1. Introduction

The human brain consists of billions of neurons which are interconnected by synapses. If “enough” synaptic inputs to the neuron fires, then the neuron will also fire. This process is called thinking. To replicate that process on computers, we need machine learning and neural networks.

Machine learning

Quite simply, machine learning allows computers to ‘learn’. Traditionally, we always got computers to do things by providing a strict set of instructions. Machine Learning uses a very different approach. Instead of giving the computer a set of instructions on how to do something, we give it instructions on how to learn to do something. For example: think of a system that can classify pictures of animals as ‘cat’, ‘dog’, or ‘mouse’. Instead of manually finding unique characteristics from images of those animals and then coding it up, machine learning takes in images of those animals and finds characteristics and differences by itself. This process of teaching the computer is referred to as training.

Deep Learning

Deep learning is a Technique for implementing Machine Learning. It uses neural networks to learn, sometimes, using decision trees may also be referred to as deep learning, but for the most part deep learning involves the use of neural networks.

Object recognition and activity recognition has always fascinated the researchers. The project deals with detecting objects and actions in videos and real time. Action classification has been widely studied over the past decade and state-of-the-art methods now achieve excellent performance. However, to analyze video content in more detail, we need to localize actions in space and time. Detecting actions in videos is a challenging task which has received increasing attention over the past few years. Recently, significant progress has been achieved in supervised action localization, see for example. However these methods require a large amount of annotation, i.e., bounding box annotations in every frame. Such annotations are, for example, used to train Convolutional Neural Networks (CNNs) at the bounding box level. Several works have suggested to generate action proposals before classifying them however they generate hundreds of proposals for a video, thus supervision is still required to label them in order to train a classifier. Consequently, all these approaches require full supervision, where action localization needs to be annotated in every frame. This makes scaling up a large dataset difficult.

The project tries to use YOLO algorithm and CNN for action detection. Dataset is prepared and downloaded (as the case may be) and the model is trained for 300 epochs. After training the model is tested on videos and real time

2. Background

2.1.OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies.

2.2.Tensorflow

TensorFlow is a framework created by Google for creating Deep Learning models. Deep Learning is a category of machine learning models (=algorithms) that use multi-layer neural networks. Machine Learning has enabled us to build complex applications with great accuracy. Whether it has to do with images, videos, text or even audio, Machine Learning can solve problems from a wide range. Tensorflow can be used to achieve all of these applications. The reason for its popularity is the ease with which developers can build and deploy applications.

2.3.Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling

fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

2.3.1. Advantages of Keras:

2.3.1.1. User friendliness. Keras is an API designed for human beings, not machines. It puts user experience front and center. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.

2.3.1.2. Modularity. A model is understood as a sequence or a graph of standalone, fully-configurable modules that can be plugged together with as few restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes are all standalone modules that you can combine to create new models.

2.3.1.3. Easy extensibility. New modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.

2.3.1.4. Work with Python. No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

2.4. Introduction to Neural Networks

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurones) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurones. This is true of ANNs as well.

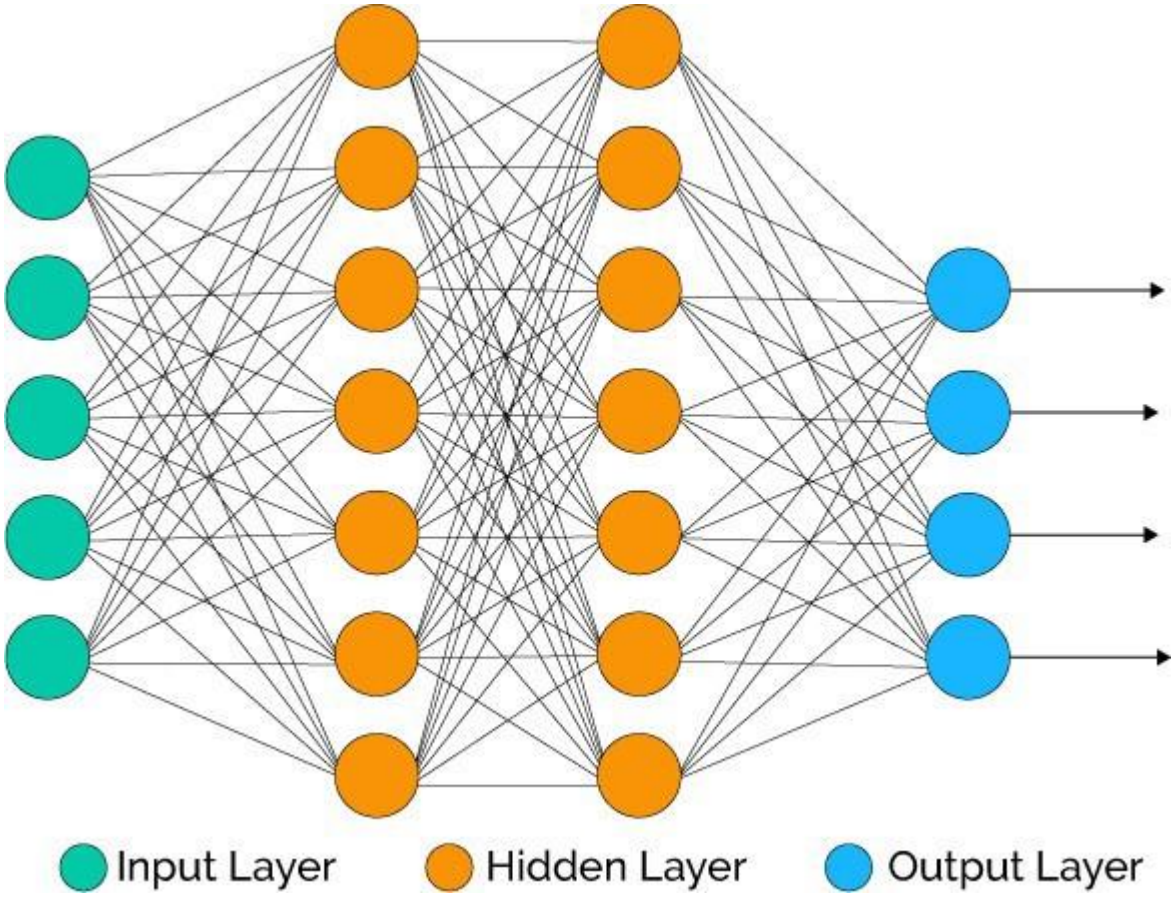
Neural network simulations appear to be a recent development. However, this field was established before the advent of computers, and has survived at least one major setback and several eras.

