plus bias if used. So non-linearity allows us to classify non-linear classification feature sets. So, we can achieve different activation functions will allow us to fit better to each of our chosen dataset classes (or similar classes).

Below I will test a combination of algorithms and activation functions. The output layer activation function will stay as **Softmax** as it is a proven and widely used activation function for multiclass problems like ours.

Table 5.3: Conv2D comparison of training algorithms and hidden layer activation functions

Optimization Algorithm	**Hidden Layer - Activation Function**	**Validation Loss – x2 Average**	**Validation Accuracy – x2 Average**
SGD – (Initial)	Relu	0.3484	0.9552
SGD	tanh	0.4700	0.9158
Adam	Relu	0.1637	0.9712
Adam	tanh	0.5076	0.8848
Adagrad	Relu	0.2140	0.9372
Adagrad	tanh	0.7718	0.7043
RMSprop	Relu	1.3856	0.8599
RMSprop	tanh	0.4580	0.9255

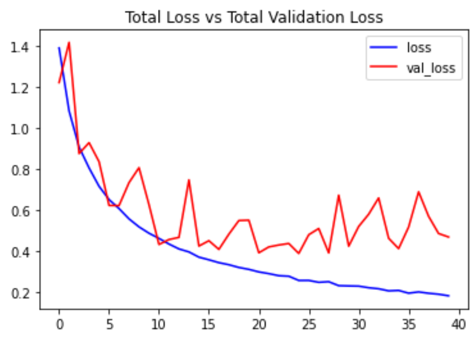


Figure 5:2: SGD/tanh Model is prone to overfitting This model is overfitting as the validation loss is gradually increasing after the initial decrease even as the training loss still steadily decreases. This behaviour was seen repeatedly in all tanh models, sometimes with even worse overfitting.

It is clear from the results that the best option is to use the training algorithm Adam and the activation function ReLu. This correlates to what most data scientists tend to use. Adam is a widely used training algorithm as it combines the best properties of other training algorithms and doesn’t limit its usefulness to just one type of architecture. Adam also handles saddle points well; so, it won’t get stuck in local minima.

ReLU is also the de facto for most deep neural networks. It is less vulnerable to problems such as the vanishing gradient problem so is widely used today. Another advantage of ReLU is that it is known to be computationally less expensive than other activation functions [19] as for example tanh contains exponent functions but ReLU only contains max functions. So, this advantage helps our goal of keeping overall resources needed low for the small business application.

Mixing activation function between layers is also a possibility but it will be not used here as it didn’t show better results from limited testing.

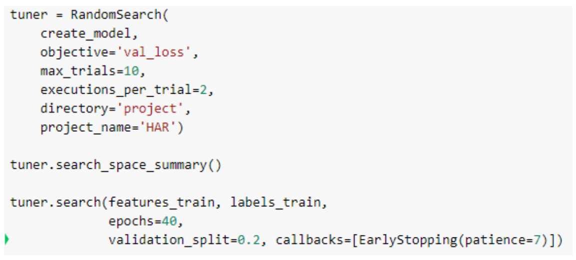
Random Search

Now that the training algorithm and hidden layer activations has been finalized it will make settling on the rest of the hyperparameters easier as there will be a much lower search space. This will save computational time. I will be using Random search to see if there may be a set of structural parameters that will yield better results than what I have achieved thus far. The rest of the parameters apart from Number of hidden layers, neurons, filters and learning rate, will be kept the same.

Table 5.4: Random Search description of search space

Hyperparameter	**Search Space**
Convolutional Filter Size	Min: 32, Max: 256, Step: 32
Number of neurons in fully connected layers	Min: 32, Max: 512, Step: 32
Learning Rate for training algorithm	3 Choices: {0.01, 0.001, 0.0001}
Number of Conv2D layers (Overall)	Min: 2, Max: 5, Step: 1
Number of Fully connected layers	Min: 1, Max: 3, Step: 1

Random search is better than Grid search because it saves time and is able to learn from pervious scores. This is because grid search tests every single combination ion the search space whilst Random Search selects random combinations and more often than not still manages to find the best solution but in a very small amount of time compared to Grid Search.

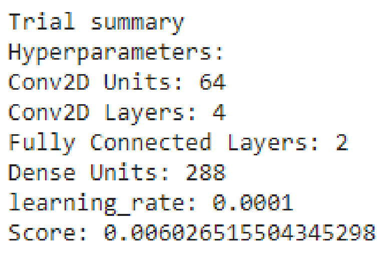


```
tuner = RandomSearch(
    create_model,
    objective='val_loss',
    max_trials=10,
    executions_per_trial=2,
    directory='project',
    project_name='HAR')

tuner.search_space_summary()

tuner.search(features_train, labels_train,
             epochs=40,
             validation_split=0.2, callbacks=[EarlyStopping(patience=7)])
```

Figure 5:3: Random Search setup

Here are the settings I chose. Running 10 trials and each trial twice should allow me to find the optimal setup or close to it. And of course, validation loss is prioritised as the important metric to rank the configurations in. I chose 40 epochs as I wanted there to be enough epochs so that the tuner finds the model with the best loss score as opposed to the model that converges the fastest.

The optimal result was:



```
Trial summary
Hyperparameters:
Conv2D Units: 64
Conv2D Layers: 4
Fully Connected Layers: 2
Dense Units: 288
learning_rate: 0.0001
Score: 0.006026515504345298
```

Figure 5:4: Random search best result

The validation loss is significantly lower than before and after using the suggested configuration I was able to see validation accuracy in the region of > 0.99. This is better than my previous results and should be a good sign that the model will perform well on a different dataset as well which will be one of my testing methods.

Adding more convolutional layers means more features such as low-level features in each image frame is detected. This can be lines and other shapes. So, each class will be able to be predicted more accurately as they are all likely to have unique low-level features so this can explain the increase in accuracy and decrease in loss. Adding more dense layers also could have increased accuracy as the have more parameters.

Adding more layers however needs to be considered carefully as it is computationally more expensive and significantly slows down training times. However, the improvement is significant so the changes will be kept.

5.2 ConvLSTM Tuning

I needed an initial model to first test this model as well. It is extremely unlikely that I will choose the most optimal hyper parameters at this stage. But by being reasonable with my choices I can get a good enough start and then change parameters accordingly in an attempt to increase accuracy and minimize loss.

