Assignment 4: Action Classification in Video

Satwiko Wirawan Indrawanto - 6201539

Basar Oguz - 6084990

This document reports the implementation and structure of a Convolutional Neural Network (Part I) aimed to classify human actions in video, as well as the results and discussion of how well this network performs with the testing data (Part II). The entire framework was written in Python, with OpenCV and TensorFlow dependencies.

I. Training and Testing Pipeline

The entire framework consists of *online* and *offline* components. The offline modules are the first in the pipeline, to be run once, prior to training, in order to prepare the data. The online modules are **train** and **predict.**

a. Data Preparation

The main data preparation module, ***dataset_generator.py***, has three main functionalities: extracting the middle frame of the video, generating the dense optical flow using the Gunnar Farneback algorithm. The final optical flow feature vector that is calculated is the collection of the maximum-magnitude (u,v) vectors that were calculated using the algorithm. The Farneback algorithm accounts for global motion, therefore it is preferable over Lucas-Kanade algorithm, for encapsulating large motions like jumping jacks that could be found in the dataset. The third data preparation module is the *fold generator,* that divides the data into the number of given folds. Some of these folds are to be used in training and whereas some in cross validation, interchangeably.

The last step in the data preparation pipeline consists of data division. We have opted for using 60% of the data for training, 20% for cross validation and the remaining 20% for testing.

b. Training

The convolutional neural network topology is set by several hyper-parameters. In order to keep the training time manageable, there are 3 convolutional layers, and two fully connected layers that are tied to the end of these convolutional layers. The first two convolutional layers (activation maps) are

created with 32 3x3 filters and the third one is created by 64 3x3 filters. We have more filters deeper in the network, because higher-level image features and semantics are more important in classifying large scale human actions. The fully connected layers, that take the higher level image features from the last convolution layer, and outputs the predicted classifications, consist of 128 neurons each with different weights, that are adjusted by gradient descent and the loss function in training. We have used the Adam algorithm, that is based on gradients, that TensorFlow provides as one of the most robust algorithms to reduce error. The learning rate we have concluded upon is 10^(-4), because smaller learning rates lead to unmanageable learning durations.

It should be noted that each of the three convolutional layers, are immediately followed by a max pooling layer and a ReLU (rectifier activation function). The pooling layer decreases the dimensionality of the activation maps deeper in the network, by only keeping the "most activated" neurons (i.e. meaningful features). The non-linear activation function ($f(x) = \max(0, x)$) eliminates "negative activations" to create non-linearity in the system.

After the 3 x Conv2D layers, we have implemented an intermediate flattening layer. This layer converts the layered output of the convolution layers to a 1D vector that the fully connected layers can use.

II. Testing and Results

The **performance measures** used to test the robustness of the neural network were: **precision**, **recall**, **F-Score** and **confusion matrices**. Given the terminology in the following table,

		Guessed class	
		True	**False**
Actual class	**True**	True positive	False negative
	False	False positive	True negative

Precision (P) = (True Positive) / (True Positive + False Positive)

Recall (R) = (True Positive) / (True Positive + False Negative)

F-Score = $2*P*R / (P + R)$

We have divided our dataset into “Training and Cross Validation” (80%) and Testing (20%) branches. The **cross validation** branch is randomly chosen as the 20% of its branch at the beginning of training. We have used the videos we took as well as some UCF101 videos to test our neural network, this means that the test set was a mixture of both datasets, providing a better indication of generalization. A **data augmentation** on the training dataset of RGB images was made by **mirroring** the input images vertically. This is theoretically valid, because none of the five Action Classes are variant to mirroring: the semantics of the action to be classified do not change with mirroring. Essentially, mirroring helps to artificially cover more variation in the possible data.

RGB Test

Initial testing (RGB Test) with the 235 RGB mid-frame images (47 per class, equal distribution with training data), that consists of the 20% of our dataset, we have collected the following performance measures:

Performance matrices (Columns represent **Predictions**, rows represent **Actuals**)

CONF_MATRIX	Brushing Teeth	Cutting in Kitchen	Jumping Jack	Lunges	Wall Pushups
Brushing Teeth	**37**	5	0	0	2
Cutting in Kitchen	3	**34**	2	2	2
Jumping Jack	0	0	**44**	1	0
Lunges	1	1	7	**36**	0
Wall Pushups	0	0	0	0	**47**