Many importand advances have been boosted by the use of inexpensive computer emulations. Following an initial period of enthusiasm, the field survived a period of frustration and disrepute. During this period when funding and professional support was minimal, important advances were made by relatively few reserchers. These pioneers were able to develop convincing technology which surpassed the limitations identified by Minsky and Papert. Minsky and Papert, published a book (in 1969) in which they summed up a general feeling of frustration (against neural networks) among researchers, and was thus accepted by most without further analysis. Currently, the neural network field enjoys a resurgence of interest and a corresponding increase in funding.

The first artificial neuron was produced in 1943 by the neurophysiologist Warren McCulloch and the logician Walter Pits. But the technology available at that time did not allow them to do too much.

Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyse. This expert can then be used to provide projections given new situations of interest and answer "what if" questions.
Other advantages include:

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
2. Self-Organisation: An ANN can create its own organisation or representation of the information it receives during learning time.
3. Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
4. Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage



2.5. Neural networks versus conventional computers

Neural networks take a different approach to problem solving than that of conventional computers. Conventional computers use an algorithmic approach i.e. the computer follows a set of instructions in order to solve a problem. Unless the specific steps that the computer needs to follow are known the computer cannot solve the problem. That restricts the problem solving capability of conventional computers to problems that we already understand and know how to solve. But computers would be so much more useful if they could do things that we don't exactly know how to do.

Neural networks process information in a similar way the human brain does. The network is composed of a large number of highly interconnected processing elements(neurones) working in parallel to solve a specific problem. Neural networks learn by example. They cannot be programmed to perform a specific task. The examples must be selected carefully otherwise useful time is wasted or even worse the network might be functioning incorrectly. The disadvantage is that because the network finds out how to solve the problem by itself, its operation can be unpredictable.

On the other hand, conventional computers use a cognitive approach to problem solving; the way the problem is to solved must be known and stated in small

unambiguous instructions. These instructions are then converted to a high level language program and then into machine code that the computer can understand. These machines are totally predictable; if anything goes wrong is due to a software or hardware fault.

Neural networks and conventional algorithmic computers are not in competition but complement each other. There are tasks are more suited to an algorithmic approach like arithmetic operations and tasks that are more suited to neural networks. Even more, a large number of tasks, require systems that use a combination of the two approaches (normally a conventional computer is used to supervise the neural network) in order to perform at maximum efficiency.

2.6. Architecture of neural networks

2.6.1 Feed-forward networks

Feed-forward ANNs (figure 1) allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any layer does not affect that same layer. Feed-forward ANNs tend to be straight forward networks that associate inputs with outputs. They are extensively used in pattern recognition. This type of organisation is also referred to as bottom-up or top-down.

2.6.2 Feedback networks

Feedback networks (figure 1) can have signals travelling in both directions by introducing loops in the network. Feedback networks are very powerful and can get extremely complicated. Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found. Feedback architectures are also referred to as interactive or recurrent, although the latter term is often used to denote feedback connections in single-layer organisations.

2.6.3 Network layers

The commonest type of artificial neural network consists of three groups, or layers, of units: a layer of "**input**" units is connected to a layer of "**hidden**" units, which is connected to a layer of **"output**" units. (see Figure 4.1)

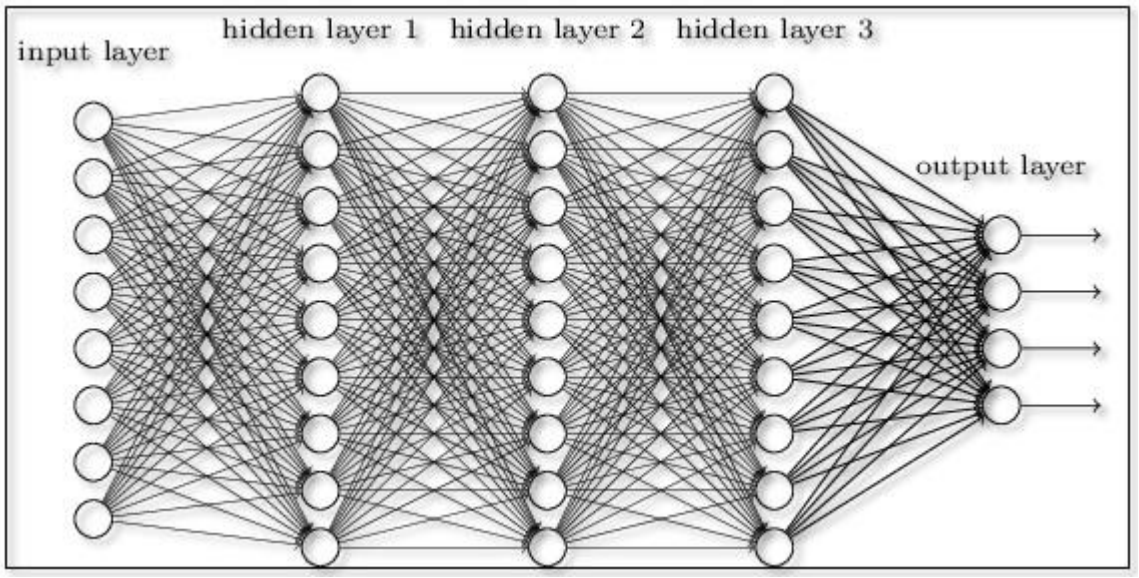
The activity of the input units represents the raw information that is fed into the network.

The activity of each hidden unit is determined by the activities of the input units and the weights on the connections between the input and the hidden units.

The behaviour of the output units depends on the activity of the hidden units and the weights between the hidden and output units.

This simple type of network is interesting because the hidden units are free to construct their own representations of the input. The weights between the input and hidden units determine when each hidden unit is active, and so by modifying these weights, a hidden unit can choose what it represents.

We also distinguish single-layer and multi-layer architectures. The single-layer organisation, in which all units are connected to one another, constitutes the most general case and is of more potential computational power than hierarchically structured multi-layer organisations. In multi-layer networks, units are often numbered by layer, instead of following a global numbering.



2.7. Applications of Neural Networks

Given this description of neural networks and how they work, what real world applications are they suited for? Neural networks have broad applicability to real world business problems. In fact, they have already been successfully applied in many industries.

