



University of Sevilla

Master's Degree in Logic, Computing and Artificial Intelligence

Master's Thesis

# **Human Action Recognition with Deep Learning**

Department of Computer Science and Artificial Intelligence

*Emre Tatbak*

Supervised by

Prof. Miguel Ángel Martínez del Amor, Ph.D

Sevilla, June 2020



Universidad de Sevilla

Máster Universitario en Lógica, Computación e Inteligencia Artificial

Trabajo Fin de Máster

Clasificación de Actividades Humanas en Vídeo

Departamento de Ciencias de la Computación e Inteligencia Artificial

Autor: Emre Tatbak

Tutor: Miguel Ángel Martínez del Amor

Sevilla, Junio 2020

## **ACKNOWLEDGMENTS**

I would like to express my special thanks of gratitude to my professor and supervisor, Miguel Ángel Martínez del Amor. I have a deep regard for him due to his endless support and kindness during my education and master's thesis period.

Besides my professor, I would like to thank my lovely family and especially my brother Enes. None of the words is enough to explain my gratitude about their support and trust.

Lastly, special thanks to Bekir Girgin, who supported me during my education and stay in Spain.

## **ABSTRACT**

Nowadays, self-learning models and artificial intelligence are popular. These systems can be seen in daily life almost in every field. Artificial intelligence makes our life easier than we expected before. Now we can drive safer and easier with self-driving cars, we can predict our monthly expenses, in medical usage we can predict cancer cells with machine learning and also many other applications.

Neural network is an effective tool for image recognition by computer vision algorithms. They work similar to human brain neural systems to recognize objects, their locations and also they can classify within multiple objects.

With this project we will see how we can detect human actions on video camera with deep learning models. Mainly our goal is train a neural network model to recognize human activities on video and live camera. Our project has two stages; firstly only human body detection in all video, then using this video clip as the input of our deep learning model. Finally we classify the actions during all video.

# Table of Contents

- 1. INTRODUCTION ..... 1
  - 1.1 Objective ..... 1
- 2. METHODS ..... 3
  - 2.1 Artificial Neural Networks ..... 3
    - 2.1.1 Training of Artificial Neural Networks ..... 4
      - 2.1.1.1 Forward Propagation ..... 5
      - 2.1.1.2 Backward Propagation ..... 5
      - 2.1.1.3 Loss Function ..... 6
        - 2.1.1.3.1 Mean Square Error ..... 7
        - 2.1.1.3.2 Mean Absolute Error ..... 7
        - 2.1.1.3.3 Cross Entropy Loss ..... 7
    - 2.2 Convolutional Neural Networks ..... 8
      - 2.2.1 Structure of Convolutional Neural Networks ..... 9
        - 2.2.1.1 Convolutional Kernel ..... 9
        - 2.2.1.2 Activation Function ..... 10
          - 2.2.1.2.1 Sigmoid Activation Function ..... 10
          - 2.2.1.2.2 Softmax Activation Function ..... 11
          - 2.2.1.2.3 Tanh Activation Function ..... 11
          - 2.2.1.2.4 ReLU Activation Function ..... 12
        - 2.2.1.3 Padding ..... 12
        - 2.2.1.4 Pooling ..... 13
        - 2.2.1.5 Over-fitting ..... 13
        - 2.2.1.6 Dropout ..... 14
        - 2.2.1.7 Flattening ..... 14
      - 2.3 Recurrent Neural Networks ..... 15
      - 2.4 Long Short- Term Memory ..... 16
      - 2.5 Software and Hardware ..... 17
        - 2.5.1 Software ..... 17
        - 2.5.2 Hardware ..... 18

3. STATE OF ART .....	19
4. DATASET .....	21
4.1 Description .....	21
4.2 Construction.....	22
4.2.1 Real-World Videos .....	23
4.2.2 Artificial Videos .....	23
4.2.3 Dataset Architecture.....	24
5. MODELS FOR ACTION RECOGNITION.....	27
5.1 Approach.....	27
5.1.1 Data Preprocessing .....	27
5.2 Empty CNN Model.....	29
5.3 Transfer Learning Models .....	32
5.3.1 Transfer Learning Approach.....	32
5.3.2 VGG16 Transfer Learning Model.....	34
5.3.3 Feature Extraction Model .....	37
5.4 Model Output .....	39
6. EXPERIMENTS WITH OTHER DATASETS .....	41
6.1 Test Dataset .....	41
6.2 Empty CNN Model Experiments .....	42
6.3 Transfer Learning Model Experiments.....	44
6.4 Feature Extractor Model Experiments.....	46
6.5 Experiment with TensorFlow Object Detection API .....	48
7. CONCLUSIONS AND FUTURE WORKS.....	50

# 1. INTRODUCTION

Human action is a process, which is done by humans in a time period. Main actions are running, speaking, eating and many other similar activities. It should be in a time period and could be repeatable.

Action recognition is a method to recognize and classify the action, which is done by human in purpose. There are several methods to classify human actions: by sensors, pose detection, location and directly by raw video analysis. These methods can achieve high accuracy when using artificial intelligence and deep learning.

One of the most famous action recognition method is Human Pose Estimation [1], which is developed by TensorFlow. It is a method, which is based on deep learning. Pose estimation detects the human skeleton joints such as nose, eyes, elbows, shoulders, knees and other parts. After that, it draws a line in between these key joints. The position of these joints gives information about human actions. With this algorithm it is possible to detect actions in the certain time.

However, at this project, video classification approach with deep learning will be explained with codes, datasets and alternative solutions. There is a time needed to finish one single action, it means only checking one frame is not enough to understand the actions. Instead of detecting certain body parts, an image sequence will be analyzed with time series models to classify the actions.

Human actions are understandable and doable by other humans easily. But for artificial intelligence it is not as easy as for human brains. The most difficult part of this work is that the same action could be done differently by different people. Although the walking action looks simple, there are various types of walking styles. This makes the project more complicated. For this reason the dataset should be collected carefully, and it should include all possible styles of the same action.

## 1.1 Objective

Action recognition by deep learning works with number of frames of the video. The algorithm was trained with thousands of video samples. For training, several different deep learning structures were used and tested to get better performance results. These results gave a chance to see what the effect of deep learning's structures are on human action recognition. Each 10 frames of videos were collected, prepared and stored to the dataset. While data preparation, dataset was cleaned with different Python libraries. The first model is based on Convolutional Neural Network (CNN) and Long Short- Term Memory (LSTM). For second model, transfer learning approach were used. As third model, feature extractor method with transfer learning was applied to the dataset.

There are two main parts in this project. The first section is detecting only human in the video. At this part, data is cleaned and prepared for action recognition model. For image processing, one of the biggest issue is reducing background noise. Background noises always make a negative effect on the model accuracy. With this method, the background noises are cleaned and it helped to train the model.

Second part is action recognition part. The model takes the frames from the first model and predicts the actions. It is a classification model with CNN and LSTM layers.

The neural network model is trained with custom dataset, which has almost 78000 images. Dataset is mostly recorded by the owner of the project. Also they are collected from video games, some movies and series. While collecting data; different people, different places, angles and different clothes were considered in order to have a variation in the dataset. It will help to avoid of over fitting during training section.

In the following chapters, firstly we will look at Methods chapter, which contains information, methods that we used for our project. Then State of Art follows as third chapter. In state of art we will explain the relevant projects that we read before, and how they inspired us for actions recognition. As forth chapter, we will see the dataset that we used during training section. Models chapter follows as fifth chapter of the project. We will give information about our models, how we built the models and model performance results. Experiments chapter follows the Model chapter. We will use testing data to show the independent performance of these three models. Finally Conclusions and Future Works follows the flow as the last chapter.



## 2. METHODS

At this section the methods, which were used during the project are explained with general details. Main purpose of this article is showing how to build a machine learning (deep learning) algorithm to understand human action recognition by computer. So before explaining methods, machine learning and deep learning terms should be explained.

Machine learning is a study of computer science. It builds mathematical models based on sample data to make a prediction with new data [2]. It learns the pattern of the system and makes decision with minimum human needs. The model is trained with historic data under the mathematical formulas. When there is a new data or one of the historic data, it makes prediction to make a decision.

Deep learning is a sub branch of machine learning with inspiration from brain working system. It has several layers, which work with raw input data. One of the most famous application of data type could be image, text or sound. It learns the best path within all the layers. Most common deep learning method is neural network.

### 2.1 Artificial Neural Networks

Artificial neural network is a mathematical computing model. It works like a structure of neurons of human brains. There is a wide range of neural network types, usually they have layers and these layers are connected each other. The model is trained with large dataset and it learns the main pattern- path under the mathematical calculations.

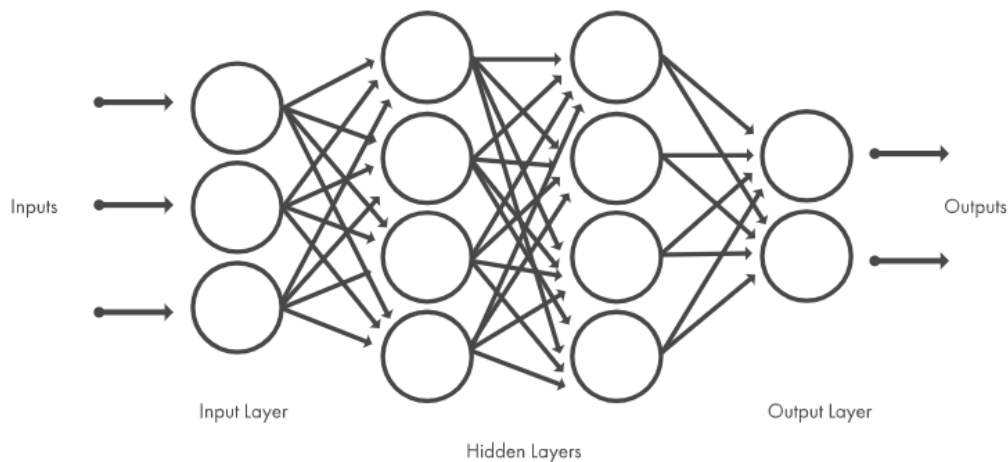


Figure 1. An example of a feed-forward neural network structure

General structure of a neural network could be seen above figure. There is an input data, which goes inside of the input layer. Then hidden layers follow input layer. Finally the outputs are defined after the output layer.

Each node has a connection with all nodes of the next layer. As it is shown at figure 1 that, there are several combinations inside of the network. With these combinations the model tries to find the best path in order to find the correct output, which is already defined (labeled classes) before.

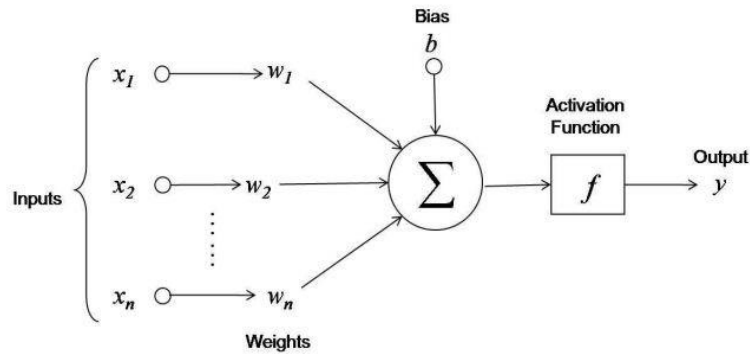


Figure 2. Neural network calculation

The mathematical explanation graph could be seen at figure 2. The  $x$  values are the features of the dataset (columns). Mainly there is a cost function and the network tries to minimize this cost as much as possible.

One of the most advantage of using neural network is, it does not require feature engineering as much as traditional machine learning technics. The neural networks are already enough complex to solve tough problems. On the other hand, the disadvantage is it requires a much bigger dataset than other machine learning models.

A basic (shallow) neural network has 2-3 hidden layers, while a deep neural network could have around 150 layers. The number of hidden layers depends on the complication of the dataset and the problem.

### 2.1.1 Training of Artificial Neural Networks

During the training of deep learning models, there is a guide, which shows how the algorithm works correct or not correct. This is the loss of each iteration. For each iteration the model calculates how the prediction value closes to the real value. If the differences between real and prediction value is high, then the model is not accurate and needs to be optimized. Basically, deep learning algorithms work with this principle.

There are three main sections in ANN model architecture. These are Forward Propagation, Loss Function and Backward Propagation.

### 2.1.1.1 Forward Propagation

Basically,  $x$  values are multiplied with weights and summation with bias value.  $f$  function means the activation function, which is applied to this sum. After each iteration it gives the  $z$  value.  $z$  goes into the activation function in order to define the output of the input data. Finally the prediction is done and the result is found for one iteration.

$$y_{pred} = f\left(b + \underbrace{\sum_{i=1}^n x_i w_i}_z\right)$$

The formula of forward propagation could be seen above.  $w$  means the weight of the each unit and  $b$  means bias value. Total amount of input unit is  $n$  and  $i$  is the current unit while calculation. Each unit can be seen at figure 2 as  $x_1, x_2 \dots x_n$ . This formula gives the result of one iteration in the model. Next is the section of backward propagation.

### 2.1.1.2 Backward Propagation

Backward propagation is one of the most important part of deep learning models. Basic idea is decreasing the loss of the model and increasing the performance. For this purpose, weights and bias values are optimized with gradient descent method.

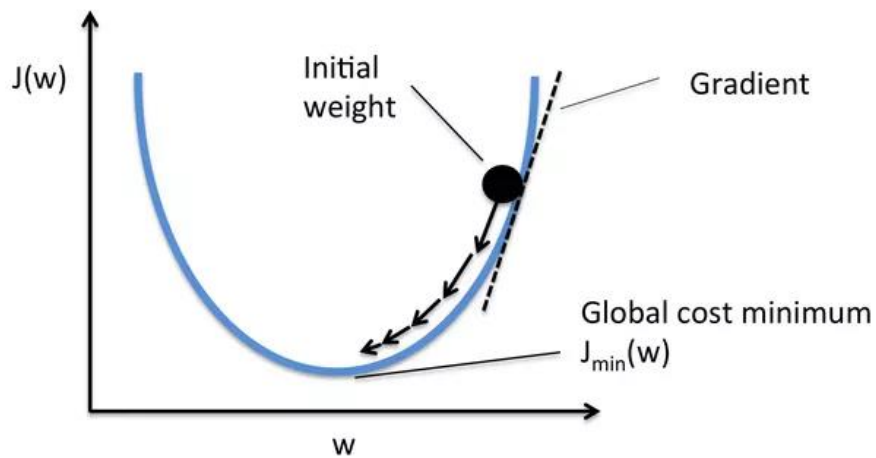


Figure 3. Idea behind Gradient Descent

$J(w)$  on the graph represents loss for each iteration according to weights  $w$ . Initial weight tries to go the bottom of the graph, it means the minimum loss value. For each step new weights are optimized and are used for the next step.

