Spring 5-20-2019

POSE ESTIMATION AND ACTION RECOGNITION IN SPORTS AND FITNESS

Parth Vyas
San Jose State University

POSE ESTIMATION AND ACTION RECOGNITION IN SPORTS AND FITNESS

A Project Report

Presented to

Dr. Ching-Seh Wu

Department of Computer Science

San Josè State University

In Partial Fulfillment

Of the requirements for the Class

CS 298

By

Parth Vyas

May 2019

The Designated Thesis Committee is Pending Approval on the Thesis Titled

POSE ESTIMATION AND ACTION RECOGNITION IN SPORTS AND FITNESS

by

Parth Vyas

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÈ STATE UNIVERSITY

May 2019

Dr. Ching-Seh Wu	Department of Computer Science
Dr. Ted Hayduk	Department of Kinesiology
Dr. Chris Pollett	Department of Computer Science

Acknowledgement

Although this quest has come to an end, the many lessons learned will last a lifetime. First and foremost I would like to thank my advisor, Dr. Ching-Seh Wu, who has guided me through the rigorous process of delivering this project. I would also like to thank Dr. Chris Pollett and Dr. Ted Hayduk for their support and guidance throughout. Finally, I would like to thank my family, friends and classmates for their support and feedback.

Abstract

The emergence of large datasets and major improvements in Deep Learning has lead to many real-world applications. These applications have been focused on automotive markets, mobile markets, stock markets, and the healthcare market. Although Deep Learning has strong foundations across many areas, the few applications in Sports, Fitness, or even Injury Rehabilitation could benefit greatly from it. For example, if you are performing a workout and you need to evaluate your form, but do not have access or resources for an instructor to evaluate your form, it would be great to have an Artificial Intelligent agent provide real time feedback through your laptop or phone. Therefore our goal in this research study is to find a foundation for an exercise feedback application by comparing two computer vision models. The two approaches we will be comparing will be pose estimation and action recognition. The latter will be covered in more depth, as we will provide an end to end approach, while the former will be used as a benchmark to compare with. Action recognition will cover the collection, labeling, and organization of the data, training and integrating with real-time data to provide the user with feedback. The exercises we will focus on during our testing and analysis will be squats and push-ups. We were able to achieve an accuracy score of 79% with our best model, given a validation set of 391 squatting images from the PennAction dataset for squat exercise action recognition.

Contents

Algorithms

Figures

Tables

Chapter 1: Introduction

The study of human body movement, also known as Kinesiology, has played a major role in human lifestyle. Ranging from simply preventing and rehabilitating injured body parts, to improving athletic performance by following rigorous training routines. Initially, these tasks have been overseen by a coach or a physical therapist. However, improvements made in technology has allowed individuals to track their own improvement by using sensory devices which provide feedback data. For example, Apple Watch which was initially released in 2015 uses an accelerometer to track step count and calculate how many calories an individual burns. Beast is another product which released in 2017 also tracks exercises using accelerometers, gyroscopes, and compasses.

The field of computer vision has had many breakthroughs in real-world applications and has shown prowess in multiple competitions. In 2009, the ImageNet dataset was an important cornerstone in building image classification models. With over 14 million images and 20,000 labels, it is used for the ImageNet Large Scale Visual Recognition Competition(ILSVRC). In 2012, the AlexNet model had achieved an error rate below 20%, while others were stuck between 20-30 shown in Figure 1.1. The reason why AlexNet was so successful was that it used a deeper model than those used previously. These results in the machine learning along with the massive amounts of data sparked interest in many companies, leading to tremendous amounts of investment into research and development in computer vision. Facebook has had major breakthroughs in identifying individuals in group pictures, and Tesla has been constantly improving its self-driving car using image recognition and collecting data through LIDAR sensors.

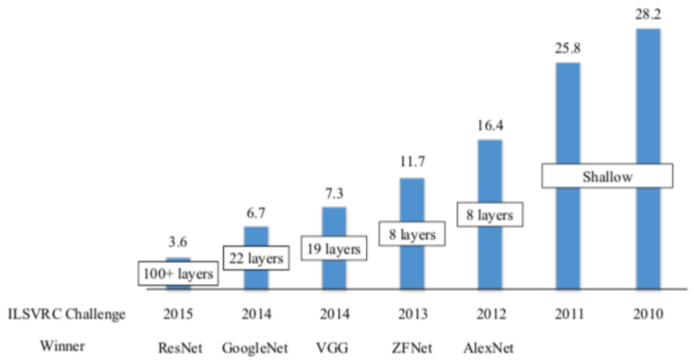


Figure 1.1: ImageNet error rates [17]

By incorporating machine learning techniques, many have begun applying these image recognition approaches for pose estimation or action recognition in videos. Pose estimation is used to identify human's in videos, and by taking it one step further, they map out joints from the body and are able to connect the joints to form a skeletal representation. There is an IOS app known as Kaia squat challenge which uses the phone's camera to track 16 points on the body and calculate angles using the relative positions of the limbs. There was also a separate study on action recognition in hockey. In this, they classified 4 different poses which were identified in a video recording of a hockey game. Although these techniques are fairly new, they have had a lot of attention and interest, with much development already in progress.

I. Problem Statement

Pose estimation is a very useful technique in computer vision. Until the construction of datasets such as COCO key points challenge, MPII human pose estimation, and VGG pose dataset, there was little to no improvement in the field. The study conducted for convolutional pose machines and 2D multi-person pose was able to successfully identify and estimate human body parts within a video. The model for 2D multi-person pose was named as OpenPose, and the architecture design is given in Figure 1. The model has two branches, the first branch predicts body part confidence maps, and the second branch predicts body part associations. The original image actually runs through a VGG-19 model for the first 10 layers in order to generate feature maps.

The part affinity field vectors provide opportunities to calculate angles between body parts. We can now measure the biomechanics of the human in an exercise or training video, and provide feedback on their routine. For example, if a user is attempting to perform a squat, we can locate the part affinity field vectors of the legs, and calculate the angles to check whether the user is applying the correct motion. This would be beneficial for individuals who do not have time to visit a personal trainer at the gym, or for an injured individual to go to a rehab center just to perform movements to restore the natural functionality of their injured body part. The feedback can be provided using the OpenCV framework, which consists of a library of programming function for computer vision. The library has support in C++, Python, and Java.

A major disadvantage in pose estimation is generating labels for the datasets. It can be a very tedious process as manual work is required to identify and label the key points within an image. This is also error prone since a missed tag or incorrect label can cause undesired results in the training process. Therefore an alternative model which identifies the action based pose once a silhouette has been extracted can alleviate the issue of rigorous manual labeling. Therefore the new approach will be specific to workouts and allow for easier data organization and labeling which can benefit the accuracy and dependability of the model identifying the workout pose. For example, if an individual would like to track their pull up form, we can train a model to learn the form required in a pull-up, and test the model in real time using the OpenCV library.

This would require an additional step to train each exercise specific model, and it would require labeled datasets. One such dataset which would help is the Penn Action dataset which includes frames captured from YouTube videos, and there are videos which are performing specified workouts such as pull ups or squats. The model can be trained in Python using the Keras deep learning library, which runs on top of TensorFlow.

In this project, the goal is to find the most effective model which would easily and accurately predict a specified workout. The pose estimation approach uses a pre-trained model, which will use geometric analysis to identify the workout performed within a video, which can be in real-time. The silhouette extracted action recognition model will be trained on datasets which contain labeled workout images. Background subtraction will be applied to the images reducing noise prior to training. Once the model is trained, it will be integrated with the webcam to perform real-time and pre-recorded video frame based analysis.

Chapter 2: Background

The motivation for this project comes from the large successes achieved by multiple computer vision machine learning models. In particular, pose estimation has had provided a variety of options with its key point challenge results. Although we do not train any models for pose estimation in our project, we use the key points provided to analyze the pose of the individual performing the workout and provide simple feedback such as how many repetitions the individual has completed. We have implemented an alternative approach to provide similar feedback. Rather than analyzing the workout with key points and angles, we train multiple models on individuals performing a given workout. Then we apply those models on either real-time or pre-recorded videos and give rep counts back to the user.

This chapter outlines some of the models which motivated our work, along with tools which would help us achieve faster results. Our goal is to be able to cover the following steps end to end:

- Gather datasets
- Augment data
- Organize and label data
- Train a few neural networks
- Integrate with OpenCV

- Compare results from our silhouette based approach to pose estimation approach

I. Pose Estimation

The idea behind pose estimation is to localize anatomical key points on an individuals body within a frame [2]. The techniques used for pose estimation combine neural networks and computer vision. They introduced the first bottom-up approach for pose estimation using Part Affinity Fields, which are 2D vectors that locate direction and orientation of limbs in the target image. By inferring a bottom-up representation of each detected key point allows for greedy parsing to achieve high-quality results, and decreases computational costs. Previously Top-Down approaches were used for generating key points on the body, where a model will crop individuals from the group in an image, and perform pose estimation on each. The issue with this approach is if the person detector fails, everything following will fall out of place since each subsequent stage is tightly coupled with the person detector.

In their approach, they used a two-branch convolutional neural network. Convolutional neural networks are similar to multi-layered perceptrons, also known as feedforward neural networks. The goal of the MLP is to take an input x, and map it to some category y given a function f [8]. The network can contain multiple layers which would each be a unique function that will apply the output of the previous layer. The convolution neural network contains convolutional layers would take an image and extract features, such as lines and edges, and pass that into the next layer. Finally, the MLP at the end will classify the final output into a category. Figure 2.1 provides an example of how the input image can be transformed through convolutions.

The CNN was used in order to train the model which would identify the key points for the individuals in the frame of reference. The model would take as input colored

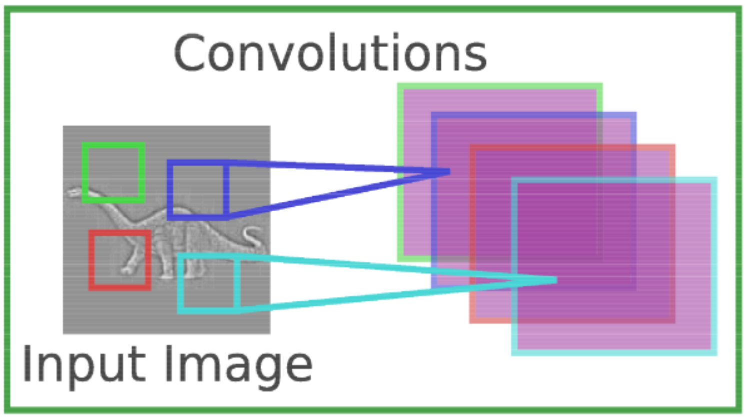


Figure 2.1: Example of feature extraction. This transformation takes an image and kernel filter, and extracts maps of 2D feature grids [9]

image of size $w \cdot h$ and pass it into the first 10 layers of VGG-19 model to generate F feature maps. Next it would pass these feature maps into Stage 1 of a multi-staged, 2-branched CNN. Branch 1 will predict key points or confidence maps $S1 = \rho(F)$ in stage 1. Branch 2 will predict part affinity fields $L1 = \phi(F)$ in stage 1. Next it will concatenate $F, S1, L1$ and use it as input in $S^{t>1}$. The $L2$ loss function for the confidence map at stage t is

$$\sum_{j=1}^{J} \sum_{p} W(p) \cdot \|S_j^t(p) - S_*^j(p)\|_2^2$$

and the part affinity loss at t is

$$\sum_{c=1}^{C} \sum_{p} W(p) \cdot \|L_c^t(p) - L_*^c(p)\|_2^2$$

.

The model achieved state-of-the-art results when testing on both COCO and MPII datasets. Although there were some false positives identifying body parts, and at times some parts were completely missed, the average precision score achieved was 75.6. Figure 2.2 shows some examples of the model's predictions, and Figure 2.3 shows the

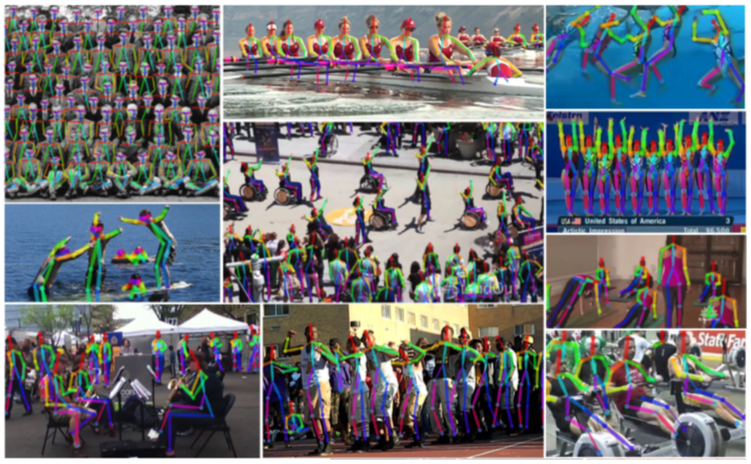
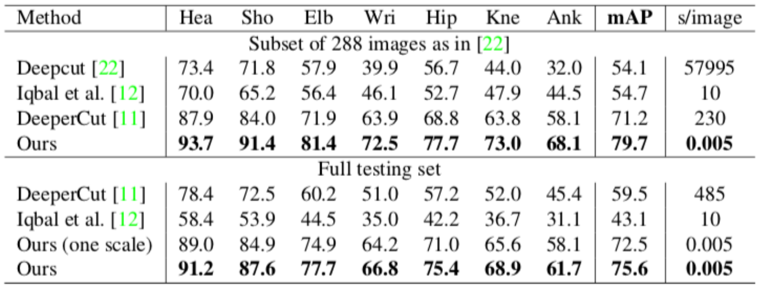


Figure 2.2: Example of pose estimation key points and body part vectors [2]

precision scores achieved on 7 different body parts, and the mean average precision score.



Method	Hea	Sho	Elb	Wri	Hip	Kne	Ank	**mAP**	s/image
Subset of 288 images as in [22]									
Deepcut [22]	73.4	71.8	57.9	39.9	56.7	44.0	32.0	54.1	57995
Iqbal et al. [12]	70.0	65.2	56.4	46.1	52.7	47.9	44.5	54.7	10
DeeperCut [11]	87.9	84.0	71.9	63.9	68.8	63.8	58.1	71.2	230
Ours	**93.7**	**91.4**	**81.4**	**72.5**	**77.7**	**73.0**	**68.1**	**79.7**	**0.005**
Full testing set									
DeeperCut [11]	78.4	72.5	60.2	51.0	57.2	52.0	45.4	59.5	485
Iqbal et al. [12]	58.4	53.9	44.5	35.0	42.2	36.7	31.1	43.1	10
Ours (one scale)	89.0	84.9	74.9	64.2	71.0	65.6	58.1	72.5	0.005
Ours	**91.2**	**87.6**	**77.7**	**66.8**	**75.4**	**68.9**	**61.7**	**75.6**	**0.005**

Figure 2.3: Precision scores across multiple models [2]

Person lab which is another bottom-up approach was researched by a team from Google. Their architecture branched into five different parts which are heatmaps, short-range offsets, mid-range offsets, person segmentation Masks, and long-range offsets. The process was the same, feed an image into a CNN which would be trained to predict the respective branch. The heatmap prediction was done through binary classification whether a pixel fell within a certain radius around a key point, while the offsets predicted a vector which pointed to one of the K different body joints through regression using L1 loss. Segmentation was predicted by logistic regression, calculating if a pixel belongs to at least one person in the image [12].