Table 5.5: ConvLSTM initial parameters

Number of filters	Number of ConvLSTM layers	Kernel Size	Hidden Dense activation function	Optimization Algorithm	Learning Rate	Batch Size
64	1	(3,3)	ReLu	SGD	0.001	8

I will test on the same 6 classes as before from the same dataset. This time the videos are being fed into the neural network in a different format due to the differences in architecture. Instead of frames being fed independent of each other; frames of each video will be sent in chronological order independent of another video. A sequence of length of 70 will be used regardless of video length: so, 70 frames in order. This will allow us to take advantage of the usefulness of LSTM's which is to learn from temporal differences in data.

Videos will be split into train, test and validation sets as before in the same ratio and shuffled so videos from different classes are fed into the model in random order. Normalisation will not be used in this model as initial testing showed worse results.

I will be using the same callback patience value of 15 as this worked well with the previous model but reduce it if it means I can keep the same performance but save time.

Initial Results – 3 Runs

Table 5.6: ConvLSTM initial results

Optimal Epochs	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy

17	0.0345	0.9980	0.6234	0.8081
14	0.0289	0.9996	0.5964	0.8283
15	0.2200	0.9299	0.5305	0.8485

The initial performance already shows good performance. However, the training performance scores seems to be consistently better. This is not a sign of overfitting as the validation scores aren't improving before worsening; in fact, validation scores are also improving but are just consistently worse than the scores on the training set:

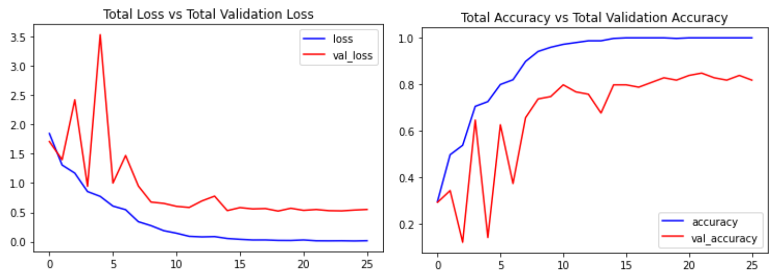


Figure 5:5:ConvLSTM2D initial run example validation accuracy and loss graphs

This is perhaps because the validation set is not representative of the overall dataset and/or is too small. This is because we are testing on individual videos now as opposed to individual frames like the previous model. So, there is fewer independent data samples because of this. Therefore, varying the validation split value will be considered: this slightly increased validation accuracy but the gap between training and validation scores stayed roughly the same. However, the change will be kept for now.

Activation function and Training algorithm

Next, I will be testing various different combinations of activation functions and optimizers before moving on to Random Search for the structural parameters as before. This is to minimize the search space of Random Search. The default and recommended activation function for ConvLSTM2D layers is "tanh". However, I will vary this and the hidden dense layer activation function to find improvements along with the training algorithm. And I will test the well-known and widely used algorithms SGD and Adam alongside RMSprop which is known to work well with recurrent/LSTM architecture (helps avoid the vanishing gradient problem.

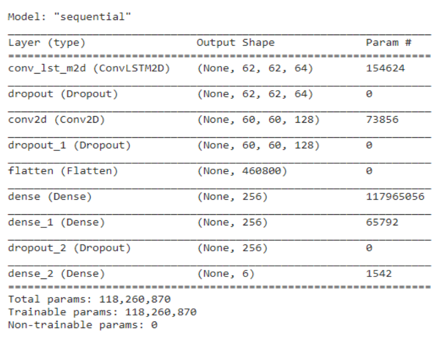
Table 5.7: ConvLSTM comparison of training algorithms and hidden layer activation functions

Optimization Algorithm	**Activation Function - ConvLSTM/Dense**	**Validation Loss – x2 Average**	**Validation Accuracy – x2 Average**
SGD	Tanh/Tanh	0.5919	0.8283
SGD	Relu/Tanh	nan	0.2222
SGD	Tanh/Relu	0.5193	0.8586
Adam	Tanh/Tanh	1.9132	0.1717
Adam	Tanh/Relu	1.1115	0.5758
RMSprop	Tanh/Tanh	1.3738	0.4444
RMSprop	Tanh/Relu	1.9533	0.5248

The ConvLSTM2D layer seems to work best with its default activation function: “tanh” whilst with ‘relu’ the model seems to not be able to fit to the data at all. This can be surprising as ‘relu’ often outperforms ‘tanh’ in most machine learning projects but for this particular layer the characteristics of ‘tanh’ allows it to perform better. Something to look out for here is that ‘relu’ is known to be sparse so more efficient in its procedure so we may be losing out on this front.

Adding Layers and Filters

Usually for these types of networks customary to add many layers and filters. This includes extra ConvLSTM2D layers and also Conv2D layers which can be used to extract further features. This should allow the accuracy to increase but will heavily slow down training times. Unfortunately to my system limitations I am unable to test extra ConvLSTM2D layers due to a “Resource allocation” error which means the GPU has been used to its maximum capacity. However, adding further Conv2D layers after the ConvLSTM2D layer helped increase accuracy and decrease loss to < 0.55. So, the final model is:



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_lst_m2d (ConvLSTM2D)    (None, 62, 62, 64)        154624
_________________________________________________________________
dropout (Dropout)            (None, 62, 62, 64)        0
_________________________________________________________________
conv2d (Conv2D)              (None, 60, 60, 128)       73856
_________________________________________________________________
dropout_1 (Dropout)          (None, 60, 60, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 460800)            0
_________________________________________________________________
dense (Dense)                (None, 256)               117965056
_________________________________________________________________
dense_1 (Dense)              (None, 256)               65792
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 1542
=================================================================
Total params: 118,260,870
Trainable params: 118,260,870
Non-trainable params: 0
```

Figure 5:6: ConvLSTM2D final model

5.3 Final Results on Test Set

Test Set classes/index: golf/0, eat/1, dribble/2, climb/3, clap/4, run/5

Below are the results whilst using the final models with the optimal hyperparameters found previously. To introduce an element of fairness I will use the same dataset, classes and epochs/callback configuration. Using a patience value of 10 for both should allow them to both find the optimal weights configuration but also penalize the models for taking too long to fit.

Three runs were taken for each model and the results are for the test set:

Conv2D

Table 5.8: Final results on test set

Model	Epochs	Test Loss	Test Accuracy
Conv2D	• 35 • 40 • 20	• 0.0059 • 0.0045 • 0.0088	• 0.9987 • 0.9984 • 0.9979
ConvLSTM2D	• 27 • 27 • 31	• 0.8744 • 0.9057 • 0.8297	• 0.8145 • 0.7984 • 0.8065

Conv2D is comfortably able to outperform the ConvLSTM mode even whilst neglecting the temporal features of videos. It seems to be able analyse the spatial features to a very high level and is almost perfect in being able to predict on the test set. However, the ConvLSTM2D model is also able to perform to a high level with ~80% accuracy not a bad score at all. If anything, the Conv2D model is a strong feature extractor/image classifier.

The reason the ConvLSTM model could be underperforming is because it inherently has less data points to work with compared to the other model which works with individual frames. So, it may be harder to fit and achieve very high accuracy. Also, ConvLSTM usually takes longer to fit as it is a more complex layer so given more time it may do better. However, the robustness of these two models will be tested later as it could be that the Conv2D model is just performing really well on this dataset but may struggle with external data.

Conv2D

Table 5.9: Conv2D Confusion Matrix and individual class scores

Class	0	1	2	3	4	5

0	1602	0	0	0	0	2
1	0	1612	0	0	1	2
2	0	0	1590	2	0	1
3	0	0	0	1584	0	1
4	0	0	0	0	1602	0
5	0	10	2	2	0	1587

	0	**1**	**2**	**3**	**4**	**5**	**Average**
Precision	1.00	0.99	1.00	1.00	1.00	1.00	0.9983
Recall	1.00	1.00	1.00	1.00	1.00	0.99	0.9983
F1	1.00	1.00	1.00	1.00	1.00	1.00	1.00

ConvLSTM2D

Table 5.10: ConvLSTM Confusion matrix and individual class scores

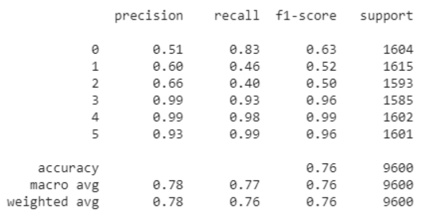
Class	**0**	**1**	**2**	**3**	**4**	**5**
0	25	0	0	0	0	1
1	0	3	0	0	1	4
2	0	0	26	2	0	0
3	1	0	3	19	2	2
4	0	1	0	4	14	1
5	1	2	1	3	1	7

	0	**1**	**2**	**3**	**4**	**5**	**Average**
Precision	0.93	0.50	0.87	0.68	0.78	0.47	0.705
Recall	0.96	0.38	0.93	0.70	0.70	0.47	0.690
F1	0.94	0.43	0.90	0.69	0.74	0.47	0.695

These results show that the first model is consistent across the board and is easily able to detect the relevant features from each class and apply it when predicting. The second model seems to be struggling with the classes “eat” and “run”. These are the only classes also where the first model doesn’t get perfect scores. These could be due to the fact that they are very “normal” actions that can be hard to distinguish between other actions. They may also have low background variation so the models are predicting other actions ahead of the correct class. Or the other way around because of them appearing as basic motions; other classes could mis-predict these two classes instead. So, to improve this perhaps a deeper model is needed or more training time is needed for the second model.

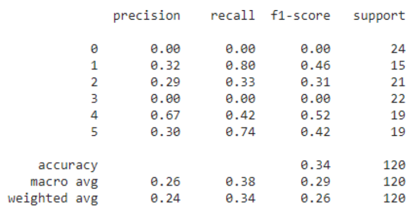
5.4 Train on KTH Dataset

I will test on the KTH dataset [23] as it is black and white dataset as opposed to RGB. This should allow me to see which of the two models rely on the background more as this would mean they are less robust. Because for example in a surveillance camera use case the models would have to learn to process low light footage or black and white footage itself from such cameras and still be able to detect human action. This can only be achieved by a model that is reliable in different conditions and on multiple types of video and this would require the models to generalise and not overfit to the training data more than ever.



```
              precision    recall  f1-score   support

           0       0.51      0.83      0.63      1604
           1       0.60      0.46      0.52      1615
           2       0.66      0.40      0.50      1593
           3       0.99      0.93      0.96      1585
           4       0.99      0.98      0.99      1602
           5       0.93      0.99      0.96      1601

    accuracy                           0.76      9600
   macro avg       0.78      0.77      0.76      9600
weighted avg       0.78      0.76      0.76      9600
```

Figure 5:8: Conv2D Result



```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        24
           1       0.32      0.80      0.46        15
           2       0.29      0.33      0.31        21
           3       0.00      0.00      0.00        22
           4       0.67      0.42      0.52        19
           5       0.30      0.74      0.42        19