Since neural networks are best at identifying patterns or trends in data, they are well suited for prediction or forecasting needs including:

sales forecasting

industrial process control

customer research

data validation

risk management

target marketing

But to give you some more specific examples; ANN are also used in the following specific paradigms: recognition of speakers in communications; diagnosis of hepatitis; recovery of telecommunications from faulty software; interpretation of multimeaning Chinese words; undersea mine detection; texture analysis; three-dimensional object recognition; hand-written word recognition; and facial recognition.

Neural networks in medicine

Artificial Neural Networks (ANN) are currently a 'hot' research area in medicine and it is believed that they will receive extensive application to biomedical systems in the next few years. At the moment, the research is mostly on modelling parts of the human body and recognising diseases from various scans (e.g. cardiograms, CAT scans, ultrasonic scans, etc.).

Neural networks are ideal in recognising diseases using scans since there is no need to provide a specific algorithm on how to identify the disease. Neural networks learn by example so the details of how to recognise the disease are not needed. What is needed is a set of examples that are representative of all the variations of the disease. The quantity of examples is not as important as the 'quantity'. The examples need to be selected very carefully if the system is to perform reliably and efficiently.

Modelling and Diagnosing the Cardiovascular System

Neural Networks are used experimentally to model the human cardiovascular system. Diagnosis can be achieved by building a model of the cardiovascular system of an individual and comparing it with the real time physiological measurements taken from the patient. If this routine is carried out regularly, potential harmful medical conditions can be detected at an early stage and thus make the process of combating the disease much easier.

A model of an individual's cardiovascular system must mimic the relationship among physiological variables (i.e., heart rate, systolic and diastolic blood pressures, and breathing rate) at different physical activity levels. If a model is adapted to an individual, then it becomes a model of the physical condition of that individual. The simulator will have to be able to adapt to the features of any individual without the supervision of an expert. This calls for a neural network.

Another reason that justifies the use of ANN technology, is the ability of ANNs to provide sensor fusion which is the combining of values from several different sensors. Sensor fusion enables the ANNs to learn complex relationships among the individual sensor values, which would otherwise be lost if the values were individually analysed. In medical modelling and diagnosis, this implies that even though each sensor in a set may be sensitive only to a specific physiological variable, ANNs are capable of detecting complex medical conditions by fusing the data from the individual biomedical sensors.

Electronic noses

ANNs are used experimentally to implement electronic noses. Electronic noses have several potential applications in telemedicine. Telemedicine is the practice of medicine over long distances via a communication link. The electronic nose would identify odours in the remote surgical environment. These identified odours would then be electronically transmitted to another site where an door generation system would recreate them. Because the sense of smell can be an important sense to the surgeon, telesmell would enhance telepresent surgery.

Instant Physician

An application developed in the mid-1980s called the "instant physician" trained an autoassociative memory neural network to store a large number of medical records, each of which includes information on symptoms, diagnosis, and treatment for a particular case. After training, the net can be presented with input consisting of a set of symptoms; it will then find the full stored pattern that represents the "best" diagnosis and treatment.

Neural Networks in business

Business is a diverted field with several general areas of specialisation such as accounting or financial analysis. Almost any neural network application would fit into one business area or financial analysis.
There is some potential for using neural networks for business purposes, including resource allocation and scheduling. There is also a strong potential for using neural networks for database mining, that is, searching for patterns implicit within the explicitly stored information in databases. Most of the funded work in this area is classified as proprietary. Thus, it is not possible to report on the full extent of the work going on. Most work is applying neural networks, such as the Hopfield-Tank network for optimization and scheduling.

2.8. Convolutional Neural Network

2.8.1. Introduction to CNN

Although neural networks can be applied to computer vision tasks, to get good generalization performance, it is beneficial to incorporate prior knowledge into the network architecture [LeC89]. Convolutional neural networks aim to use spatial information between the pixels of an image. Therefore, they are based on discrete convolution.

Convolutional neural networks are deep artificial neural networks that are used primarily to classify images (e.g. name what they see), cluster them by similarity (photo search), and perform object recognition within scenes. They are algorithms

that can identify faces, individuals, street signs, tumors, platypuses and many other aspects of visual data.

Convolutional networks perform optical character recognition (OCR) to digitize text and make natural-language processing possible on analog and hand-written documents, where the images are symbols to be transcribed. CNNs can also be applied to sound when it is represented visually as a spectrogram. More recently, convolutional networks have been applied directly to text analytics as well as graph data with graph convolutional networks.

The efficacy of convolutional nets (ConvNets or CNNs) in image recognition is one of the main reasons why the world has woken up to the efficacy of deep learning. They are powering major advances in computer vision (CV), which has obvious applications for self-driving cars, robotics, drones, security, medical diagnoses, and treatments for the visually impaired.

The first thing to know about convolutional networks is that they don't perceive images like humans do. Therefore, you are going to have to think in a different way about what an image means as it is fed to and processed by a convolutional network.

Convolutional networks perceive images as volumes; i.e. three-dimensional objects, rather than flat canvases to be measured only by width and height. That's because digital color images have a red-blue-green (RGB) encoding, mixing those three colors to produce the color spectrum humans perceive. A convolutional network ingests such images as three separate strata of color stacked one on top of the other.

So a convolutional network receives a normal color image as a rectangular box whose width and height are measured by the number of pixels along those dimensions, and whose depth is three layers deep, one for each letter in RGB. Those depth layers are referred to as *channels*.

As images move through a convolutional network, we will describe them in terms of input and output volumes, expressing them mathematically as matrices of multiple dimensions in this form: 30x30x3. From layer to layer, their dimensions change for reasons that will be explained below.

You will need to pay close attention to the precise measures of each dimension of the image volume, because they are the foundation of the linear algebra operations used to process images.

Now, for each pixel of an image, the intensity of R, G and B will be expressed by a number, and that number will be an element in one of the three, stacked two-dimensional matrices, which together form the image volume.

Those numbers are the initial, raw, sensory features being fed into the convolutional network, and the ConvNets purpose is to find which of those numbers are significant signals that actually help it classify images more accurately. (Just like other feedforward networks)

Rather than focus on one pixel at a time, a convolutional net takes in square patches of pixels and passes them through a *filter*. That filter is also a square matrix smaller than the image itself, and equal in size to the patch. It is also called a *kernel*, which

will ring a bell for those familiar with support-vector machines, and the job of the filter is to find patterns in the pixels.