Other top-down approaches include pose machine and deep pose both use the same datasets, leeds sports dataset and FLIC(frames labeled in cinema), for training. Pose machine prevents the vanishing gradient problem by incorporating some intermediate supervised learning in their deep neural network. Deep pose also uses a deep neural net, which contains a CNN along with a regressor and a refiner, it contains 40 million parameters and follows a holistic approach compared to a part by part approach [15] [14]. Forecasting human dynamics from static images uses an encoder decoder with an RNN within its model structure. The Encoder takes the input image, applies a few convolutions to target features, apply an LSTM and upscale with the decoder and classifies heatmap key points [3].

II. Pose Trainer

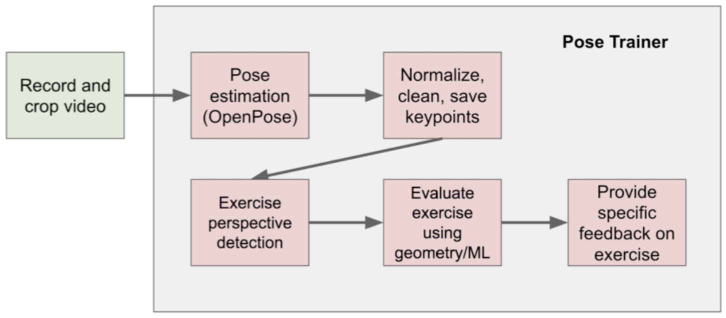

Pose Trainer
Record and crop video
Pose estimation (OpenPose)
Normalize, clean, save keypoints
Exercise perspective detection
Evaluate exercise using geometry/ML
Provide specific feedback on exercise

Figure 1. Pose Trainer system pipeline, as described in Technical Approach.

Figure 2.4: Pipeline of the process used in pose trainer [4]

The pose estimation provided very promising outputs which caught many researcher's interests. One group of student's from Stanford took the pose estimation model and applied geometric analysis to the key points that were generated. Figure 2.4 shows a pipeline from taking an input image to outputting feedback to the user. Before they begin their geometric approach, they normalize all test images by taking the average between the neck and the right and left hip, this way the calculations are generalized regardless of the height of individuals. The first exercise which they estimate is the bicep curl. Here they have a starting point, which is 180°. Next, they have two fail states, one is if the individual performing the exercise goes beyond 35°and the second is if they stay below 70° [4]. Figure 2.5 shows an example of the model in action.

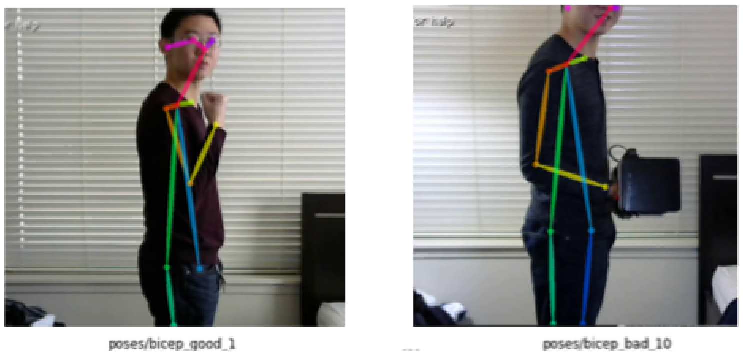


Figure 2.5: Pose trainer test on bicep workout [4]

The second approach they use a machine learning approach where given two videos, they measure the Euclidean distance between two given key points. Given two key point sequences $Q \in R^m$ and $C \in R^n$, they construct a distance matrix $D \in R^{m \cdot n}$, where $D_{i,j}$ is the Euclidean distance between points q_i and c_j [4].

Although they do not specify the source of their training data, the results they achieve are great. They were able to classify all good form videos correctly in their geometric model. There was a 20% chance that the analysis returned a false positive, otherwise, it was able to distinguish incorrect workouts. The feedback provided was in the form of textual messages. For example, if the user was performing shoulder shrugs, and failed to reach the proper height, the following message would appear “Your shoulders do not go through enough motion. Squeeze and raise your shoulders more through the exercise” [4].

III. Spatial-Temporal Based Approach

Many approaches to action recognition and classification have focused on CNN models. Baccouche, $et\ al.$ [1], Donahue, $et\ al.$ [6] and Karpathy, $et\ al.$ [10] combined long short-term memory models with CNN. By using an LSTM, this allowed for more complex memory gates which allow the network to learn to gorget previous hidden states, and update hidden states with new information. An additional benefit is they are not restrained to fixed length inputs. Donahue, $et\ al.$ model for activity recognition takes the input as a sequence of frames, passes it into CNN layer which generates the feature maps, followed by an LSTM layer which outputs the result and labels the image. Baccouche, $et\ al.$ takes a model with 10 layers which contain convolutions, rectification, and subsampling repeated twice, followed another convolution and ending with two fully connected layers. Donahue and Karpathy used the UCF-01 datasets, while Baccouche used the KTH for training and testing.

These frameworks and techniques are invaluable in computer vision. Pose estimation opens endless possibilities across many use cases. In our case, by extracting the key points and mapping the body parts, we can now take these values and vectors and calculate angles to provide feedback to users. The second approach includes extraction of the human silhouette and classification of the pose or action based on the shape of the silhouette using convolutional neural networks. Once the workout begins, we can make use of the OpenCV framework in Python, which allows users to add objects and texts to images and videos. We can take the feedback which is occurring in real time, and display this to the user as they are performing the workout.

IV. Action Recognition

Action recognition in sports tackles the problem of quickly and efficiently analyzing player performance. Fani, *et al*. [7] provides their solution using a CNN called action recognition hourglass network (ARHN) and apply it to ice hockey videos. In their approach, they generate and take an annotated dataset of hockey images, which are video frames of hockey players, falling into four categories of "Cross-Overs, Straight Skating, Pre-Shot, and Post-Shot". Figure 2.6 shows an overview of their framework.

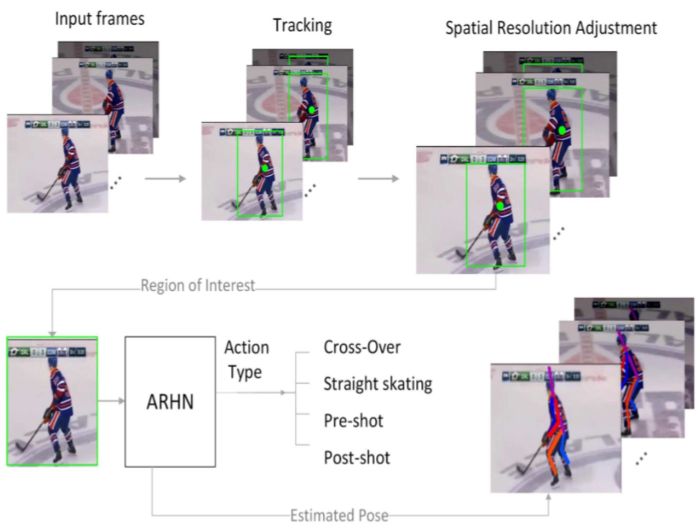


Figure 2.6: Action recognition framework [7]

They begin by converting a video into a sequence of frames. In each frame, they determine the center of the player in focus. Then they adjust the size of the frame into a 720 x 1280 pixel image, and subsequently taking a 250 x 250 region of interest using the center of the body of the player in focus and feeding it into the ARHN network. The network generates pose estimation by finding the heatmaps of the joints, which is finally used to estimate the players' action. The architecture of their ARHN consists of three components. The first takes the input image and generates heatmaps which define a pose, the second which takes latent features and transform them into a common frame of reference, and finally the third which classifies the action with six fully connected layers.

The first component of their ARHN follows an hourglass pattern, which scales the image down into 4x4 pixel feature maps. This is done by applying convolutions, max pooling. The next step is taking these feature maps and applying up-sampling using nearest neighbor. The output of the first component is a set of 16 heatmaps which identify the joints of the individual from the input image. The second part of the ARHN takes the heatmaps and approximates the joint based on the location peak of the heatmap. Once all the joints are mapped, the points are joined together to form a 40-dimensional vector known as the canonical pose. The pose also has the calculations of the angles between the joints which are used to perform the final step in the action recognition. Finally, the last step is to pass the canonical pose into the six fully connected layers, which classify the pose and return a label.

The dataset which was used had gone through extensive manual labeling since there weren't any datasets published which matched the requirements for their model. The name of the dataset is HARPE which is a collection of hockey videos converted to frames. The labels for the frames are specified into the four categories mentioned above. Low-quality frames are removed by manual detection, which is a meticulous process. Each frame contains 16 annotations on the body parts and an additional 2 which indicate the ends of the hockey sticks. The accuracy was measured through mean average precision, and the results had classifications above 60%.

V. Background Subtraction

In order to target and isolate the person who is performing a workout, Szucs, *et al*. [13] propose their own unique way of subtracting the background. Other methods include Gaussian mixed model which is a non-parametrized approach and the complexity increases with the amount of training and kernel-based which is parametrized however their approach received better results. Their approach is specific to their problem since they need multiple frames and the person moves horizontally allowing for a portion of the image which does not include the person extracted and appended to the next frame which has the person moving in the opposite direction, therefore gathering the full static background. Then once the static background has been collected, the image is subtracted from the original image with the individual inside, and the resulting subtraction leaves just the silhouette of the target allowing for tracking motion.

GrabCut an extension of graph-cut is another approach which takes the input image, based on user specifications takes a rectangular coordinate of the region of interest and performs segmentation and background subtraction. It segments the initial image into foreground, background, and unlabeled pixels, which is then defined into an array of grey values with a range from 0 to 1. The background is given a 0 and the foreground is given a value of 1. An energy function is then calculated given a generated histogram of the grey scale pixels.

VI. Silhouette Based Classification

Although pose estimation is accurate and generates a map of key points, it may not be required for recognizing and classifying poses. An alternative approach presented by Dedeoğlu, *et al*. [5] is based on extracting the humans silhouette and then classifying an action or pose based. The labeled poses are walking, punching, and kicking. Once the silhouette was generated, a normalized histogram is used to calculate the Euclidean distance to classify the action.

Luberg, *et al*. [11] also takes a similar approach, initially identifying the human in the frame using histogram oriented gradient. Once the human is detected, the next step is to extract the silhouette, Luberg makes use of the GrabCut algorithm which takes an image and divides it into foreground and background. The training is run through a CNN which contains convolutional layer 32 x 3 x 3, pooling 2 x 2, which is repeated twice, followed by a hidden layer of 64 nodes. The output is classified into two categories, either walking or standing. The dataset used was GRATZ-01 and contains 226 images, however, with data augmentation the number of images increased to 2260 with over 1000 images per action.

Chapter 3: Methodology

I. Dataset Collection and Preprocessing

As specified in the background, the workouts that will be analyzed and tested must be trained by our model. The primary dataset in which the models trained on was the PennAction dataset, collected from the study on "Strongly-supervised Representation for Detailed Action Understanding" [16]. There are a total of 15 unique actions, with a set of frames extracted from videos in which individuals are performing a cycle of each action. Examples of frames extracted from a video with individuals performing a squat and a pull-up, respectively, can be found in Figure 3.1.

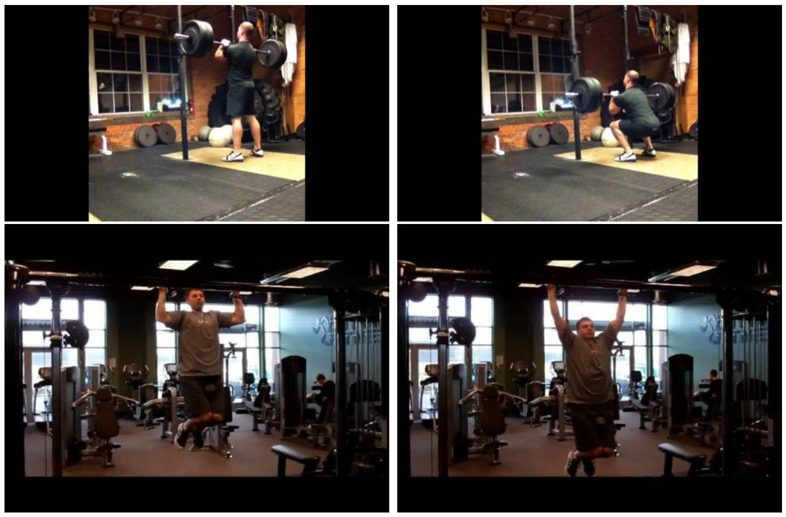


Figure 3.1: PennAction frames [16]

There are 231 sets of squat cycles performed, with a total of 1959 images to train on. There are 199 sets of pull up cycles, and a total of 1277 pull up images. The images were then resized to 350 pixels by 350 pixels for consistency. Finally, in order to apply background subtraction, the frames were put together into videos. Additionally we have also collected datasets from UCF-101 and kinetics action video dataset, however, we used those datasets for the testing phase and not training. This decision was made because the videos had either too much noise, or the individuals were not performing single cycles. Therefore we decided it was best to keep them for testing purposes, and if we need to improve our models, we could manually pick and separate the frames and aggregate them to our training set.

II. Background Subtraction

To reduce the number of parameters and noise, while improving accuracy on the training of the model we apply background subtraction on the training sets. The approach used is a Gaussian-based mixture model, which takes the probability density across the frames to estimate the background [18]. The background subtraction equation which finds the probability that a pixel is in the background is given here,

$$\frac{p(BG|\vec{x}^{(t)})}{p(FG|\vec{x}^{(t)})} = \frac{p(\vec{x}^{(t)}|BG)p(BG)}{p(\vec{x}^{(t)}|FG)p(FG)}$$

and the GMM initializes the background given in the following formula.

$$\hat{p}(\vec{x}|X_T, BG+FG) = \sum_{m=1}^{M} \hat{\pi}_m(\vec{x}; \hat{\vec{\mu}}_m, \hat{\sigma}_m^2 I)$$

Then it recursively updates the mean, covariance, and weights over an exponentially decaying constant α. The reason we chose to use the GMM version compared to a non-parametric version is that it is optimal when we have static scenes, in our case the scenes stay the same, and only the individual performing the workout is moving. Table 3.1 shows the general steps followed in both the GMM approach which we have used and the non-parametric approach which can be applied to dynamic scenes.

The pseudocode for the background subtraction implementation used and applied to our dataset is presented in algorithm 1. We begin by setting the input directory which contains the workout videos, followed by the output directory containing the background subtracted videos. We loop over each video, create the background subtractor, set the frame width and height, and initialize the video with these properties. Next, we enter an infinite while loop, exiting once the last frame of the video is reached. Inside the loop, the background subtractor is applied to each frame, and the newly generated frame is saved to the output video we initialized earlier. Finally, we catch any error exceptions thrown and handle them appropriately.