Table I: Confusion matrix RGB test using separated testing data from the UCF101 set

CONF_MATRIX	Brushing Teeth	Cutting in Kitchen	Jumping Jack	Lunges	Wall Pushups
Brushing Teeth	**2**	0	0	0	0
Cutting in Kitchen	0	**2**	0	1	0
Jumping Jack	0	0	**2**	0	0
Lunges	0	0	1	**1**	1
Wall Pushups	0	1	1	0	**1**

Table II: Confusion matrix RGB test using our own created testing data

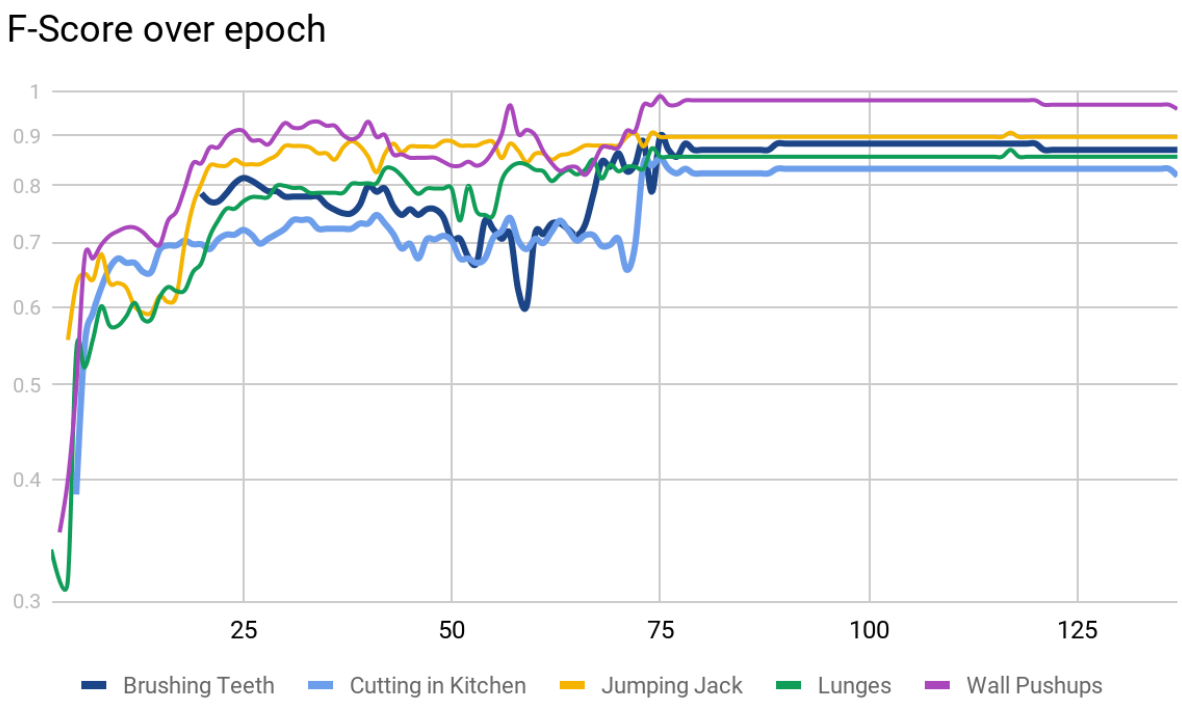
Performance measures (Columns represent **Predictions**, rows represent **Actuals**)

PERF_MATRIX	Brushing Teeth	Cutting in Kitchen	Jumping Jack	Lunges	Wall Pushups
Precision (P)	0.84	0.79	0.98	0.80	1.00
Recall (R)	0.90	0.85	0.83	0.92	0.92
F-Score	0.87	0.82	0.90	0.86	0.96

Table III: Performance measure RGB test using separated testing data from the UCF101 set (average F-Score is 0.88)

PERF_MATRIX	Brushing Teeth	Cutting in Kitchen	Jumping Jack	Lunges	Wall Pushups
Precision (P)	1.00	0.67	1.00	0.33	0.33
Recall (R)	1.00	0.67	0.50	0.50	0.50
F-Score	1.00	0.67	0.66	0.40	0.40

Table IV: Performance measure RGB test using our own created testing data (average F-Score is 0.64)