    accuracy                           0.34       120
   macro avg       0.26      0.38      0.29       120
weighted avg       0.24      0.34      0.26       120
```

Figure 5:7: ConvLSTM Result

It is suprising that the Conv2D is able to outperform the ConvLSTM2D model in this test however there may be any reasons for this. Once reason is that the Conv2D optimal model chosen is much deeper than the ConvLSTM model so it is able to extract more features beyond the surface level pixels.

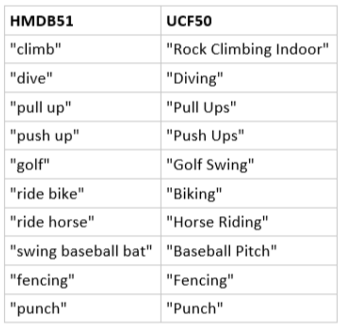
5.5 Test on different dataset

I will test on a different dataset to ‘hmdb51’ but with similar actions to analyse whether the model’ are robust enough for real world use. As of now we have only tested the model on the same dataset as it was trained on. This makes it easier for the model to predict on the test set as we can expect the videos to all be vaguely similar, especially since we are randomizing the videos each time before training; this means that videos from the same source can be in all three sets (train, validation and test) potentially.

Therefore, training on ‘hmdb51’ and testing on a different dataset seems to be a viable test that tests the robustness of the two models. Especially since in real life use if a model is only useful on the same data/similar data it has been trained on then it is useless as there is nothing to gain from that, in a business perspective particularly. Thinking back to the aforementioned possible business applications this seems logical, particularly in a surveillance system application, because we would expect new footage to be captured constantly and then be fed into the model. This would require model’s to constantly adapt and work well on different types of data with different backgrounds, that’s why this test is actually quite significant.

I will use the ‘UCF50’ as the test set. So, the models will be trained using the same procedure as before on the ‘hmdb51’ dataset then tested on the same 6 classes in the ‘UCF50’ dataset.

The potential classes to be used are below:



HMDB51	UCF50
"climb"	"Rock Climbing Indoor"
"dive"	"Diving"
"pull up"	"Pull Ups"
"push up"	"Push Ups"
"golf"	"Golf Swing"
"ride bike"	"Biking"
"ride horse"	"Horse Riding"
"swing baseball bat"	"Baseball Pitch"
"fencing"	"Fencing"
"punch"	"Punch"

Figure 5:3: Similar classes from hmdb51 and UCF50 datasets

I will choose 6 classes for this test and will try to mix classes that have a unique and vivid background such as “golf” with those with more ordinary backgrounds such as “punch”. The process will be easy to implement; we just need to make sure that the model knows which class is which. This can be achieved simply by ordering the classes as before in the training step.

Comparing the classes further will also be required. This is because the models will understandably perform less well than expected if the similar actions are still very different in types of source. But on the other hand, they will perform better than expected if there are repeated videos in the datasets/very similar videos/videos from the same source (such as same movie).

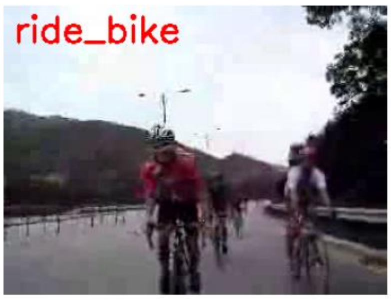


Figure 5:10 UCF50: Biking

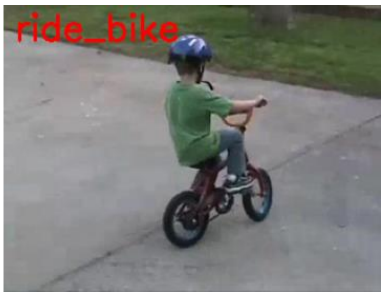


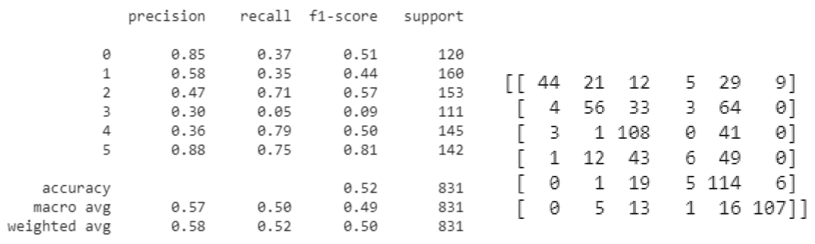
Figure 5:9 hmdb51: ride_bike

Figure 5:4 and 5:3 shows two quite different videos from similar datasets. They differ in background and in main human subject so it would be quite challenging for the models but if they are able to generalise well then, the bike should be the deciding factor that allows them to predict dependably.

For the Conv2D model, implementing this would be a bit tricky as so far, we have only tried training on frames from different videos together without feeding in individual videos. So, the model has to be adapted to work to predict individual videos. The rolling average method was used; whereby every other frame from a video is predicted in order and the average prediction (the mode) of all the frames is the final prediction for the whole video.

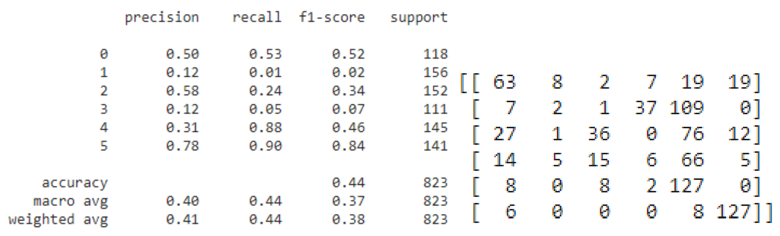
I will be training on the following classes from hmdb51: ["pullup", "punch", "dive", "fencing", "ride_bike", "golf"] and predicting on the following classes from UCF50: [“Pull Ups”,” Punch”,” Diving”,” Fencing”,” Biking”,” Golf Swing”]

Results



```
              precision    recall  f1-score   support

           0       0.85      0.37      0.51       120
           1       0.58      0.35      0.44       160      [[ 44  21  12   5  29   9]
           2       0.47      0.71      0.57       153       [  4  56  33   3  64   0]
           3       0.30      0.05      0.09       111       [  3   1 108   0  41   0]
           4       0.36      0.79      0.50       145       [  1  12  43   6  49   0]
           5       0.88      0.75      0.81       142       [  0   1  19   5 114   6]
                                                           [  0   5  13   1  16 107]]
    accuracy                           0.52       831
   macro avg       0.57      0.50      0.49       831
weighted avg       0.58      0.52      0.50       831
```

Figure 5:11 Conv2D Results



```
              precision    recall  f1-score   support

           0       0.50      0.53      0.52       118
           1       0.12      0.01      0.02       156   [[ 63   8   2   7  19  19]
           2       0.58      0.24      0.34       152    [  7   2   1  37 109   0]
           3       0.12      0.05      0.07       111    [ 27   1  36   0  76  12]
           4       0.31      0.88      0.46       145    [ 14   5  15   6  66   5]
           5       0.78      0.90      0.84       141    [  8   0   8   2 127   0]
                                                        [  6   0   0   0   8 127]]
    accuracy                           0.44       823
   macro avg       0.40      0.44      0.37       823
weighted avg       0.41      0.44      0.38       823
```

Figure 5:12 ConvLSTM Results

The results show that both models underperform on video types that are quite different to what they have been trained on. Both models were strong on the “Golf” class. This makes sense because even if the videos were of different origin to the ones in the “hmdb51” class, they are all likely to have very similar background with predominantly green scenery in the background. The “fencing” class was surprisingly the weakest. Despite having quite unique actions, the fact that it had different backgrounds perhaps let it down. This is a problem with the models as it means spatial features are being learned more than temporal for the ConvLSTM. And the Conv2D is failing to detect small features in different backgrounds such as a fencing sabre or bicycle.

The other classes on average performed with 50% accuracy which is still quite poor depending on the models’ use cases. In a surveillance environment this would be too low as false positive predictions with actions such as criminal activity would lead to a lot of trouble with the emergency services being alerted for no good reason.

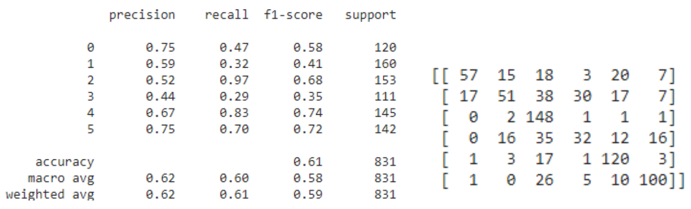
Another reason for this poor performance could be that both models are unable to sufficiently capture the complexities of the original dataset (“hmdb51”). Perhaps because the models are not too deep, only surface features are being detected. This would mean small objects in the frames would be neglected. However, I was unable to further deepen my networks due to hardware/GPU limitations but I will compare to a pre-trained network using transfer learning (so no need to train the whole network) to analyse if this is the case because an ImageNet pre-trained network such as VGG16 should be able to extract more features from each video frame.

5.6 Comparison with Pre-Trained network

Pre-trained networks are proven feature extractors so we can test them with the same dataset and configurations to see if my hardware is a limiting factor for performance and if a much deeper network is likely to perform better in Human Activity Recognition. So, I will therefore know if the weakness of the two models is extracting deeper features.

If the pre-trained network also underperforms then it is clear that the hardware I have is the limiting factor and perhaps with better GPU then my models will also perform better like the pre-trained model no doubt would. So, seeing how VGG16 performs on limited hardware we can predict how the current developed models could perform with better hardware. VGG16 is a proven network that has performed well on the ImageNet dataset [11]. Also using transfer learning should help with our problems on training with insufficient data as we will be using pre-trained weights that have been trained on the ImageNet dataset. This should allow for higher accuracy rates.

The complete same tests will be run as before, with training/testing on the “hmdb51” dataset then further testing the same classes on the “UCF50 dataset: "pullup", "punch", "dive", "fencing", "ride_bike" and "golf".