All the layers are connected each other with weight values. During calculation of the output layer, all these input and hidden layers are used. During backward propagation, chain rule [3] is used to derivate of each weights. Formula of the chain rule can be seen below [4].

$$\frac{\partial J_{total}}{\partial w_n} = \frac{\partial J_{total}}{\partial out} \cdot \frac{\partial out}{\partial net} \cdot \frac{\partial net}{\partial w_n}$$

In the formula,  $J_{total}$  means total cost,  $w_n$  is the current weight,  $out$  is the output value of the layer and  $net$  is the input of the current layer.

The below formula explains how to update weights of each unit.  $n$  means number of iteration at that time and  $\mu$  is the learning rate. Learning rate manages the optimization steps. If learning rate is chosen too low, optimization can take longer than expected. If it is too high, it can never be optimized.

$$w_{(n+1)} = w_{(n)} - \mu \frac{\partial J_{total}}{\partial w_{(n)}} J(w)$$

This loop goes until the Global cost minimum point  $J_{min}(w)$ . Finally the model is optimized with minimum loss value.

### 2.1.1.3 Loss Function

Loss function is one of the most important tool in deep learning. It is used during the optimization of weight and bias values. For each iteration there is a value, which is predicted by deep learning model. Prediction and the real values are checked with loss function in order to improve the model performance. If the loss value is high, it means the model parameters are not correct and they should be optimized.

There are various loss functions in deep learning and machine learning applications. The performance of these functions depends on the type of the problem; it could be a binary classification, multi class classification, regression or others. They are classified into two categories as classification and regression losses.

Some popular loss functions are Mean Square Error, Mean Absolute Error, Hinge Loss and Cross Entropy. They are briefly introduced next.

### 2.1.1.3.1 Mean Square Error

Mean square error (MSE) is average of the square differences between predicted and real values. It is a regression loss function and mostly used with logistic regression models.

$$MSE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}$$

At the formula  $i$  means the index of the current data,  $n$  is the total number of examples,  $y$  is the true value and  $\hat{y}$  is the predicted value.

### 2.1.1.3.2 Mean Absolute Error

Mean absolute error (MAE) is average of the absolute differences between predicted and real values. MAE is more robust against outliers, because it does not square the values.

$$MAE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n}$$

The terms of the formula are same as MSE.

### 2.1.1.3.3 Cross Entropy Loss

Cross Entropy is a probabilistic measurement between 0 and 1. It is also known as logistic loss. It is so common in classification problems.

$$Cross\ Entropy\ Loss = -(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

The terms of the formula are same as MSE.

## 2.2 Convolutional Neural Networks

Convolutional neural networks (CNN) were invented at 1980s. The main purpose of this model is image processing, given that it is inspired from the visual cortex of mammals. CNN is one of the most important tools in deep learning world. Image classification, image recognition, object detection and many similar works are using CNN algorithms to understand and get information from the images.

CNN can be used not only for images, but also for other data types: audio, temporal series, etc. In what follows, CNN will be explained as when applied to images, since it is their natural application.

CNN model checks all the images pixel by pixel. The work principle of CNN is similar to ANN, it takes the input image, then calculates the weights and bias values under the feature importance. After sums all the weights with features, it calculates loss values with each iteration and uses an activation function to give a logic result. The important values could be edges, sharp color changes and different object in the image.

While training, CNN model uses feature extraction in between each layer. The aim of feature extraction is reducing complexity of the image and taking distinctive information [5] such as corners, sharp shapes, objects, lines etc. Each layer extracts features from the previous one during the training path. Feature extraction is used in almost all computer vision algorithms.

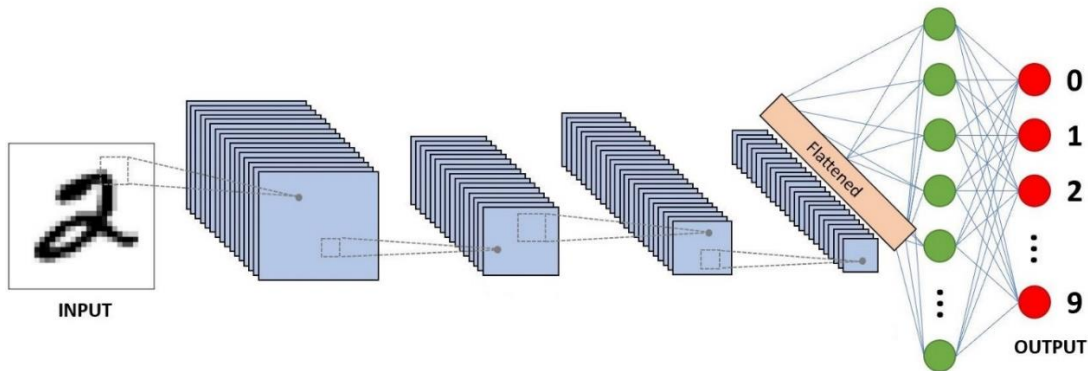


Figure 4. A CNN model to classify handwritten digits

As it is seen at figure 4, an input image goes into the CNN model. It takes one pixel to the convolutional layer and transfer the pixel information to the next layer. The pixel information is a number, which could be between 0 and 255 as RGB or RGBA. And finally the output is a digit number between 0 and 9.

## 2.2.1 Structure of Convolutional Neural Networks

CNN algorithm has several components in a deep learning model. These components can be different, which depends on the work purpose. These components help to simplify the images and understand the pixel information. Also they help to optimize weights and bias values. A simple classification model has these components:

- Convolutional Kernel
- Activation Function
- Padding
- Pooling
- Dropout
- Flattening

### 2.2.1.1 Convolutional Kernel

Convolutional kernel is a matrix, which is working as a filter for CNN layers. The main task of convolutional kernel is resembling the original image. With this method, the image size could be reduced and simplified to use by algorithm.

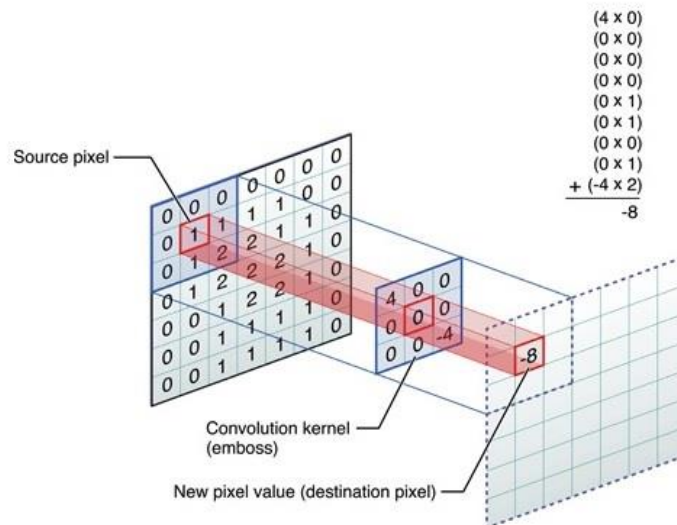


Figure 5. Calculation of convolutional kernel

Filter can be 3x3, 5x5 or different sizes, which is defined by the user. The filter shifts over the all image pixels, it goes from left to right then from top to bottom. At this example from figure 5 [6], the kernel size is 3x3 and it shifts 25 times on the image. Each time, there is a matrix multiplication between source pixel and convolutional kernel. Finally the output goes to 5x5 matrix with these results.

If the source image has multiple channels like RGB then the kernel has the same depth as as 3x3x3, 5x5x3 etc. For RGB example, the kernel has 3 channels and it shifts 3 times on the image channels.

### 2.2.1.2 Activation Function

Activation function gives the output of the node. It converts all the values to a logic output like “yes” or “no”. They are used after just convolutional layers. Activation functions could be categorized as linear and non-linear. However, non-linearity is the form that is wanted to have in the model. Mostly non-linear activation functions are used for deep learning. There are several types of activation functions, and the most used will be discussed next:

Sigmoid, Softmax, Tanh, Arctan, ReLU, PReLU etc.

#### 2.2.1.2.1 Sigmoid Activation Function

Sigmoid is a non-linear activation function. It gives the outputs between 0 and 1. It is quite popular with probability prediction models. The most famous partner of sigmoid is logistic regression model.

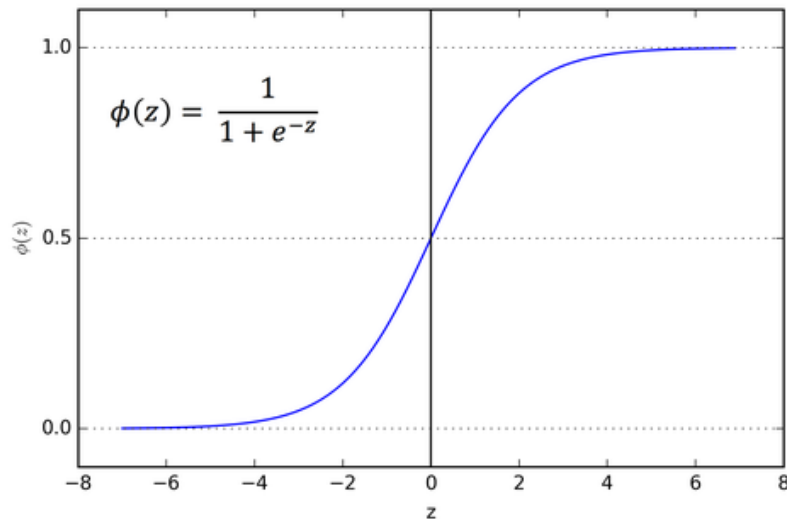


Figure 6. Sigmoid Function

As it is seen from figure 6, the output is always between 0 and 1. For logistic regression, there is a filter to classify the outputs. If the output of the sigmoid is lower than 0.5 than it classifies as 0, if higher than 0.5 than it classifies as 1. Sigmoid is mostly used for binary classification methods. It is not usually employed to hidden layers because of vanishing gradient problem.

In the formula,  $z$  means sum of all multiplications of weights and  $x$  and bias values (see section 2.1.1.1).



### 2.2.1.2.2 Softmax Activation Function

For multi class classification and binary classification problems, softmax is a useful activation function. It gives the probability of each class like sigmoid function. When one class's probability increases, the rest decreases.

The formula could be written as:

$$\phi(z) = \frac{e^{z_i}}{\sum_{j=0}^k e^{z_j}}$$

In the formula,  $z$  means sum of all multiplications of weights and  $x$  and bias values,  $k$  is the total number of examples,  $j$  is the current index of the normalization and  $i$  is the current index of the example.

### 2.2.1.2.3 Tanh Activation Function

Tanh activation function is similar to sigmoid function. Since sigmoid only gives positive results, tanh tolerates negative outputs as well. It works between -1 and 1.

The formula can be written as:

$$\phi(z) = \frac{1 - e^{-2z}}{1 + e^{-2z}}$$

In the formula,  $z$  means sum of all multiplications of weights and  $x$  and bias values.

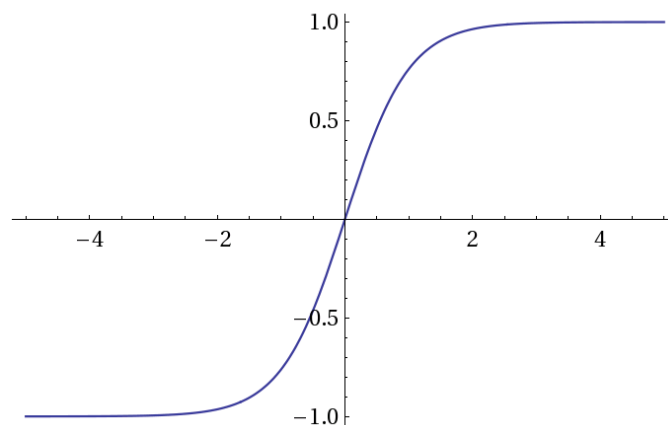


Figure 7. Tanh activation function [7]

### 2.2.1.2.4 ReLU Activation Function

ReLU can be used for all neural networks, but only with hidden layers. When the input is positive, ReLU gives the same output. When the input is zero or less then the output is always zero. While it is linear with positive values, it is zero with negative values. ReLU is the most common activation function in deep learning.

The formula is:

$$\phi(z) = \max(z, 0)$$

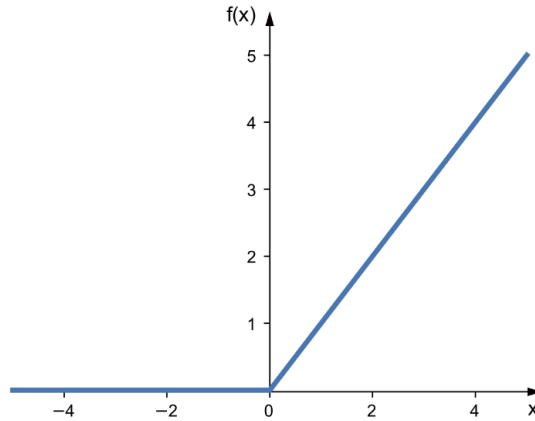


Figure 8. ReLU activation function

ReLU is the simplest activation function in usage, for this reason it is the most popular one.

### 2.2.1.3 Padding

When using convolutional kernels, the image size is decreased and simplified. However sometimes it could affect to lose information at the corners or edges. To avoid this problem, padding is the correct choose. Padding covers the border of the image with zero values and extends the size of the pixels. When the filter shifts the image, it does not lose the information at the borders.