Table 3.1:
A brief summary of he GMM and the non-parametric Background Subtraction algorithm

General steps
Classify the new sample $\vec{x}^{(t)} p(\vec{x}^{(t)} \vert X_T, BG) > c_{thr}$
Update $p(\vec{x}^{(t)} \vert X_T, BG + FG)$
Update $p(\vec{x}^{(t)} \vert X_T, BG)$
These step are repeated for each new video frame [18]

Since background subtraction requires a sequence of frames, the workout frames were combined into an audio video interleave and run through the GMM background subtractor. Finally, the last bit of image processing is complete, however before we can pass the data to the model we need to split and organize the workouts into two sets. The split involved for squats was the individual in standing form and working towards parallel as one set. The other set was the individual in a fully parallel squat form. Pull-ups followed the same split idea, with the individual at rest versus the individual pulled up over the bar. The labels for the squat sets were either standing or squatting, and the labels for pull ups are pull up and not pull up. Since these are binary classifications, the labels can be changed to any name, but the data split remains the same. Sample output from frames extracted out of the background subtracted videos can be found in Figure 3.2. In the next chapter, we will take a look at the input, architecture, and output of our model.

Algorithm 1 Background subtraction

```
 1: procedure APPLY BACKGROUND SUBTRACTION ON PRE RECORDED VIDEOS
 2:     inputpath← video input directories
 3:     outputpath ← video output directory
 4:     for video in inputpath do:
 5:         if "DS" not in video then
 6:             cap ← VideoCapture(path_to_video)
 7:             fgbg ← createBackgroundSubtractorMOG2(frame)
 8:             height ← cap.height
 9:             width ← cap.width
10:             out ← VideoWriter(outputpath, width, height)
11:             while TRUE do
12:                 ret, frame ← cap.read()
13:                 if ret == TRUE then
14:                     fgmask ← fgbg.apply(frame)
15:                     out.write(fgmask)
16:                 end if
17:             end while
18:             cap.release
19:             out.release
20:             cv.destroyAllWindows
21:         end if
22:         Throw Exception
23:     end for
24: end procedure
```

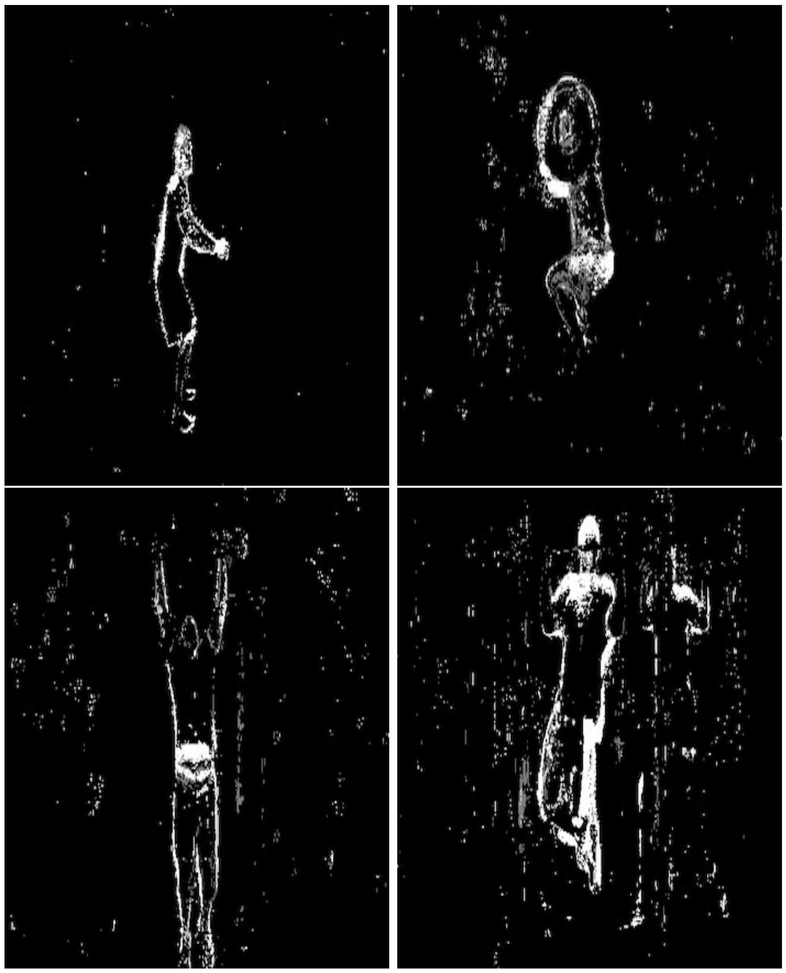


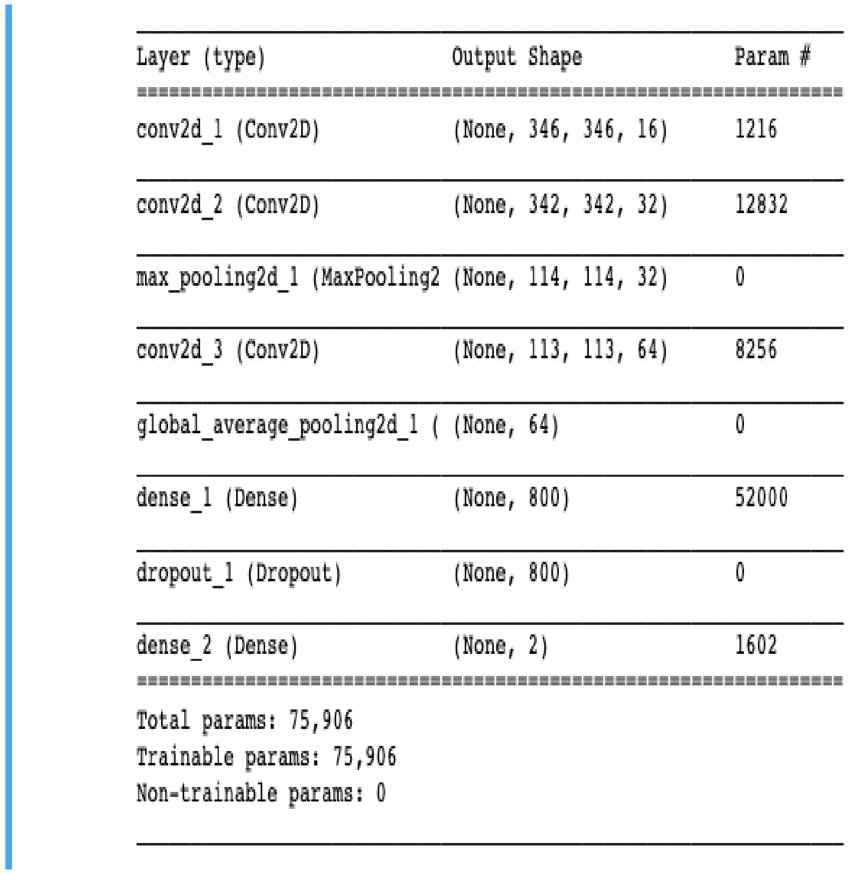
Figure 3.2: Background subtraction squats

III. Model Architecture

Now that the data has been processed, and we have the silhouette of the individual organized and labeled, we pass an input list of 350 by 350 pixels into the neural network. Over the course of the training phase, a few different models with hyper-parameter adjustments are proposed. The initial model architecture involved three convolutions with rectified linear units as the activation function. ReLU is used to avoid running into the vanishing gradient problem, since the derivative is zero for negative input, and 1 for positive input, compared to the tanh 3.4 function which flatlines on inputs less than -2 and greater than 2.

$$ReLU(x) = \begin{cases} x & x > 0 \\ 0 & otherwise \end{cases}$$

$$ReLU'(x) = \begin{cases} 1 & x > 0 \\ 0 & otherwise \end{cases}$$



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 346, 346, 16)      1216
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 342, 342, 32)      12832
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 114, 114, 32)      0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 113, 113, 64)      8256
_________________________________________________________________
global_average_pooling2d_1 ( (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 800)               52000
_________________________________________________________________
dropout_1 (Dropout)          (None, 800)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 1602
=================================================================
Total params: 75,906
Trainable params: 75,906
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.3: Initial model layers

Figure 3.3 provides an overview of the architecture, including the total number of parameters at each layer. We have a total of 8 layers including pooling and dropout layers. The first convolution contains 16 filters with a kernel size of 5. The second

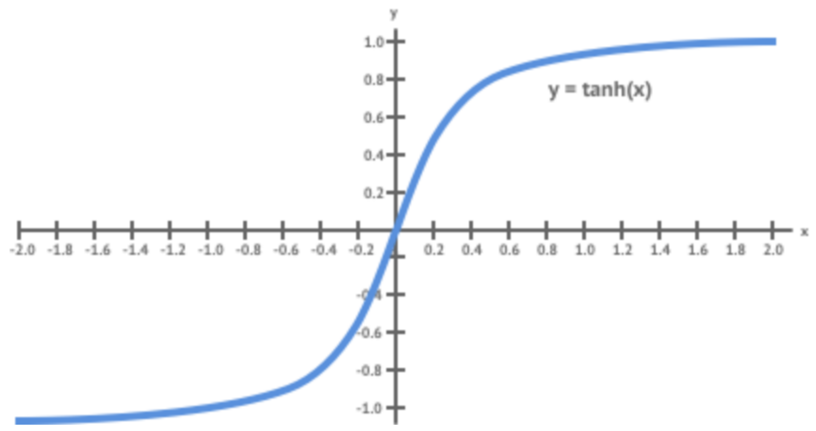


Figure 3.4: Tanh(x)

convolution contains 32 filters with a kernel size of 5. These convolutions allow the model to extract features and learn spatial locality. The feature maps then run through a max pooling of size three by three which takes the max value from the local area. Then we apply another convolution with 64 filters, and a 3 by 3 kernel, pass that into a global average pooling to flatten the output before running through the fully connected layer. Before the final output layer, we have given a dropout to prevent overfitting. The output is given as a binary classifier, and depending on the workout, the classification label is given in the form halfway through the workout cycle, or in starting/ending position. The output activation function used was softmax followed by a categorical cross entropy to calculate the loss. The alternative activation function used was sigmoid when using binary cross entropy to calculate the loss.

$$Softmax(x_{output}) = \frac{exp(x_{output})}{\sum_k exp(x_k)}$$

$$Softmax'(x_{output}) = Softmax(x_{output}) - y_{output}$$

$$Sigmoid(x_{output}) = \frac{1}{1 + x^{-x}}$$

$$Sigmoid'(x_{output}) = Sigmoid(x) \cdot (1 - Sigmoid(x))$$

Figure 3.5 shows the architecture of the initial model used to train over squat images. In the second architecture we simply added additional training epochs, increased from 100 to 150.

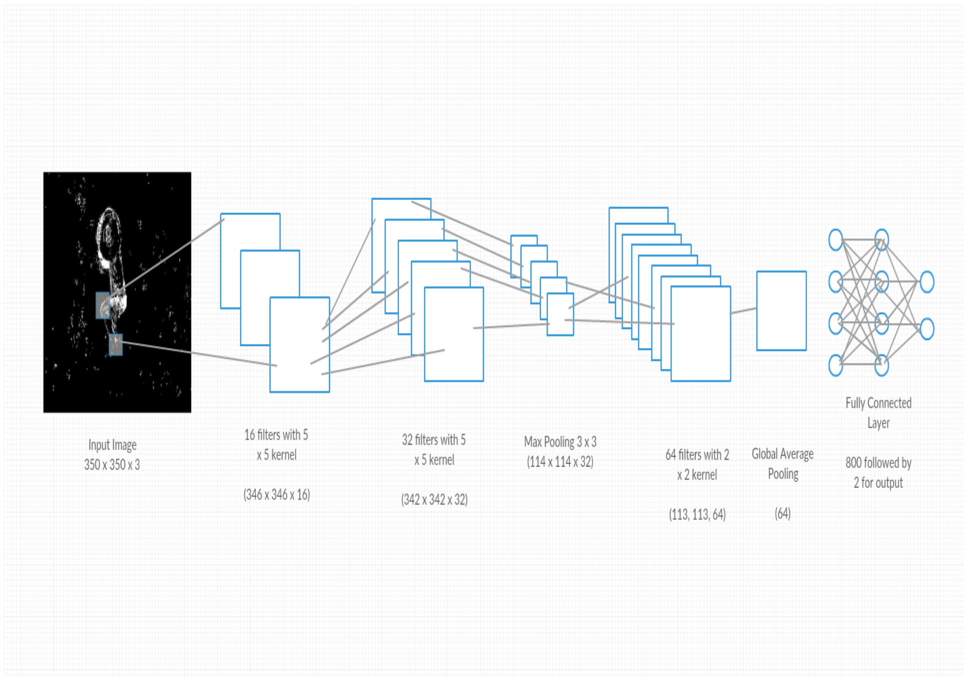


Figure 3.5: Initial network diagram, 8 total layers which include convolutions, pooling, dropout and fully connected layers. Output given and 0 or 1 which is mapped to class label.

The next set of models consisted of a pre-trained model sitting before three convolutions, max-pooling, dropout and fully connected layers. Figure 3.7 shows our model represented as a table of layers, which include the dimensions and parameters at each layer, and the total of trainable parameters at the end. The pre-trained models were VGG and InceptionV3, and the weights were initialized with ImageNet. The models were much deeper, InceptionV3 had 27 total layers in front of our 6 layers and VGG-16 had 16 layers in front of our 6 layers. The reasoning behind adding the pre-trained layer was it provided learned features from the ImageNet dataset. Therefore by training on the squat or pull up dataset, shared features will be easily adjusted within the weight updates. The initial pre-trained model architecture had its training epoch set to 10, and subsequent model epochs were increased to 50-150 allowing for additional training. Both of these models look like the second proposed model architecture, with the pre-trained layers inserted before.