```
              precision    recall  f1-score   support

           0       0.75      0.47      0.58       120
           1       0.59      0.32      0.41       160
           2       0.52      0.97      0.68       153
           3       0.44      0.29      0.35       111
           4       0.67      0.83      0.74       145
           5       0.75      0.70      0.72       142

    accuracy                           0.61       831
   macro avg       0.62      0.60      0.58       831
weighted avg       0.62      0.61      0.59       831
```

```
[[ 57  15  18   3  20   7]
 [ 17  51  38  30  17   7]
 [  0   2 148   1   1   1]
 [  0  16  35  32  12  16]
 [  1   3  17   1 120   3]
 [  1   0  26   5  10 100]]
```

Figure 5:13: Pre-trained model results

This model clearly performed better than the two models. It shows that even without learning from temporal features, neural networks can still achieve very good results in terms of video classification/human activity recognition. It is also a predominantly Conv2D model like one of the models we developed so it just goes to show that a deeper model is needed for this type of problem. Because it is able to learn and detect more obscure features in each frame and patterns between frames. This could also mean that a deeper ConvLSTM network could perhaps generalise better than the one we developed and so get higher rates of accuracy when being tested on a different dataset or on new forms of input data in a proper real-world application.

Also, as you can see in Figure 5:7, the model did less well in the “golf” class than the other datasets but managed to do well across all classes overall. This shows that the model is able to generalise for all classes and it is starting to properly identify what differs between each class beyond just the background.

5.7 Resource usage

Google Colab Pro Specs: Tesla P100 GPU, 16GB GPU RAM and 24GB RAM.

Using Wandb [23] it is possible to investigate which model uses resources the most efficiently:

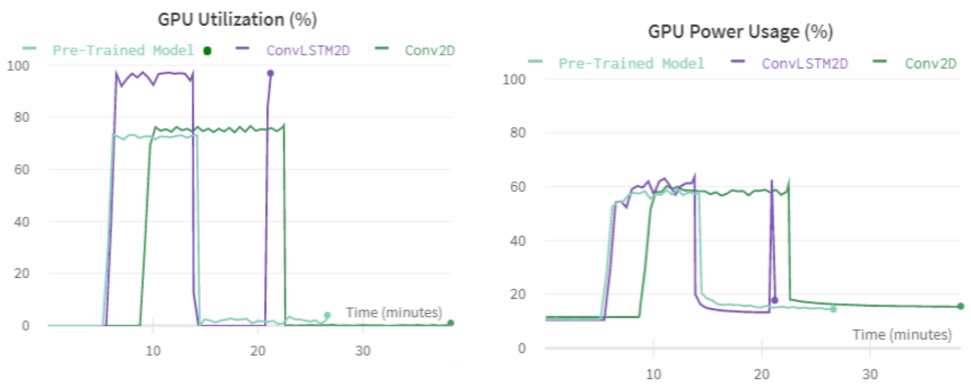


Figure 5:14: GPU stats for all three models

It is clear to see that the ConvLSTM requires more out of the GPU than the others and uses slightly more power also. The fact that the Conv2D model trains for longer shouldn't be misunderstood however as call back was used for all three models and in this instance, it took the Conv2D model longer to train.

Considering these stats, a small business might consider the other two models over the ConvLSTM model due to wanting to save power and therefore money. It is clear to me that the ConvLSTM model requires more GPU due to training over whole videos at a time but the proposed benefit of this, which is learning from temporal information, just isn't being seen. This puts the other two models ahead as they are able to get better scores on the test set and when testing on a different dataset altogether. Complete resource usage results are shown in Appendix C.

6 Conclusions and Future work

After analysing various methods for Human Activity Recognition, I was able to settle on two deep learning methods that have been used mainly for other applications such as Image Classification and Precipitation forecasting and apply them to video classification and compare the two methods to see which was the all-round best in terms of performance scores, computation expenses and difficulty of implementation.

Out of the two methods I tested, Conv2D seems to be the better method. Not just because of the performance scores but also because it's hyperparameters were able to be tuned much easier in comparison to the ConvLSTM model. This means that for data scientists it is easier to implement this type of algorithm in day to day running. The Conv2D model is also more effective in videos with low variation in background such as in the KTH dataset test. This is an indication that if this type of model is used to analyse night-time footage from CCTV then the Conv2D model will perform better.