Optical Flow Test

In addition to RGB based training and testing, an Optical Flow input was given to train the network from scratch. We tested our optical flow converted datas using the same neural network topology that we used for RGB training. The following confusion matrix and performance scores were obtained with testing using the optical frame with the maximum magnitude from each video, as the feature vector input (tensor input):

Performance measures (Columns represent **Predictions**, rows represent **Actuals**)

CONF_MATRIX	Brushing Teeth	Cutting in Kitchen	Jumping Jack	Lunges	Wall Pushups
Brushing Teeth	**10**	5	0	3	7
Cutting In Kitchen	3	**14**	2	2	4
Jumping Jack	2	0	**21**	1	1
Lunges	1	8	1	**14**	1
Wall Pushups	3	0	0	1	**21**

Table V: Confusion matrix optical flow test

PERF_MATRIX	Brushing Teeth	Cutting in Kitchen	Jumping Jack	Lunges	Wall Pushups
Precision (P)	0.40	0.56	0.84	0.56	0.84
Recall (R)	0.53	0.52	0.86	0.67	0.62
F-Score	0.45	0.54	0.86	0.61	0.71

Table VI: Performance measure optical flow test

III. Discussion

Using Single RGB Frames

Training and testing the neural network with a single mid-frame, does not account for motion, the high-level meaning of the deeper activation maps in the CNN only account for how big and where the large "segments" in the image are. Therefore, when similar "segments" like human bodies in similar scales do different motions, it causes some confused guesses (false positives). For example 12% of the Jumping Jack predictions were actually Lunges, because typically, what distinguishes these two moves is not what *is* on the frame but what movement that subject does. However, single-frame RGB testing already can excel at distinguishing between types of actions that have different main subjects (large inter-class variation in the dataset). For example there is a large much inter-class variation between *brushing teeth* and 3 "standing moves", and therefore only 1 case of brushing teeth prediction was actually a "standing move".

Using Optical Flow

When taking optical flow into account, however, the movement of the pixels are taken into account and the object sizes and colors are disregarded. This produces slightly different results than a RGB based network.

Based on the experiment that we did, optical flow surprisingly had worse performance than RGB. This is mainly caused by how we calculated the optical flow for each video. Since we only take the optical frame with the largest movement, the network is only good for identifying high movement actions. In this case with our data is Jumping Jack, followed by Wall Pushups. Using a single 128x128x2 dimension optical flow input from each video from training is not the optimal solution. Training took a lot longer while validation loss keeps increasing.

Performance graphs of training using optical flow can be found below.

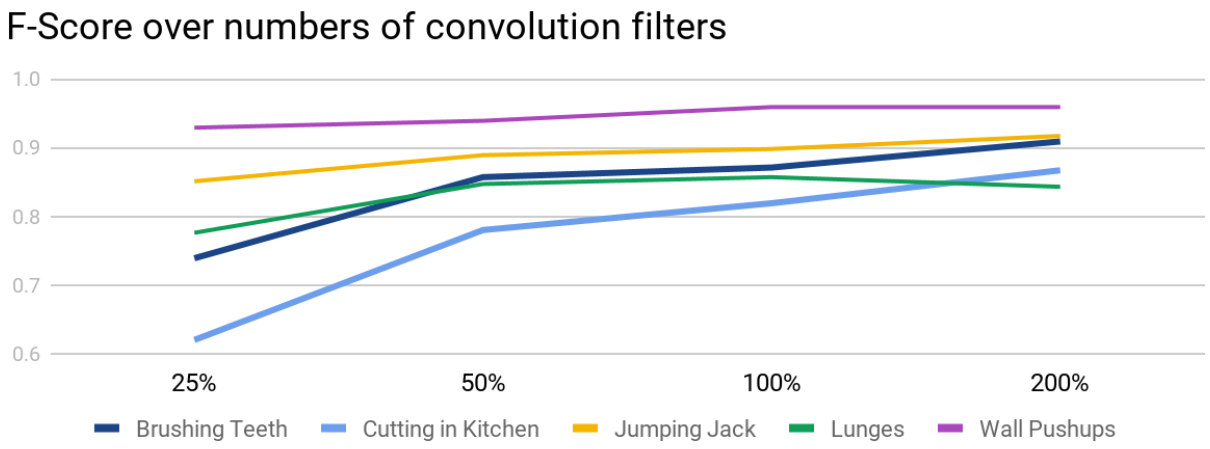
Different Network Topologies

We have also experimented with different network topologies, especially with the filter (neuron) counts that produce activation maps. Since every filter in a convolutional neural network is aimed at detecting a feature in the image, less filters would mean less features to train the network at detecting. Less filters lead to thinner convolutional layers and ultimately leads to a less robust classifier.