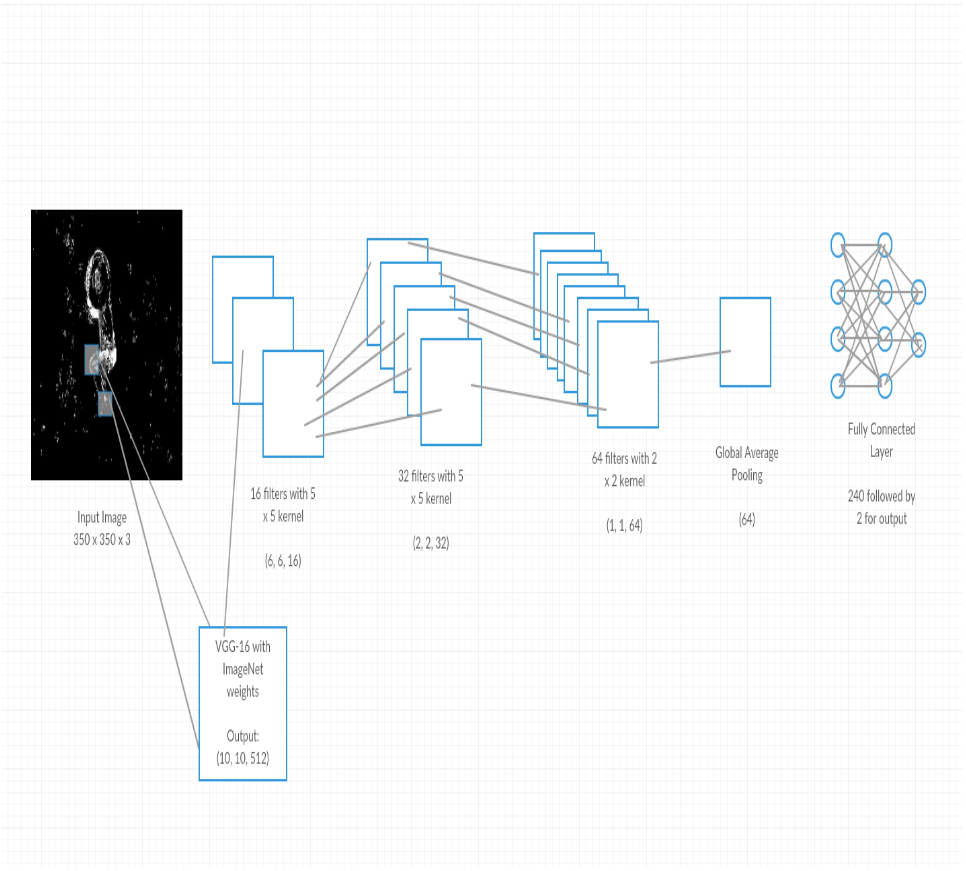

Input Image
350 x 350 x 3
16 filters with 5 x 5 kernel
(6, 6, 16)
32 filters with 5 x 5 kernel
(2, 2, 32)
64 filters with 2 x 2 kernel
(1, 1, 64)
Global Average Pooling
(64)
Fully Connected Layer
240 followed by 2 for output
VGG-16 with ImageNet weights
Output: (10, 10, 512)

Figure 3.6: Pre trained network diagram

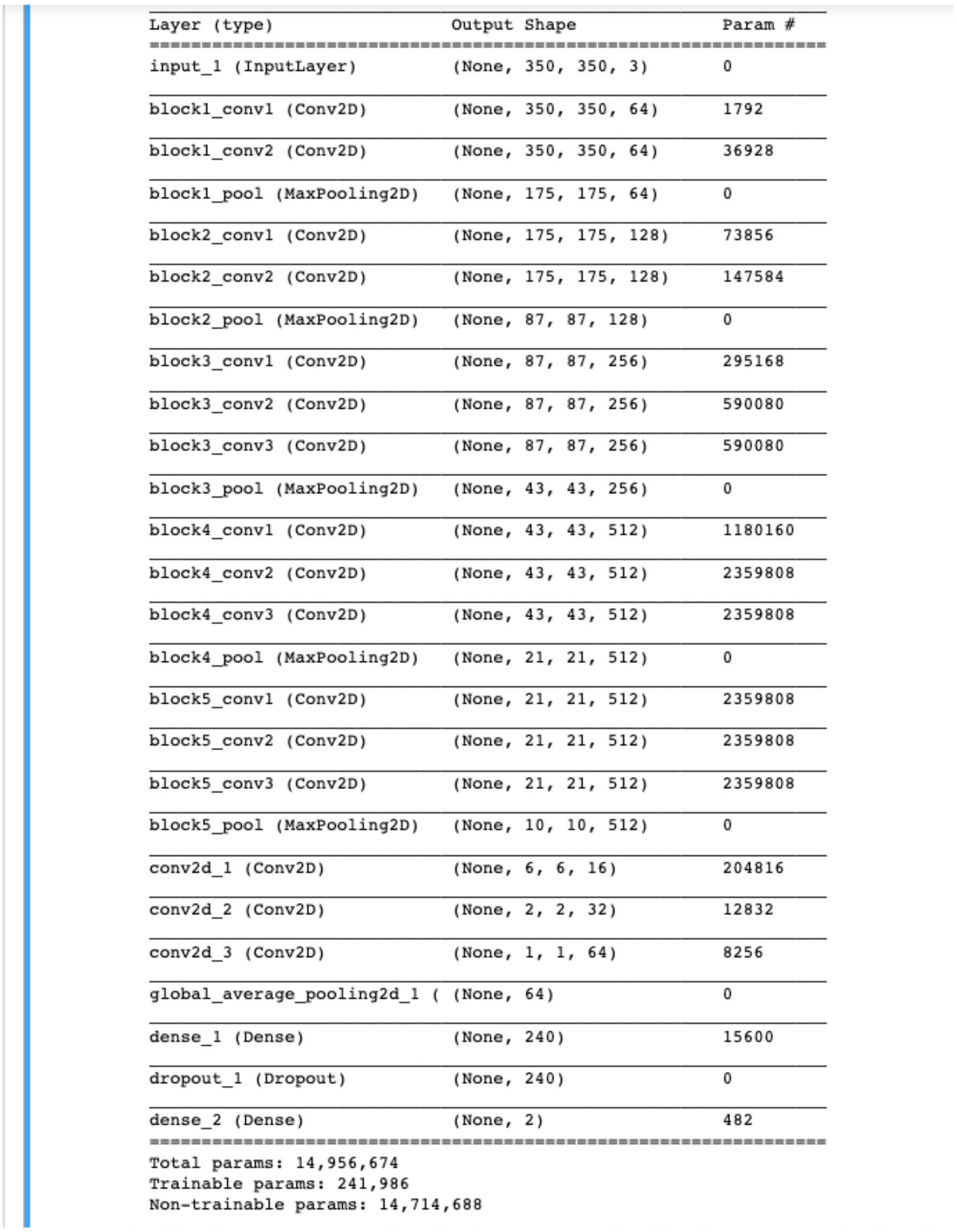

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 350, 350, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 350, 350, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 350, 350, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 175, 175, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 175, 175, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 175, 175, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 87, 87, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 87, 87, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 87, 87, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 87, 87, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 43, 43, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 43, 43, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 43, 43, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 43, 43, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 21, 21, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 21, 21, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 21, 21, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 21, 21, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 10, 10, 512)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 6, 6, 16)          204816
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 2, 2, 32)          12832
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 1, 1, 64)          8256
_________________________________________________________________
global_average_pooling2d_1 ( (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 240)               15600
_________________________________________________________________
dropout_1 (Dropout)          (None, 240)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 482
=================================================================
Total params: 14,956,674
Trainable params: 241,986
Non-trainable params: 14,714,688
```

Figure 3.7: VGG + initial model layers

Finally, the last set of model architectures are similar to the previous pre-trained model, with the slight variation coming from the pre-trained layer. Rather than using the InceptionV3, which is a deeper layered model, we replaced it with the VGG model. This allowed for a quick training run, and the weights were initialized using ImageNet.

Similar to the previous pre-trained design, we either had convolutions, max pooling and a fully connected layer following the pre-trained layer, or just the fully connected layer at the end. With slight changes to the datasets throughout training, we will discuss the experiments and why we updated the datasets in the following chapter.

IV. OpenCV Camera Integration

To apply our model with real-time data we turned towards the OpenCV framework, which has a vast computer vision developer community. OpenCV has a variety of tools for computer vision. OpenCV was initially part of Intel research in 1999, with projects focused on real-time ray tracing and 3D display walls. A few application areas included with OpenCV are:

- Facial Recognition
- Gesture Recognition
- Object Identification
- Segmentation
- Motion Tracking
- Augmented Reality

GPU interfaces used by OpenCV include CUDA and OpenCL. Although OpenCV implements its own set of machine learning statistical models, we stuck with the Keras framework since we have had more experience with its implementation. The tools that were used for this project range from loading and augmenting images, to generating pre-recorded videos to test along with live feed on our trained model. Algorithm 2 shows the pseudocode that generates videos for background subtraction. The code begins by setting the source and destination folders for the image and videos, respectively. The source contains multiple directories, each with a set of frames of the individual performing one cycle of the workout. The destination contains a generated video with the frames contained in the folder, and the video's name is set by taking the source folder's name. The next step is to iterate through each directory within the dataset, in our case the PennAction dataset, and begin by saving each image into an array and sorting it in numerical order. While still inside the loop, the next step is to initialize the video height and width by taking those values from any frame. Finally, we create a new video, iterate through and concatenate each frame, and save the video with the given destination.

Algorithm 2 Video generator

```
procedure GENERATE .AVI
    image_srcdir ← image frames directories
    image_outdir ← video output directory
    for image in image directory do:
        sorted(image, lambda x: images)
        frame ← image[0]
        height, width, layers ← frame.shape
        VideoWriter(image_outdir, width, height)
        for image in imagearray do:
            video.write
        end for
    end for
end procedure
```

The next step, as explained earlier, is to apply background subtraction on the video, extract the frames from it and train the model. Once we had the models trained over the background subtracted frames, algorithm 3 shows the steps for real-time and prerecorded video testing. We begin by loading the model that we have trained on a particular workout. The models were saved in hierarchical data format, and this tool is provided by Keras. Next, we have a flag to specify whether we want to test our model with a pre-recorded video or live feed. Next, depending on the flag that was initially set, we initialize the capture mode. Now that the video type is selected, we begin iterating through each frame of the video. This portion sits inside an infinite while loop, in which we will specify the exit criteria. Once we enter the loop, the first step is to apply the background subtraction if the input is coming from the camera. The frames are then resized to the dimensions provided to the trained model, which is 350 width and 350 height. Next, we append the frame to an array since our input into our trained model was an array, and it would require the additional dimension. Now we take our model and begin predicting the label on each input frame as its getting fed in. If the label matches the exercise position, we increment a counter and output the exercise position. Finally, if the counter reaches 5, we save an image of the users exercise pose and exit out. Otherwise, we run until the end of the video, and if it is a live video, then the user must quit by pressing the escape key on the keyboard.

V. Pose Estimation Feedback

Although pose estimation itself only provides the key points across the individual in the frame, we extended the codebase to calculate the angles given these key points and provide feedback on the workout that is being performed. We simply use the inverse cosine function given the Law of Cosines

$$arccos(c) = \frac{a^2 + b^2 - c^2}{2ab}$$

to calculate angles throughout the workout. For example, if the individual is performing squats, we will make sure the user reaches parallel or the angle between the thigh and calf is at 90 degrees. In order to get the line segments a, b, c we must take the x and y coordinates of the key points provided by the pose estimation model, and apply the

Algorithm 3 Model tester

```
1: procedure TEST MODELS
2:     model.load(path_to_model)
3:     camFlag ← TRUE
4:     if input == webcam then
5:         VideoCapture(0)
6:     else
7:         VideoCapture(path_to_video)
8:         camFlag ← FALSE
9:     end if
10:    while TRUE do
11:        ret, frame ← video.read()
12:        if ret == TRUE then
13:            if camFlag == TRUE then
14:                createBackgroundSubtractorMOG2.apply(frame)
15:            end if
16:            resizeFrame(350,350)
17:            model.predict(inputFrameArray)
18:            if prediction == 0 then
19:                exerciseForm (Depends on workout)
20:                if exerciseCount == 5 then
21:                    write(exerciseFrame)
22:                end if
23:            end if
24:        end if
25:    end while
26: end procedure
```

following functions in order to get the respective segments.

$$a = \sqrt{(point1_x - point3_x)^2 + (point1_y - point3_y)^2}$$

$$b = \sqrt{(point1_x - point2_x)^2 + (point1_y - point2_y)^2}$$

$$c = \sqrt{(point2_x - point3_x)^2 + (point2_y - point3_y)^2}$$

Once we have these segments calculated, we can pass them into the inverse cosine function to get the angle needed for the workout. We have added a helper function that maps each key point to a readable value, which in our case was the body part that the key point was associated with. For example, if we wanted to get the values of the joints located between the left wrist and shoulder, we would pass in the body parts "left wrist, left elbow, left shoulder" as values into our "getBodyPartPoints" function.

The pseudocode in algorithm 4 highlights the pose estimation implementation for counting the number of correct exercises, in our case either squat or pull-ups. The first step is to take in the video, model, and workout as arguments and initializing them for use later. We load the model provided through the argument, in our testing, we stuck with "MobileNet_Thin" which was trained on images of size 432 by 368 pixels. Afterward, we begin to iterate through an infinite for loop, performing inference with our model on each frame provided. The inference is taken and now the key points values are given. We take those key points and draw out the skeletal structure of each individual present. Now that we have the key points available, we can use them to map our the body parts, such as the wrist, elbow, ankle. These mappings are given as a coordinate on a 2-dimensional plane. This brings us to the workout analysis portion, where we calculate the angles between specific joints related to the exercise performed. The workout should be provided as an argument, and the thresholds vary depending on the workout. For pull-ups, we calculate the angle from the elbow joint, making sure it is initially between 180 and 150, followed by when the individual pulls themselves up to an angle between 90 and 50, and finally back down to an angle in between 180 and 150. This sequence provides us with one cycle of the workout, which we count the total number of cycles or repetitions, and output it back to the user. If the input is live feed, we would require an exit key, otherwise, the while loop will exit once the video ends.

Algorithm 4 Pose estimation calculator