However, there are many other tests and work to be done on this topic that can make this piece of work more impactful. For example, one improvement that can be made is to carefully choose the types of videos to use when training. This could be achieved by perhaps combining datasets together, this is so that we can see how each model will perform when each and every type of video in a particular class is trained on. For example, in the action "pull up" if we include videos from different places such as gym, home, garden and less frequent places such as beach then the model is more likely to be able to perform to a high standard as it won't underperform on the more unique videos. It will especially perform better in real world use on never tested before videos.

Also, another test that can be performed is to analyse how the models may improve if the image height and width constants are increased. Unfortunately, due to computation/resource constraints, we were not able to increase the value for both above 64. Anything higher induced a "Resource Allocation" error whilst loading the videos or during training. So, training with larger images will allow the model to read more information from the images. More so than the current tests allowed us to, this may mean the models are able to improve their understanding of images beyond the background and on subtle information such as objects and the human's stance and pose. Coupling this with many layers should help build more clever models in the future.

7 References

[1] ‘Understanding LSTM Networks’, Github.io. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed: 24-Apr-2021].

[2] M. Babiker, O. O. Khalifa, K. K. Htike, A. Hassan, and M. Zaharadeen, ‘Automated daily human activity recognition for video surveillance using neural network’, in 2017 IEEE 4th International Conference on Smart Instrumentation, Measurement and Application (ICSIMA), 2017, pp. 1–5.

[3] J. M. Chaquet, E. J. Carmona, and A. Fernández-Caballero, ‘A survey of video datasets for human action and activity recognition’, Comput. Vis. Image Underst., vol. 117, no. 6, pp. 633–659, 2013.

[4] Z. Wu, X. Wang, Y.-G. Jiang, H. Ye, and X. Xue, ‘Modeling spatial-temporal clues in a hybrid deep learning framework for video classification’, in Proceedings of the 23rd ACM international conference on Multimedia, 2015.

[5] S. Saha, ‘A comprehensive guide to convolutional neural networks — the ELI5 way’, Towards Data Science, 15-Dec-2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed: 24-Apr-2021].

[6] M. Kirkup and M. Carrigan, ‘Video surveillance research in retailing: ethical issues’, Int. j. retail distrib. manag., vol. 28, no. 11, pp. 470–480, 2000.

[7] Keras Team, ‘Developer guides’, Keras.io. [Online]. Available: https://keras.io/guides/. [Accessed: 24-Apr-2021].

[8] Nips.cc. [Online]. Available: https://papers.nips.cc/paper/2015/file/07563a3fe3bbe7e3ba84431ad9d055af-Paper.pdf. [Accessed: 24-Apr-2021].

[9] A. Xavier, ‘An introduction to ConvLSTM - Neuronio - Medium’, Neuronio, 25-Mar-2019. [Online]. Available: https://medium.com/neuronio/an-introduction-to-convlstm-55c9025563a7. [Accessed: 24-Apr-2021].

[10] Keras Team, ‘Keras Applications’, Keras.io. [Online]. Available: https://keras.io/api/applications/. [Accessed: 28-Apr-2021].

[11] ‘VGG16 - convolutional network for classification and detection’, Neurohive.io, 20-Nov-2018. [Online]. Available: https://neurohive.io/en/popular-networks/vgg16/. [Accessed: 28-Apr-2021].

[12] S. Gautam, P. Kaur, and M. Gangadharappa, ‘An overview of human activity recognition from recordings’, in 2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), 2018, pp. 921–928.

[13] A. Sargano, P. Angelov, and Z. Habib, ‘A comprehensive review on handcrafted and learning-based action representation approaches for human activity recognition’, Appl. Sci. (Basel), vol. 7, no. 1, p. 110, 2017.