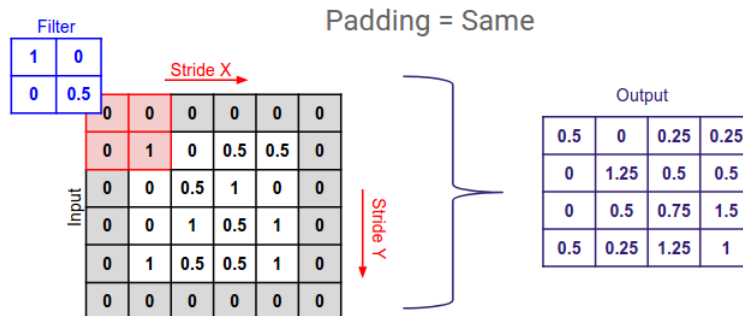


Figure 9. Same padding

As it is described at figure 9, same padding avoids lose information at the borders. For training, it is important to keep all information as much as possible.

#### 2.2.1.4 Pooling

Pooling method down samples the image kernel. It helps to decrease computational cost by reducing the size of the image. Also it helps to avoid over fitting by providing abstracted form of the kernel. There are several type of pooling as max pooling, average pooling etc.

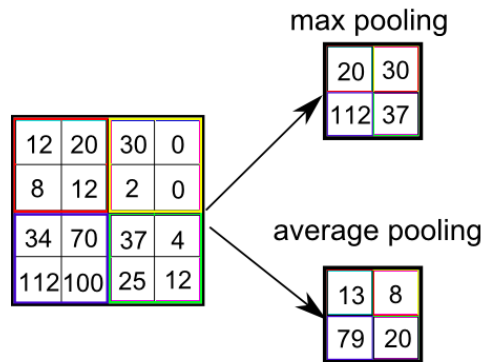


Figure 10. Pooling method

Figure 10 shows how to down sample image by max pooling and average pooling. Pooling size is 2x2 and it runs over all image. Then it checks the each selected 2x2 matrix. With max pooling, filter takes the maximum value and places to new output of 2x2 matrix. Average pooling takes average of each 2x2 matrix and places to the output matrix.

#### 2.2.1.5 Over-fitting

Over-fitting happens when the model learns the training data too well [8] and cannot predict new data. The model can stack in some specific neurons and learns noise in the learning way and cannot learn more important path. Neural networks are often over-fitted because of the similarities inside of the dataset, choosing wrong hyper parameters, wrong model structures and other reasons. This case damages the performance of the model. It can be detected with validation and testing data. When training loss is low and validation-testing loss is higher than expected, the model is probably over-fitted.

### 2.2.1.6 Dropout

Dropout is a tool that helps to prevent over-fitting during the training of the model (a.k.a. regularization method). Dropout ignores random neurons; it means it does not take into account these neurons during the calculation of forward and backward propagation [9].

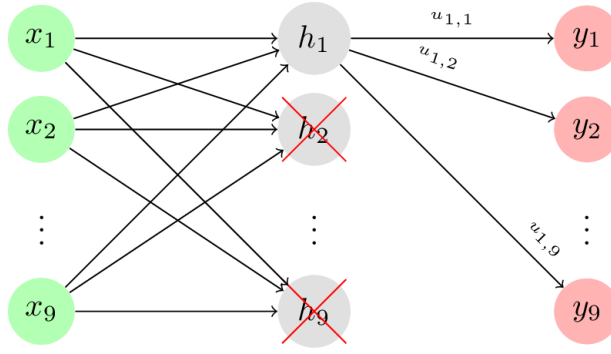


Figure 11. Dropout removes some neurons

### 2.2.1.7 Flattening

Flattening is the final section of CNN model. With flattening the input shape is converted into 1-dimensional array form. As it is seen at figure 12, all pixels are long 1-d vector. After flattening the model is ready for the next layer.

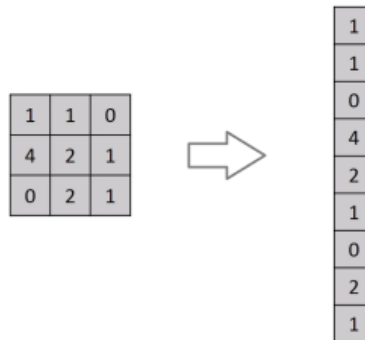


Figure 12. Flattening

## 2.3 Recurrent Neural Networks

Recurrent Neural Network (RNN) is a special deep learning method to work with time series. Basically it has own memory to remember previous case in dataset. It checks the previous data, keeps the information in the memory and use that memory to make a prediction. Mostly it is used in natural language processing (NLP), speech recognition, time series problems and video classification.

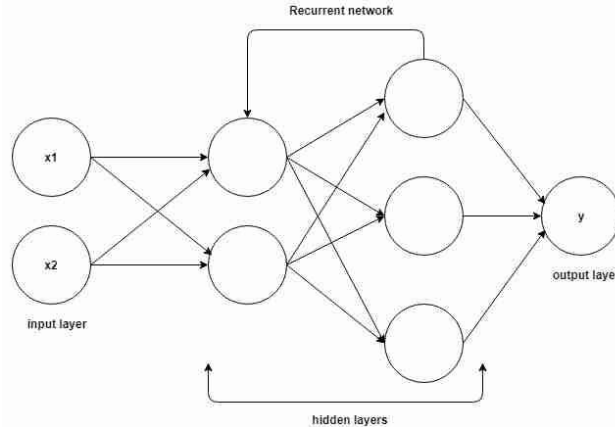


Figure 13. RNN structure

As it was described before, in the working principle of neural networks there are forward and backward propagations. However, the input of the calculation was only about current data. Here in RNN the loop is unfolded and the units are duplicated for a certain time steps, then the output of the current data stays in the memory and goes to the next data's calculation as an input. At figure 13, prediction goes to the output layer and also they are kept in the memory to be used again in hidden layers.

Loss calculation for backward propagation, instead of one data RNN model checks all historic time-series predictions. It means there are  $n$  times derivation for the loss function. Then it optimizes all the time series weights.

$$h_{(t)} = \phi(w_{x_{(t)}} + Uh_{(t-1)})$$

As above formula explains,  $h_{(t)}$  means the output of the hidden layers at  $t$  time,  $U$  is a special weight for previous values. The previous  $h_{(t-1)}$  is multiplied with  $U$  values. Finally  $h_{(t)}$  goes to activation function.

## 2.4 Long Short- Term Memory

Long short- term memory (LSTM) is an updated version of RNN. With LSTM it is easy to remember important past data. It was designed, because RNN is not enough for time series models in deep learning. With RNN there is a gradient vanishing problem, which happens with long sequences data. LSTM trains the dataset with back-propagation like RNN model. The main differences is the structure. While in RNN there is one section, LSTM has three different sections; Input Gate, Forget Gate and Output Gate.

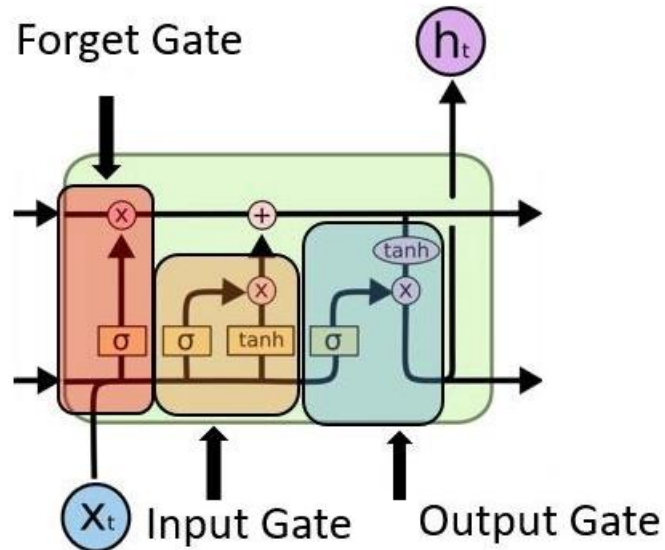


Figure 14. Long short term memory structure [10]

In Input Gate, sigmoid and tanh functions work together to decide which input should modify the memory. Sigmoid gives the result as 0 or 1 and tanh makes an effect on the performance with values between -1 and 1 and finally they are multiplied. The formula can be seen below.

$$i_t = \sigma(w_i[h_{(t-1)}, x_t] + b_i)$$
$$\hat{C}_t = \tanh(w_c[h_{(t-1)}, x_t] + b_c)$$

The function of the Forget Gate is removing useless or non-necessary details from the model. It is done by sigmoid function. Sigmoid looks at the previous hidden layer values  $h_{(t-1)}$  and current input  $x_t$  and decides to keep in the memory or not.  $b_i$ , and  $b_c$  are bias values,  $w_i$  and  $w_c$  are weights.

$$f_t = \sigma(w_f[h_{(t-1)}, x_t] + b_f)$$

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t$$

Final section is Output Gate. Here sigmoid decides which part is going to be used as output. Then tanh gives probabilistic result between -1 and 1. Then finally this both results are multiplied and go out of Output Gate as the main output. The formula variables are similar to input gate formula.

$$o_t = \sigma(w_o[h_{(t-1)}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

## 2.5 Software and Hardware

During action recognition project, the author used some specific software and hardware to train the model, show the results and improve the performance.

### 2.5.1 Software

All the process during collecting dataset, data preparation, training the neural network models are done with Python. Python is an open sourced programming language. It was created by Guido van Rossum at 1991[11]. Currently Python is one of the most famous programming language in the world.

OpenCV and Numpy libraries were used for collecting data and data processing parts. OpenCV is an open sourced library for image processing applications. OpenCV was used for changing image formats, reading, resizing and reshaping the images.

During data preparation, Numpy is the correct library to make the mathematical calculations. It was used by author to create multi-dimensional arrays and matrices. For image processing Numpy is an essential library.

For Training and evaluating of the model, TensorFlow library is the main tool during the project. TensorFlow is an open sourced library, which was created by Google Brain Team at 2015. TensorFlow is a numerical computation tool meant for automatic differentiation. It was invented specifically for deep learning projects.

During training, Keras and LSTM of TensorFlow tools are used to create the neural network model. Also TensorFlow Object Detection API helps to detect only person in the image. As the preprocessing of training and testing, these tools are used by the author. This method will be explained in the following sections.

### **2.5.2 Hardware**

Deep learning has intense computational requirements during the calculation of the neural network models. There are thousands of calculations inside of a simple model. GPU power makes these heavy calculations faster than CPU. Google Colab GPUs are used during training section. Google Colab has NVIDIA Tesla K80 GPU and 24 GBs of Ram.



### 3. STATE OF ART

Recent years there are significant demands on human action recognition with deep learning by researchers. They try to build new models with different neural network structures, train their models with large dataset and use different approaches for this purpose. However video classification or action recognition is still not enough accurate and needs to be researched more. For this reason there are several methods on this research area.

In the research of Jeff Donahue and team members, Long-term Recurrent Convolutional Networks for Visual Recognition and Description [12] was published on 17 November 2014. Basically researchers used CNN and LSTM blocks to memorize frames of the action. There is an action, which is spread into n-times images. Within these images there is a relationship to recognize this action. The researchers focus on this sequence and try to find the pattern relation to explain the action. With LSTM, they proposed they can capture temporal information from the video. As the structure of the model, they used CNN to check the image information, after CNN they connected image information to LSTM blocks. It means for each image they can keep the sequence information with LSTM.

As article explains they took 16 frames as one action and they used these frames for the input of the deep learning model. On the project, the color format of the images is RGB model. They calculate each image frames and take the average of the total sequence. While training and validation they used UCF101 dataset [13], which consists 13320 videos and it is categorized to 101 actions. Dataset is collected from YouTube for action recognition by Khurram Soomro, Amir Roshan Zamir and Mubarak Shah. It can be seen at figure 15.

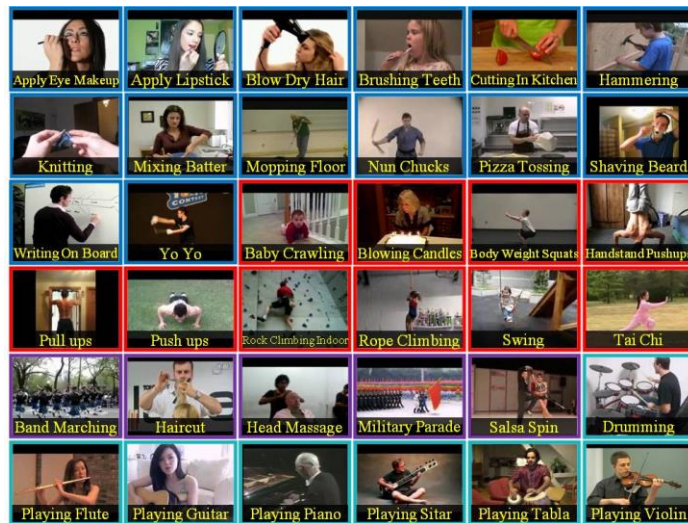


Figure 15. Sample of the dataset

Below figure explains the structure of the training and prediction model. Input images go into CNN modules. Then LSTM helps to keep information in the memory. Finally output classifies the actions. The figure on the left, shows how the model learns image sequences. There are  $T$  times images in the sequence and the model gives  $T$  times output. The right figure shows prediction of the model. Each image gives an output as  $y$  value, then Average block takes the average of the results as the output. For this example the output is predicted as HighJump.

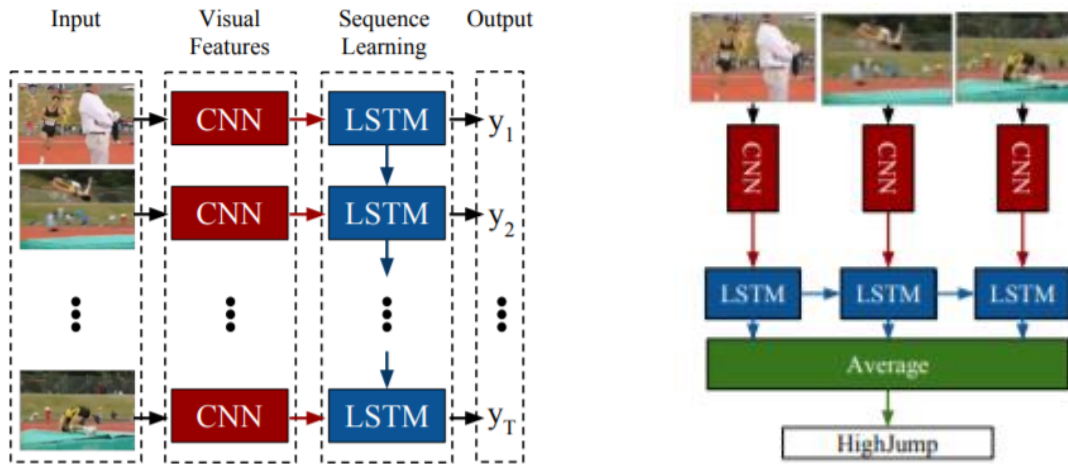


Figure 16. Training and prediction structure

Second project on this research is Learning Spatiotemporal Features with 3D Convolutional Networks [14] by Du Tran and their research fellows. The paper was published in 2 December 2014. UCF101 dataset was used while training and testing the deep learning model.