```
1: procedure GET POSE AND CALCULATE ANGLES
2:     video ← args.video
3:     model ← args.model
4:     workout ← args.workout
5:     TfPoseEstimator(get_graph_path(model))
6:     while TRUE do
7:         inference(frame)
8:         draw_humans(frame, inference)
9:         bodypartpoints ← getBodyPartPoints(frame, inference)
10:        if workout == "pullups" then
11:            angle ← calculate_angles_arccos(bodyparts)
12:            if angle $\leq$ 180 AND angle $\geq$ 150 AND cycle == 0 then
13:                $cycle++$
14:            else if angle $\leq$ 90 AND angle $\geq$ 50 AND cycle == 1 then
15:                $workoutCounter++$
16:                $cycle--$
17:            end if
18:        else if workout == "squats" then
19:            angle ← calculate_angles_arccos(bodyparts)
20:            if angle $\leq$ 180 AND angle $\geq$ 170 AND cycle == 0 then
21:                $cycle++$
22:            else if angle $\leq$ 90 AND angle $\geq$ 60 AND cycle == 1 then
23:                $workoutCounter++$
24:                $cycle--$
25:            end if
26:        end if
27:    end while
28:    print $workoutCounter$
29: end procedure
```

Chapter 4: Experiments and Results

During the experimentation phase, we broke down the process into a few steps which were repeated over two different types of workouts. Initially, we began with squats, and once we had a working end to end model, we added pull-ups. Our goal was to choose the Silhouette model with the highest accuracy. Finally, we want to compare the Silhouette model approach to the pose estimation model approach, and decide which approach to use as our foundation for real-time exercise feedback. Here is the list of steps taken to test our feedback model:

- Collect, download, and organize datasets with individuals performing specified workouts
- Combine frames into videos
- Apply background subtraction on the videos
- Extract frames from the background subtracted videos
- Separate and label frames into categories(either using thirds or manual selection)
- Resize all frames to 350 by 350
- Pass frames into our neural network and begins training the model
- Monitor accuracy and error rates to make changes to the given process

- Choose model with top results
- Apply model to OpenCV Video Capture to test videos and real-time data
- Repeat with other exercises
- Compare with pose estimation approach

The first step in our experimentation process was to collect, download and organize the training data we will use for our neural network models. Although this task is trivial since we just had to download and save the datasets, organizing it required a bit of code to parse the directories, find the exercise labels which were given in .mat files, and move them into a separate directory. This way we do not have to repeatedly parse and locate the datasets, and if we make any alterations we would still have a backup.

The next step was to combine the frames into videos, which initially generated the videos with the frames out of order. This was due to the directory parser which had a non-numerical ordering. This issue was resolved by simply adding a sort method which ordered the images numerically and subsequently joined in the correct order and saved as an audio video interleave. Diagram 4.1 shows the process behind generating a single squat video.

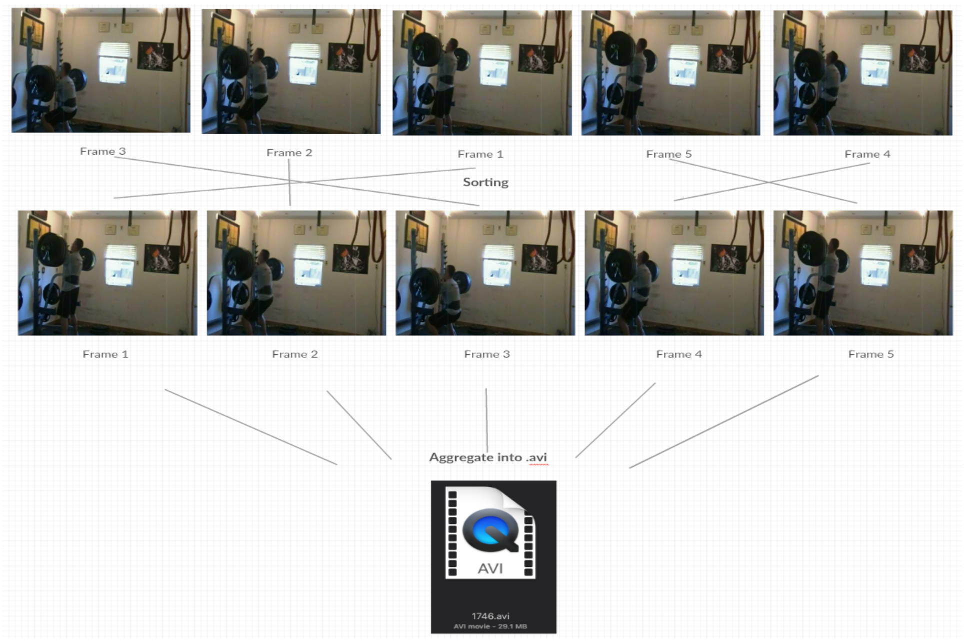


Figure 4.1: Squat video generated

During background subtraction, we were able to successfully generate the black and white silhouette, however, we did run into an issue where the first frame was filled with background pixels. This seems to be caused by the algorithm, therefore we went ahead and discarded that image, which ultimately reduced our training set, however it was a necessary step. Once the background subtraction was completed, the frame extraction is straight forward, and the set of frames are stored in directories with the original video name followed by the frame number.

Now all that's left before the training process is resizing and separating the data based on the given class labels. The labels for squats were standing(neutral) and squatting, and for pull-ups, it was either pulled up or non-pulled up(neutral). Although our labeling was straightforward, we still needed to tag the image appropriately. The original plan was to split the data set into thirds:

- First third split of standing or neutral position

- Next third split into full flexion or midway through a repetition
- Last third of the split would contain the neutral position once again

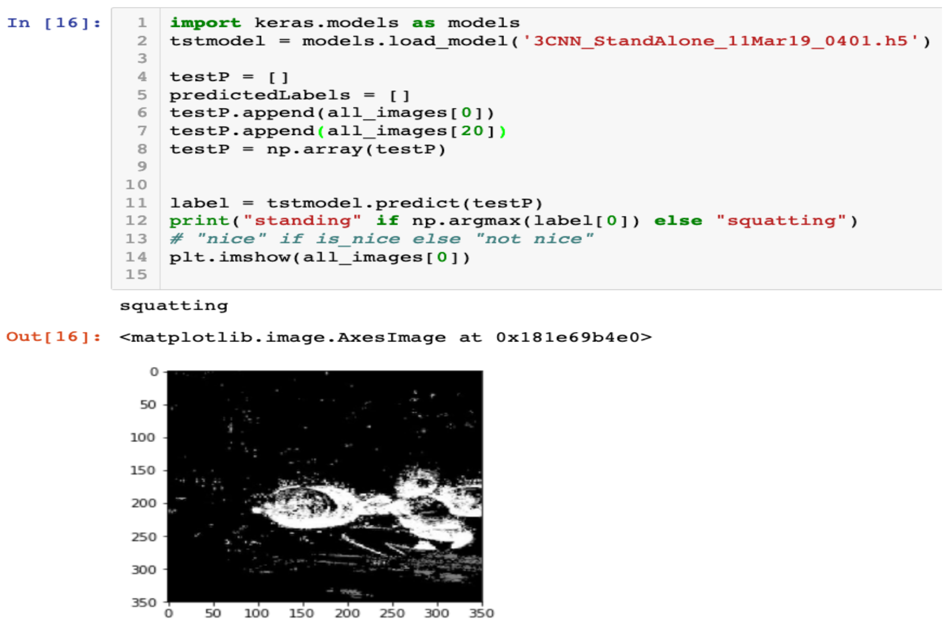
Ultimately this would contain a full cycle of one repetition of a given workout. However, when looking at the split data, there were mixed images, those that belonged to the neutral position directory found in flexed position directory and vice versa. Therefore the splitting process was completed manually, where I personally went through the directory and separating images based on what the position looked like to me. This was a very strenuous process, since manually looking at over 1000 black and white Silhouettes and organizing them strained my eyes. The last step was much easier, just rescale the image to 350 by 350.

I. Convolutional neural network 1

Our training experimentation starts with the three convolutional layered models, and the exercise being trained was squats. We had 1090 frames where the individual is standing, and 869 frames with the individual in squatting position. The hyper-parameters set for this model are training and validation split was set to 80% and 20% respectively, total epochs or iterations at 100, and a batch size set to 64 to prevent overfitting.

The total time it took to run through 100 epochs was 34,420 seconds, or roughly $9\frac{1}{2}$ hours. The results from this model showed promise, with an accuracy topping off at 49%. Figure 4.3 illustrates the training steps, plotting the loss and accuracy value taken at each step. The loss curve on the graph seems to show improvement, up until around 60 epochs where it starts to flatten. For accuracy, it improves the most between 20 and 40 epochs and seems to flatline around 80 epochs. The minimum loss recorded was 1.8 which occurred on the 99th epoch. The max accuracy occurred during the 92nd epoch.

In the test set, we were able to achieve 31%. This test was on images the model had not seen at all. An example prediction can be found in Figure 4.2



```
In [16]: import keras.models as models
         tstmodel = models.load_model('3CNN_StandAlone_11Mar19_0401.h5')

         testP = []
         predictedLabels = []
         testP.append(all_images[0])
         testP.append(all_images[20])
         testP = np.array(testP)


         label = tstmodel.predict(testP)
         print("standing" if np.argmax(label[0]) else "squatting")
         # "nice" if is_nice else "not nice"
         plt.imshow(all_images[0])

