Tennis Stroke Classifier

Introduction:

Tennis is one of my favorite sports to play. I also like to watch the professionals showcase their skills in the tournaments. Personally, whenever I play a tennis match, I always liked to see the result of the match, to analyze my strokes and make any improvements upon feedback. However, processing the video captured and editing the video to see particular strokes is a tedious and time consuming task.

In this project, I would like to build a machine learning algorithm using computer vision deep learning principles to classify three major strokes of tennis(forehand, backhand and serve). This will be a first step in my final goal of building a software that takes in a video and auto-edits it to give an index of all the strokes played. This final software is beyond the scope of this project. I'll focus mainly on the three stroke classification problem.

In addition, i'll be analyzing various techniques used in developing this classifier. My goal is to find and use a deep learning classifier that maximizes the classification accuracy on a test set.

I was unable to find any similar work that has been done for this kind of tennis stroke classification problem. However, this paper "Learning using CNNs for Ball-by-Ball Outcome Classification in Sports" which analyzes the video clips for a different classification problem. Some of the ideas of analyzing video clips have been taken from this paper.

Problem Statement:

Given a small clip, the deep learning classifier algorithm should accurately identify the kind of stroke that was played. I'll be using accuracy as a metric to identify the best classifier algorithm. Accuracy is defined as (total number of accurate predictions/total number of predictions made)*100. Accuracy is measured on a test set. Since there are 3 categories(backhand, forehand and serve), naive random guess(benchmark/baseline) would have an accuracy of 33%. I'm trying to achieve atleast 90% accurate classifier.

Accuracy has been chosen as a metric because I would like the classifier to predict maximum number of correct classifications. For this to happen, I need to verify and maximize the correct predictions. Therefore, accuracy has been chosen as a metric. The goal of the classifier is to maximize the accuracy, thereby, increasing total number of correct predictions.

I would like to use convolutional neural network techniques combined with other state of the art object detection techniques such as YOLOv2(you only look once).

Benchmark:

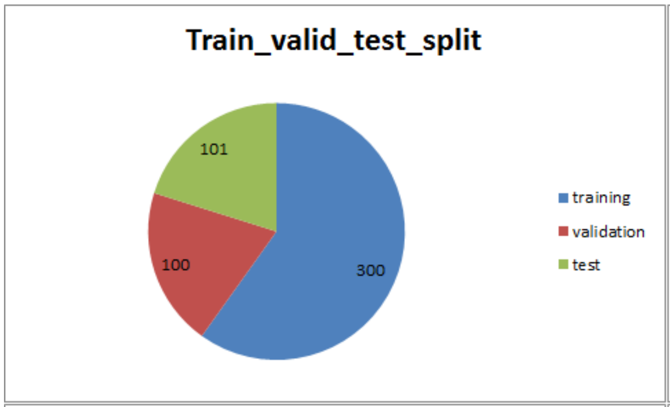
A naive random guess benchmark would yield an accuracy of 33%. I'm trying to achieve above 90% accuracy.

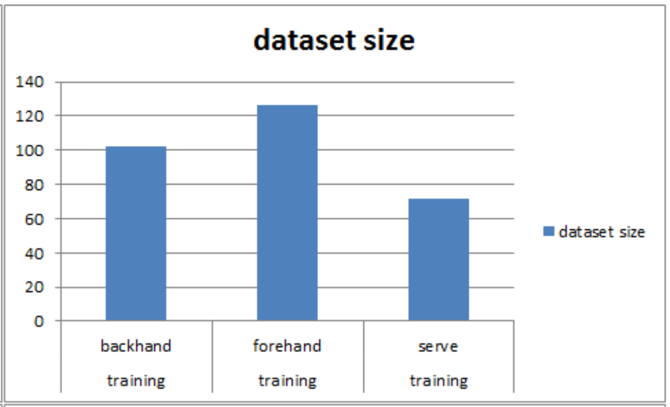
Input Datasets and visualization:

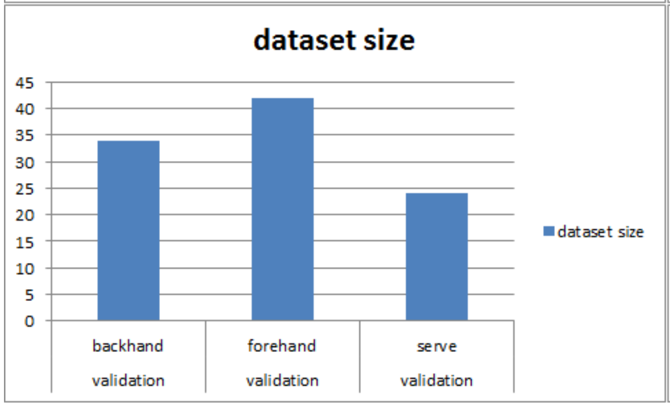
Since this is a one of a kind problem that I am trying to solve, there are no easily application specific datasets available online. So, I took a few recorded videos of tennis matches played by my friends and also some I found online to create input datasets.

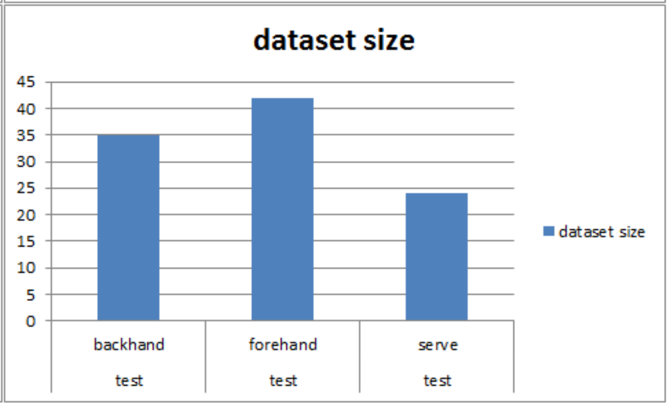
In total , I used 5 different tennis matches played in different environments by different people. Also, since some strokes can take a fraction of a second to execute, I slowed down the video to 10fps(frames per second) from the original 30fps video. I then proceeded to edit the videos, to make many 2-10second clips from the 10fps video, and classified them in accordance to the stroke played. I made 3 folders one for each stroke(i.e backhand, forehand and serve).

I had a total of 171 clips for "backhand", 210 clips for "forehand" and 120 clips for "serve". I then divided these clips into training validation and testing sets using 60-20-20 random split. This resulted in having 102 "backhand" clips, 126 "forehand" clips and 72 "serve" clips of the training set. 34 backhand clips, 42 "forehand" clips and 24 "serve" clips of the validation set. 35 backhand clips, 42 "forehand" clips and 24 "serve" clips of the test set.