By reducing the number of filters to 25% of the default number (the default neural network has 128 number of filters), training went much faster to converge to the desired validation error range but the results for guessing unseen images dropped down compared to the default number of filters. Fast paced actions, such as Jumping Jacks and Wall Push Ups, were guessed more correctly than others while slow paced actions were only guessed a little bit more than 50%. The worst was for Cutting in Kitchen as it has the slowest pace of all the actions. It was still a decent classifier as the average F-Score is 0.78.

Increasing the number of filters from 25% to 50% made the classifier more robust. Average of the F-Score is 0.86. Time it took to train was between 25% numbers of filters and the default settings.

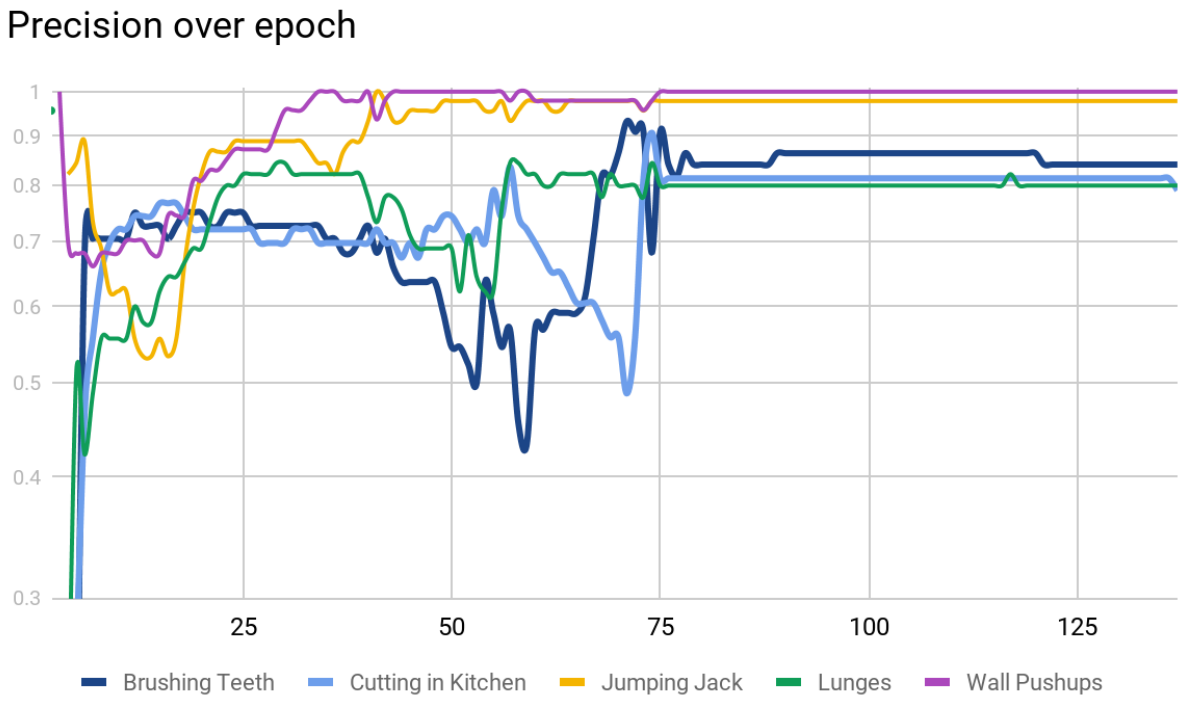
Doubling the number of filters gave us the most robust classifier as it has an average F-Score of 0.92. The drawback with this setting is that training took a lot of time and by having more epochs to train, the classifier is more prone to overfitting (can only recognise trained data). As we can see from the graph, F-Score for Lunges went slightly down with longer training although still correct guess the unseen images.