         squatting
Out[16]: <matplotlib.image.AxesImage at 0x181e69b4e0>
```

Figure 4.2: Squat image test

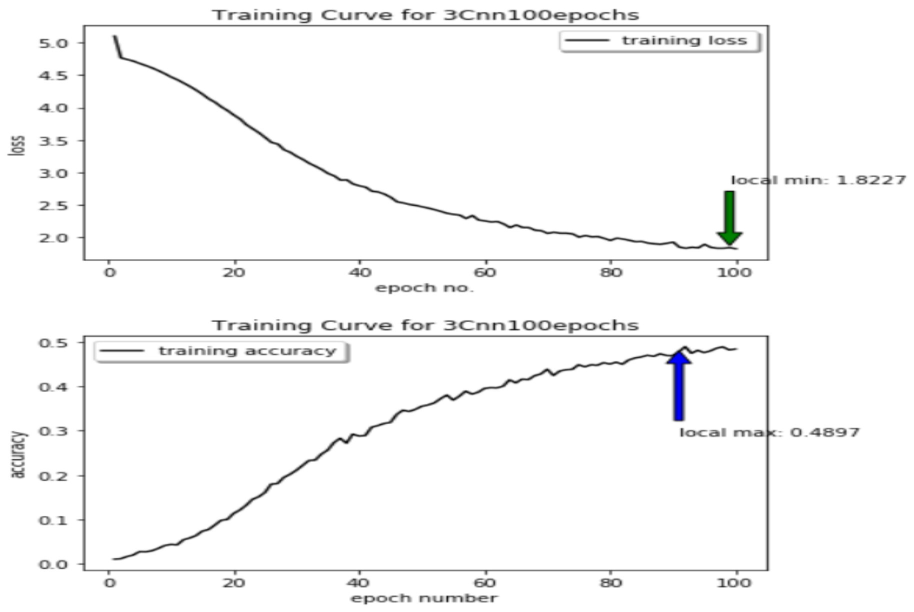


Figure 4.3: Initial training model result

II. Convolutional neural network 2

This model had a few alterations to the hyper-parameters. Updates made to the hyper-parameters include increasing the number of epochs to 150 and reducing the batch size to 16 to allow the model to learn more about the features. With this model, we were able to achieve a lower loss value of .2 and an increase in the accuracy to 91% as seen in Figure 4.4.

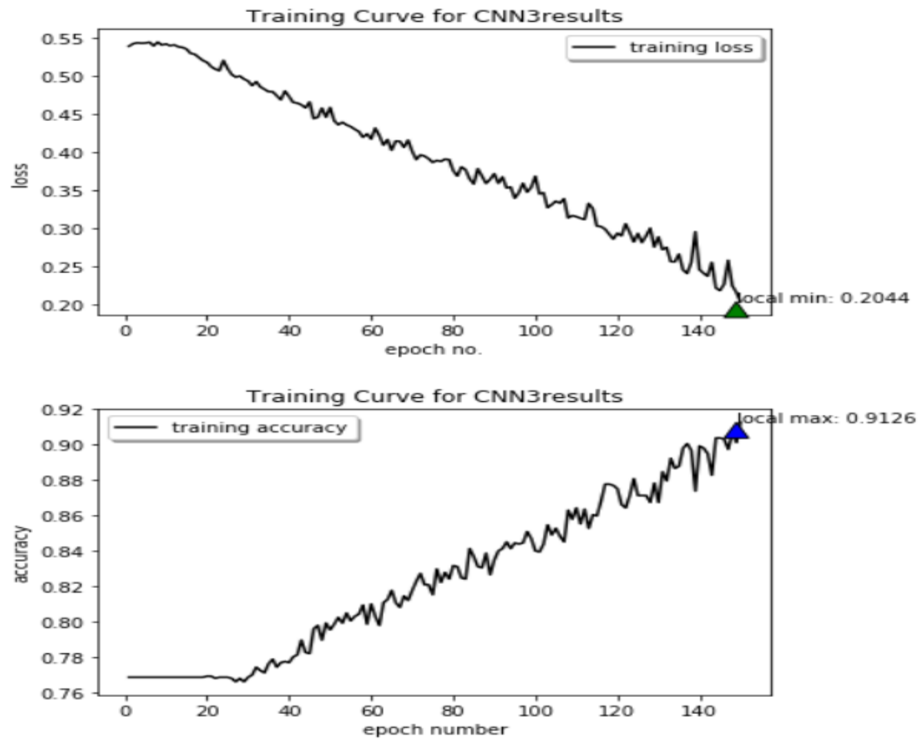


Figure 4.4: 150 epoch training model result

Although the accuracy on the training set was high, when we ran our model on the validation set, we had achieved an accuracy of 79%, and on the test set, we achieved an accuracy of 42%. The loss and accuracy curves on the validation set shown in Figure 4.5, do not look as logarithmic, which is fine since the weights are not updating when we feed in those images. The key point from the validation test is that we were able to achieve higher accuracy than our previous model architecture by making slight adjustments to the hyper-parameters. The tradeoff is the overall run time increased to 56,821 seconds or about 15 hours.

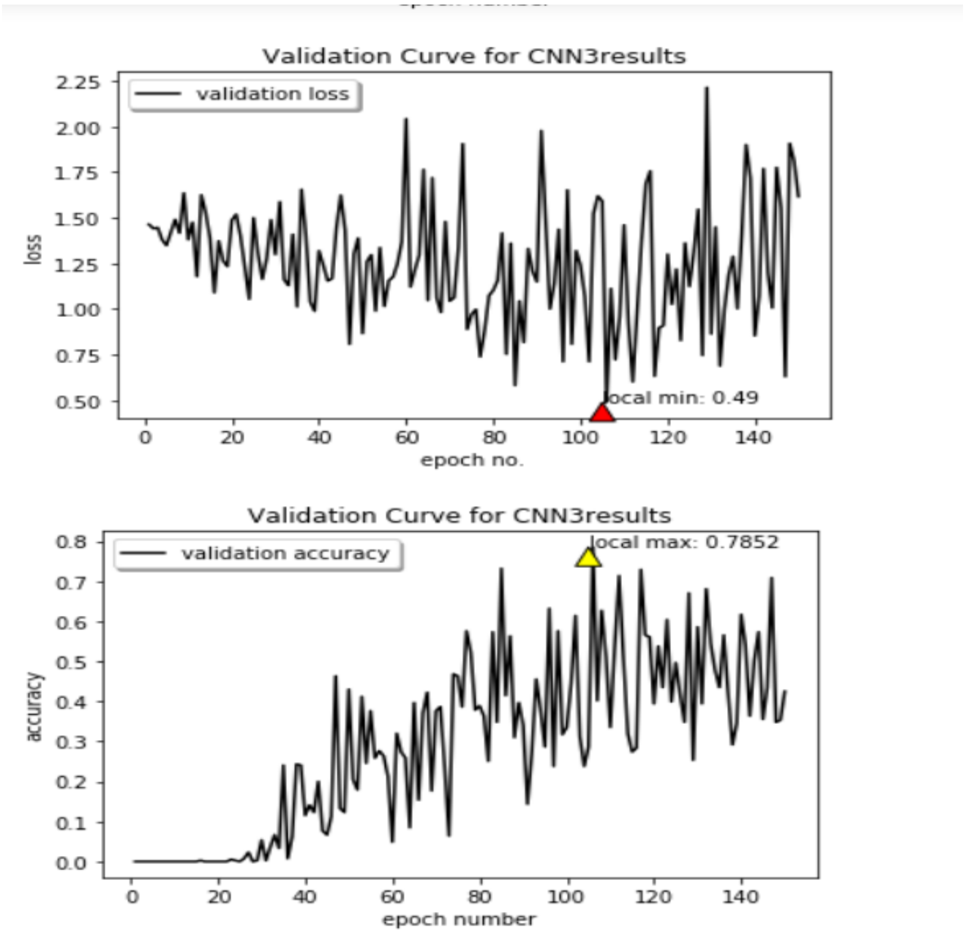


Figure 4.5: 150 epoch validation model result

III. Pre Trained InceptionV3 Convolutional neural network 1

One of the biggest turning points in our project is credited to using a pre-trained network as the initial layer to our network. We tested two different sets of pre-trained networks, the first was InceptionV3. InceptionV3 is a very deep layered network, with over 14 million parameters. This would take too much time to train, therefore we made the InceptionV3 layers non-trainable. We set the epoch to 10 iterations with this architecture setup. The curve in Figure 4.6 which contained the validation split showed improvement, although at the 10th epoch it capped at 30%.

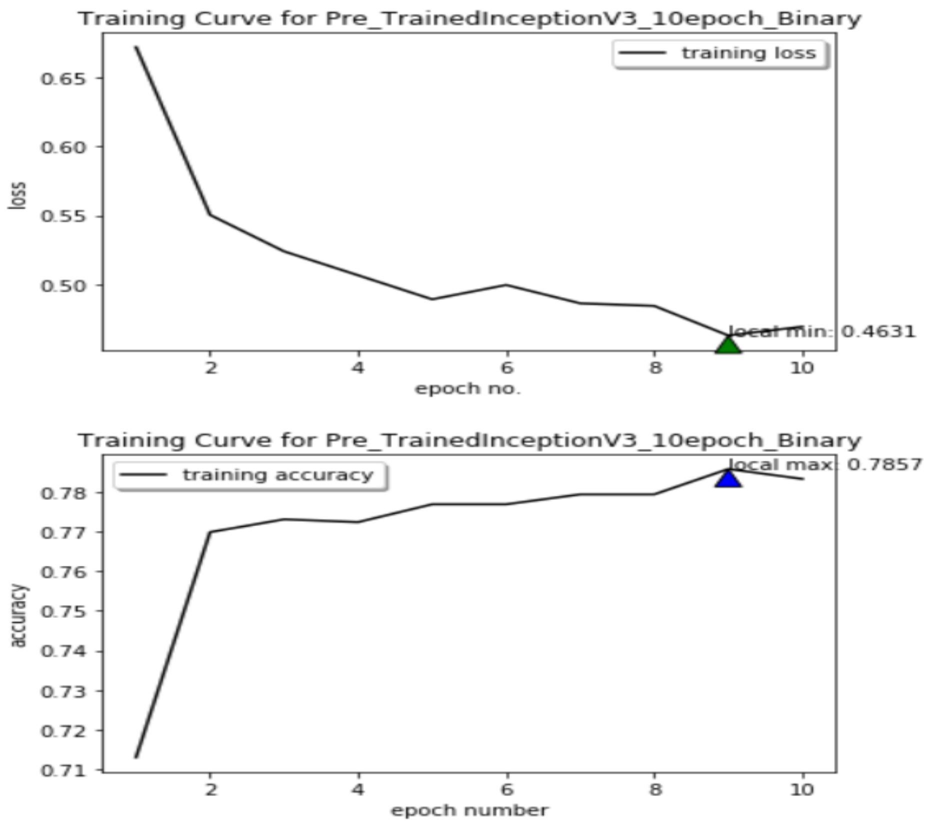
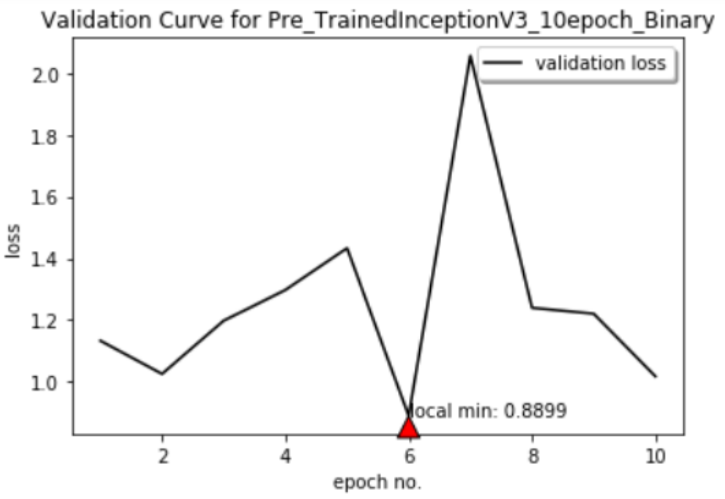


Figure 4.6: 10 epoch pre trained result

Figure 4.7 shows the curve which contained the training set reaching a max of 78% and a low loss score of .46. This was at only 10 epoch which showed great promise, therefore in our next test, we increased the number of epochs to see if we could achieve better results.



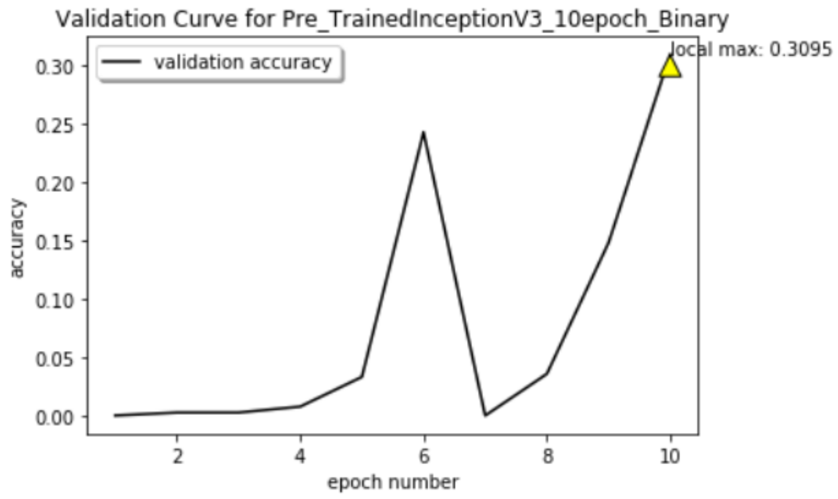


Figure 4.7: 10 epoch pre trained validation result

Figure 4.6 shows the accuracy score achieved on test data that the model had not seen before. The only problem we noticed was this model did horribly on the test set. It got all test images incorrect. We decided we needed to increase the epochs, however, we were still able to see progress with only 10 epochs, and total training time of a little over 1 hour.

IV. Pre Trained InceptionV3 Convolutional neural network 2

This model had much better results than the previous pre-trained model. The only changes we made to this architecture was increasing the epoch iteration to 50, the batch size remained 16 and we kept the split 80 training 20 validation. Starting with the training set, Figure 4.8 shows the training loss and accuracy curves. The loss score achieved was .38 with an 80% accuracy score. This was an improvement over the previous model.

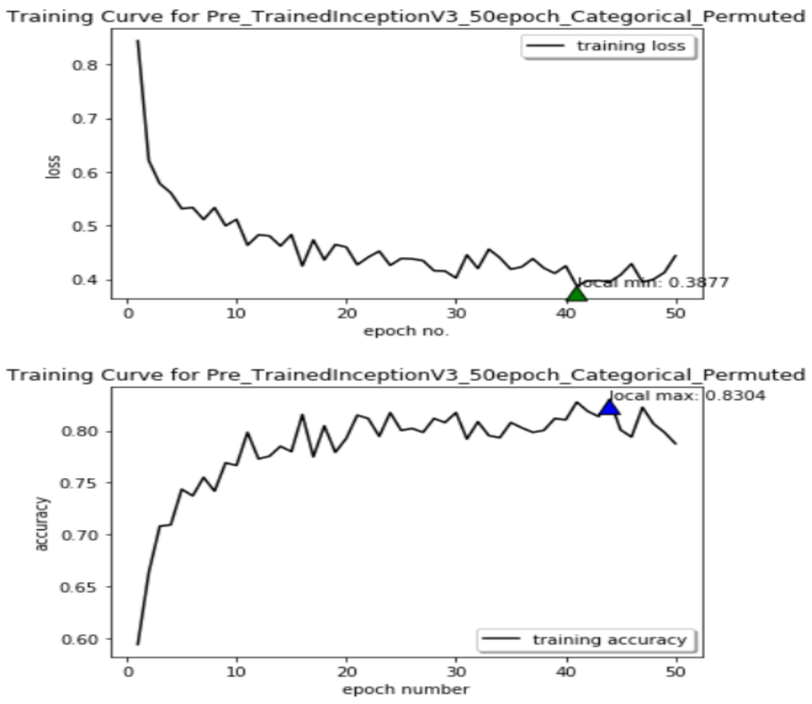
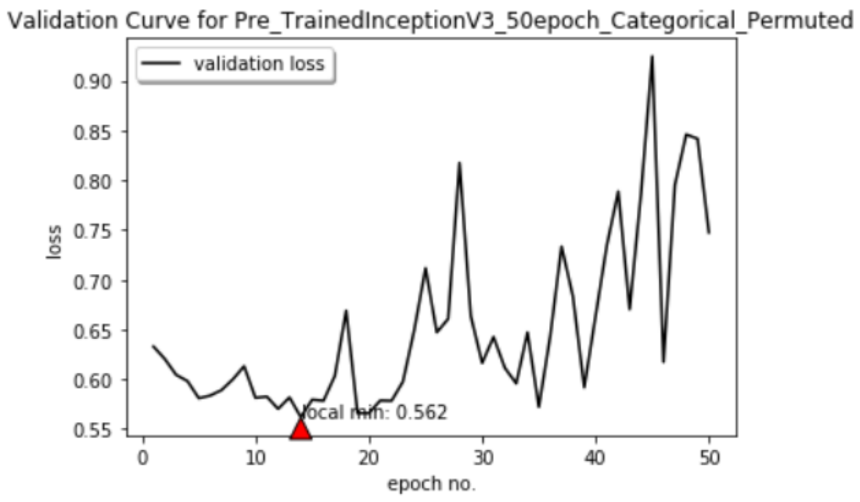


Figure 4.8: 50 epoch pre trained result

The validation curves are shown in Figure 4.9 seem to be increasing in loss, as the minimum loss was achieved around 14 epochs. The accuracy on the validation fluctuated a lot, and around the 39th epoch, we can see the accuracy peak at 73%.



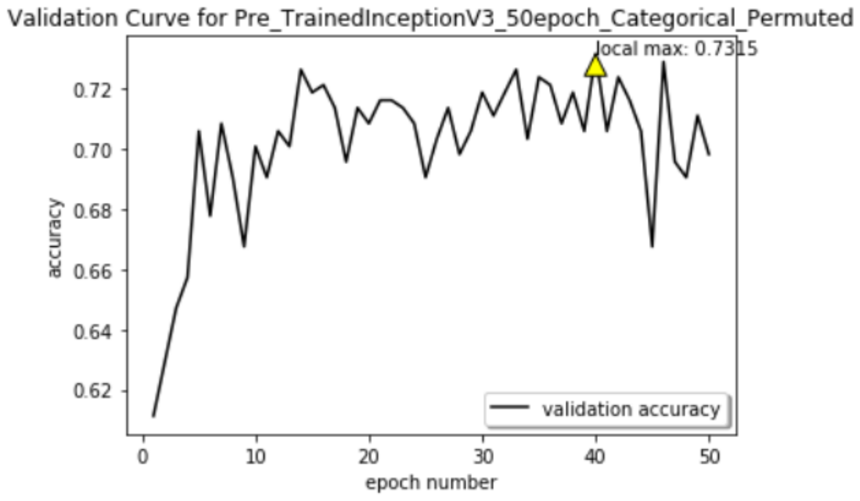
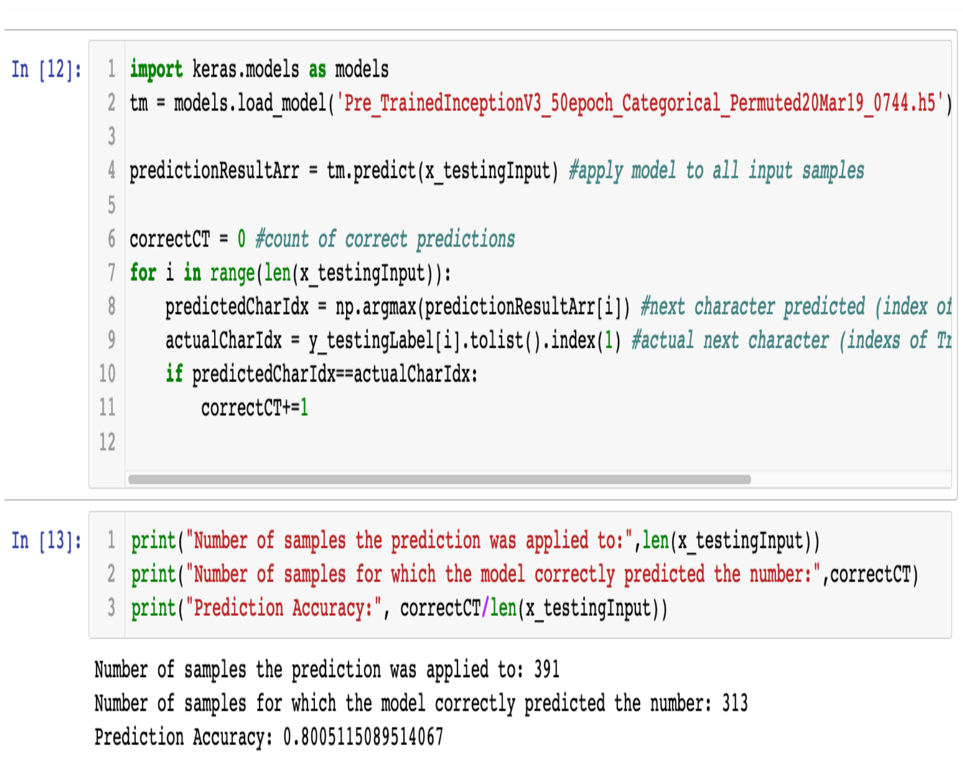


Figure 4.9: 50 epoch pre trained result

As for the test set, we were able to achieve a score of 80% as seen in Figure 4.10. At this point, this was the highest score achieved by any of our models' given data which it had not seen before. The only cost of this model was the training time which was 612, still less than the 150 epoch without the pre-trained layers.