Algorithm Inputs and Data Preprocessing:

A video can be thought of as a set of images/frames concatenated together. In this case, since the input video is 10fps(frames per second), we will have 10 images per second of the clip. For example, if an input video clip is 5 seconds, we will have 5*10=50 input images to consider when analyzing the video clip. Each image is of the resolution 854x480 pixels. Where each pixel has 3 channels (RGB-RedGreenBlue). Therefore we have each image of size 854x480x3. And analyzing 50 of these images for a video clip is a very computationally expensive task for any algorithm.

To mitigate this problem, I divide a video clip into 5 equal parts. Then I randomly select one image from each divided part. Therefore, I will end up with a total of 5 images(size 854x480x3) per input video clip. Now 5x854x480x3 is a much more reasonable input for an algorithm to be efficient. Please note that, this number of equal divisions is one of the hyper-parameter that can be tweaked to achieve an optimal solution. Also, it is interesting to point out that, we can have a total of 10^5 different combination of 5 set of images representing the same 5seconds-10fps video clip. This can be a potential way to increase number of input datasets to train the algorithm on. However, I this project I do not explore this particular observation.

Also, all the algorithms I implemented use input video clip's each image size of 299x299x3. I use opencv libraries to resize each of the images from 854x480x3 to 299x299x3.

In Addition, My final most accurate algorithm uses input video clips to be cropped to include just the person of interest + the tennis racquet. This is done using YoloV2(you only look once) state of the art object detection algorithm. I do this using "Yolo-crop-video_conversion.ipynb" file in the "full_to_yolo_conversion" folder. This is done because, when we look at the video clip, we generally look at the person of interest and his motion to determine the classification of the shot. In a similar fashion, I wanted the algorithm to look at the person of interest only to make a classification judgement. As will be seen in the results section, this leads to significant improvement in the accuracy.

To summarize, we have 2 sets of input video clips that we analyze.

1. Full unedited video clips that will be divided into 5 equal parts, and 1 image is chosen from each division at random. Also, each image is resized using opencv to 299x299x3. Thereby having 5 images per video clip. Therefore, input matrix is 5x299x299x3. These video clips can be found in the folders "train_full", "valid_full" and "test_full".

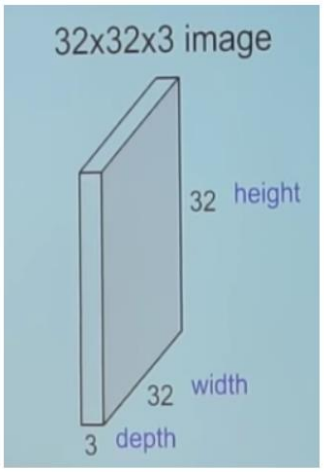
2. Edited video clips using YoloV2 person detection and cropping techniques described above. These video clips will be divided into 5 equal parts, and 1 image is chosen from each division at random. Also, each image is resized using opencv to 299x299x3. Thereby having 5 images per video clip. Therefore, input matrix is 5x299x299x3. These video clips can be found in the folders "train_yolo", "valid_yolo" and "test_yolo".

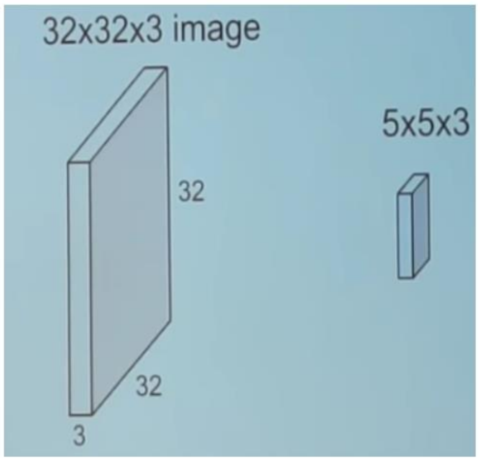
Algorithms Implemented:

Convolution Neural Networks(CNN): This is state of the art deep learning image processing/computer vision technique where we use filters to parse through various parts of the image to find basic features such as edges/contrast/brightness etc. in the shallow layers and object detection features such are eyes/nose/wheels/fur etc in the deeper layers. Further details on Convolutional neural networks can be found in the following link https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8

In a more detailed fashion with pictures, CNN's can be described as follows:

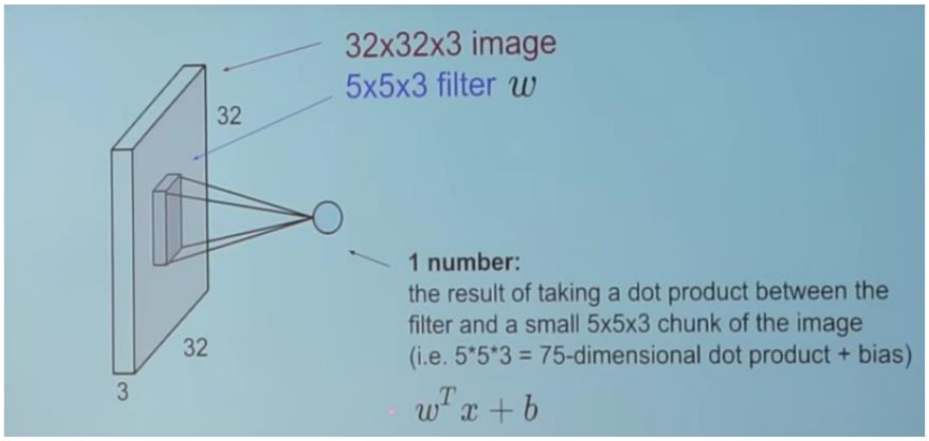
- Let us say we have an input image. It will have information of all the pixels. Each pixel will have 3 channels(Red, Blue and Green). For example let us say we have input image of size 32x32x3.
- We take a filter say of size 5x5x3 and use it to do a dot product across all areas of the image.





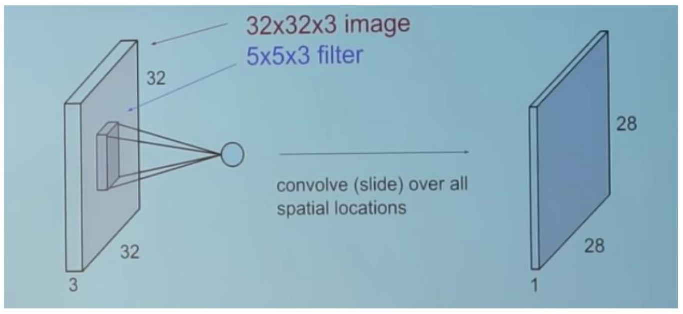
- 1. Example of a RGB image (let's call it 'input image')

2. Convolving an image with a filter


32x32x3 image
5x5x3 filter w
32
32
3
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)
$w^T x + b$

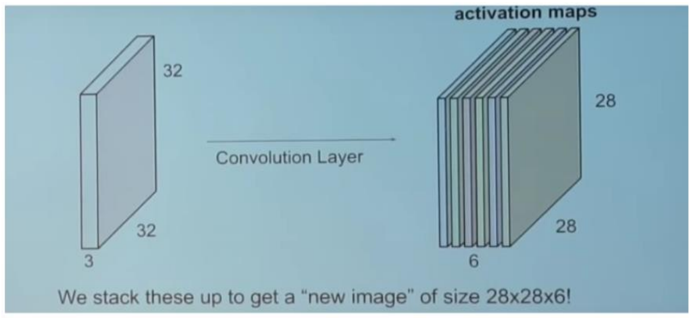
- 3. This is how it looks

- When we perform dot product across all 5x5x3 areas of an image(convolution), we end up with a convolutional layer corresponding to that particular filter. Can be seen in picture 4.