2.8.2 Examples

Imagine two matrices. One is 30x30, and another is 3x3. That is, the filter covers one-hundredth of one image channel's surface area.

We are going to take the dot product of the filter with this patch of the image channel. If the two matrices have high values in the same positions, the dot product's output will be high. If they don't, it will be low. In this way, a single value – the output of the dot product – can tell us whether the pixel pattern in the underlying image matches the pixel pattern expressed by our filter.

Let's imagine that our filter expresses a horizontal line, with high values along its second row and low values in the first and third rows. Now picture that we start in the upper lefthand corner of the underlying image, and we move the filter across the image step by step until it reaches the upper righthand corner. The size of the step is known as *stride*. You can move the filter to the right one column at a time, or you can choose to make larger steps.

At each step, you take another dot product, and you place the results of that dot product in a third matrix known as an *activation map*. The width, or number of columns, of the activation map is equal to the number of steps the filter takes to traverse the underlying image. Since larger strides lead to fewer steps, a big stride will produce a smaller activation map. This is important, because the size of the matrices that convolutional networks process and produce at each layer is directly proportional to how computationally expensive they are and how much time they take to train. A larger stride means less time and compute.

A filter superimposed on the first three rows will slide across them and then begin again with rows 4-6 of the same image. If it has a stride of three, then it will produce a matrix of dot products that is 10x10. That same filter representing a horizontal line can be applied to all three channels of the underlying image, R, G and B. And the three 10x10 activation maps can be added together, so that the aggregate activation map for a horizontal line on all three channels of the underlying image is also 10x10.

Now, because images have lines going in many directions, and contain many different kinds of shapes and pixel patterns, you will want to slide other filters across the underlying image in search of those patterns. You could, for example, look for 96 different patterns in the pixels. Those 96 patterns will create a stack of 96 activation maps, resulting in a new volume that is 10x10x96. In the diagram below, we've relabeled the input image, the kernels and the output activation maps to make sure we're clear.

The next layer in a convolutional network has three names: max pooling, downsampling and subsampling. The activation maps are fed into a downsampling layer, and like convolutions, this method is applied one patch at a time. In this case, max pooling simply takes the largest value from one patch of an image, places it in a new matrix next to the max values from other patches, and discards the rest of the information contained in the activation maps. Only the locations on the image that

showed the strongest correlation to each feature (the maximum value) are preserved, and those maximum values combine to form a lower-dimensional space.

Much information about lesser values is lost in this step, which has spurred research into alternative methods. But downsampling has the advantage, precisely because information is lost, of decreasing the amount of storage and processing required.

2.8.3. Process

The process involves:

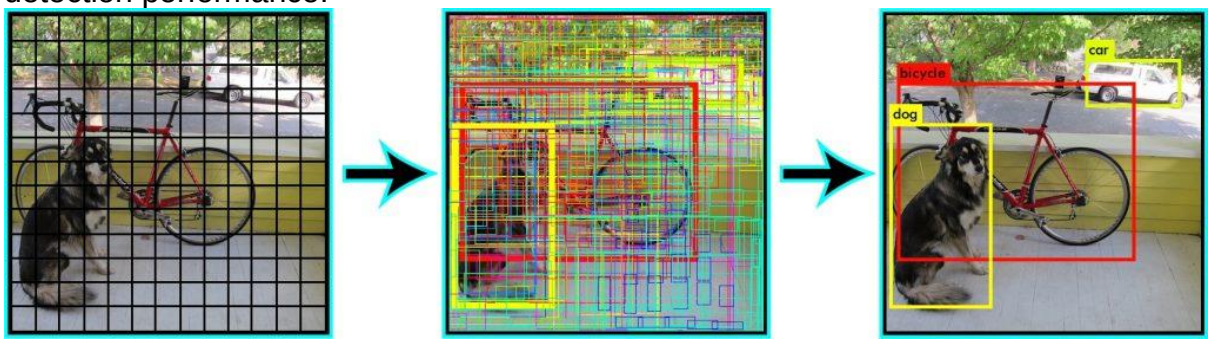
- The actual input image that is scanned for features. The light rectangle is the filter that passes over it.
- Activation maps stacked atop one another, one for each filter you employ. The larger rectangle is one patch to be downsampled.
- The activation maps condensed through downsampling.
- A new set of activation maps created by passing filters over the first downsampled stack.
- The second downsampling, which condenses the second set of activation maps.
- A fully connected layer that classifies output with one label per node.

As more and more information is lost, the patterns processed by the convolutional net become more abstract and grow more distant from visual patterns we recognize as humans. So forgive yourself, and us, if convolutional networks do not offer easy intuitions as they grow deeper.

2.9. YOLO

YOLO stands for You Only Look Once which is an object detection algorithm.

Object detection is a single regression problem involving image pixels, bounding box coordinates, anchor boxes and class probabilities.
A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance.



2.9.1 Advantages of Yolo

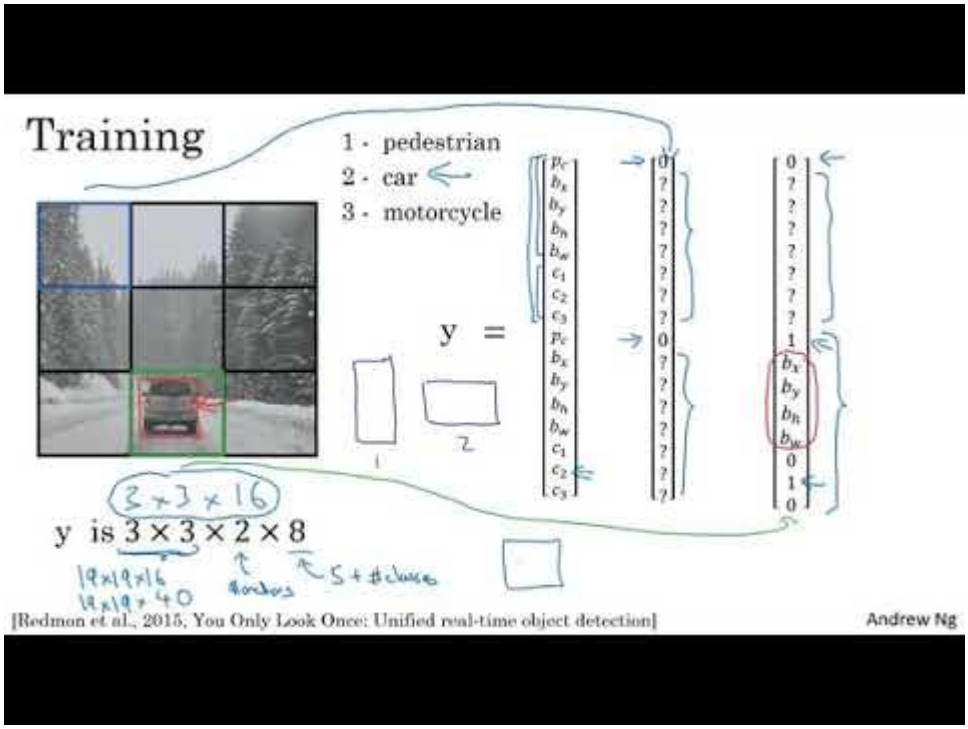
This unified model has several benefits over traditional methods of object detection.