They used 3D convolutional neural network (3D ConvNet) for the project. For one action they took 16 frames, which means almost 2 seconds. All images were resized to 112x112 with RGB color model. Basic one action size is 16x112x112x3, which is the input of the deep learning model. The network architecture has learning rate as 0.003, mini-batch as 30, and it is stopped after 16 epochs.



Figure 17. Basic structure of the model [14]

We got ideas from these two papers to solve action recognition with deep learning models. The effective way to use neural network is connecting CNN layers with LSTM layers.

## 4. DATASET

This chapter provides all information concerning the dataset, created specifically for this project: how it was collected, which kinds of video and video categories were used, what kind of data processing were applied and also some examples will be seen.

### 4.1 Description

In the project there are several types of videos, which were collected and were used for training the models. The author trained multiple TensorFlow models to show the importance of artificial neural networks.

Dataset contains a total of 78000 images. These images were converted from multiple different video clips. The videos were recorded by the author specifically for action recognition. Also they were collected from some video games to make a variation at dataset in order to avoid over fitting. These 78000 images include some artificial images as well, which were created with data augmentation technics.



Figure 18. Overview of the dataset

The dataset has 4 different categories; shooting, fighting, running and sitting. A sample could be seen at figure 18. These actions are interesting in daily life, for this reason they were chosen for the project. Especially violence, shooting and running detection is important to help security and safety. One of the important goals of the author is using artificial intelligence technics to improve the life quality and help the community.

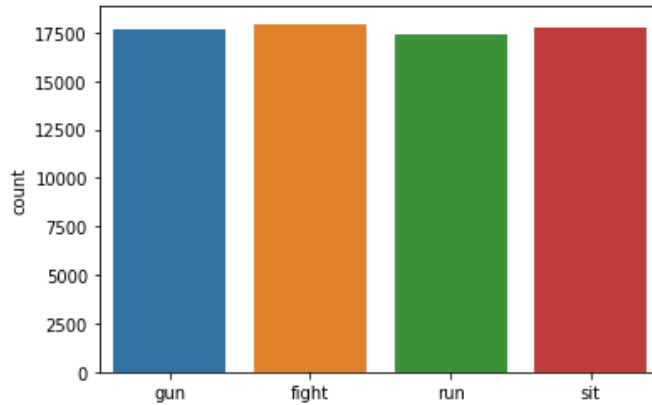


Figure 19. Count plot of training dataset

As it can be seen from above figure, the dataset is almost balanced and contains 17700 shooting, 17950 fighting, 17420 running and 17730 sitting action images.

Almost all dataset was recorded by the author. Public datasets are not used for the project. Because the technic is a way different than other projects. Also it could be seen that the dataset only has human body, like inside of a bounding box. Because the background, outside of the human body was removed with rectangle box, to make it ready to use for training section. The reason of background elimination is to train a neural network considering only the changes on the body shape at the moment. Background noises always affect negative to the performance of the model.

## 4.2 Construction

There are two different data types in training dataset. First one is real-world videos. At real-world videos, the author recorded himself to collect data. Second type is called artificial videos. These video images were collected from some video games in order to increase the data size and also give a variation to the dataset.

### 4.2.1 Real-World Videos

While collecting data, the author recorded himself and other people to focus only human body in the center of the video. The human does actions during the video. While doing these actions, they should do these actions with different sequences, different lengths and with different body shapes in aim to avoid over-fitting of the model. If all the actions are the same and collected by only one person, the neural network model could be over-fitted. It can only learn trained dataset, then it cannot predict the actions by other people. Also the person at the video wears different clothes. Some of the videos were collected at different time of the day with considering light effect.

The data augmentation was applied to all dataset to avoid over-fitting and increase the size of the dataset. Data augmentation is an artificial method to increase the data size. Mainly the image shapes, image quality, zooming in and out, brightness or colors are modified during augmentation process. Finally there are new images, which are created and added to the dataset. This method helps training process.

ImageDataGenerator module of Keras library was used for data augmentation method. The brightness of images were modified in the range of 1.5 and 1.7. The author modified images with zooming in and zooming out as well. Finally the dataset is doubled and ready to use for training section.

The other point for real-world images is video recording angle. In real world, videos could be recorded by different angles. After a training images with one angle, the author realized that there was a need to collect more data with different angles. For this reason all action videos were recorded with different angles. Some examples could be seen at Figure 18.

### 4.2.2 Artificial Videos

As it was mentioned before, dataset includes video- game images. The idea is similar; in the game there is a person, who is doing these actions with the same sequences. While fighting, running and sitting, the images were recorded and saved to dataset. The video- game is called Knight Online. There is a high definition character, which has a human appearance. The character's actions are similar to human actions, for this reason the author decided to collect data from this game.



Figure 20. Overview of the video-game images

Dataset overview could be seen above figure. While character is doing actions, the frames are being saved to dataset.

### 4.2.3 Dataset Architecture

To record real world and video games images, some Python libraries were used. These libraries are grabscreen, ImageGrab of PIL, Numpy and OpenCV. After recording all videos and images, there are some preprocessing sections. Only human body is needed to train the model. For this reason all background is removed from images. While cropping background, Numpy and OpenCV libraries were used.

Finally dataset includes only human body and the actions. The images were needed to convert to Gray Scale format for empty CNN model. For transfer learning they are kept in RGB format.

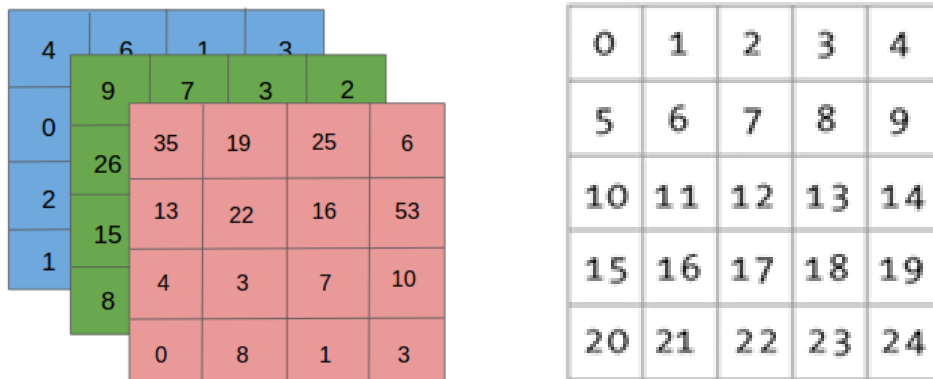


Figure 21. RGB and Gran Scale matrix

The both RGB and Gray Scale formats could be seen at figure 21. The images were converted from 3d matrix to 2d matrix format. From 3d to 2d, the images lose their colors and they only have gray colors.

After all, the image shape is modified from  $(w, h, 3)$  to  $(w, h, 1)$ . Third number shows the channel of image.

Convolutional neural networks only understand input as images. It is not possible to use videos directly as an input. For this reason the videos should be converted to image format and then kept as image sequences.

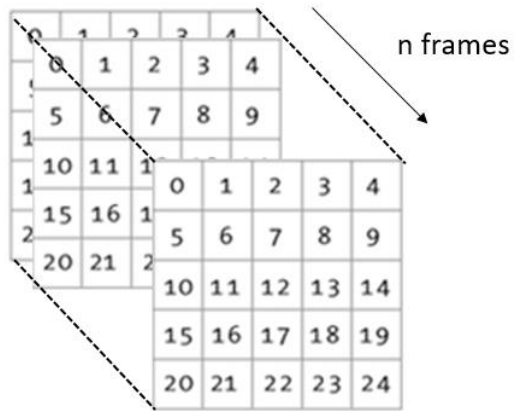


Figure 26. 3D image sequence

All dataset is reshaped from shape of  $(n\_images, w, h, 1)$  to  $(n\_sequences, 10, w, h, 1)$ .

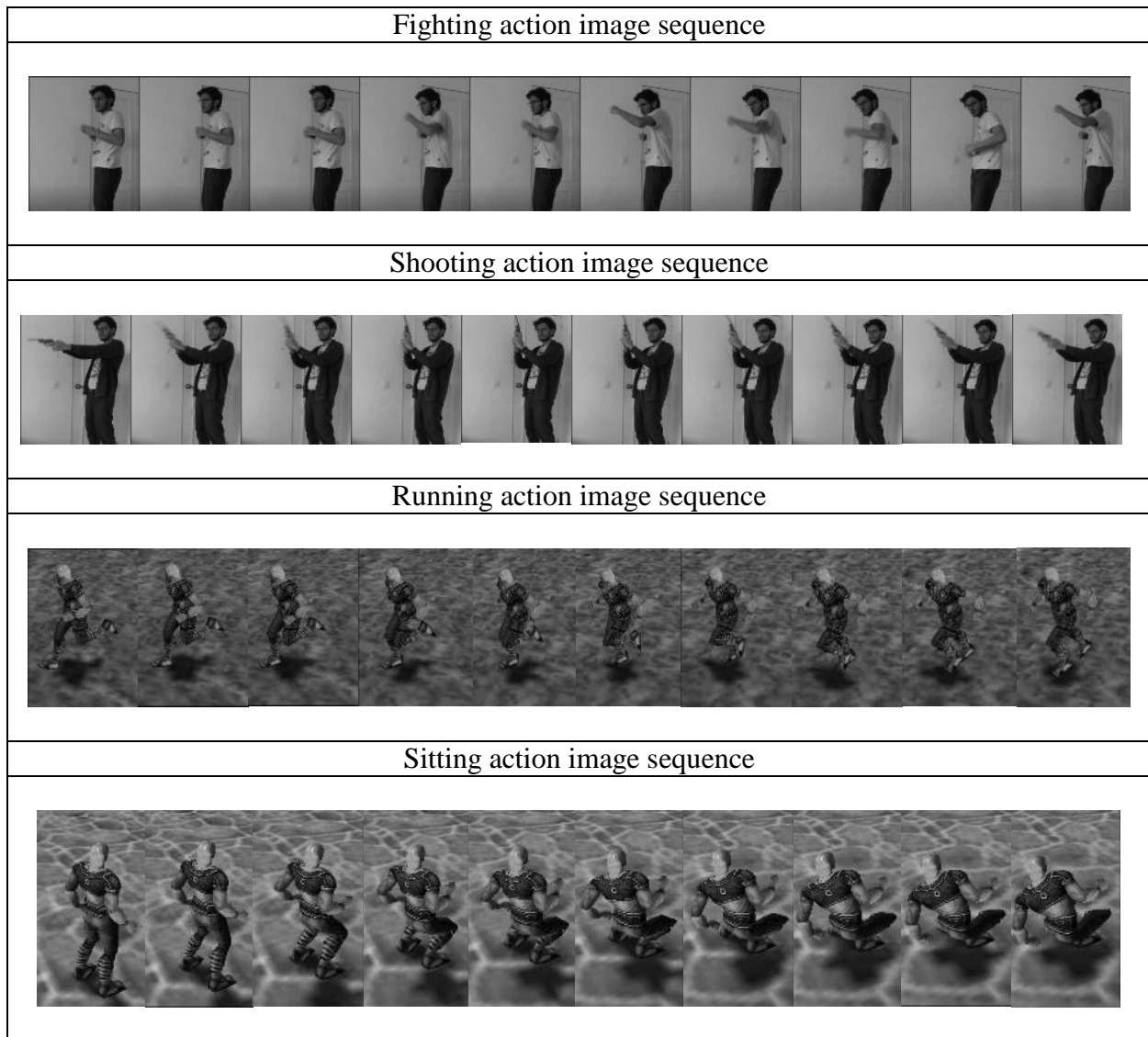


Figure 27. Dataset categories with sequences

Above figure shows each actions from dataset. The images collected from video, then reshaped as 10 frames. After several try and combinations, author decided to take each 10 frames as one action. It can be seen from above figures that, it is possible to recognize an action in 10 images.

During the training, these sequences will be used as the target of the model.



## 5. MODELS FOR ACTION RECOGNITION

This section contains information about neural network models, which are built by the author of the project. The author gives details to explain the approaches and the technology behind of the algorithms.

### 5.1 Approach

The main goal of this project is building and comparing deep learning models to recognize human actions in the video. Different structure of deep neural network model has different effect on each dataset. While sometimes small changes can increase the performance of the model, sometimes it does not effect on the model. For this reason the author decided to train multiple neural network models to get the best performance. While doing this, also it is important to see what the differences is, what the effect of the model is and how to choose the correct model to have better results.

There are three models, which were trained for action recognition project.

- First model is trained from Empty CNN model with Keras library of TensorFlow.
- Second model is trained with Transfer Learning method.
- Third model is based on transfer learning from a pre-trained model using feature extractor.

For transfer learning, the author used Keras library of TensorFlow.

Three models have different requirements while data pre-processing, training and testing sections. Also it will be seen that, the results, training time and model weights are different in these three models.

#### 5.1.1 Data Preprocessing

Data preprocessing section is almost the same for all models. The dataset, which is recorded and collected by the author was used for training. Transfer learning pre-trained models only support to use RGB images for training. For this reason the images are not converted to GrayScale during data processing. For training Empty CNN model, all images are converted to GrayScale as it was explained at section 4.

```
[ (184, 144), (175, 132), (184, 144), (175, 132), (184, 144), (175, 132),  
  (184, 144), (175, 132), (152, 144), (143, 132), (232, 144), (229, 132),  
  (208, 144), (206, 132), (216, 144), (207, 132), (184, 144) (175, 132),  
  (184, 144), (175, 132) (264, 144), (261, 132), (264, 144), (261, 132),  
  (264, 272), (261, 264), (264, 272), (261, 264) ]
```

Figure 28. Dataset image sizes

Figure 28 shows all image sizes in the dataset. It can be seen that the images are not fixed sizes and not the same quality. For this reason the author resized all the images to a fix size as (200,145,3) for transfer learning and (200,145,1) for Empty CNN model. For deep learning, neural network input images should be in the fix sizes.