- 

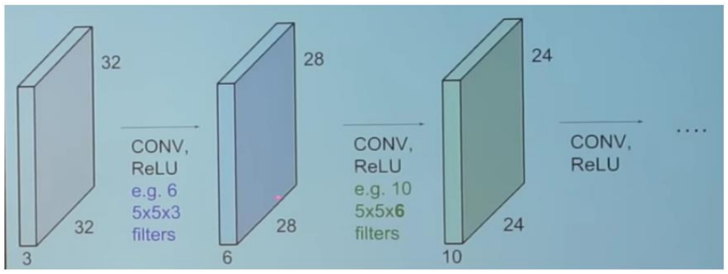
4. This!

- Now when we perform convolution using many filters, we end up with activation maps, which is basically a concatenation of convolution performed on the input image with multiple filters. Let us say we use 6 filters, then we have the structure shown in the following picture5

- 

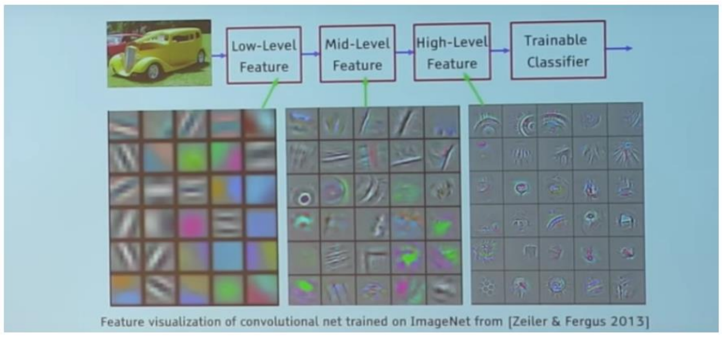
5. Convolution Layer

- Now we use this 28x28x6 layer as an input layer for the next set of filters(say 10 filters). We will end up with the following sequence of layers. If we use more number of layers in sequence(sometimes parallel as well) we use, we will end up with a deeper Convolutional neural network.

- 

6. Convolution Layers in sequence

- Here is an example of CNN showing how the filters react to various features of the image.
- As mentioned earlier, lower level features show features such as edges/contrast/brightness etc. and higher level features give a more detailed glimpse of higher order features such as eyes, wheels, etc.

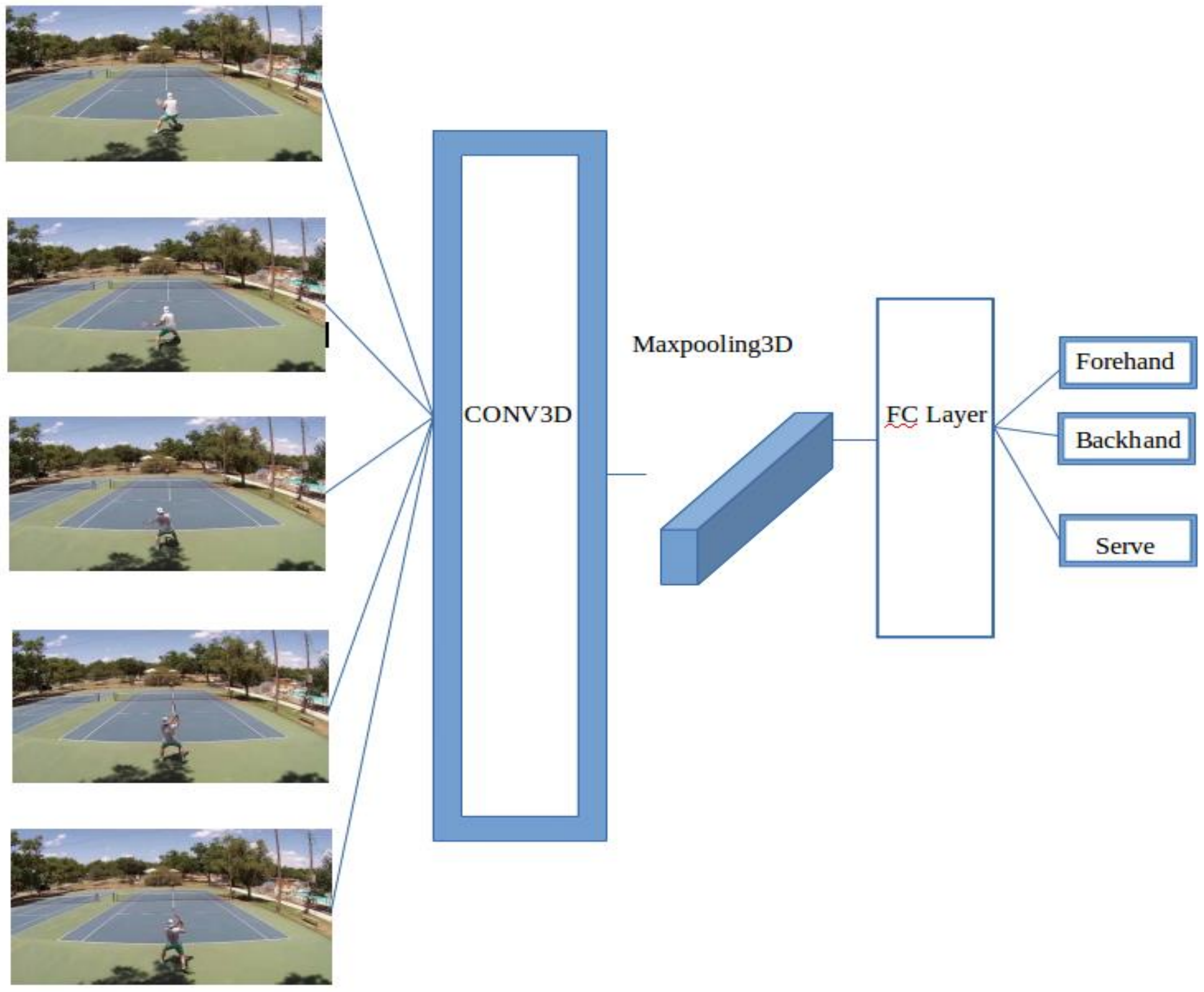

Low-Level Feature
Mid-Level Feature
High-Level Feature
Trainable Classifier
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