- First, YOLO is extremely fast.
- Since we frame detection as a regression problem we don't need a complex pipeline. We simply run our neural network on a new image at test time to predict detections. Our base network runs at 45 frames per second with no batch processing on a Titan X GPU and a fast version runs at more than 150 fps. This means we can process streaming video in real-time with less than 25 milliseconds of latency.
- YOLO reasons globally about the image when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance. Fast R-CNN, a top detection method, mistakes background patches in an image for objects because it can't see the larger context. YOLO makes less than half the number of background errors compared to Fast R-CNN.
- YOLO learns generalizable representations of objects. When trained on natural images and tested on artwork, YOLO outperforms top detection methods like DPM and R-CNN by a wide margin.
- Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected inputs.

2.9.2 Theory

Our network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously. This means our network reasons globally about the full image and all the objects in the image. The YOLO design enables end-to-end training and realtime speeds while maintaining high average precision.

Our system divides the input image into an S × S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object.
Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. Formally we define confidence as $\Pr(\text{Object}) * \text{IOU}$.
If no object exists in that cell, the confidence scores should be zero. Otherwise we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth



Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally the confidence prediction represents the IOU between the predicted box and any ground truth box. Each grid cell also predicts C conditional class probabilities, Pr(Classi |Object). These probabilities are conditioned on the grid cell containing an object. We only predict one set of class probabilities per grid cell, regardless of the number of boxes B.
At test time we multiply the conditional class probabilities and the individual box confidence predictions,
, which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.

Suppose we try to train three objects pedestrian, car and motorcycle. If you are using 2 anchor boxes and a 3*3 image then output volume will be 3*3*2*8=144

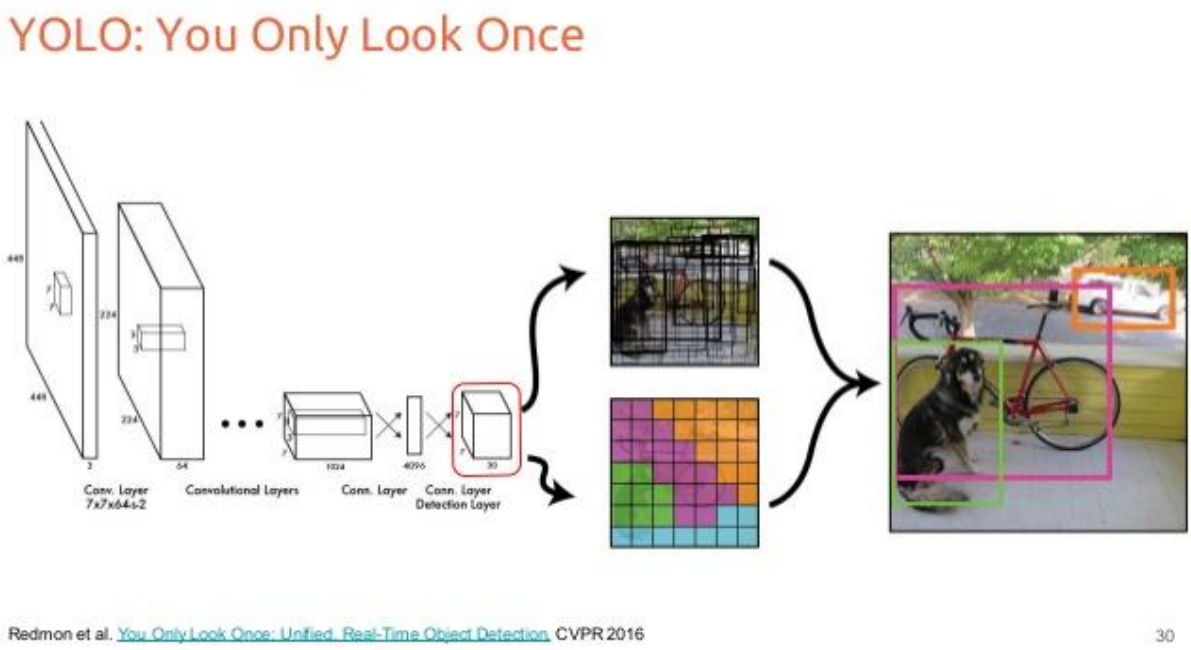
8 because each bounding box has 8 predictions. First is if an object is present or not, if not then all other not matter and can be don't care. Whereas if an object is present then the other predictions are the (x, y) coordinates that represent the centre of the box relative to the bounds of the grid cell, The width and height are predicted relative to the whole image and last three tell what is the object.

For each grid cell we get two bounding boxes and some will have really low probability. So then we get rid of low probability predictions. For each class use non max suppression to generate final class.

Non max suppression suppresses the lower probability bounding box and outputs the bounding box with highest probability. Discard all the bounding boxes with IOU>=0.5 with the box output.

The image is converted to final matrix which is then converted to the output volume using CovNet.

YOLO imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class. This spatial constraint limits the number of nearby objects that our model can predict. Our model struggles with small objects that appear in groups, such as flocks of birds. Since our model learns to predict bounding boxes from data, it struggles to generalize to objects in new or unusual aspect ratios or configurations. Our model also uses relatively coarse features for predicting bounding boxes since our architecture has multiple downsampling layers from the input image. Finally, while we train on a loss function that approximates detection performance, our loss function treats errors the same in small bounding boxes versus large bounding boxes. A small error in a large box is generally benign but a small error in a small box has a much greater effect on IOU. Our main source of error is incorrect localizations.