It is chosen (200,145), because the author avoids lose pixel information of the images. If the images are resized to lower values then it is not easy to learn by neural network models. (200,145) is the optimum values of all dataset.

There are 4 categories as explained in dataset section. These categories are converted to binary format with One-Hot Encoding method. One-hot encoder converts string categories to 0 and 1 values. So they are meaningful for neural network.

```
Fighting ---> array([0., 0., 0., 1.])
Shooting ---> array([0., 0., 1., 0.])
Runnig ---> array([0., 1., 0., 0.])
Sitting ---> array([1., 0., 0., 0.])
```

Figure 29. Classes after One-Hot Encoding

After one-hot encoding method, binary format can be seen at figure 29 for all classes. Number of classes also define last layer of neural network models. So it means the last layer of both models should be 4.

Before model training part, there is a last process to prepare dataset. Total dataset has almost 78000 images for training. It should be split to training and validation data. While training, at the end of the each epoch there is a model validation to see how the performance of the model. The user can stop training, when the loss is at the minimum stage and model accuracy is maximum stage.

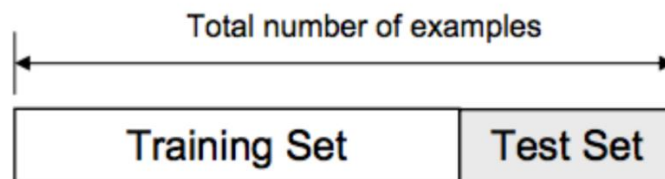


Figure 30. Train-Test Set separation

For this purpose, `train_test_split` module of SkLearn library helps to separate dataset. The dataset was split as 85% of training and 15% of validation set. Before train-test splitting, the data is already converted to sequences, as it was mentioned at Dataset section. So mixing data is not important, because all dataset is sequence format now.

## 5.2 Empty CNN Model

The Keras model was trained from scratch without pre-trained models. The model was built and all the hyper parameters were chosen by the author. The model was built with Sequential of Keras library. Firstly, CNN layer added to the sequential model with TimeDistributed, as activation function tanh and ReLU are applied to CNN layer. MaxPooling and Dropout follows the model. There are several CNN model were added to the model, then Flatten feature comes after all CNN layers. After that, LSTM layer was added and Dense layers follow to the LSTM layers. To give the output, last layer of the model is added as Dense layer with 4 size with softmax activation function.

Convolutional layers only work with image inputs and they are not compatible with sequence inputs. Image sequence has an extra dimension as 5-d shape. In order to use image sequence, TimeDistributed of Keras library should be applied. TimeDistributed works as a bridge between each image in the sequence. It passes information from current image to the next image. It applies the same layer to several inputs and produces one output per each input. It is good to be noted that, TimeDistributed is applied only for convolutional layers. It is not used for LSTM or Dense layers.

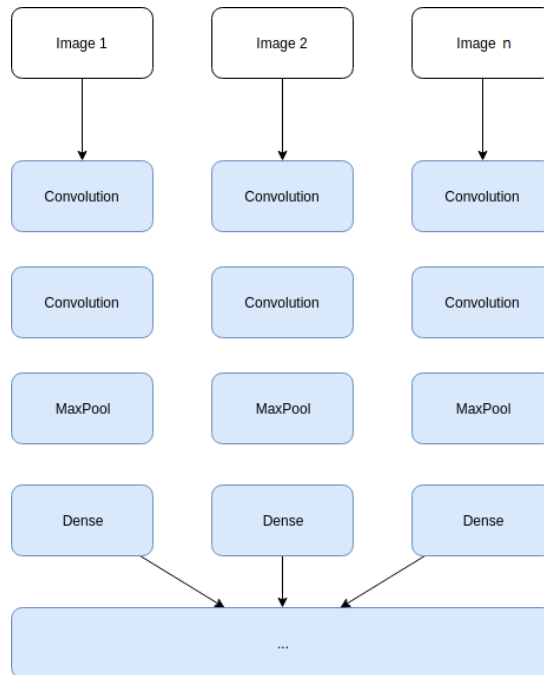


Figure 31. TimeDistributed model for image sequence [15]

Image sequence structure can be seen above. There are 10 images in each sequence and with TimeDistributed, the layers are applied to each image.

Layer (type)	Output Shape	Param #
time_distributed_12 (TimeDis	(None, 10, 145, 200, 96)	960
time_distributed_13 (TimeDis	(None, 10, 145, 200, 96)	0
time_distributed_14 (TimeDis	(None, 10, 143, 198, 128)	110720
time_distributed_15 (TimeDis	(None, 10, 143, 198, 128)	0
time_distributed_16 (TimeDis	(None, 10, 71, 99, 128)	0
time_distributed_17 (TimeDis	(None, 10, 71, 99, 128)	0
time_distributed_18 (TimeDis	(None, 10, 69, 97, 96)	110688
time_distributed_19 (TimeDis	(None, 10, 69, 97, 96)	0
time_distributed_20 (TimeDis	(None, 10, 34, 48, 96)	0
time_distributed_21 (TimeDis	(None, 10, 34, 48, 96)	0
time_distributed_22 (TimeDis	(None, 10, 156672)	0
lstm_2 (LSTM)	(None, 10, 100)	62709200
dense_4 (Dense)	(None, 10, 512)	51712
dropout_8 (Dropout)	(None, 10, 512)	0
dense_5 (Dense)	(None, 10, 256)	131328
dropout_9 (Dropout)	(None, 10, 256)	0
dense_6 (Dense)	(None, 10, 4)	1028
Total params: 63,115,636		
Trainable params: 63,115,636		
Non-trainable params: 0		

Figure 32. Empty CNN Model structure

The model summary is shown above figure. Model has 63,115,636 total parameters. As optimizer Stochastic Gradient Descent (SGD) with 0.001 *learning rate*, *momentum* as 0.9 and *nesterov* as True boolean is applied to the model. Mean squared error (MSE) function is chosen as the loss function. Action recognition is a continuous scoring problem as it is applied TimeDistributed function. TimeDistributed gives the result as number of images inside of the sequence. The output of the model is a probabilistic result of each classes, for this reason MSE is the good choice for action recognition model [16]. The batch size and number of epochs are chosen as 5 and 40.

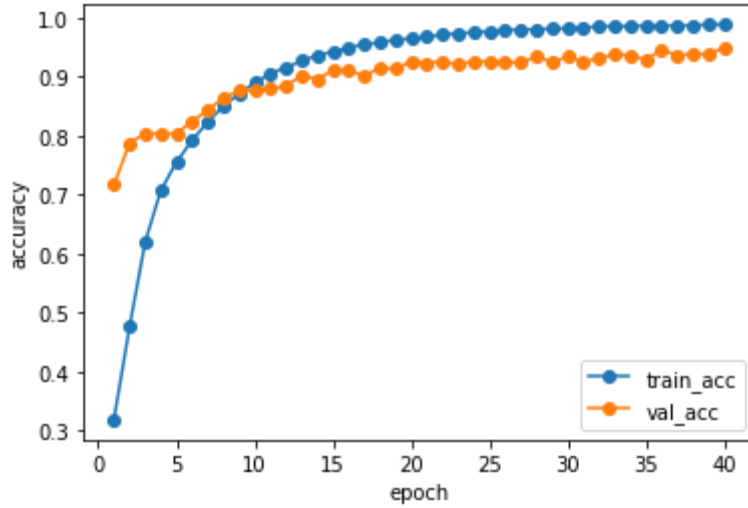


Figure 33. Accuracy graph for both train and validation set

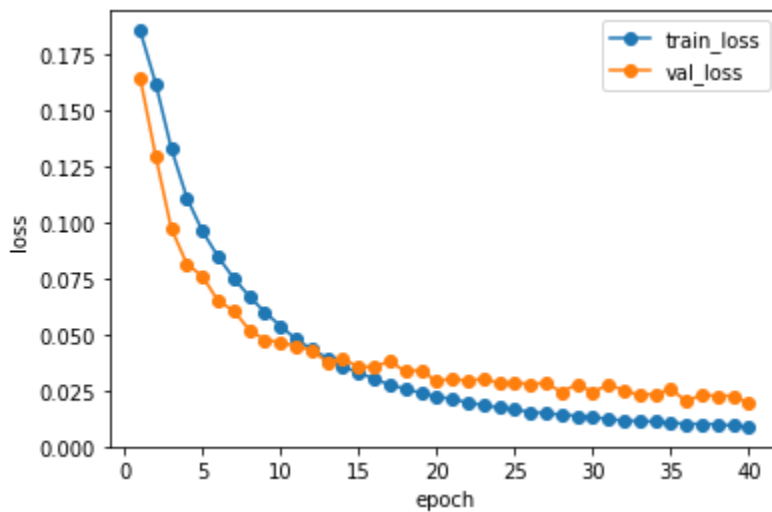


Figure 34. Loss graph for both train and validation set

Figure 33 and figure 34 give information about training results. The models reached 98% of training and 95% of testing accuracy after 40 epochs. Also, loss values drop under 0.01 for training and 0.02 for validation set.

### 5.3 Transfer Learning Models

Other action recognition model was trained with transfer learning approach. VGG16 pre-trained Keras model helped to use transfer learning method. Two VGG16 models were trained with transfer learning. First transfer learning model is VGG16 with full layers. Second model is Feature Extractor VGG16 from block4\_pool layer. In the following chapters all details will be explained.

#### 5.3.1 Transfer Learning Approach

Transfer learning is a method to use pre-trained models to build a new model. Instead of training a model from scratch, this method allows to use existing weights, which are already trained with millions of parameters and images. With transfer learning the author avoids to train longer training time and having more data. It can be seen from figure 35 that, there is a Base model. This was trained with very large dataset under long training time.

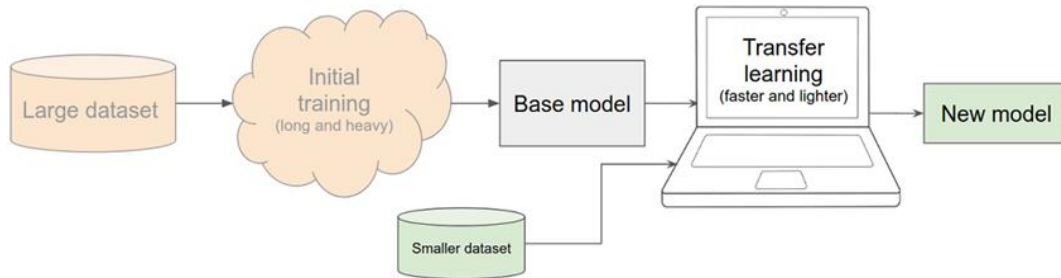


Figure 35. Transfer learning

The definition of transfer learning is based on domain  $D$  and task  $T$ . The domain  $D$  consists of features  $\chi$  and probability distribution  $P(X)$ , where  $X = \{x_1, x_2, x_3, \dots, x_n\}$  and  $D = \{\chi, P(X)\}$ .

The task consist of two components, which are label space  $y$  and predictive function  $f(\cdot)$ . Task function is  $T = \{y, f(\cdot)\}$ .

With a source domain  $D_s$  and source learning task  $T_s$ , transfer learning aims to improve the learning of the target predictive function  $f_T(\cdot)$  in  $D_T$  with using the knowledge in the source domain and learning tasks. Here  $D_s \neq D_T$  and  $T_s \neq T_T$  [17].

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121

Figure 36. Keras pre-trained models

A sample of Keras pre-trained models is shown at figure 36. These models were trained with ImageNet [18] dataset, which is an open sourced dataset for deep learning applications. ImageNet has more than 14 million images under 1000 categories.

The author chose VGG16 pre-trained model for action recognition model. The model achieves 90.1% top-5 accuracy in ImageNet. It was trained for week with NVIDIA Titan Black GPUs [19]. VGG16 has enough parameters to train and predict action recognition model, it is also well accurate. For this reason author decided to use this model.

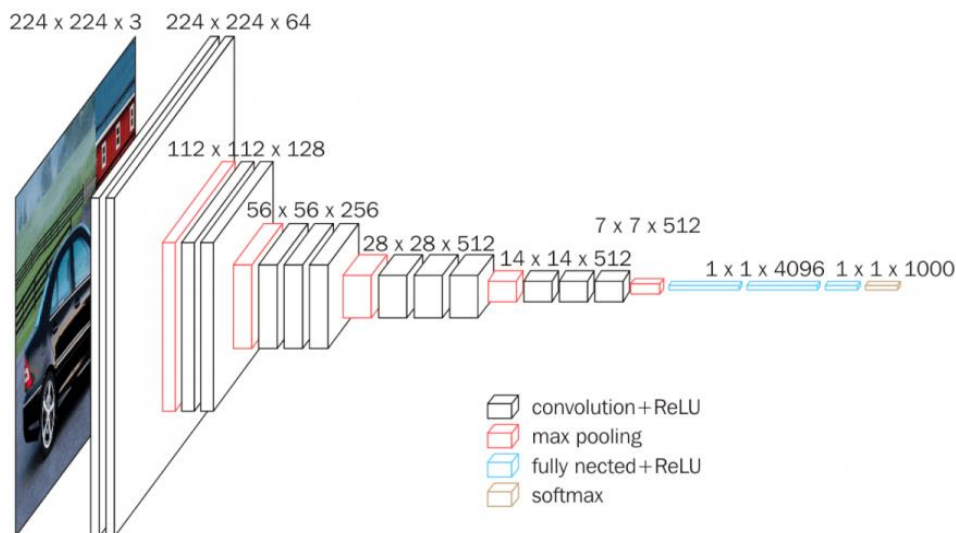


Figure 37. VGG16 model structure

The model structure can be seen above figure. The input is fixed size as 224x224 RGB color images. The input images pass to convolutional layers, after that they go through pooling layers. Max pooling layers follow each convolutional layers. Finally the images leave the model with softmax activation function.

### 5.3.2 VGG16 Transfer Learning Model

First the model is downloaded the same size as the dataset from Keras library. As it was mentioned before, pre-trained models were trained with RGB images. So dataset has to be used as RGB format. After data implementation, dataset shape is (7080, 10, 145, 200, 3).