-

7. Filters in a trained network

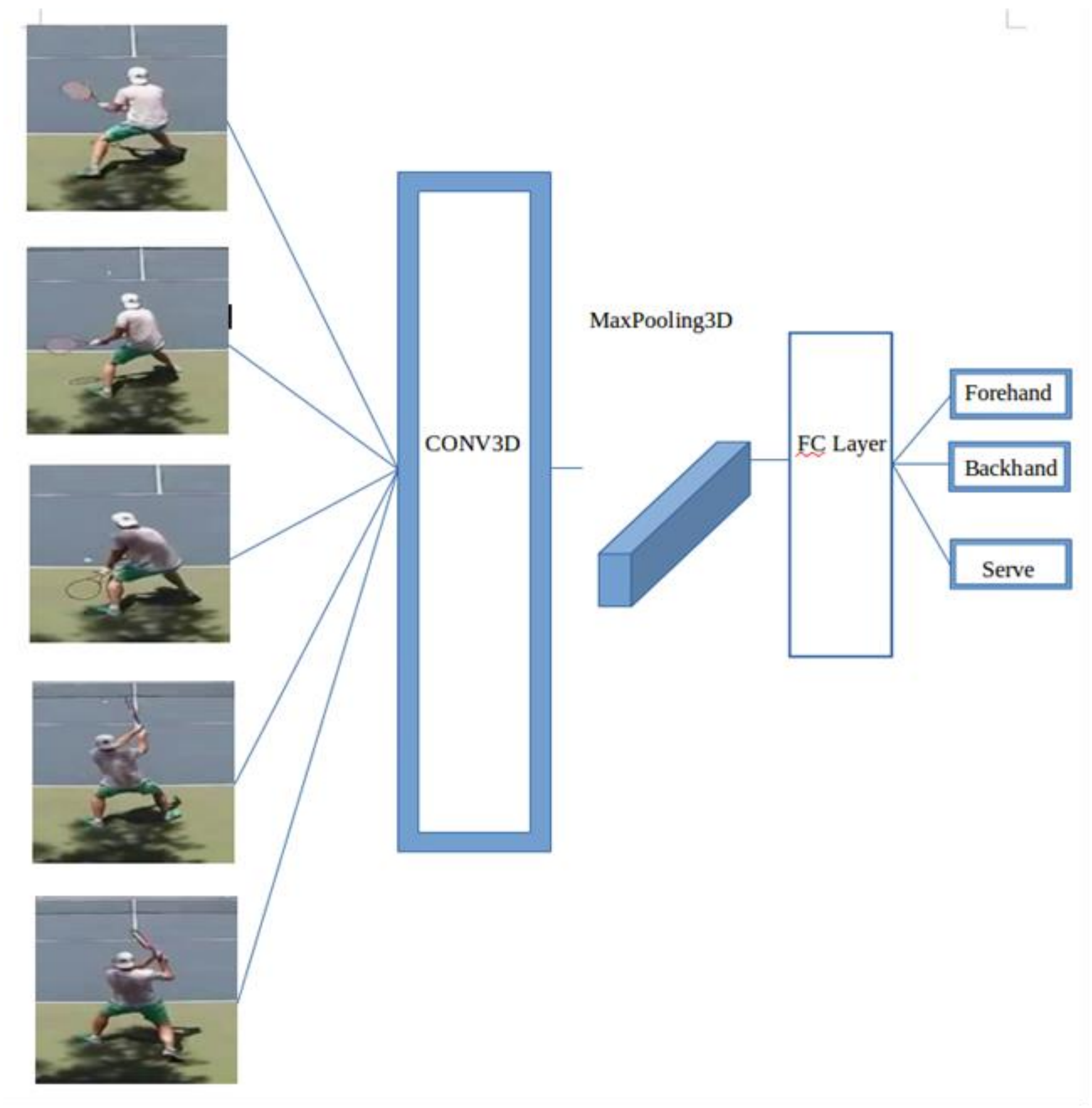
All the algorithms I use have basis on CNN. I use keras library with tensorflow backend to implement all my algorithms. I implement and compare performance of 6 algorithms as outlined below with pictures:

1. Full video clip; With Convolution3d: In this technique, I take 5 input images of effective size 5x299x299x3. Parse it through Conv3d layer, then Maxpooling3D layer, flatten and dense layers. I was unable to use deeper layers because of the very limited input size. I do use techniques such as dropout to overcome high-variance/overfitting. I also use model_checkpoint