[14] ‘Extract images from video in Python - GeeksforGeeks’, Geeksforgeeks.org, 29-Aug-2018. [Online]. Available: https://www.geeksforgeeks.org/extract-images-from-video-in-python/. [Accessed: 03-May-2021].

[15] T. Stöttner, ‘Why Data should be Normalized before Training a Neural Network’, Towards Data Science, 16-May-2019. [Online]. Available: https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-c626b7f66c7d. [Accessed: 06-May-2021].

[16] J. Brownlee, ‘How to use data scaling improve deep learning model stability and performance’, Machinelearningmastery.com, 03-Feb-2019. [Online]. Available: https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/. [Accessed: 06-May-2021].

[17] S. Kumar, ‘Overview of various Optimizers in Neural Networks’, Towards Data Science, 09-Jun-2020. [Online]. Available: https://towardsdatascience.com/overview-of-various-optimizers-in-neural-networks-17c1be2df6d5. [Accessed: 09-May-2021].

[18] A. Saha, ‘The need for activation function along with hidden layers in a Neural Network’, Analytics Vidhya, 30-Jan-2020. [Online]. Available: https://medium.com/analytics-vidhya/the-need-for-activation-function-along-with-hidden-layers-in-a-neural-network-8b37b4bffa42. [Accessed: 09-May-2021].

[19] Chris, ‘ReLU, Sigmoid, Tanh: activation functions for neural networks – MachineCurve’, Machinecurve.com, 04-Sep-2019. [Online]. Available: https://www.machinecurve.com/index.php/2019/09/04/relu-sigmoid-

and-tanh-todays-most-used-activation-functions/. [Accessed: 10-May-2021].

[20] Keras Team, ‘Accuracy metrics’, Keras.io. [Online]. Available: https://keras.io/api/metrics/accuracy_metrics/. [Accessed: 14-May-2021].

[21] ‘Categorical crossentropy’, Peltarion.com. [Online]. Available: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy. [Accessed: 14-May-2021].

[22] ‘Accuracy, precision, recall & F1 score: Interpretation of performance measures - exsilio blog’, Exsilio.com, 09-Sep-2016. [Online]. Available: https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/. [Accessed: 14-May-2021].

[23] ‘Recognition of human actions’, Kth.se. [Online]. Available: https://www.csc.kth.se/cvap/actions/. [Accessed: 21-May-2021].

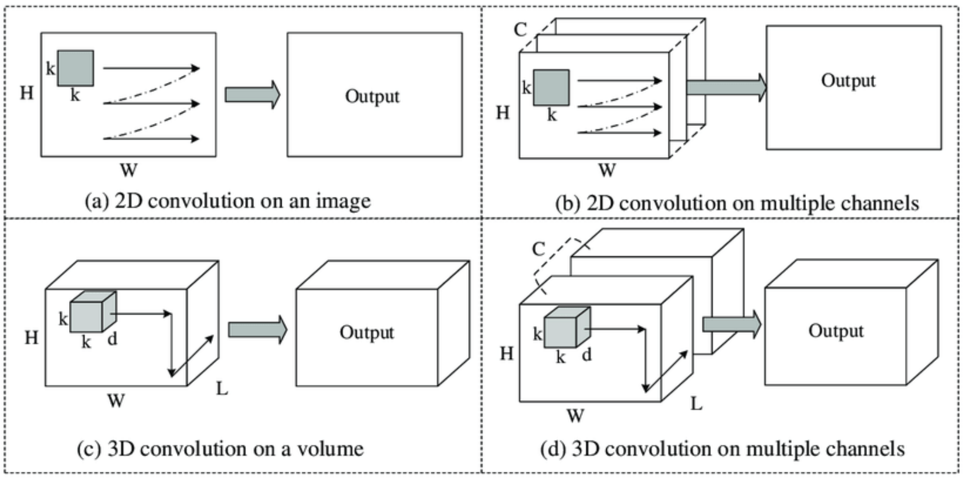
[24] ‘Weights & Biases – Developer tools for ML’, Wandb.ai. [Online]. Available: https://wandb.ai/site. [Accessed: 24-May-2021].

[25] A. Y. Shdefat, A. A. Halimeh, and H. C. Kim, ‘Human activities recognition via smartphones using supervised machine learning classifiers’, Prim. Health Care, vol. 08, no. 01, 2018.

[26] Y. Zheng, H. Bao, and C. Xu, ‘A method for improved pedestrian gesture recognition in self-driving cars’, Aust. J. Mech. Eng., vol. 16, no. sup1, pp. 78–85, 2018.

8 Appendices

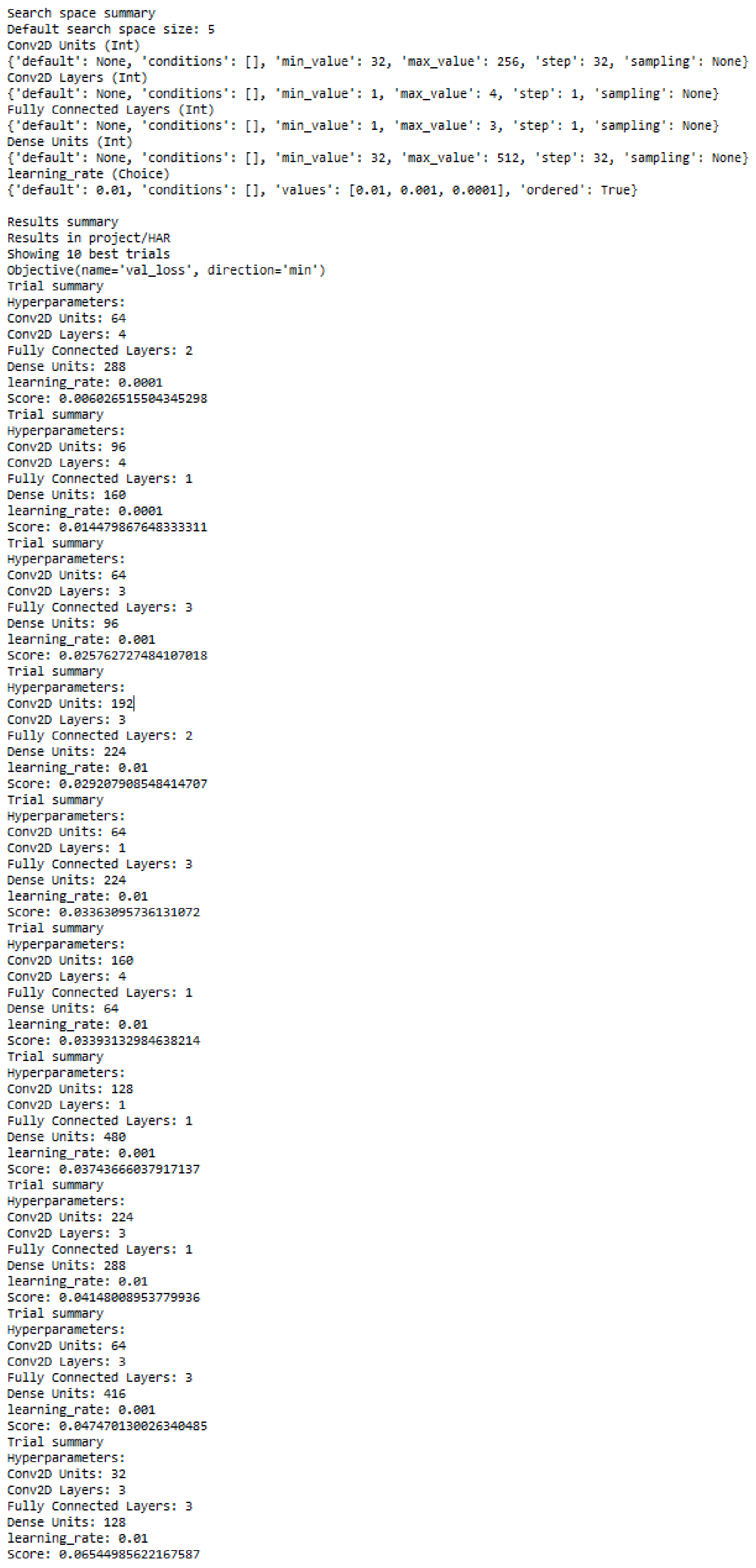
8.1 Appendix A



H
k
k
W
Output
(a) 2D convolution on an image
C
H
k
k
W
Output
(b) 2D convolution on multiple channels
H
k
k
d
W
L
Output
(c) 3D convolution on a volume
C
H
k
k
d
W
L
Output
(d) 3D convolution on multiple channels

8.2 Appendix B

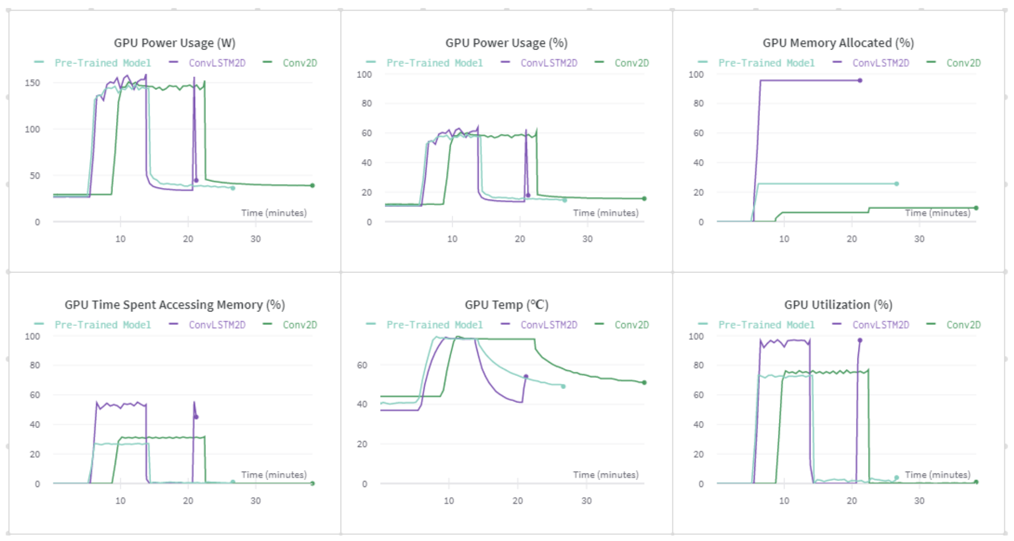
Below are the full results of the Random Search I performed for the Conv2D model. The search space is described at the top and the top ten results follows in order of best to worst combination of parameters.

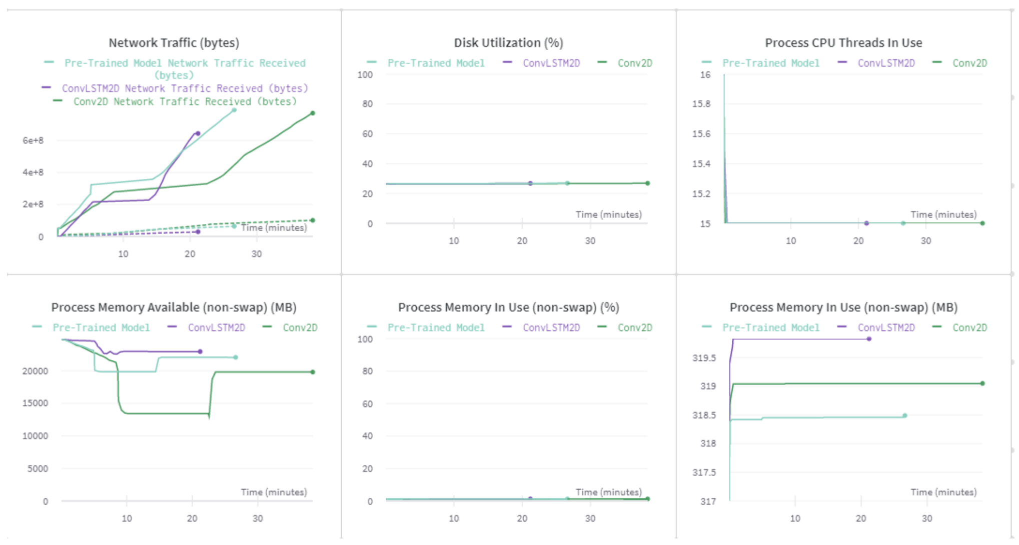


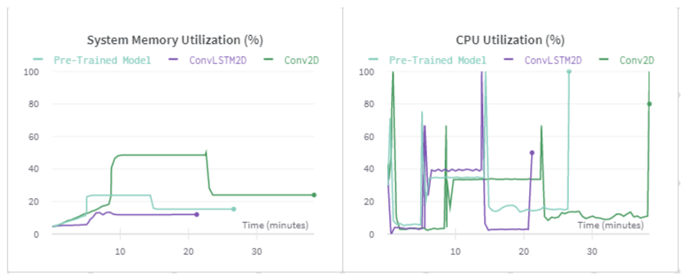
```
Search space summary
Default search space size: 5
Conv2D Units (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 256, 'step': 32, 'sampling': None}
Conv2D Layers (Int)
{'default': None, 'conditions': [], 'min_value': 1, 'max_value': 4, 'step': 1, 'sampling': None}
Fully Connected Layers (Int)
{'default': None, 'conditions': [], 'min_value': 1, 'max_value': 3, 'step': 1, 'sampling': None}
Dense Units (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 'sampling': None}
learning_rate (Choice)
{'default': 0.01, 'conditions': [], 'values': [0.01, 0.001, 0.0001], 'ordered': True}