Error analysis of YOLO compared to Fast R-CNN shows that YOLO makes a significant number of localization errors. Furthermore, YOLO has relatively low recall compared to region proposal-based methods. Thus we focus mainly on improving recall and localization while maintaining classification accuracy.

2.9.3 YOLOv2

By adding batch normalization on all of the convolutional layers in YOLO we get more than 2% improvement in mAP. Batch normalization also helps regularize the model. With batch normalization we can remove dropout from the model without overfitting.

For YOLOv2 we first fine tune the classification network at the full 448 × 448 resolution for 10 epochs on ImageNet. This gives the network time to adjust its filters to work better on higher resolution input. We then fine tune the resulting network on detection. This high resolution classification network gives us an increase of almost 4% mAP

Convolutional With Anchor Boxes. YOLO predicts the coordinates of bounding boxes directly using fully connected layers on top of the convolutional feature extractor. Predicting offsets instead of coordinates simplifies the problem and makes it easier for the network to learn. We remove the fully connected layers from YOLO and use anchor boxes to predict bounding boxes. Using anchor boxes we get a small decrease in accuracy. YOLO only predicts 98 boxes per image but with anchor boxes our model predicts more than a thousand. Without anchor boxes our intermediate model gets 69.5 mAP with a recall of 81%. With anchor boxes our model gets 69.2 mAP with a recall of 88%. Even though the mAP decreases, the increase in recall means that our model has more room to improve.

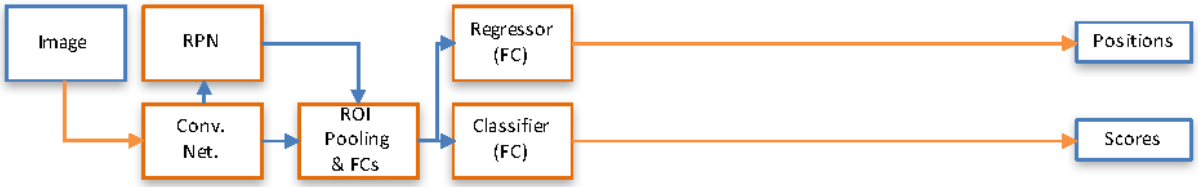
Fine-Grained Features.This modified YOLO predicts detections on a 13 × 13 feature map. While this is suffi- cient for large objects, it may benefit from finer grained features for localizing smaller objects. Faster R-CNN and SSD both run their proposal networks at various feature maps in the network to get a range of resolutions. We take a different approach, simply adding a passthrough layer that brings features from an earlier layer at 26 × 26 resolution. The passthrough layer concatenates the higher resolution features with the low resolution features by stacking adjacent features into different channels instead of spatial locations, similar to the identity mappings in ResNet. This turns the 26 × 26 × 512 feature map into a 13 × 13 × 2048 feature map, which can be concatenated with the original features. Our detector runs on top of this expanded feature map so that it has access to fine grained features. This gives a modest 1% performance increase.

During training we mix images from both detection and classification datasets. When our network sees an image labelled for detection we can backpropagate based on the full YOLOv2 loss function. When it sees a classification image we only backpropagate loss from the classification specific parts of the architecture.

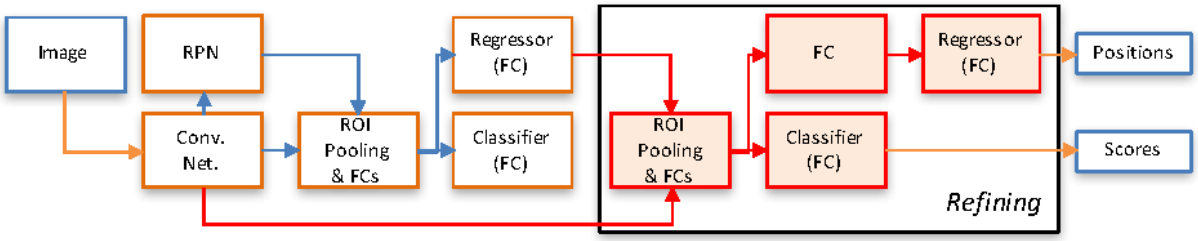
YOLOv2, is state-of-the-art on standard detection tasks like PASCAL VOC and COCO. At 67 FPS, YOLOv2 gets 76.8 mAP on VOC 2007. At 40 FPS, YOLOv2 gets 78.6 mAP, outperforming state-of-the-art methods like Faster RCNN with ResNet and SSD while still running significantly faster.

2.10. Faster RCNN

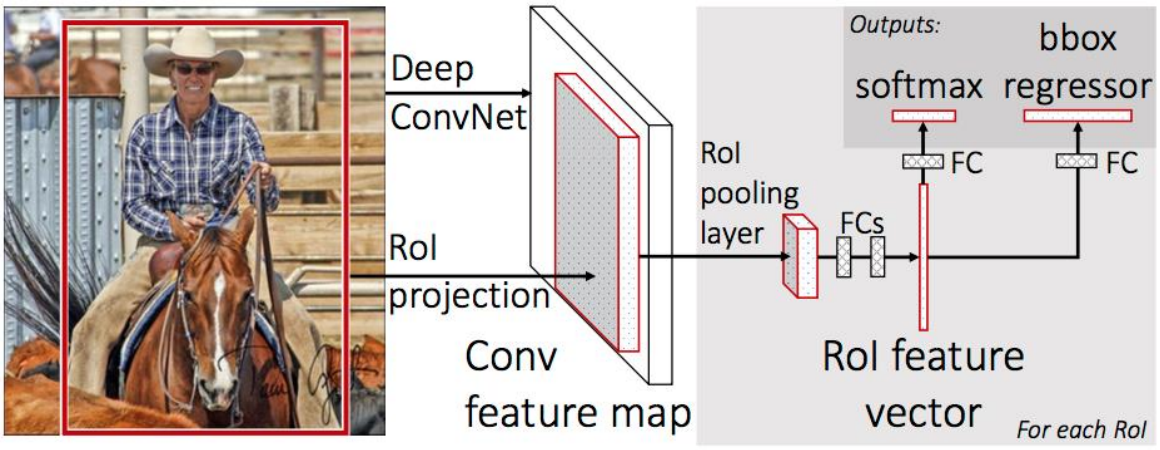
Faster RCNN is an extension of the R-CNN and Fast R-CNN object detection techniques. All three of these techniques use convolutional neural networks (CNN). The difference between them is how they select regions to process and how those regions are classified. R-CNN and Fast R-CNN use a region proposal algorithm as a pre-processing step before running the CNN. The proposal algorithms are typically techniques such as EdgeBoxes or Selective Search, which are independent of the CNN. In the case of Fast R-CNN, the use of these techniques becomes the processing bottleneck compared to running the CNN. Faster R-CNN addresses this issue by implementing the region proposal mechanism using the CNN and thereby making region proposal a part of the CNN training and prediction steps.