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 145, 200, 3)]	0
block1_conv1 (Conv2D)	(None, 145, 200, 64)	1792
block1_conv2 (Conv2D)	(None, 145, 200, 64)	36928
block1_pool (MaxPooling2D)	(None, 72, 100, 64)	0
block2_conv1 (Conv2D)	(None, 72, 100, 128)	73856
block2_conv2 (Conv2D)	(None, 72, 100, 128)	147584
block2_pool (MaxPooling2D)	(None, 36, 50, 128)	0
block3_conv1 (Conv2D)	(None, 36, 50, 256)	295168
block3_conv2 (Conv2D)	(None, 36, 50, 256)	590080
block3_conv3 (Conv2D)	(None, 36, 50, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 25, 256)	0
block4_conv1 (Conv2D)	(None, 18, 25, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 25, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 25, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 12, 512)	0
block5_conv1 (Conv2D)	(None, 9, 12, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 12, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 12, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 6, 512)	0
Total params: 14,714,688		
Trainable params: 14,714,688		
Non-trainable params: 0		

Figure 38. VGG16 Pre-trained model summary

Default VGG16 model can be seen at figure 38. After last layer, the model should be flattened for custom input shape, which is 5 dim (None, 10, 145, 200, 3) for action recognition model. Firstly, Global Average Pooling is used for this purpose. But author realized that, there is a problem during detection of the actions. Action is a sequence problem and taking average of the convolutional layers occurs information lose, for this reason Flatten feature was applied. While giving input shape, TimeDistributed feature is added to the model. With TimeDistributed, convolutional part (transfer learning model) can use sequence information for training.



After VGG16, LSTM layer follows the model structure. It is connected directly to the end of the transfer learning part. Size of the LSTM layer is selected as 100. Also dropout is added after the LSTM layer. A hidden dense layer follows the structure. It was added as the size of 512 with *tanh* activation function and dropout layer. After all, the output layer is added to the neural network model. The size of output dense layer should be the same as number of classes. Because one hot encoding was applied during data processing to convert string classes to binary format. As activation function, *Softmax* is added to the hidden layer. Model summary can be seen at below figure.

Layer (type)	Output Shape	Param #
input_7 (InputLayer)	[(None, 10, 145, 200, 3)]	0
time_distributed_2 (TimeDist)	(None, 10, 12288)	14714688
lstm_2 (LSTM)	(None, 10, 100)	4955600
dropout_4 (Dropout)	(None, 10, 100)	0
dense_4 (Dense)	(None, 10, 512)	51712
dropout_5 (Dropout)	(None, 10, 512)	0
dense_5 (Dense)	(None, 10, 4)	2052
Total params: 19,724,052		
Trainable params: 19,724,052		
Non-trainable params: 0		

Figure 39. Summary of training model

As it is shown above, the model has 19,724,052 total parameters as the same amount of trainable parameters. As optimizer Stochastic Gradient Descent (SGD) of Keras is selected with *learning rate* as 0.0001, *momentum* as 0.9 and *nesterov* as *True* boolean. Mean squared error (MSE) function is chosen as loss function.

The batch size and number of epochs are chosen as 10. After each epoch, model is saved to the training directory. Finally, model with the best score is chosen as the main model.

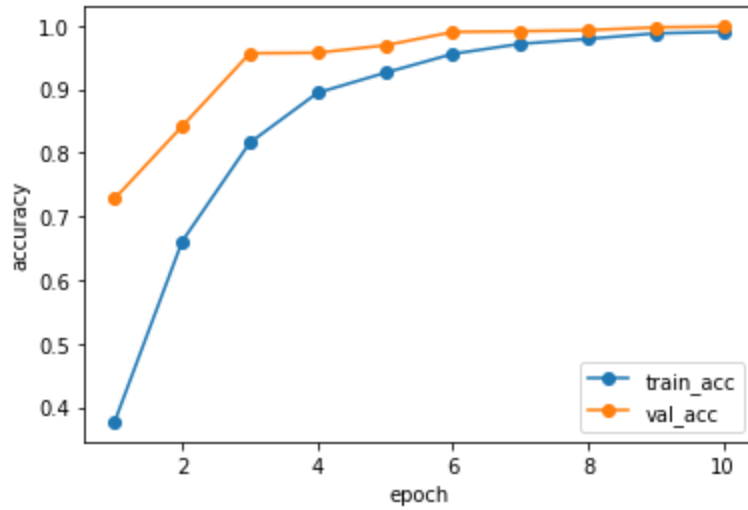


Figure 40. Accuracy graph for both train and validation set

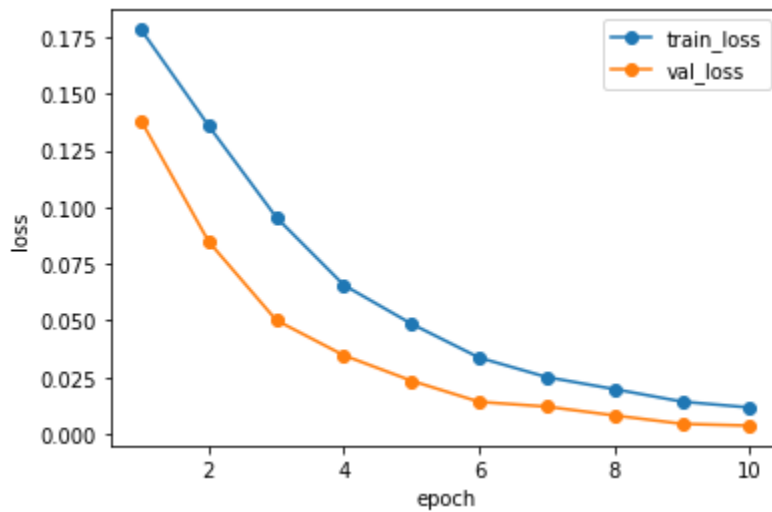



Figure 41. Loss graph for both train and validation set

After 10 epoch model accuracy is around 99% for both train and validation set. The loss is significantly decreased to 0.01 for both train and validation set.

### 5.3.3 Feature Extraction Model

Pre-trained models can be retrained from several part of the model. It means it is possible to choose a layer and start training after that layer. This method is called Feature Extraction. The author decided to cut the last block- block4\_pool of VGG16 and add new layers to train custom model. The reason behind of this idea is detecting actions, instead of images. When the features is extracted from earlier layers, the model has lower abstraction and it is ready to apply for action recognition.

Layer (type)	Output Shape	Param #
input_9 (InputLayer)	[(None, 145, 200, 3)]	0
block1_conv1 (Conv2D)	(None, 145, 200, 64)	1792
block1_conv2 (Conv2D)	(None, 145, 200, 64)	36928
block1_pool (MaxPooling2D)	(None, 72, 100, 64)	0
block2_conv1 (Conv2D)	(None, 72, 100, 128)	73856
block2_conv2 (Conv2D)	(None, 72, 100, 128)	147584
block2_pool (MaxPooling2D)	(None, 36, 50, 128)	0
block3_conv1 (Conv2D)	(None, 36, 50, 256)	295168
block3_conv2 (Conv2D)	(None, 36, 50, 256)	590080
block3_conv3 (Conv2D)	(None, 36, 50, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 25, 256)	0
block4_conv1 (Conv2D)	(None, 18, 25, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 25, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 25, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 12, 512)	0
block5_conv1 (Conv2D)	(None, 9, 12, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 12, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 12, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 6, 512)	0
Total params: 14,714,688		
Trainable params: 14,714,688		
Non-trainable params: 0		



Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 145, 200, 3)]	0
block1_conv1 (Conv2D)	(None, 145, 200, 64)	1792
block1_conv2 (Conv2D)	(None, 145, 200, 64)	36928
block1_pool (MaxPooling2D)	(None, 72, 100, 64)	0
block2_conv1 (Conv2D)	(None, 72, 100, 128)	73856
block2_conv2 (Conv2D)	(None, 72, 100, 128)	147584
block2_pool (MaxPooling2D)	(None, 36, 50, 128)	0
block3_conv1 (Conv2D)	(None, 36, 50, 256)	295168
block3_conv2 (Conv2D)	(None, 36, 50, 256)	590080
block3_conv3 (Conv2D)	(None, 36, 50, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 25, 256)	0
block4_conv1 (Conv2D)	(None, 18, 25, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 25, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 25, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 12, 512)	0
Total params: 7,635,264		
Trainable params: 7,635,264		
Non-trainable params: 0		

Figure 42. VGG16 structure after Feature Extraction

After feature extraction, total number of parameters reduced to 7,635,264 as the same amount of trainable parameters. The same layers as transfer learning model are added to the new model structure: 100 LSTM layer, Dropout, Dense with tanh activation function and Dense with softmax activation function. MSE and SGD are applied to training model.

Layer (type)	Output Shape	Param #
input_5 (InputLayer)	[(None, 10, 145, 200, 3)]	0
time_distributed_1 (TimeDist)	(None, 10, 55296)	7635264
lstm_1 (LSTM)	(None, 10, 100)	22158800
dropout_2 (Dropout)	(None, 10, 100)	0
dense_2 (Dense)	(None, 10, 512)	51712
dropout_3 (Dropout)	(None, 10, 512)	0
dense_3 (Dense)	(None, 10, 4)	2052
Total params: 29,847,828		
Trainable params: 29,847,828		
Non-trainable params: 0		

Figure 43. Summary of the model

Above figure shows the structure of the training model. The model has total parameters and trainable parameters as 29,847,828. The batch size and number of epochs are chosen as 10. As transfer learning model, all training epochs are saved and chosen the most accurate model.

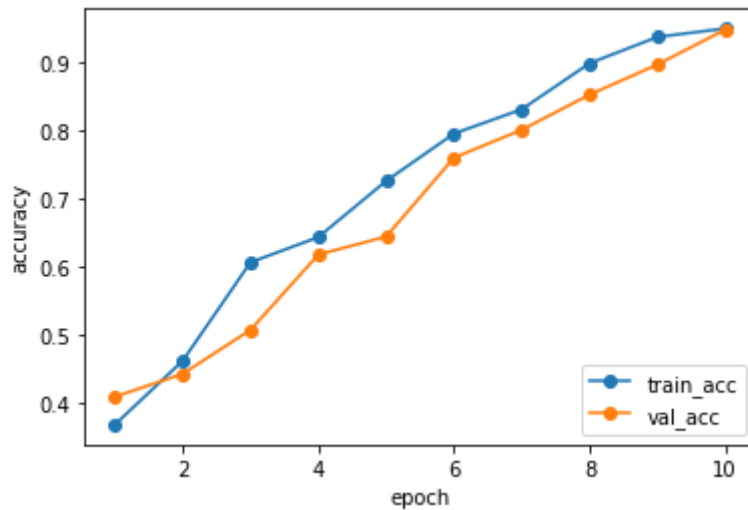


Figure 44. Accuracy graph for both train and validation set

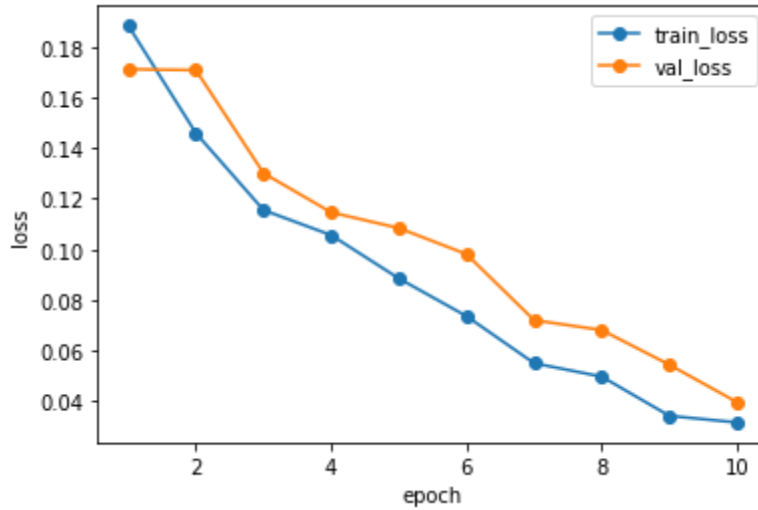


Figure 45. Loss graph for both train and validation set

At the end of the training, the model accuracy reached around 97% for both training and validation data. Training loss decreased to 0.03 and validation loss is around 0.005.

### 5.4 Model Output

The output of the neural network model is ten times of each classes. Because one sequence contains ten images, which means one action. Inside of these ten classes, each result contains four results, because of the last layer of the model. The last layer gives one-hot encoding output and dataset has four different actions.

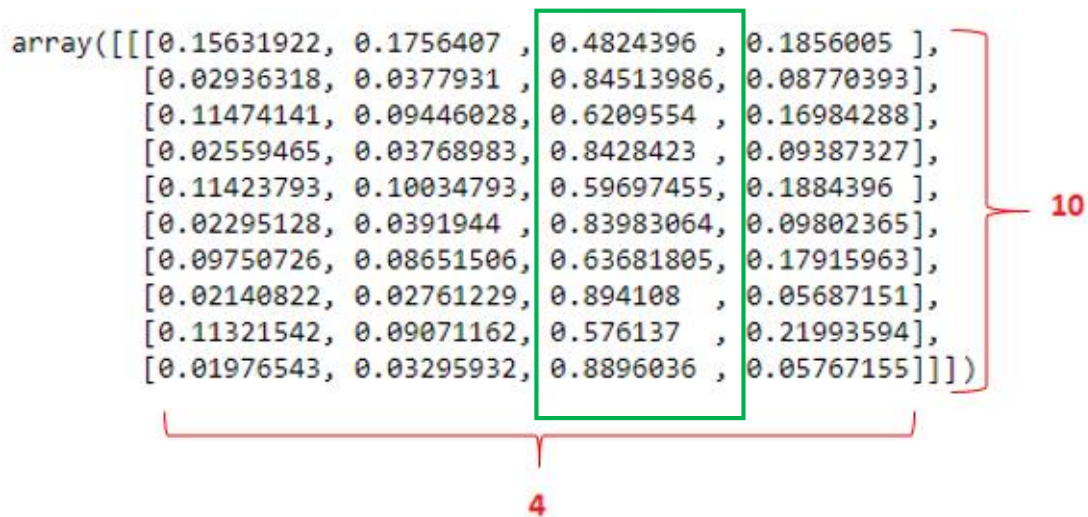


Figure 46. Model output for one action

Figure 46 shows prediction result of sitting action. This result is a probabilistic result of all classes of the model. The sum of the each rows equals 1. It can be seen that third column with green box, the values are smoothly higher than other columns, it means the model predicts this image sequence as third action. The columns represent fighting, running, sitting and shooting in order.