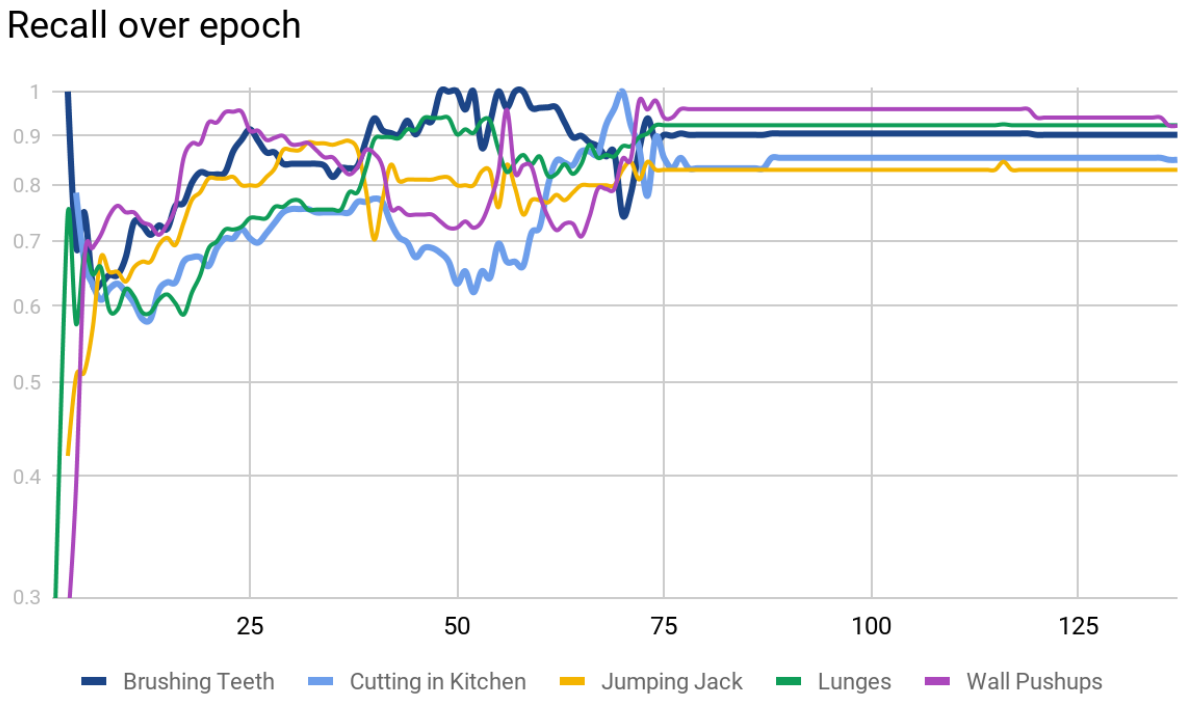


In conclusion of the network topology, we can say that by increasing the number of filters (neurons), we can achieve a more robust classifier as there are more details to be detected by the system while sacrificing time to train and having the probability of overfitting.

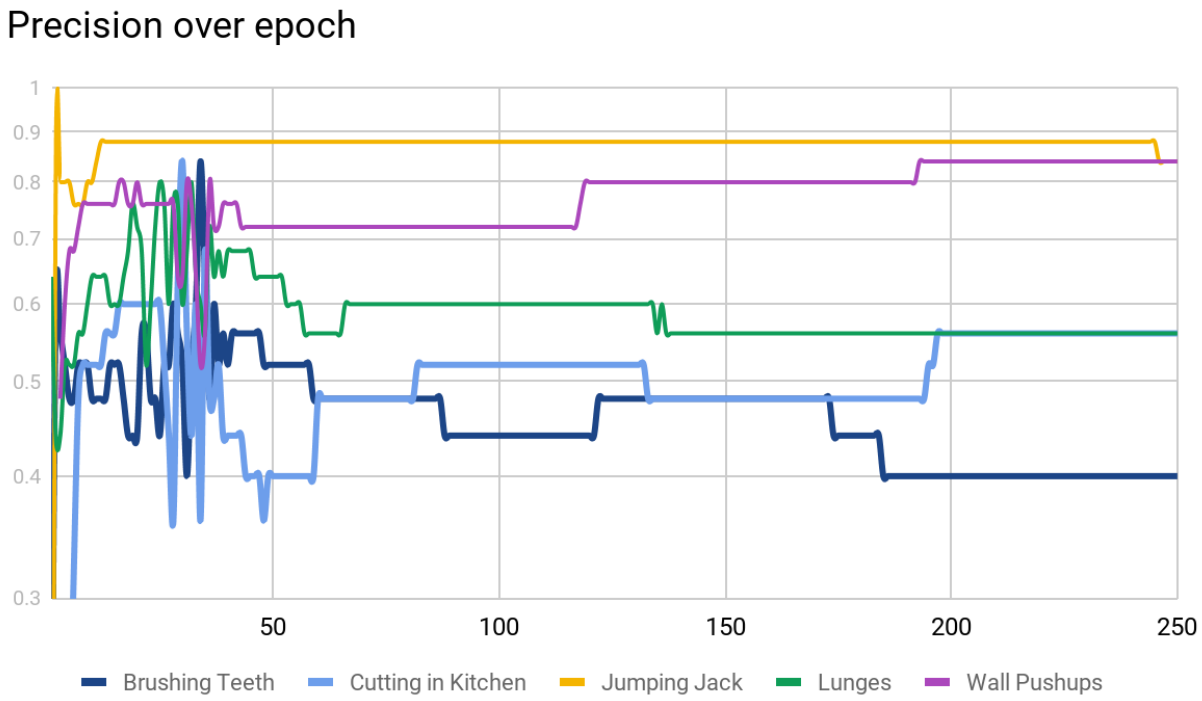
Confusion and performance matrices of the different topologies of the neural network can be found below.