(a) Faster-RCNN



(b) Refining Faster-RCNN



3. Research so far

3.1. Video Classification

Convolutional Neural Networks (CNN) have been adopted widely for image classification problems. As CNNs demonstrate significant success in learning powerful and interpretable image features, more and more researchers start to deploy CNN on video classification problems. The main challenge is to capture not only the appearance information present in single, static frames, but also complex temporal evolution.

Among various video classification tasks, human action recognition is the key problem due to its wide application in surveillance camera, robotics and human computer interface (HCI) Robustly classifying real videos (UCF-101) into arbitrary free-form activities is a challenging task mainly because of background clutter, viewpoint changes, and drastically diverse dynamics in the observed motion . Existing research has shown that a two-stream approach, namely spatial stream plus temporal stream performed significantly better than training on raw stacked frames. This is inspired by two-stream hypothesis -- the human visual cortex contains two pathways: the ventral stream (which performs object recognition in the scene) and the dorsal stream (which recognizes motion). In order to get rid of the background noise and represent the action dynamics, we propose to detect and segment out human action from the background scene.
In this project, we investigate how single-shot object detection method can be integrated into the existing two-stream video classification pipeline to leverage its performance. In particular, we adopt You Only Look Once (YOLO) object detection approach to localize human action. Specifically, our two-stream architecture includes one spatial stream which performs action recognition from still video frames, and one temporal stream where adaptive cropping windows are generated by the fine-tuned YOLO detector and frames are cropped and stacked to represent the action from motion. Both streams are then fed into two separate CNNs and further fused together to get the classification result.

3.2 Related work

Classic video classification approach usually involves three major steps: First, extracting local visual features that describe a region of the video, either densely or as a sparse set of interest points. Second, the extracted features are encoded into high-level local spatial-temporal video descriptions . Lastly, a classifier (such as an SVM or Softmax) is trained on the resulting representation to cast the final class prediction.

There has also been a number of attempts for developing a deep architecture for video recognition. Karpathy and his colleagues were one of the first pioneers in using deep convolutional neural networks for video classification. They suggested a

multiresolution, foveated two-stream architecture. Input frames are fed into two separate streams of processing: a low-resolution full image frame and a high-resolution center crop. Their key assumption was that main information of the video was captured in the center of the frame. In addition, this method directly feeds video frames into CNN without considering the motion between consecutive images. Therefore, model is expected to implicitly learn spatial-temporal motion-dependent features, which can be a difficult task.

Subsequently, a research that addressed the shortcomings mention before is Two-Stream Convolutional Networks for Action Recognition in Videos. In addition to a spatial stream, which operates on original video frames

they developed a temporal stream using Optical Flow. Optical flow is defined as a set of displacement vector fields between pairs of consecutive frames. Essentially, optical flow represents the difference between frames and captures motion-dependent features. However, this approach is susceptive to changes in camera orientation and position. Nevertheless, this study gave us fresh thoughts for extracting motion from videos.

A state-of-the-art research on this topic is two-stream semantic region based CNNs (SR-CNNs) .They leveraged existing human/object detectors and proposed an architecture which leverages semantic cues (e.g. scene, person and object) for action understanding. Specifically, for an image that has more than one person, they tried to capture the “actor” rather than “bystanders”. They also described how they recovered missing detection results in individual frames and refined locations of bounding boxes. This is a very thought-provoking project that we look up to. Wang and his colleagues demonstrated that, for current action recognition benchmarks, scene context acts as a very strong cue for action recognition. Incorporating human action using R-CNNs has given action recognition accuracy a significant boost. However, approaches like R-CNN includes complex pipeline for generating potential bounding boxes in an image, running a classifier on these proposed boxes and post-processing to refine these boxes. This is slow and hard to optimize as each of these components must be trained separately. Thus, we propose to use YOLO to provide semantic information about the activities portrait in the videos over time. YOLO was first proposed by Joseph Redmon etc. in 2016 where they construct object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. In the YOLO framework, a single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation, which leads to extremely fast detection speed. However, YOLO also suffers from a variety of shortcomings such as high localization errors and low recalls compared to state-of-the-art detection systems. In 2017 YOLO evolves into its second version YOLOv2 (YOLO 9000) by pooling a variety of techniques such as batch normalization, dimension clusters, direct location prediction and multi-scale training [10]. In this project, we integrate fine-tuned YOLOv2 detector into our two-stream CNN architecture, generating bounding boxes to localize the human action in the temporal stream.

3.3 Fully-supervised action localization

Initial attempts for temporal and spatio-temporal action localization are based on a sliding-window scheme and focus on improving the search complexity. Other approaches rely on figure-centric models. For instance, consider the human position as a latent variable and infer it jointly with the action label.
Use a human detector and build human tracks using KLT features tracks. The human tracks are then classified with HOG-3D descriptors. Our approach is also based on human tracks but is significantly more robust to huge variations in pose and appearance.
Several recent methods for action localization are based on action proposals to reduce the search complexity. Build action localization candidates by hierarchically merging supervoxels and use dense trajectory features for tube classification. Similarly, van Gemert et al.cluster trajectories and use the resulting tubes for action detection.
In parallel, several works have attempted to further improve the quality of tubes. Most of these methods generate thousands of proposals for a short video and require ground-truth annotations to label the proposals in order to learn a proposal classifier. In contrast, our approach relies on only a few human tube proposals per video and can, thus, reduce spatial supervision to one annotated frame without drop of performance. Recently, CNNs for human action localization have merged. These approaches rely on RCNNs for both appearance and motion, classifying region proposals in individual frames. Detection tubes are obtained by combining class-specific detections with either temporal linking based on proximity [6], or with a class specific tracking-by-detection approach. Both strategies need to be run independently for each action. State-of-the heart approaches rely on Faster R-CNN trained on appearance and flow. Note that all these methods make extensive use of bounding box annotations in every frame for training.