The author took the maximum values of each rows and keep them as classes. To convert these probabilistic values to classes, argmax formula of Numpy module is applied to the array result. After argmax, the array looks like array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2]). Finally, mode function of statistics library of Python is applied and the final result is 2. Mode function counts all array and takes the highest amount of the list.

## 6. EXPERIMENTS WITH OTHER DATASETS

In this section, the experiments related to testing dataset will be seen. Training is done with training and validation dataset. On the other hand there is also another dataset which is independent to training data. Test data is always necessary to predict independent results to measure the real model performance.

### 6.1 Test Dataset

As it was mentioned earlier chapters, the author only trained the model with recorded videos and artificial videos. Test dataset was collected from YouTube videos and UCF101 dataset [13]. Test data contains 8630 images with four categories. These images are collected RGB and Grayscale format. RGB images will be used with transfer learning models, grayscale images with Empty CNN model.

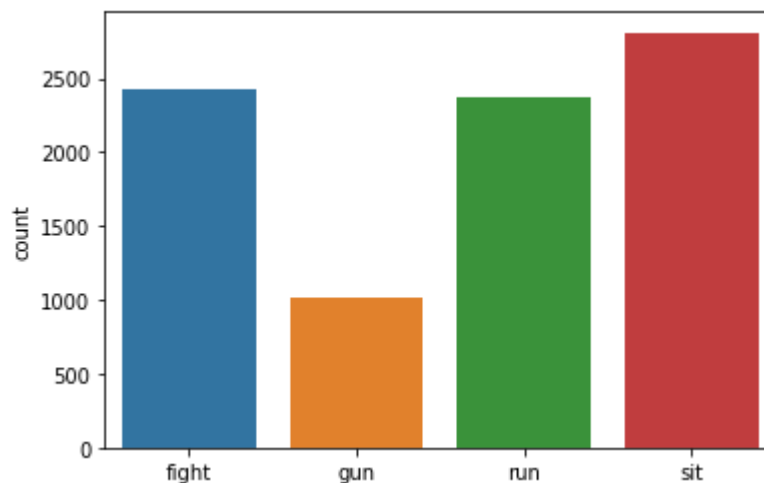


Figure 47. Count plot of test data

As figure 47 shows that, test dataset contains 2430 fighting, 1020 shooting, 2370 running and 2810 sitting action images.

## 6.2 Empty CNN Model Experiments

In this part, the test data will be predicted by Empty CNN model. GrayScale images will be used to show the performance of the model with independent test data.

Test accuracy of the model is  $0.607$  within 850 actions.

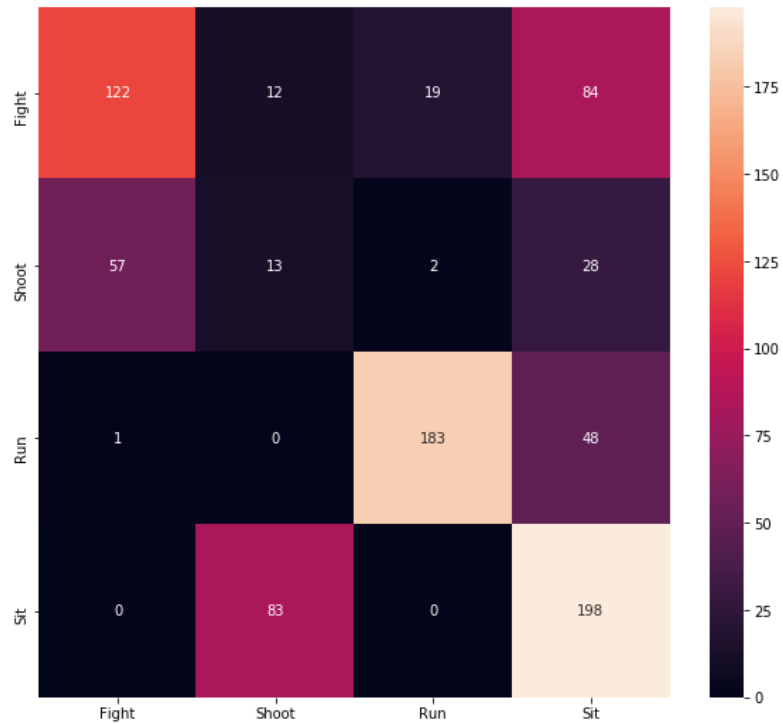


Figure 48. Confusion matrix of test data

Above figure shows test result per each classes with confusion matrix. It can be seen that shooting class is classified with lowest accuracy, it will be discussed after next figure.



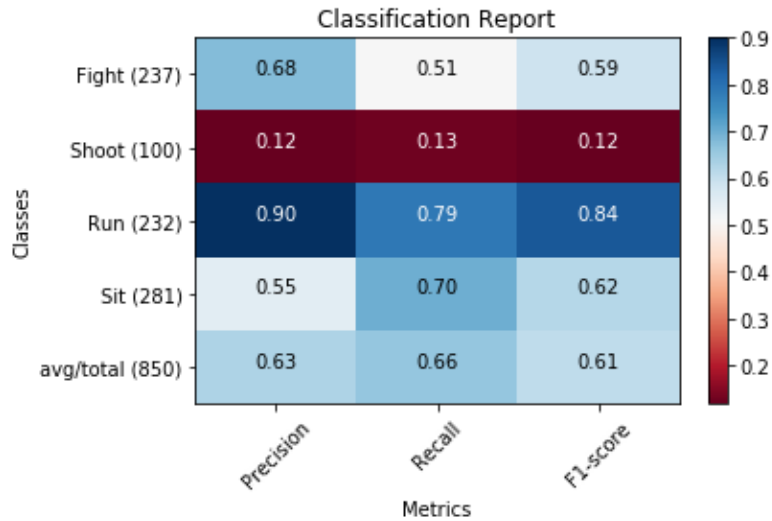


Figure 49. Classification report of test data

$$Precision = \frac{TP}{TP+FP}$$

$$Recall = \frac{TP}{TP+FN}$$

$$F1 = \frac{2*precision*recall}{precision+recall}$$

Above chart gives information about classification performance of each classes of the model. It is also known as classification report with precision, recall and f1-score. Precision is a measure to determine the cost of false positives. Recall is a measure to determine the cost of false negative results. F1-score measures the balance between precision and recall.

84 actions of the fighting class misclassified as sitting class. For this reason recall metric is lower than precision and f1-score. Also there are some false negatives predictions as shooting and running classes.

Shooting is the lowest accurate classes in the model and all metrics are lower than average of the rest of the classes. 57 and 28 actions are classified as fighting and sitting actions. Precision is the lowest metric, because 83 actions of the sitting class were classified as shooting class. Also it should be considered that, shooting class has lower example in testing data.

Running has the highest precision, because there are only 21 actions, which were misclassified as running class. Because of the 48 false negative actions, the recall is lower than expected. Also it has the highest f1-score in the testing data, thanks to high precision value.

70% of the sitting actions are classified correct. Only false negative actions classified as shooting class. There are many actions, which were misclassified as sitting class. This explain the lower precision value.

It can be seen from the results, mostly classes are predicted correct, except shooting class. Shooting action is the lowest accurate class. On the other hand, running is the most accurate class in the model performance.

### 6.3 Transfer Learning Model Experiments

In this section test data will be applied to transfer learning model. The model will predict the results and then the performance will be measured and shown. RGB color images will be used for transfer learning model.

Test data accuracy of the model is  $0.65077$  within 839 actions.

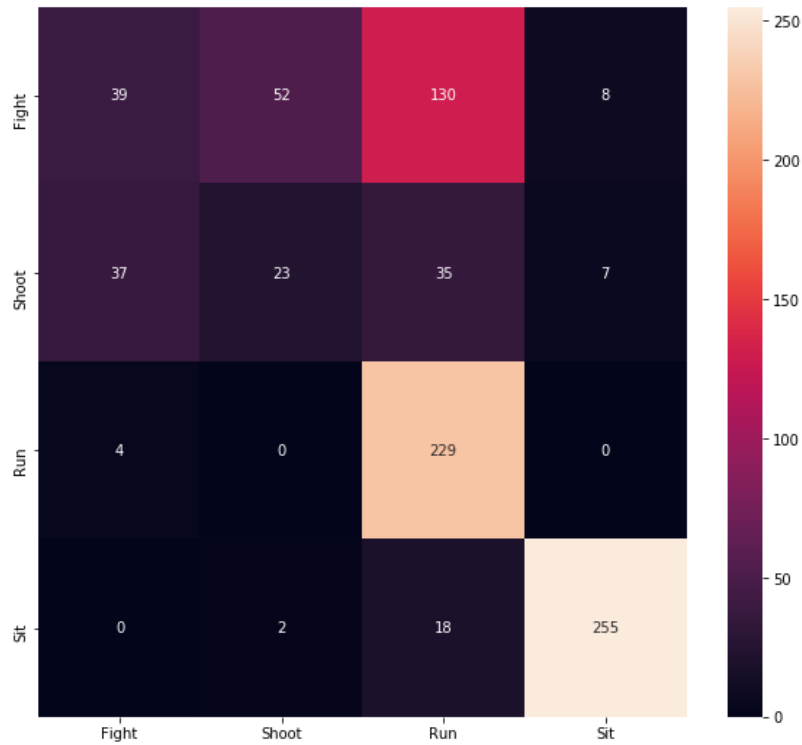


Figure 50. Confusion matrix of test data

Figure 50 shows confusion matrix results. It can be seen that run and sit actions have high accuracies. On the other hand fight and shoot action have lower accuracies. Most of the shooting action misclassified as running action.

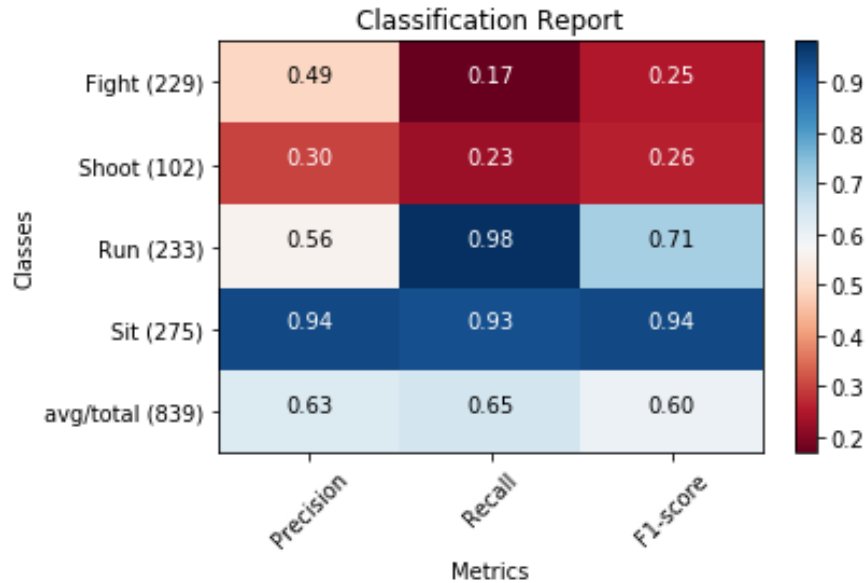


Figure 51. Classification report of test data

Fighting and shooting classes have the lowest recall. Because they have high false negative predictions. 130 of the fighting actions were misclassified as running action, this misclassification affected negatively to the precision metric. Also while fighting, the person moves and sometimes doing running actions, this can be the reason that why fighting was misclassified as running class. Only 39 of 229 actions were classified correctly.

Shooting class has the lowest precision. It can be seen from figure 50 that some of the fight classes are classified as shooting class. 79 of 102 shooting actions were misclassified as other classes.

Running class has the highest recall, it is obvious that it only has 4 false negative predictions. However, on the other side 130 fighting, 35 shooting and 18 sitting actions were misclassified as running. This is the reason of the having low precision of the running class.

The best performance of the model belongs to the sitting class. There are only 15 false positives, so this explains the high precision metric of sitting class.

## 6.4 Feature Extractor Model Experiments

In this section test data will be applied to feature extractor model, which was cut from block4\_pool layer. The model will predict the results and then the performance will be measured and shown. RGB color images will be used for feature extractor model.

Test data accuracy of the model is  $0.6419$  within 835 actions.

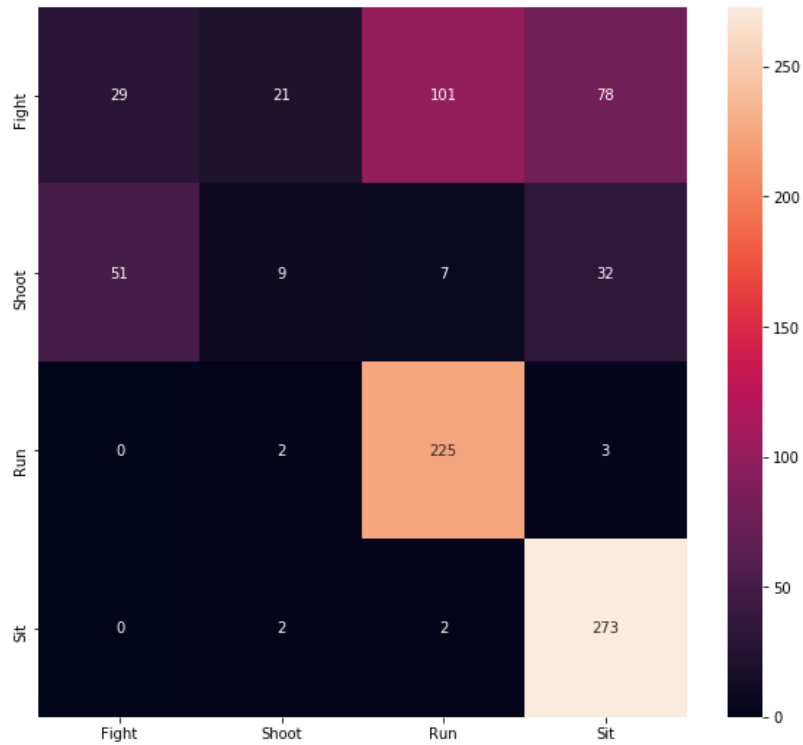


Figure 52. Confusion matrix of test data

Figure 52 shows results of feature extractor model. It can be seen that there are more false detection of sitting class. Also shooting class is barely classified when it is compared with other classes. True classification of the running and sitting classes are still high.