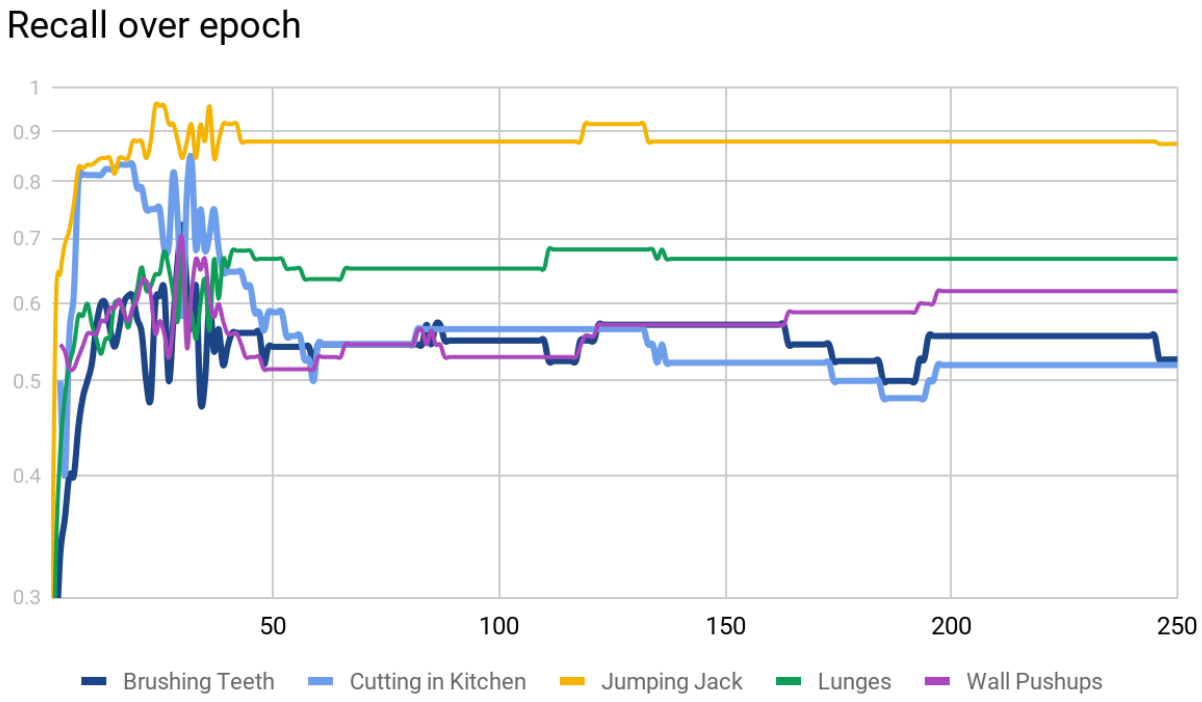
Single RGB Frame: Precision and Recall Graph





Optical Flow: Precision and Recall Graph





Different Network Topologies: Confusion Matrices

CONF_MATRIX	Brushing Teeth	Cutting in Kitchen	Jumping Jack	Lunges	Wall Pushups
Brushing Teeth	**34**	5	2	1	2
Cutting In Kitchen	12	**22**	2	4	3
Jumping Jack	0	0	**43**	2	0
Lunges	2	1	8	**33**	1
Wall Pushups	0	0	1	0	**46**