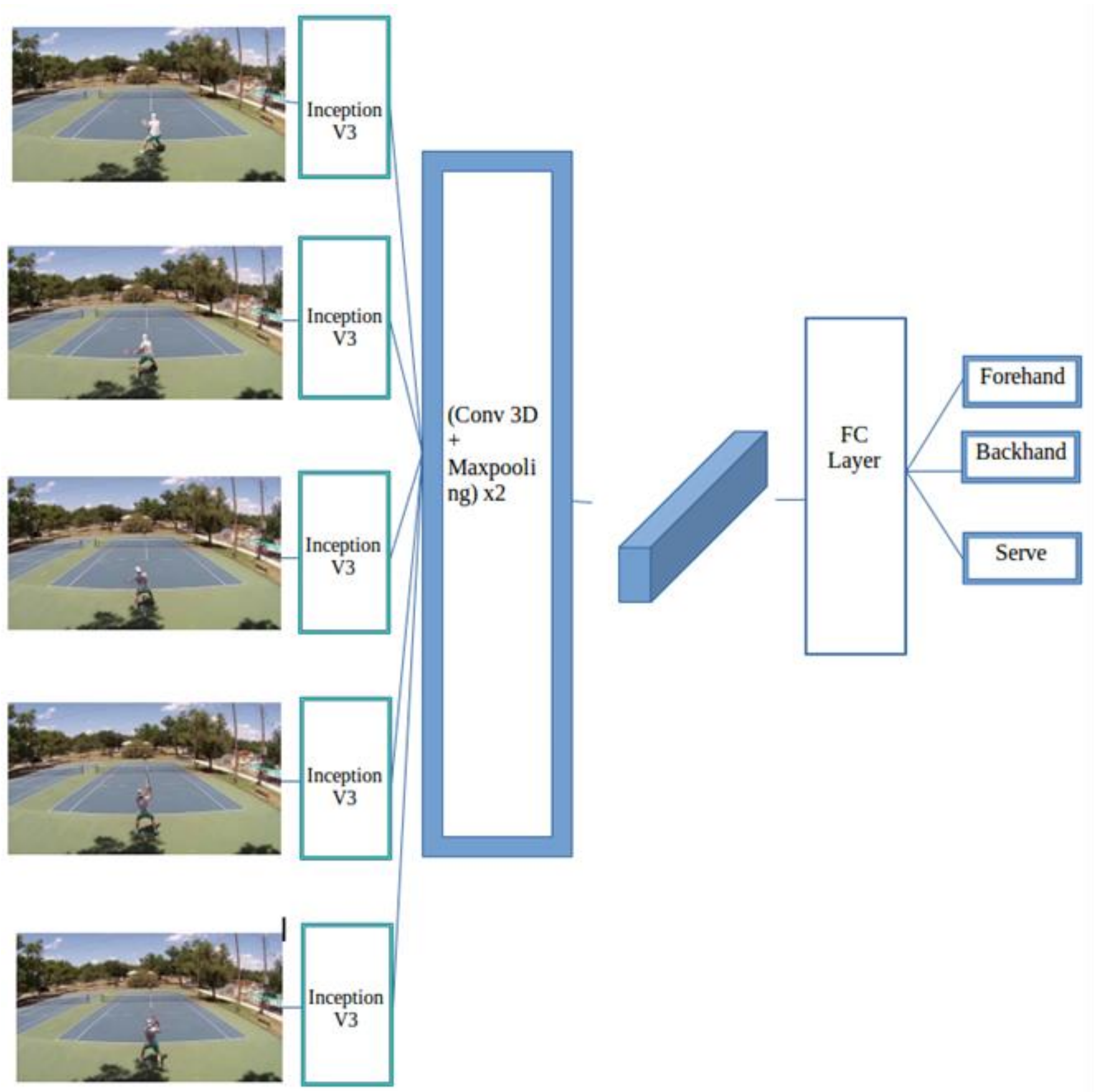


to save the algorithm weights of a particular epoch with best validation accuracy. This technique can be visually seen in the above picture:

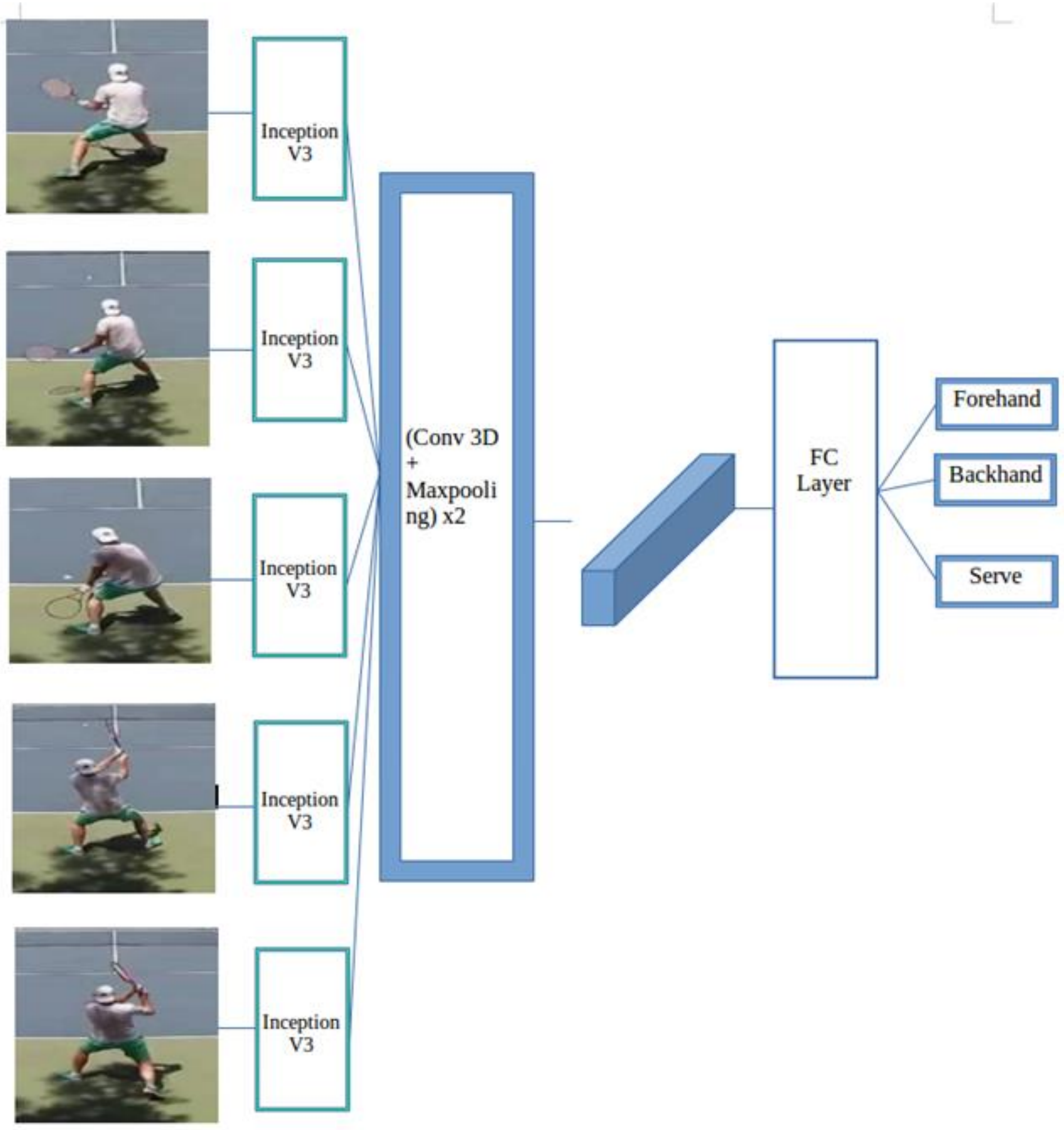
2. Yolo video clip; Convolution3D: In this technique, I use yolo-person-of-interest-cropped video clip. I take 5 input images of effective size 5x299x299x3. Parse it through Conv3d layer, then Maxpooling3D layer, flatten and dense layers. I was unable to use deeper layers because of the very limited input size. I do use techniques such as dropout to overcome high-variance/overfitting. I also use model_checkpoint to save the algorithm weights of a particular epoch with best validation accuracy. Please note the images which have been cropped to just have the person of interest in sight. This technique can be visually seen as follows:



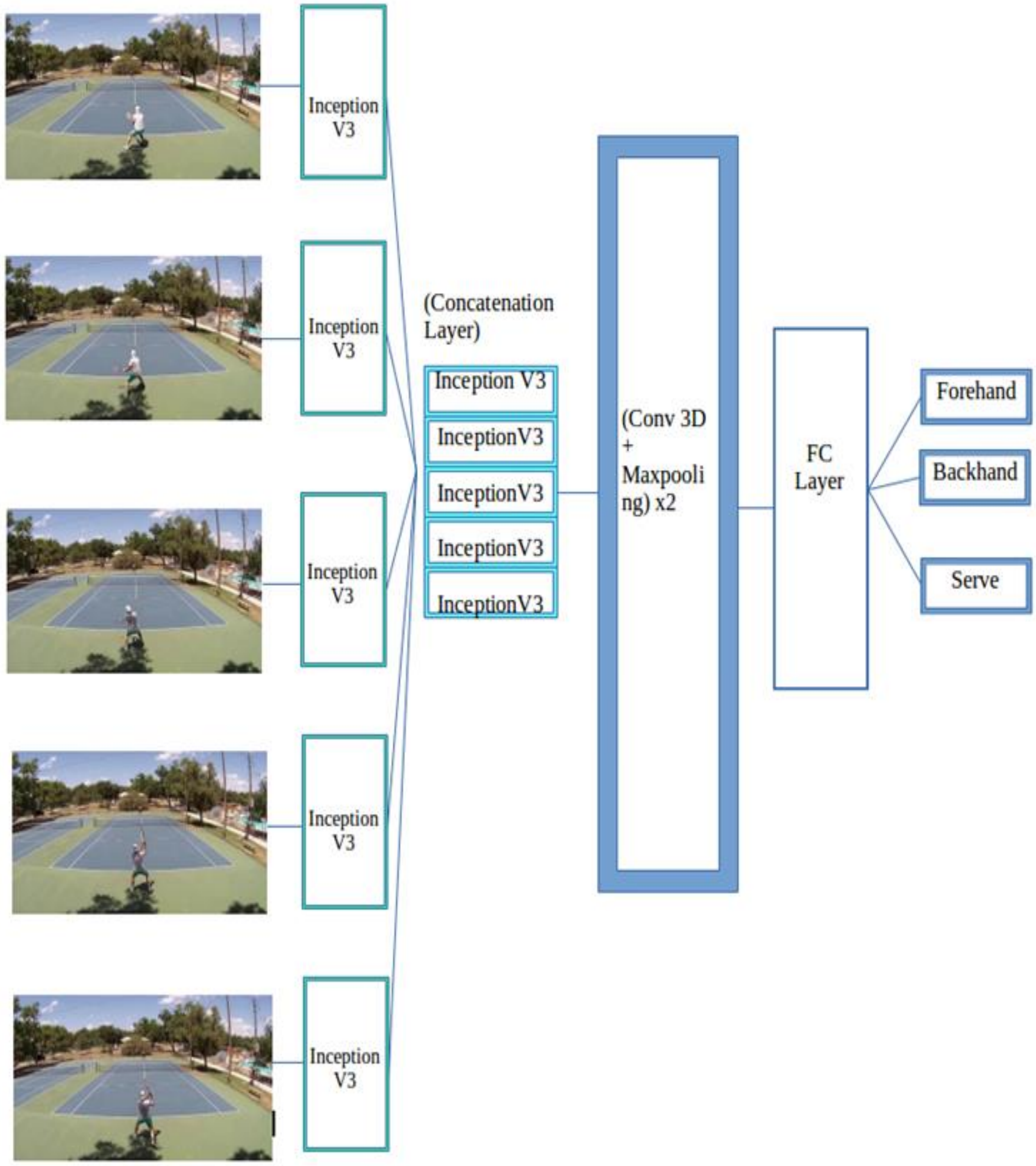
3. Full video clip; With transfer learning + Convolution3d: Transfer learning is a technique where an open source multi layer deep, pre-trained models such are InceptionV3 with weights are used (these weights have been trained on huge amount of data to obtain high level features such as person,car,bike, etc). We use bottleneck feature extraction technique that uses the resultant output, just before flattening of the layers. So we give this model our set of 5 images, we get an output set of 5 tensors which contain high level feature information. We then use these 5 tensor outputs and parse it through Conv3D layer as shown below to classify the video clip. This architecture can be visually seen below:



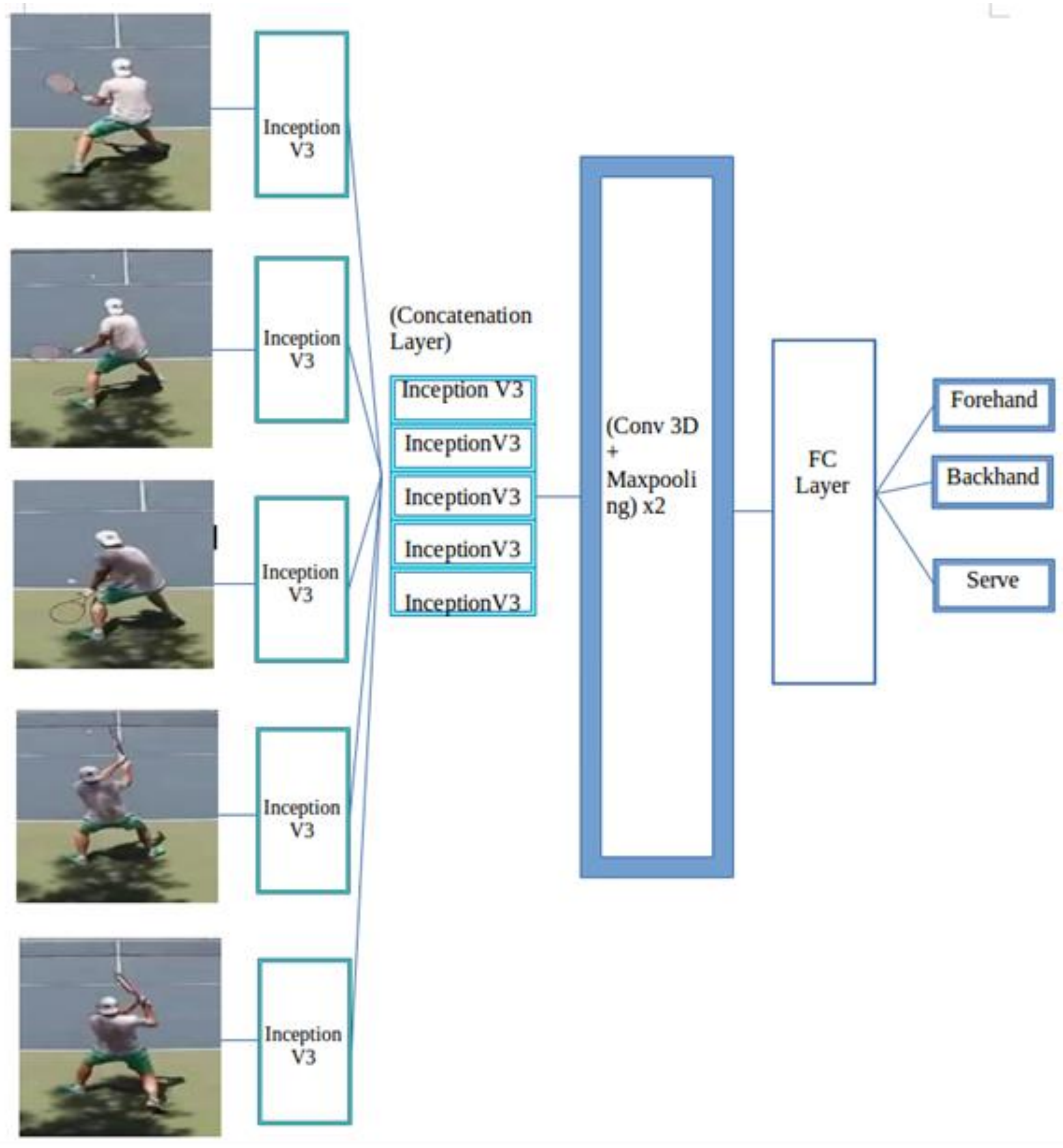
4. Yolo video clip; With transfer learning + Convolution3d: In this technique, I use yolo-person-of-interest-cropped video clip. Transfer learning is a technique where an open source multi layer deep, pre-trained models such are InceptionV3 with weights are used (these weights have been trained on huge amount of data to obtain high level features such as person,car,bike, etc). We use bottleneck feature extraction technique that uses the resultant output, just before flattening of the layers. So we give this model our set of 5 images, we get an output set of 5 tensors which contain high level feature information. We then use these 5 tensor outputs and parse it through Conv3D layer as shown below to classify the video clip. This architecture can be visually seen below:



5. Full video clip; With transfer learning + Concatenate + Convolution2d: This technique is very similar to technique 3, but here, we use the 5 output tensors of the resultant transfer learning model, to concatenate together to form one tensor(i.e reduce dimensions). This one tensor which contains information of the whole video clip is parsed through Convolution2D layers to make a classification judgement. This can be visually seen in the picture below:



6. Yolo video clip; With transfer learning + Concatenate + Convolution2d: In this technique, I use yolo-person-of-interest-cropped video clip. This technique is very similar to technique 4, but here, we use the 5 output tensors of the resultant transfer learning model, to concatenate together to form one tensor(i.e reduce dimensions). Then this one tensor which contains information of the yolo video clip is parsed through Convolution2D layers to make a classification judgement. This can be visually seen in the picture below:



Challenges faced:

- Data collection and video clip generation: this was a time consuming task to classify every video clip into their respective categories. Also, since some of the strokes are faster than a second, it was very hard to capture the video clips accurately. Hence I chose to convert the video from 30fps to 10fps using opencv to make video clips indexing and capturing easy.
- Yolo video clip generation: this was a quite tricky since the original YoloV2 was implemented in darknet C language. After a bit of trial and error with various python yolo sources available online, I found "https://github.com/experiencor/basic-yolo-keras" to be the most helpful in implementing yolo algorithm. I just followed the procedures outlined in this repository to implement yolo algorithm.

Dataset retrieval and Algorithm implementation files and links:

1. Full video clips to Yolo cropped video clips: this implementation can be seen in the folder "full_to_yolo_conversion". The file is a jupyter file "Yolo_crop_video_conversion.ipynb". This file has the complete yoloV2 architecture. It uses the pretrained "weights_coco.h5" to make person detection and bounding box coordinates, to create cropped videos as shown in folders "train_yolo", "valid_yolo" and "test_yolo".
2. Algorithms on Full video clips: The three algorithms described above(Conv3d, InceptionV3+Conv3d, and InceptionV3+concatenate+Conv2d) which are implemented on on full video clips datasets can be seen in the jupyter file "full_vid.ipynb" in the main folder.
3. Algorithms on Yolo video clips: The three algorithms described above(Conv3d, InceptionV3+Conv3d, and InceptionV3+concatenate+Conv2d) which are implemented on on full video clips datasets can be seen in the jupyter file "yolo_vid.ipynb"in the main folder.

Results:

Accuracy results can be seen in the table below.

Algorithms	**Accuracy** on test set
Benchmark/baseline; Naïve random guess	33%
Full Video Clips; Conv3d	52.47%
Full Video Clips; Transfer learning; InceptionV3 +Conv3D	75.25%
Full Video Clips; Transfer learning; InceptionV3 +Concatenate + Conv3D	78.22%
Yolo Video Clips; Conv3d	76.23%
Yolo Video Clips; Transfer learning; InceptionV3 +Conv3D	91%
Yolo Video Clips; Transfer learning; InceptionV3 +Concatenate + Conv3D	92%

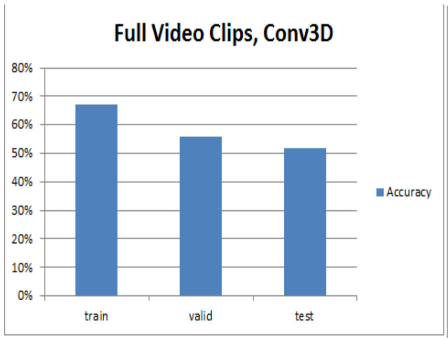
From the results summarized in the table above, we see that algorithms using Yolo video clips achieve maximum accuracy. This can be explained simply by the fact that the classification only depends on the person of interest and his/her actions in the video clips. Since, Yolo video clips isolate the person of interest, we can safely predict the classification of the video clip to the highest degree.

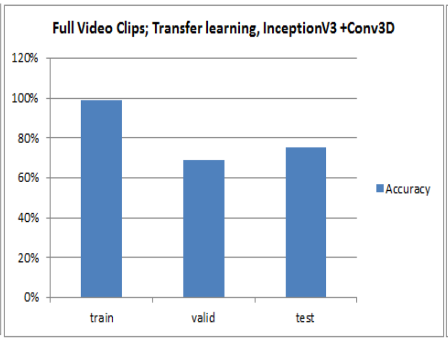
It is also interesting to point out that algorithms "Yolo Video Clips; Transfer learning; InceptionV3 +Conv3D" and "Yolo Video Clips; Transfer learning; InceptionV3 +Concatenate + Conv3D" both have similar accuracies making them both ideal choices for Tennis stroke classification problem

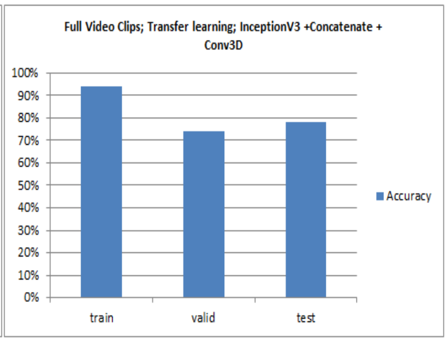
The model should work with the above accuracy for all new sample clips that have been taken, only if they are played in hard courts. If a video clip has a tennis match in clay or grass courts, the above algorithm will not work well. This is because, the training has been done on hard courts and none of the image/video augmentation techniques have been applied. Ideally, if the training set is larger with various samples of video clips from clay, grass, hard courts (all kinds of tennis courts), we should get better accuracy with just slight modification to above models.