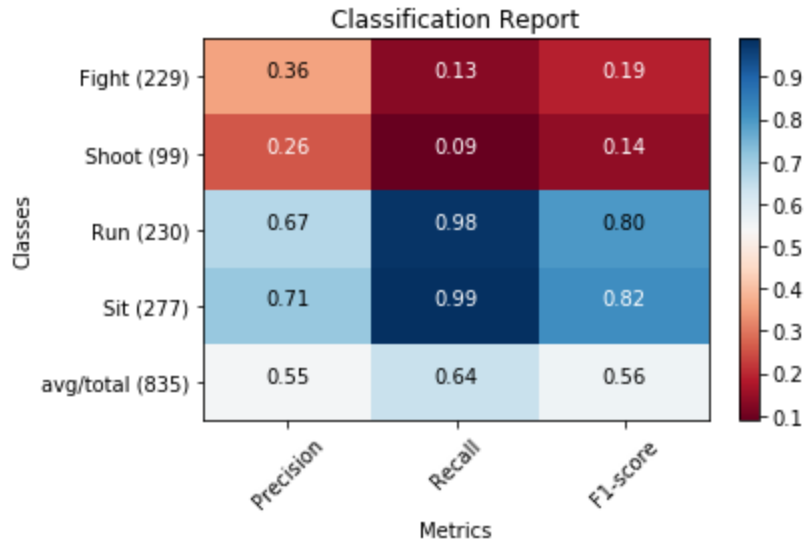


Figure 53. Classification report of test data

Classification report shows clearly that shooting class has the lowest performance in compare of other classes. Only 9 actions were classified correct, the algorithm misclassified 51 classes as fighting class.

Model misclassified fight action as running and siting actions. This explains why fighting class has a low recall metric, there are 179 false negative in the classification. Only 29 actions were classified correct, meantime 21 actions were classified as shooting class.

Running action mostly classified correct, when it is compared with shooting and fighting classes. However there are many false positives because of the fighting action, this explains why running has lower precision metric and recall and f1-score. There are only 5 false negative values, which helps to improve recall of the action.

In the sitting action, the similar results as running action can be seen. 273 of 277 actions were classified correctly and 4 were misclassified. Because of the false positive of fighting and shooting actions, sitting has a low precision. There are 115 false positive classes in the results. Also f1-score is calculated as 0.82.

## 6.5 Experiment with TensorFlow Object Detection API

Object detection is a computer vision algorithm, which works to identify and locate the certain objects in video and images. It is one of the most important and challenging application in deep learning universe. There is two types of object detection; the first one is detection with bounding boxes, which shows the objects with boxes around them. Second one is marking pixel of the objects in the image, it is also known as segmentation. For this project TensorFlow Object Detection API is used by the author. TensorFlow Object Detection API was developed by TensorFlow team [20].

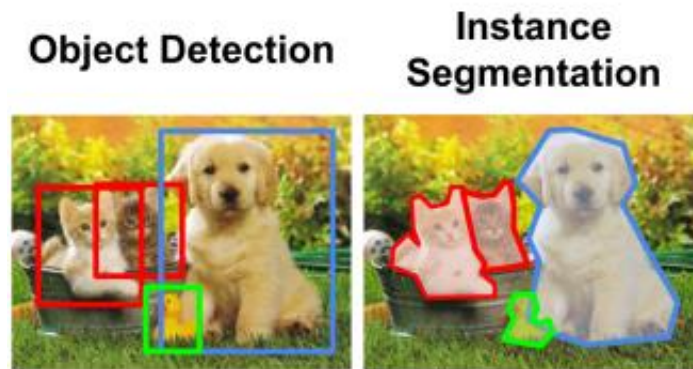


Figure 54. Object detection with bounding boxes and instance segmentation

In this project, object detection was applied to detect only human body in the the video and images. Other background noises are removed and only human body is kept in the video. Single Shot Detector (SSD) MobileNet pre-trained model [21] was used as the base model. SSD MobileNet is suitable for live videos and it has 74.3% mAP, which is quite enough for human detection.

Normally object detection detects 80 classes as person, car, table, animals etc... Only human class is kept inside of the label map, then the model does not detect other objects. When object detection detects a person in the video, the coordinates of the bounding boxes are kept and saved as an image format to the prediction data list. After detecting 10 images, they are resized as the same size of the Keras model and converted to GrayScale or kept as RGB format. Finally action recognition model predicts these sequences and gives the output as the actions.

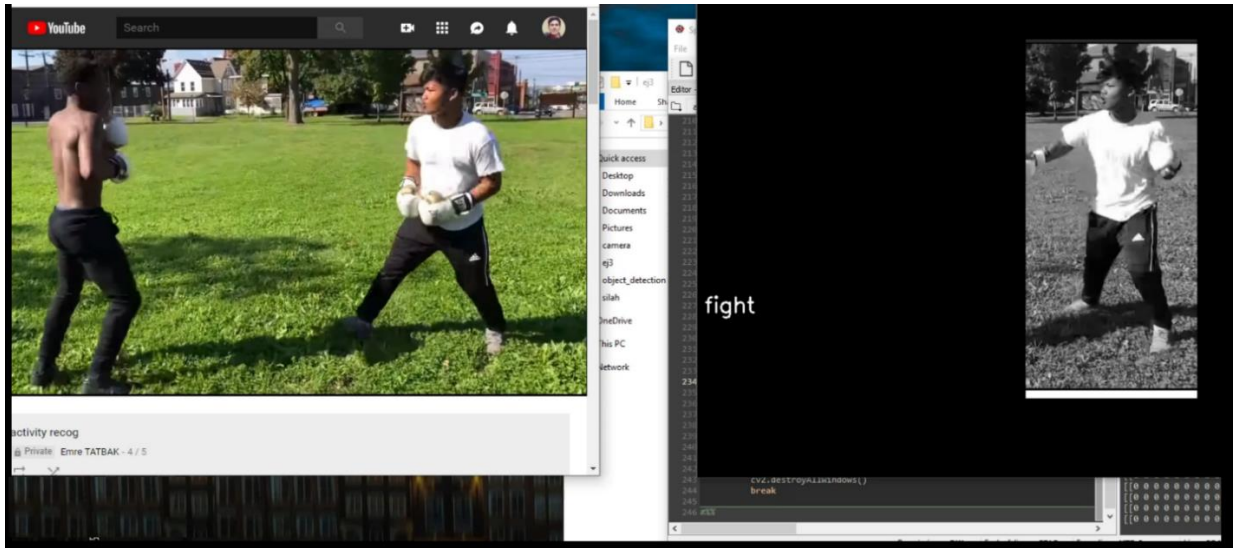


Figure 55. Action Recognition model with object detection

Overview of the application can be seen above figure. On the left screen there is a live YouTube video. Object detection detects the person with white t-shirt from video. On the right screen the overview of the detection can be seen as inside of the box. This image is the input of the action recognition model. The output of the model can be seen as fight action on the right screen. This application was developed by the author to show how action recognition model works with live videos [22]. Every steps as described before in the data processing section are applied automatically with the combination of object detection. Below figure explains how to combine both models in real time detection. Object detection finds the person in the video, then this person is cropped and kept in the memory until collection of 10 frames. Finally this sequence goes into the action recognition model to predict the action.

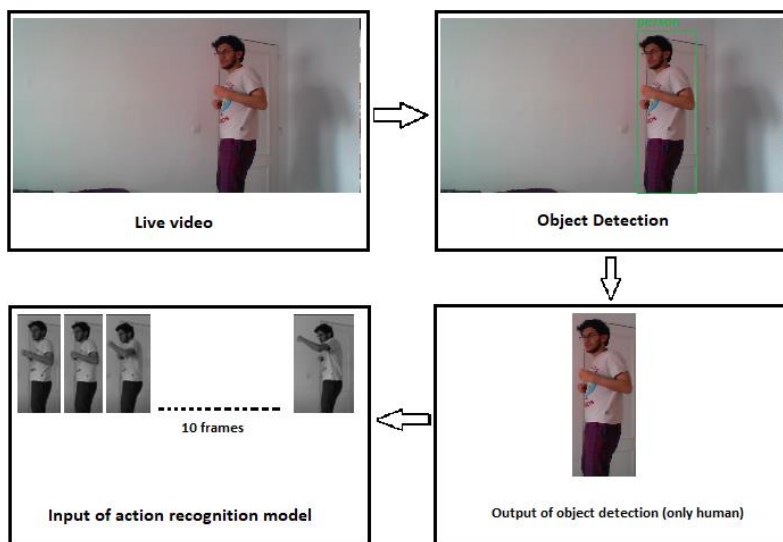


Figure 56. The connection of object detection with action recognition model

## 7. CONCLUSIONS AND FUTURE WORKS

The goal of this project was detecting human actions. We used video and images as an input data of our project. To detect human actions, we used deep learning approach and we tried to explain all details that we used during the all project. We applied Convolutional Neural Network of deep learning models to our images and we found some interesting results after all. These results are based on the dataset that was trained by our models. We recorded ourselves and did the four actions in front of the camera. We investigated all the steps and prepared our model for training section.

During training we used three different models in order to show the results and compare them to get a better idea for future works. Firstly we built our Empty CNN model. This model includes CNN layers, TimeDistributed to create image sequences, LSTM model to keep sequence information in the memory and Dense layer for classification. Our model reached 99% of training and 95% of validation accuracy. With test data we got 60% accuracy. Second model was trained with transfer learning approach. We used VGG16 pre-trained model to train our dataset. We achieved 99% of training and validation accuracy, also we got 65% of testing accuracy. Last model was trained with feature extractor VGG16 pre-trained model. We cut the model from block4\_pool layer and trained the new model. We got similar results as transfer learning: 97% of training, 96% of validation and 64% of test accuracy. The structure of transfer learning and feature extractor models were built similar as Empty CNN model.

Model	Accuracies		
	Train	Validation	Test
Empty CNN Model	98	95	60
Transfer Learning	99	99	65
Feature Extractor	97	96	64

Figure 57. Model performance results

The performance results can be seen from figure 57. Our model achieved almost 99% of accuracy quickly. This is because of the similarity inside our dataset. As it was mentioned before that, we recorded ourselves and collected data from videogames. These videos are similar each other and there is no a big variance between the images. So when the loss is calculated, the model predicts easily to the validation data, because the images are already similar to testing data. This explains why our model learnt quickly.



The performance differences between test and training can be seen from the table. As we mentioned above paragraph, the reason is because of our training dataset. This dataset has similar actions and it does not contain a big variance. Mostly there is one person and they do the actions during the video. In real life, the video frames are not clear as training data. This explains why there is a difference between train-validation and testing data. The model mostly learnt training dataset and if there is a different data, it cannot predict as well as training data.

After comparing the results and checking all metrics, we can say that Empty CNN model with gray scale images gives better precision and recall performance. All classes were predicted with average of 60% of accuracy, except shooting class. On the other hand, Transfer Learning models give the best performance with 64-65% of accuracies. However, Transfer Learning models cannot predict fighting class as good as Empty CNN model. Also Transfer Learning models trained much faster than Empty CNN model, this is also another positive side of using Transfer Learning.

Cutting transfer learning model from block4\_pool helped to increase the performance of sitting action class. However, we cannot say the same as the other classes. It is good to experience to train with block4\_pool feature extractor model.

For this project we aimed to show that our model works accurate with similar dataset, such as similar angle, having all part of the human body inside of the video and similar video quality. For future works we need to improve our training dataset. We will add variation to the dataset with different videos, different angles, different person, different video quality, more data augmentation and more images. This can help to improve the performance of the model.

## References

- [1] Google Creative Lab, Real-time Human Pose Estimation in the Browser with TensorFlow.js, 2018
- [2] Tom Mitchell, McGraw Hill, Machine Learning, 1997
- [3] oregonstate.edu, Chain Rule in Leibniz Notation, 2019
- [4] Matt Mazur, A Step by Step Backpropagation Example, 2015
- [5] Alpaydin Ethem, Introduction to Machine Learning, 2010
- [6] DATAI Team- kaggle.com, Convolutional Neural Network (CNN) Tutorial, 2018
- [7] Ahmed Menshawy, Deep Learning By Example- oreilly.com,
- [8] Jason Brownlee, Master Machine Learning Algorithms, 2016
- [9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, 2014
- [10] Aditi Mittal, Understanding RNN and LSTM- towardsdatascience.com, 2019
- [11] python.org
- [12] Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, Trevor Darrell, Long-term Recurrent Convolutional Networks for Visual Recognition and Description, 2014
- [13] Khurram Soomro, Amir Roshan Zamir and Mubarak Shah, UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild, 2012
- [14] Du Tran, Lubomir Bourdev , Rob Fergus , Lorenzo Torresani , Manohar Paluri Facebook AI Research, Dartmouth College, Learning Spatiotemporal Features with 3D Convolutional Networks, 2014
- [15] Patrice Ferlet, How to work with Time Distributed data in a neural network- medium.com, 2019
- [16] Michael Mathieu, Camille Couprie, Yann LeCun, Deep Multi-Scale Video Prediction Beyond Mean Square Error, 2015
- [17] Yuan-Pin Lin, Tzyy-Ping Jung, Improving EEG-Based Emotion Classification Using Conditional Transfer Learning, 2017
- [18] Imagenet Researchers, image-net.org, 2010

[19] Popular networks- neurohive.io, VGG16 – Convolutional Network for Classification and Detection, 2018

[20] TensorFlow Developers,  
[github.com/tensorflow/models/tree/master/research/object\\_detection](https://github.com/tensorflow/models/tree/master/research/object_detection), 2017

[21] Sik-Ho Tsang, SSD- Single Shot Detector, 2018

[22] Emre Tatbak, Action Recognition video- [youtube.com/watch?v=BmoVg3BCwF0](https://youtube.com/watch?v=BmoVg3BCwF0), 2019