```
In [12]:
import keras.models as models
tm = models.load_model('Pre_TrainedInceptionV3_50epoch_Categorical_Permuted20Mar19_0744.h5')

predictionResultArr = tm.predict(x_testingInput) #apply model to all input samples

correctCT = 0 #count of correct predictions
for i in range(len(x_testingInput)):
    predictedCharIdx = np.argmax(predictionResultArr[i]) #next character predicted (index of
    actualCharIdx = y_testingLabel[i].tolist().index(1) #actual next character (indexs of Tr
    if predictedCharIdx==actualCharIdx:
        correctCT+=1
```

```
In [13]:
print("Number of samples the prediction was applied to:",len(x_testingInput))
print("Number of samples for which the model correctly predicted the number:",correctCT)
print("Prediction Accuracy:", correctCT/len(x_testingInput))
```

```
Number of samples the prediction was applied to: 391
Number of samples for which the model correctly predicted the number: 313
Prediction Accuracy: 0.8005115089514067
```

Figure 4.10: Test Set Result

V. Pre trained VGG convolutional neural network

This was the final set of models trained and tested. We tried multiple variations changing the epoch, batch size, permuting images, and manually fixing labels. We took the best one of those variations of the VGG pre-trained model and show the results for it. We were able to achieve 97% accuracy on our training set and loss score of .1. The curves are shown in Figure 4.11 also look logarithmic and reaching a flat line. It’s possible that we had overtrained it on our training set, but our validation and test results say otherwise.

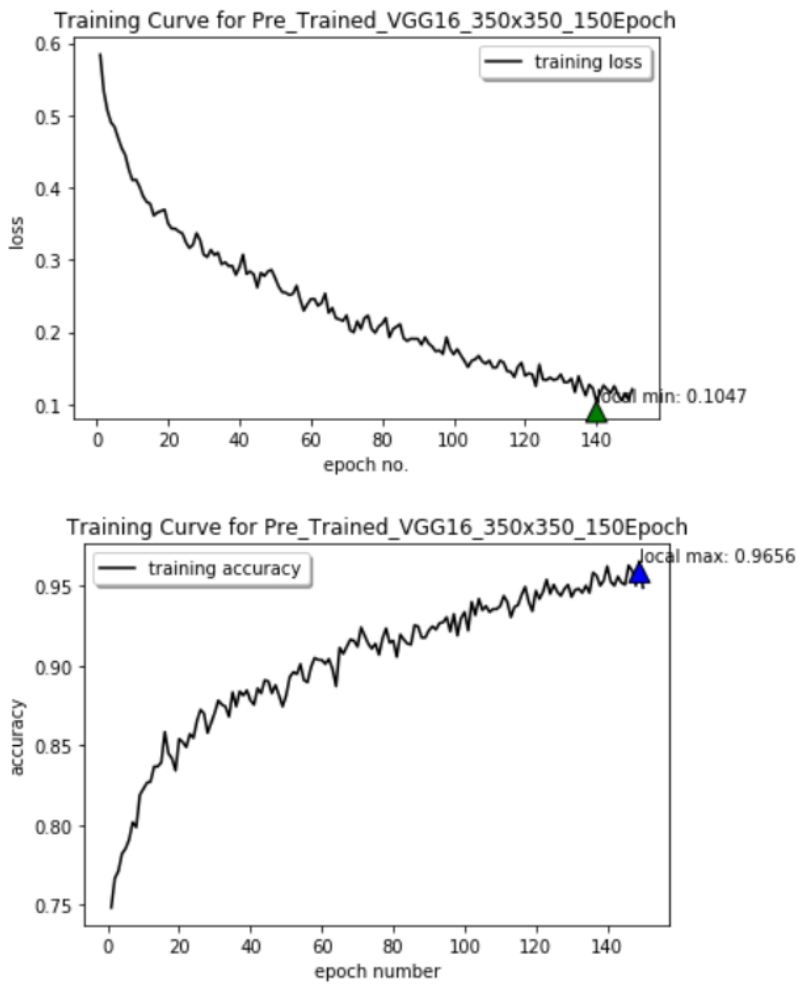
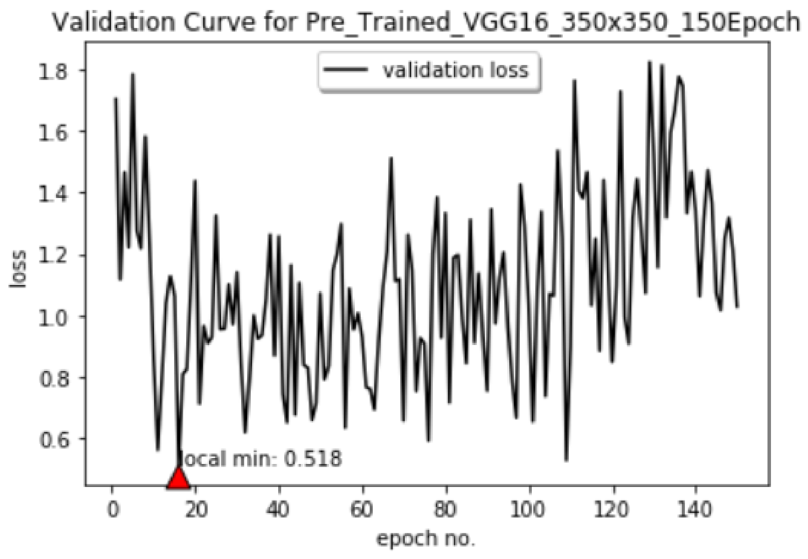


Figure 4.11: Training result VGG

The validation loss and accuracy curves are a bit odd, however, the scores achieved are the best out of all models we have trained over. With accuracy on validation set reaching 79% and loss score of .52, we had one more test left.



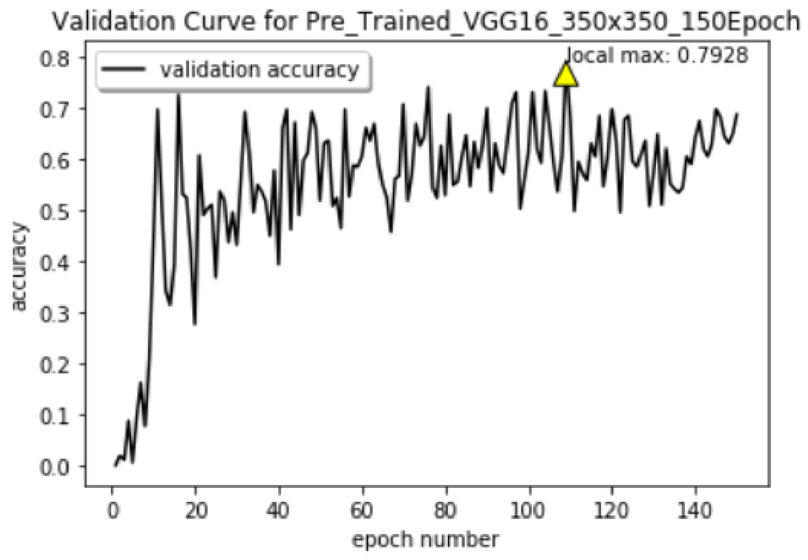
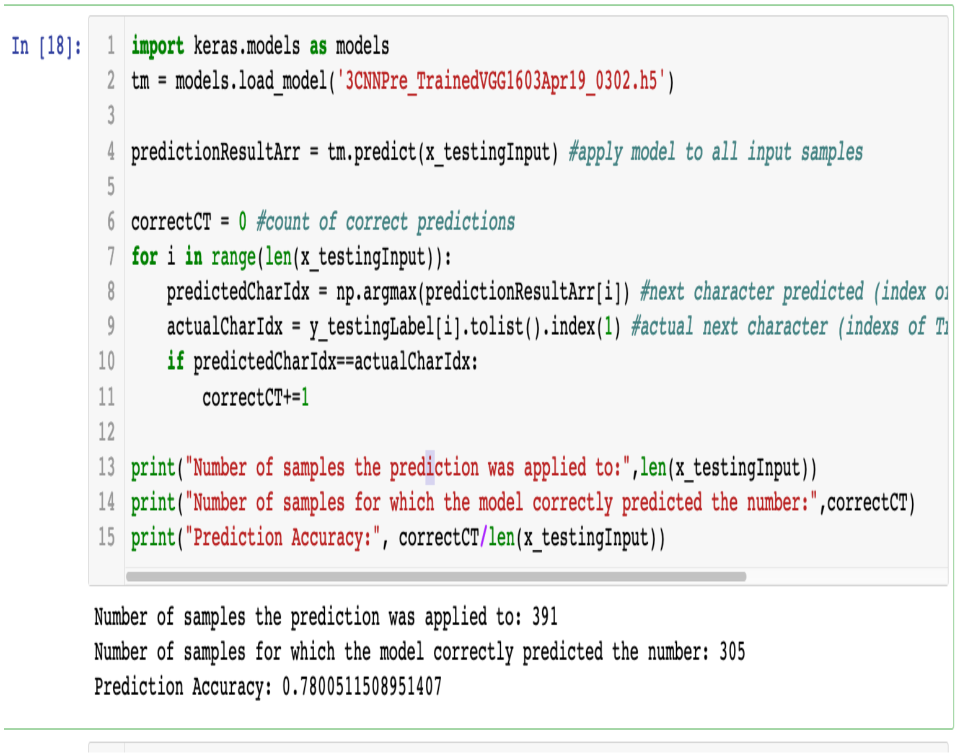


Figure 4.12: Validation result VGG

In the final test, we were able to achieve 78% accuracy shown in Figure 4.13, which although it was lower than the InceptionV3 model, when we tested the two models with our live feed webcam, we noticed much better predictions made by the VGG model.



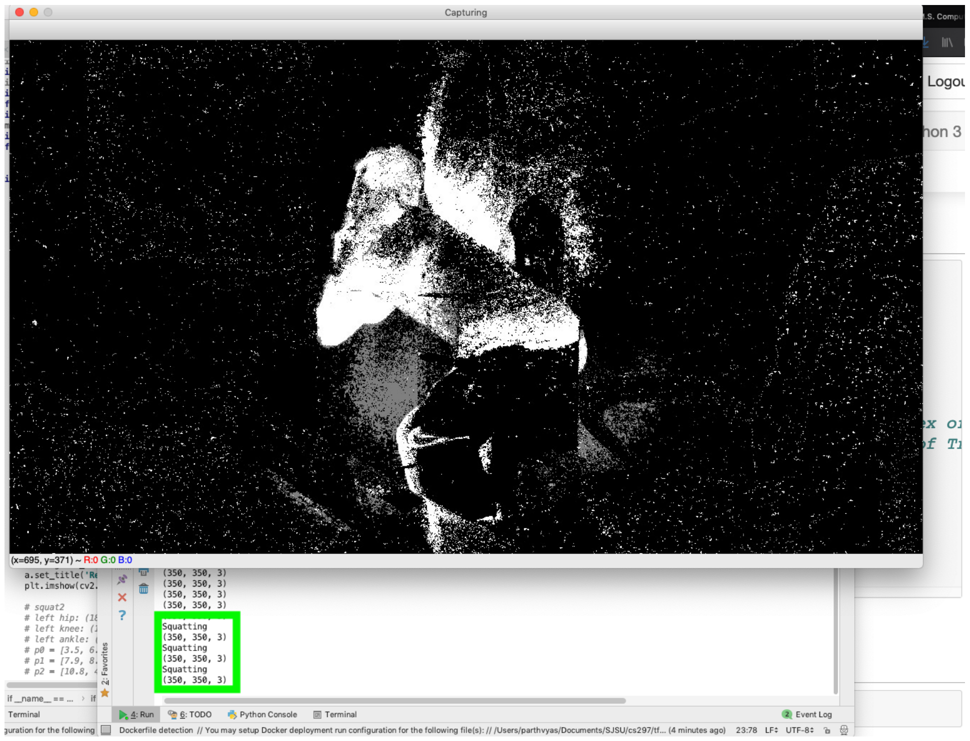
```
In [18]: import keras.models as models
         tm = models.load_model('3CNNPre_TrainedVGG1603Apr19_0302.h5')

         predictionResultArr = tm.predict(x_testingInput) #apply model to all input samples

         correctCT = 0 #count of correct predictions
         for i in range(len(x_testingInput)):
             predictedCharIdx = np.argmax(predictionResultArr[i]) #next character predicted (index o
             actualCharIdx = y_testingLabel[i].tolist().index(1) #actual next character (indexs of T
             if predictedCharIdx==actualCharIdx:
                 correctCT+=1

         print("Number of samples the prediction was applied to:",len(x_testingInput))
         print("Number of samples for which the model correctly predicted the number:",correctCT)
         print("Prediction Accuracy:", correctCT/len(x_testingInput))
```

```
Number of samples the prediction was applied to: 391
Number of samples for which the model correctly predicted the number: 305
Prediction Accuracy: 0.7800511508951407
```

Figure 4.13: VGG test set result

VI. Pose estimation vs. Silhouette based action recognition

Once we had our final working silhouette action recognition model, we applied the test to both squats and pull-ups using pre-recorded videos. I personally tested the squat model with the live webcam feed. The model was able to identify when the individual performing the exercise was in the labeled form, however, there were still some false positives, where it would incorrectly identify neutral position when in the flexed position. Figure 4.14 and 4.15 shows our results when run with webcam and pre-recorded videos.

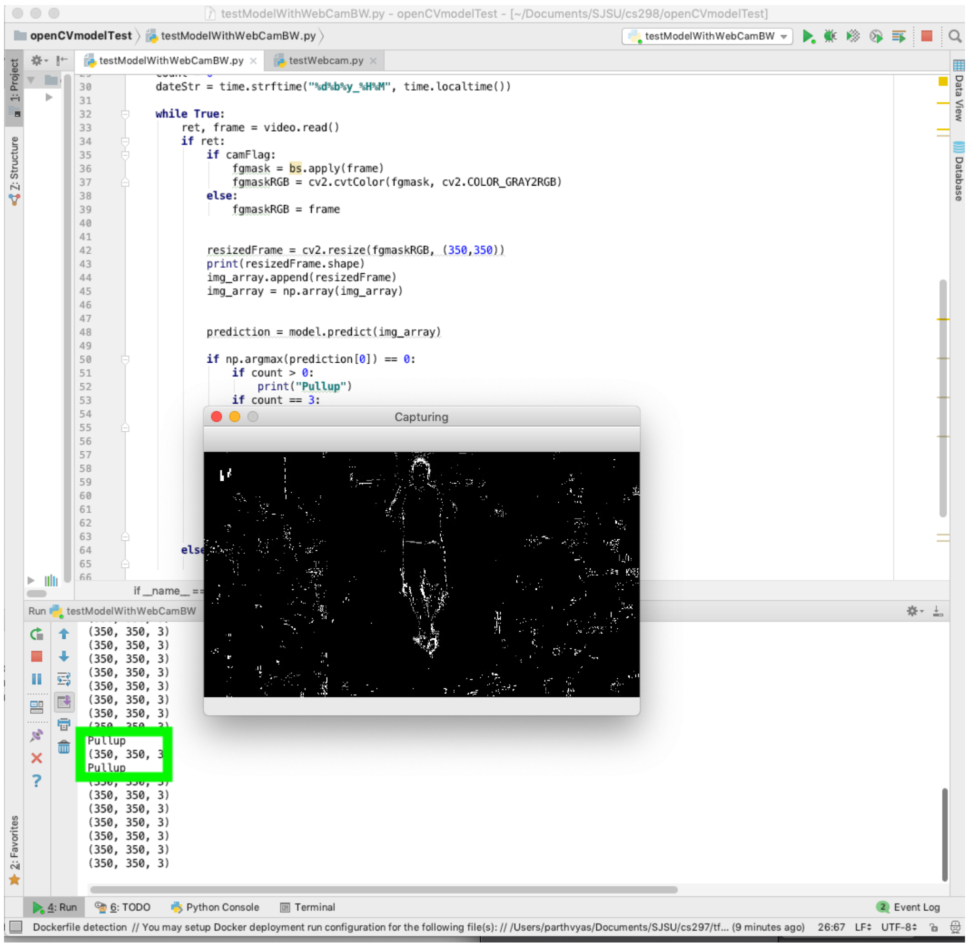


```
a.set_title('Re
plt.imshow(cv2.