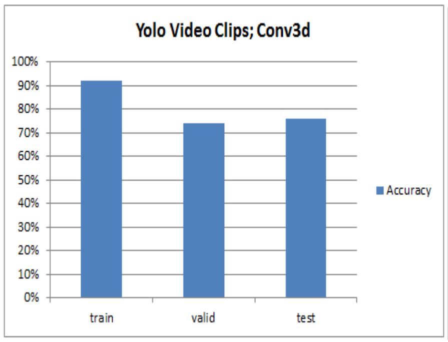
Robustness of the results:

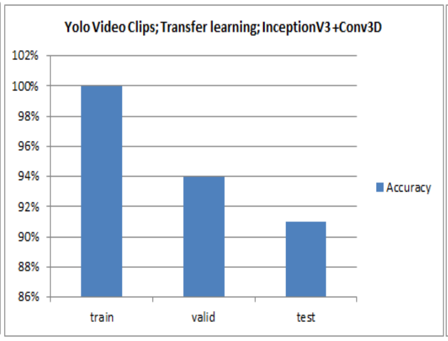
Following graphs show how the results varied across training, validation and test sets. As expected, we see train, validation and test datasets having similar performance.

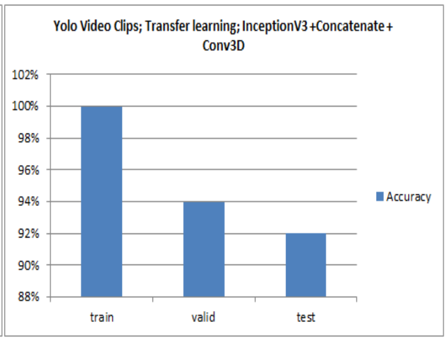






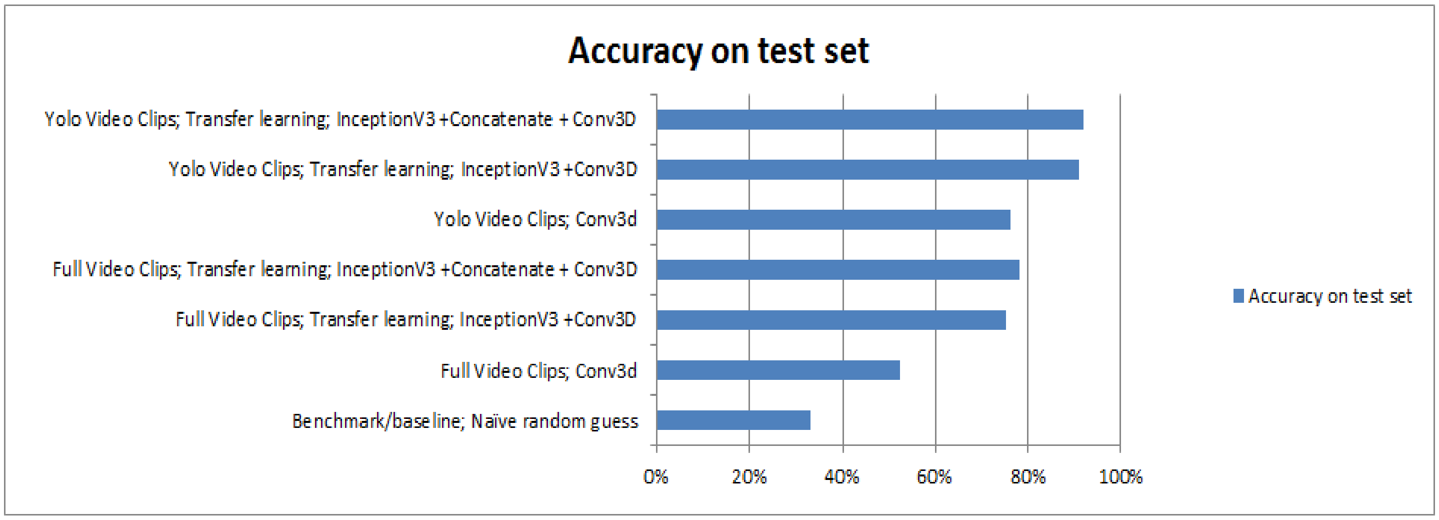






Conclusion:

I have successfully studied and implemented 6 different algorithms and achieved an accuracy of 91% at the least for the tennis stroke classification problem. Naive Random guess would yield an accuracy of 33%. So, this is a significant improvement in classifying the strokes. Visual summary of 6 different algorithms can be seen below:



In addition, Step-by-Step implementation of the project can be summarized as follows:

- Gather full tennis match videos
- Classify the full tennis match video into video clips and arrange them in their respective classification categories.
- Run Conv3D algorithm, transfer learning+Conv3D algorithm, transfer learning + concatenate + conv2D algorithm and measure their respective algorithm accuracies. Modify the algorithm parameters by iteratively changing parameters and verifying any improvement in accuracy.
- Realized that accuracies are nowhere close to the accuracies I desired for this classification problem. Hence, after careful observation of the video, I figured out that cropping the person of interest and only applying CNN algorithms on the resultant video clips will lead to much better accuracy. After thorough research on various open source person classification with bounding boxes algorithms available online, I concluded that YoloV2 is the fastest and one of the most accurate algorithms.
- Used YoloV2 algorithm to crop original video clips to make a yolo_video clips that contain maximum information of the person of interest.
- Ran Conv3D algorithm, transfer learning+Conv3D algorithm, transfer learning + concatenate + conv2D algorithm and measure their respective algorithm accuracies. Modify the algorithm parameters by iteratively changing parameters and verifying any improvement in accuracy.
- Figured out that this implementation of using yolo cropped person of interest video clips leads to a much higher and very desirable accuracy >90%

Future Work:

One major way to improve the accuracy would be to gather a much larger dataset. Having larger datasets is one of the reasons where deep learning algorithms shine. With a larger dataset, I can implement a much more complicated/deeper convolutional layers that can lead to much improved accuracy.

In Addition, having an input dataset with video clips that have tennis matches played in various kinds of courts (ex: hard court, grass court or clay courts) will lead to a greater improvement in accuracy for various possible test clips. Image/video augmentation techniques could be potentially used to make this happen as well.

This is a beginning to a full fledged solution where, if given a video, the software processes the data file and systematically indexes all the strokes. This would be a very useful feature for me so I can spend more time in analyzing my tennis game, rather than looking at the whole video and manually editing the video.

References:

(1) Kalpit Dixit, Anusha Balakrishnan:Deep Learning using CNNs for Ball-by-Ball Outcome Classification in Sports. http://cs231n.stanford.edu/reports/2016/pdfs/273_Report.pdf

(2) https://github.com/experiencor/basic-yolo-keras

(3) https://en.wikipedia.org/wiki/Convolutional_neural_network

(4) Udacity Deep learning dog classifier project.

(5) YoloV2 project. https://pjreddie.com/darknet/yolo/

(6) https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8