Table VII: Performance measure using 25% numbers of filters

CONF_MATRIX	Brushing Teeth	Cutting in Kitchen	Jumping Jack	Lunges	Wall Pushups
Brushing Teeth	**36**	5	0	0	3
Cutting In Kitchen	4	**32**	2	3	2
Jumping Jack	0	0	**44**	1	0
Lunges	0	2	7	**36**	0
Wall Pushups	0	0	1	0	**46**

Table VIII: Performance measure using 50% numbers of filters

CONF_MATRIX	Brushing Teeth	Cutting in Kitchen	Jumping Jack	Lunges	Wall Pushups
Brushing Teeth	**40**	2	0	0	2
Cutting In Kitchen	3	**36**	0	2	2
Jumping Jack	0	0	**44**	1	0
Lunges	1	2	7	**35**	0
Wall Pushups	0	0	0	0	**47**

Table IX: Performance measure using 200% numbers of filters

Different Network Topologies: Performance Matrices

PERF_MATRIX	Brushing Teeth	Cutting in Kitchen	Jumping Jack	Lunges	Wall Pushups
Precision (P)	0.77	0.51	0.96	0.73	0.98
Recall (R)	0.71	0.79	0.77	0.82	0.88
F-Score	0.74	0.62	0.85	0.78	0.93

Table X: Performance measure using 25% numbers of filters

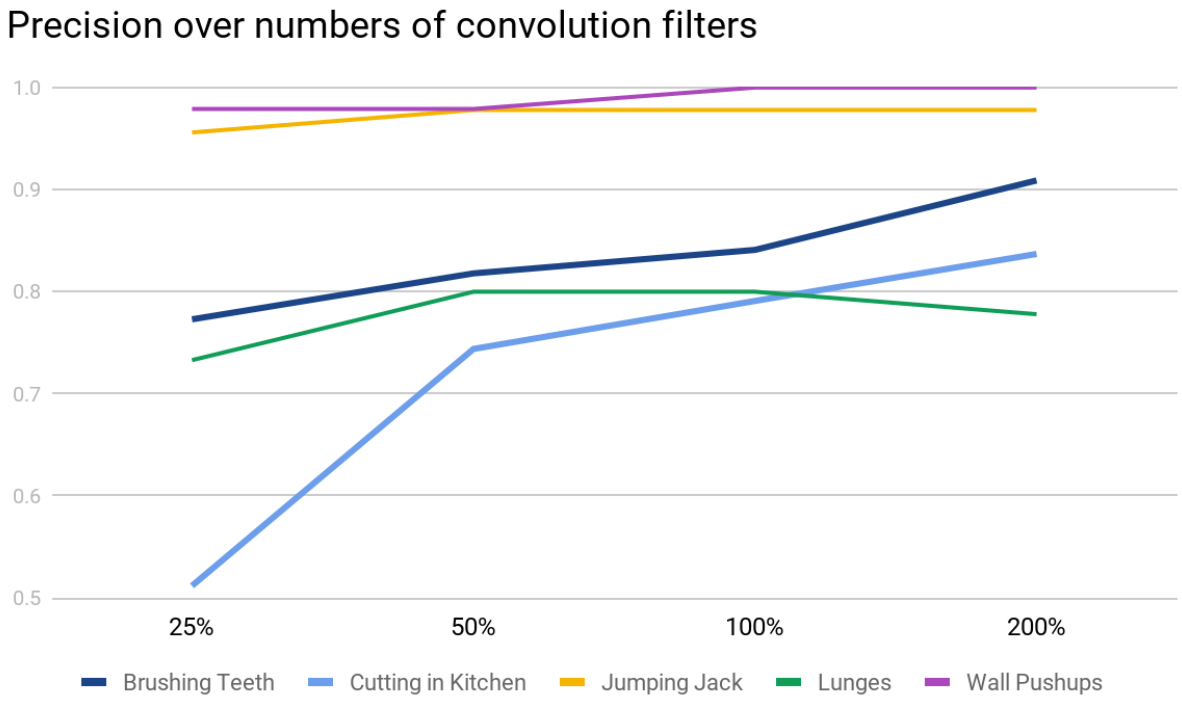
PERF_MATRIX	Brushing Teeth	Cutting in Kitchen	Jumping Jack	Lunges	Wall Pushups
Precision (P)	0.82	0.74	0.98	0.80	0.98
Recall (R)	0.90	0.82	0.81	0.90	0.90
F-Score	0.86	0.78	0.89	0.85	0.94