# squat2
# left hip: (18
# left knee: (1
# left ankle: (
# p0 = [3.5, 6.
# p1 = [7.9, 8.
# p2 = [10.8, 4
```

```
(350, 350, 3)
(350, 350, 3)
(350, 350, 3)
(350, 350, 3)
Squatting
(350, 350, 3)
Squatting
(350, 350, 3)
Squatting
(350, 350, 3)
```

```
guration for the following   Dockerfile detection // You may setup Docker deployment run configuration for the following file(s): // /Users/parthvyas/Documents/SJSU/cs297/tf... (4 minutes ago)   23:78   LF÷   UTF-8÷
```

Figure 4.14: Final integration with silhouette action model


testModelWithWebCamBW.py - openCVmodelTest - [~/Documents/SJSU/cs298/openCVmodelTest]
openCVmodelTest
testModelWithWebCamBW.py
testWebcam.py
Capturing
Run testModelWithWebCamBW
4: Run
6: TODO
Python Console
Terminal
Event Log

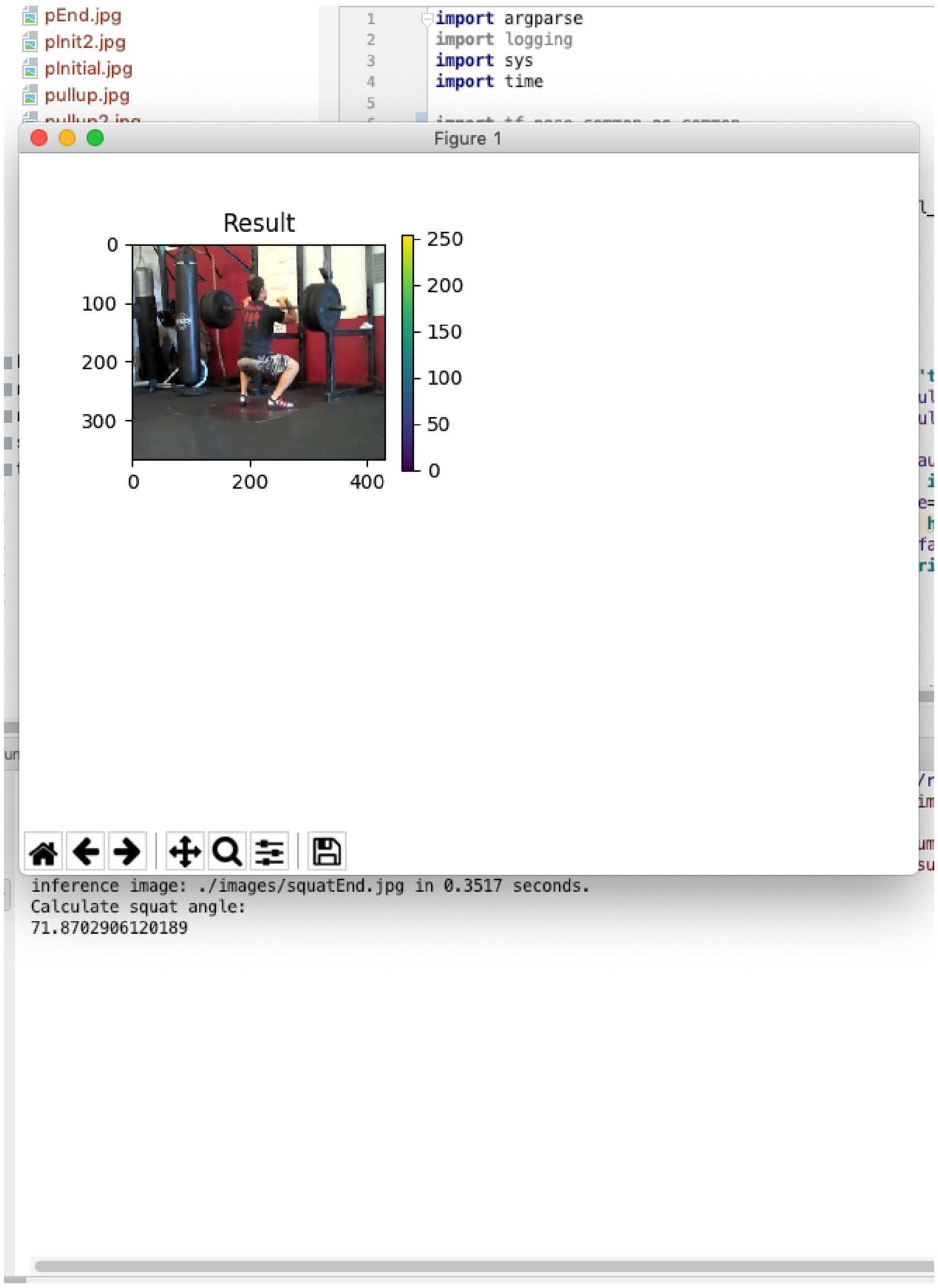
```
    dateStr = time.strftime("%d%b%y_%H%M", time.localtime())

    while True:
        ret, frame = video.read()
        if ret:
            if camFlag:
                fgmask = bs.apply(frame)
                fgmaskRGB = cv2.cvtColor(fgmask, cv2.COLOR_GRAY2RGB)
            else:
                fgmaskRGB = frame


            resizedFrame = cv2.resize(fgmaskRGB, (350,350))
            print(resizedFrame.shape)
            img_array.append(resizedFrame)
            img_array = np.array(img_array)


            prediction = model.predict(img_array)

            if np.argmax(prediction[0]) == 0:
                if count > 0:
                    print("Pullup")
                if count == 3:
```

```
(350, 350, 3)
(350, 350, 3)
(350, 350, 3)
(350, 350, 3)
(350, 350, 3)
(350, 350, 3)
(350, 350, 3)
Pullup
(350, 350, 3
Pullup
(350, 350, 3)
(350, 350, 3)
(350, 350, 3)
(350, 350, 3)
(350, 350, 3)
(350, 350, 3)
```

```
Dockerfile detection // You may setup Docker deployment run configuration for the following file(s): // /Users/parthvyas/Documents/SJSU/cs297/tf... (9 minutes ago)   26:67   LF   UTF-8
```

Figure 4.15: Final integration with silhouette action model

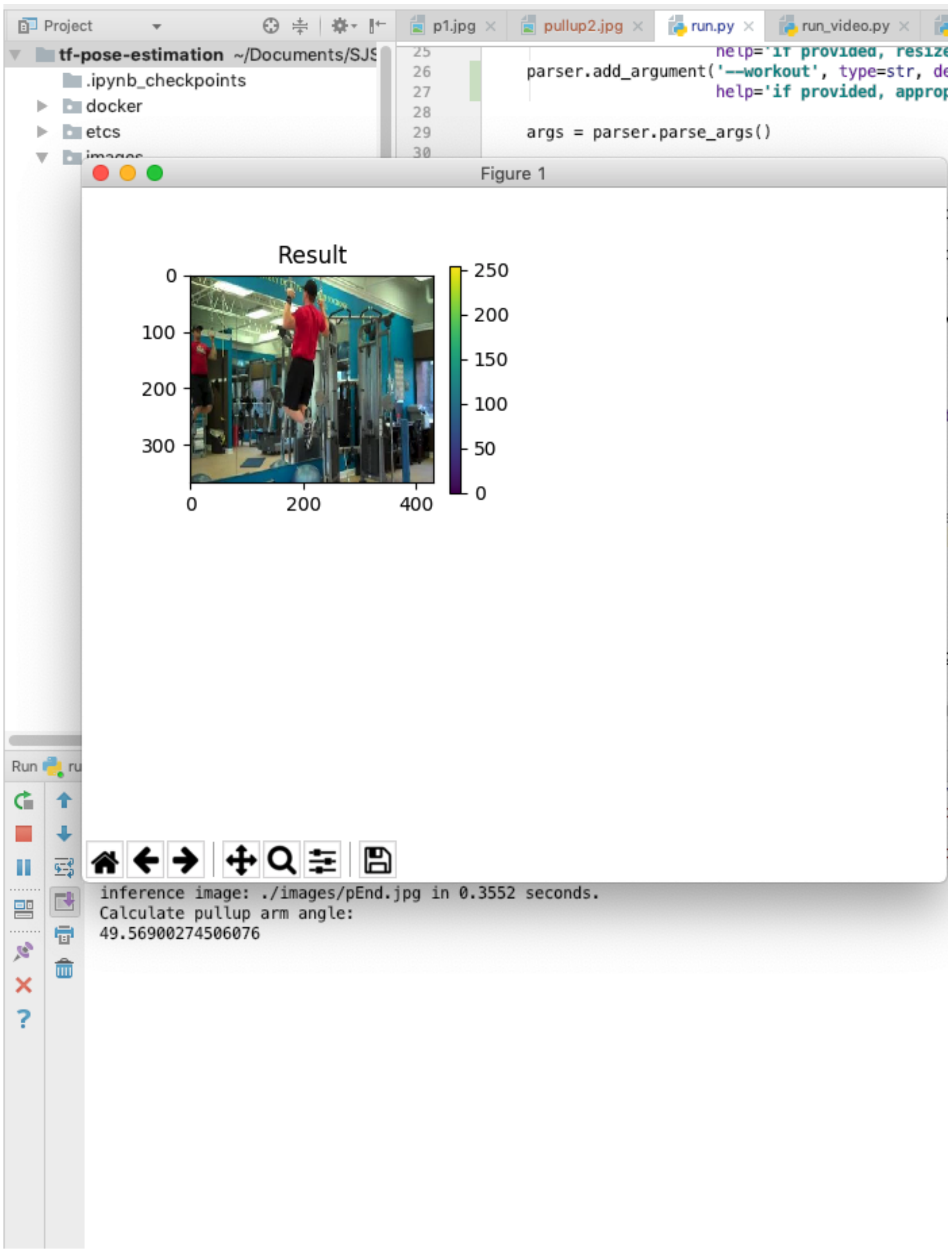
The pose estimation setup we have implemented had great results as well, giving us the angle of the positions the individual is in throughout the entire workout. This gives us more control over the entire workout estimation, and since we have more accurate data, we can provide more precise feedback. This was trained by Cao, *et al*. [2] and we are only using the output key points to calculate the angles. Figure 4.16 and Figure 4.17 shows the output from those results, which provides a solid foundation to continue building a feedback model on.



```
import argparse
import logging
import sys
import time
```

```
inference image: ./images/squatEnd.jpg in 0.3517 seconds.
Calculate squat angle:
71.8702906120189
```

Figure 4.16: Final integration with pose estimation model squats



```
        help='if provided, resize
parser.add_argument('--workout', type=str, de
        help='if provided, approp

args = parser.parse_args()
```

```
inference image: ./images/pEnd.jpg in 0.3552 seconds.
Calculate pullup arm angle:
49.56900274506076
```

Figure 4.17: Final integration with pose estimationmModel pull ups

Chapter 5: Conclusion and Future Work

Our work concludes with an end to end trained convolutional neural network on action recognition, in which silhouettes are generated by subtracting the static background from the moving foreground, and intermediate workout actions are classified. This model provides a foundation for a workout application in sports and fitness using workout specific action recognition. The pose estimation model which we enhanced specifically for our goal of providing live feedback to the end user also had great results, and a lot of room to improve and build on. Our top model was able to achieve a testing accuracy score of 79% on squatting images from the PennAction dataset. In our future work, we would like to add additional exercises and sports forms to train on. Since we had a little over 1000 images for each workout to train our action recognition model, we would like to increase the number of training and validation images for more accuracy. The necessary hardware to train and test would be required with the increase in datasets, such as a dedicated graphics card to speed up the number of calculations and processing. Our training and tests were conducted on a MacBook Pro with 2.7GHz i7 processor, 16GB of DDR3 RAM, and an AMD RadeonPro 455 Graphics card which was unable to integrate with the Keras framework used. Ultimately, our goal is to figure out better ways to provide feedback to the user, not only with critique, but also motivation to keep the individual pushing forward through their workouts.

Bibliography

[1] Moez Baccouche, Franck Mamalet, Christian Wolf, Christophe Garcia, and Atilla Baskurt. Sequential deep learning for human action recognition. In Albert Ali Salah and Bruno Lepri, editors, *Human Behavior Understanding*, pages 29–39, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[2] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh. Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. *ArXiv e-prints*, November 2016.

[3] Yu-Wei Chao, Jimei Yang, Brian Price, Scott Cohen, and Jia Deng. Forecasting human dynamics from static images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[4] Steven Chen and Richard Yang. Pose trainer: Correcting exercise posture using pose estimation. *ResearchGate*, 03 2018.

[5] Yiğithan Dedeoğlu, B. Uğur Töreyin, Uğur Güdükbay, and A. Enis Çetin. Silhouette-based method for object classification and human action recognition in video. In Thomas S. Huang, Nicu Sebe, Michael S. Lew, Vladimir Pavlović, Mathias Kölsch, Aphrodite Galata, and Branislav Kisačanin, editors, *Computer Vision in Human-Computer Interaction*, pages 64–77, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[6] Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan,

Sergio Guadarrama, Kate Saenko, and Trevor Darrell. Long-term Recurrent Convolutional Networks for Visual Recognition and Description. *arXiv e-prints*, page arXiv:1411.4389, November 2014.

[7] M. Fani, H. Neher, D. A. Clausi, A. Wong, and J. Zelek. Hockey action recognition via integrated stacked hourglass network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 85–93, July 2017.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[9] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the Best Multi-Stage Architecture for Object Recognition? *IEEE*, August 2009.

[10] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

[11] Karl-Kristjan Luberg. Human body poses recognition using neural networks with class based data augmentation. 2018.

[12] G. Papandreou, T. Zhu, L.C. Chen, S. Gidaris, J. Tompson, and K. Murphy. PersonLab: Person Pose Estimation and Instance Segmentation with a Bottom-Up, Part-Based, Geometric Embedding Model. *arXiv*, March 2018.

[13] Gábor Szucs and Bence Tamás. Body part extraction and pose estimation method in rowing videos. *Journal of Computing and Information Technology*, 26:29–43, 01 2018.

[14] Alexander Toshev and Christian Szegedy. Deeppose: Human pose estimation via

deep neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[15] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional Pose Machines. *arXiv e-prints*, page arXiv:1602.00134, January 2016.

[16] W. Zhang, M. Zhu, and K. G. Derpanis. From actemes to action: A strongly-supervised representation for detailed action understanding. In *2013 IEEE International Conference on Computer Vision*, pages 2248–2255, Dec 2013.

[17] Zizhao Zhang, Fuyong Xing, Hai Su, Xiaoshuang Shi, and Lin Yang. Recent advances in the applications of convolutional neural networks to medical image contour detection. 08 2017.

[18] Zoran Zivkovic and Ferdinand van der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern Recogn. Lett.*, 27(7):773–780, May 2006.