Results summary
Results in project/HAR
Showing 10 best trials
Objective(name='val_loss', direction='min')
Trial summary
Hyperparameters:
Conv2D Units: 64
Conv2D Layers: 4
Fully Connected Layers: 2
Dense Units: 288
learning_rate: 0.0001
Score: 0.006026515504345298
Trial summary
Hyperparameters:
Conv2D Units: 96
Conv2D Layers: 4
Fully Connected Layers: 1
Dense Units: 160
learning_rate: 0.0001
Score: 0.014479867648333311
Trial summary
Hyperparameters:
Conv2D Units: 64
Conv2D Layers: 3
Fully Connected Layers: 3
Dense Units: 96
learning_rate: 0.001
Score: 0.025762727484107018
Trial summary
Hyperparameters:
Conv2D Units: 192
Conv2D Layers: 3
Fully Connected Layers: 2
Dense Units: 224
learning_rate: 0.01
Score: 0.029207908548414707
Trial summary
Hyperparameters:
Conv2D Units: 64
Conv2D Layers: 1
Fully Connected Layers: 3
Dense Units: 224
learning_rate: 0.01
Score: 0.03363095736131072
Trial summary
Hyperparameters:
Conv2D Units: 160
Conv2D Layers: 4
Fully Connected Layers: 1
Dense Units: 64
learning_rate: 0.01
Score: 0.03393132984638214
Trial summary
Hyperparameters:
Conv2D Units: 128
Conv2D Layers: 1
Fully Connected Layers: 1
Dense Units: 480
learning_rate: 0.001
Score: 0.03743666037917137
Trial summary
Hyperparameters:
Conv2D Units: 224
Conv2D Layers: 3
Fully Connected Layers: 1
Dense Units: 288
learning_rate: 0.01
Score: 0.04148008953779936
Trial summary
Hyperparameters:
Conv2D Units: 64
Conv2D Layers: 3
Fully Connected Layers: 3
Dense Units: 416
learning_rate: 0.001
Score: 0.047470130026340485
Trial summary
Hyperparameters:
Conv2D Units: 32
Conv2D Layers: 3
Fully Connected Layers: 3
Dense Units: 128
learning_rate: 0.01
Score: 0.06544985622167587
```

8.3 Appendix C

Below is a permanent link to the report containing computational costs of all three models when entirely run:







A PDF of these full results will also be supplied alongside this report