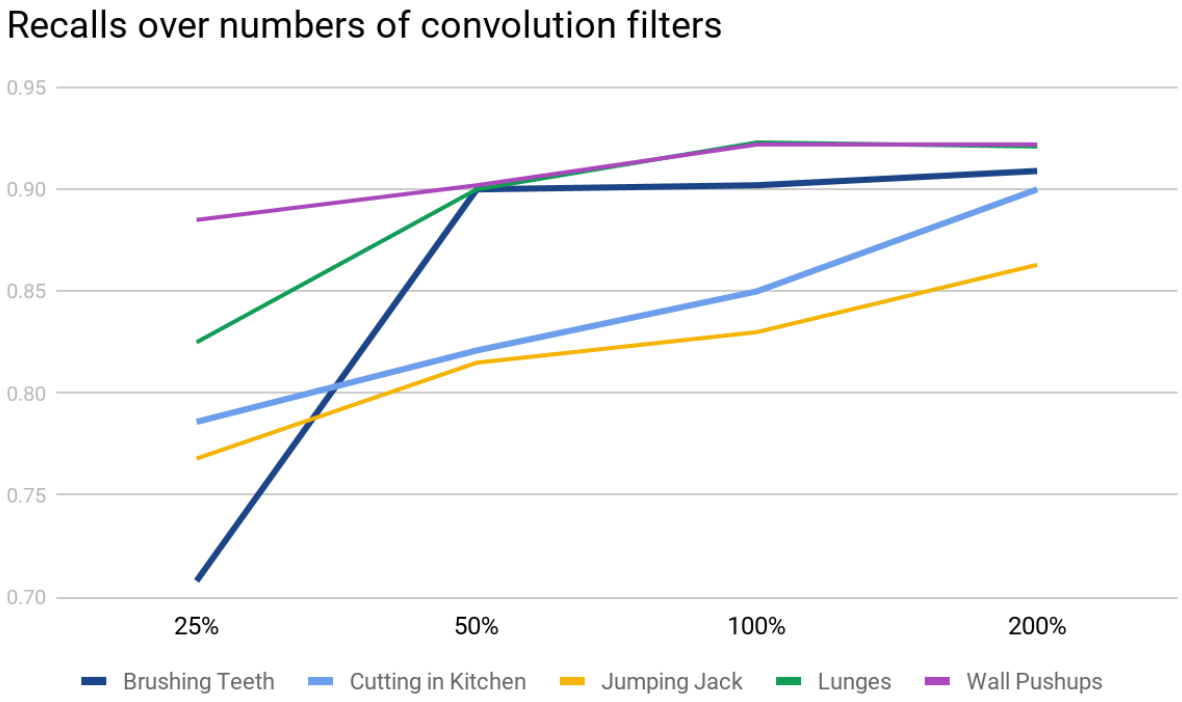
Table XI: Performance measure using 50% numbers of filters

PERF_MATRIX	Brushing Teeth	Cutting in Kitchen	Jumping Jack	Lunges	Wall Pushups
Precision (P)	0.90	0.84	0.98	0.78	1.00
Recall (R)	0.90	0.90	0.86	0.92	0.92
F-Score	0.90	0.87	0.92	0.84	0.96

Table XII: Performance measure using 200% numbers of filters

Different Network Topologies: Precision and Recall Graph





Points done for assignment 4:

- Cross validation
- Performance measurement
- Training a CNN
- Testing a video
- Test performance on your test sets and own recordings
- Test performance using optical flow instead of RGB-frames
- Test performance on different network topologies
- Train on mirrored frames
- Test performance using different sets of parameters

Instructions to run the program:

1. The action data folder should be in /data/ucf-101 and custom data should be in /data/own.
2. Run ***dataset_generator.py*** to generate middle frames and mirrored images.
3. 20% of the data should be handpicked and moved to folder /test/ucf-101 in their respective action folders. This will be used as cross validation.
4. Run ***train.py*** to train the convolutional neural network and generate performance measures each epoch.
5. Run ***predict.py*** to run predict unseen images from /test/ucf-101 using the train network and generate performance measure.