3.4 UCF101

We use UCF-101 Human Action dataset [6] as it offers a good balance between number of action classes and the variety of actions. Besides, it is adopted in almost all research papers in the video classification area. This dataset contains 101 action classes. Each action class consists of 25 groups and each group has 4 to 7 video clips. The whole dataset contains 13,320 video clips. The clips in one group share some common features, such as back-ground or actors. The mean clip length is 7.21 sec and the total duration is 1600 mins. The min clip length is 1.06 sec and the max clip length is 71.04 sec. All clips have fixed frame rate and resolution of 25 FPS and 320×240 respectively. We pre-process all the videos, discretized into frames of images at 5FPS. We further split the entire UCF-101 dataset into train and test set by a ratio of 4:1 -- in each action classes, the first 20 groups of videos belong to training set and the other 5 groups belong to test set.

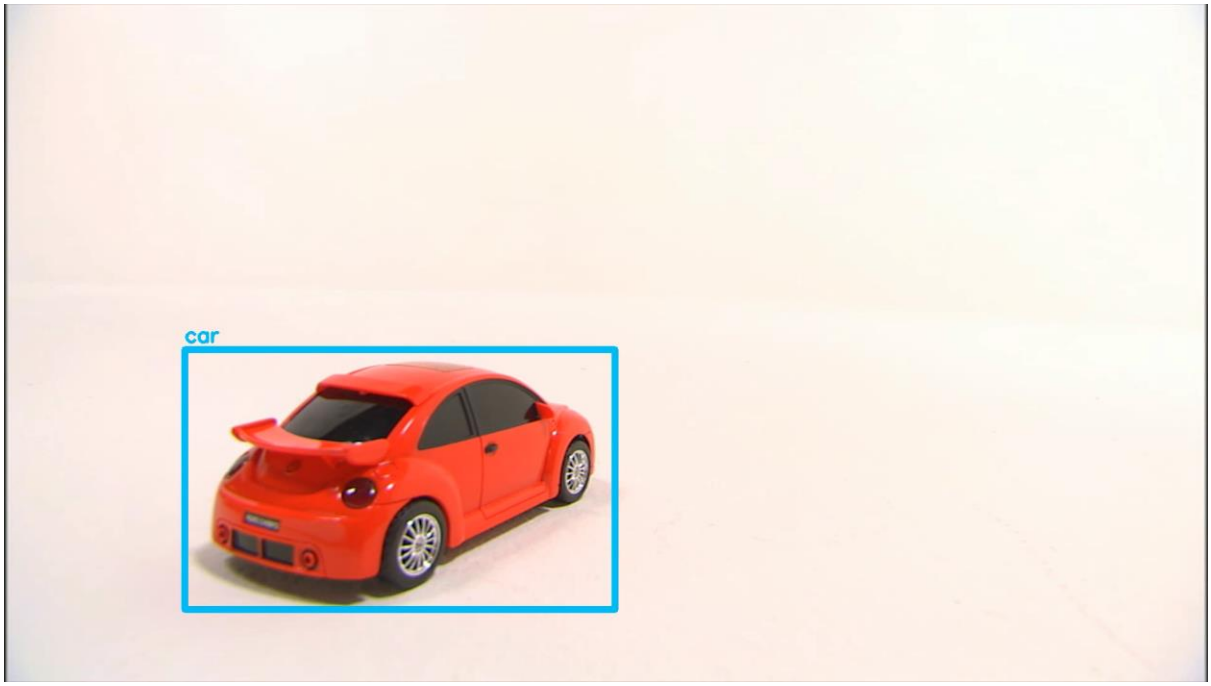
4. Object Detection using YOLOv2

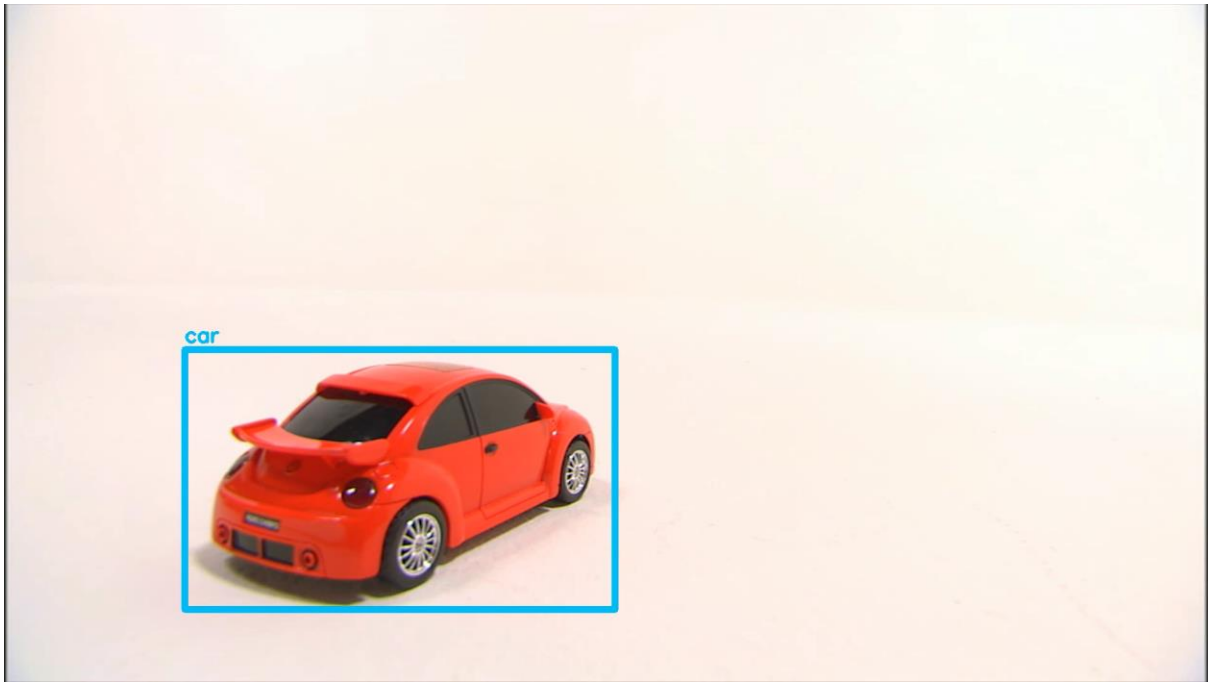
4.1. Object detection using command prompt

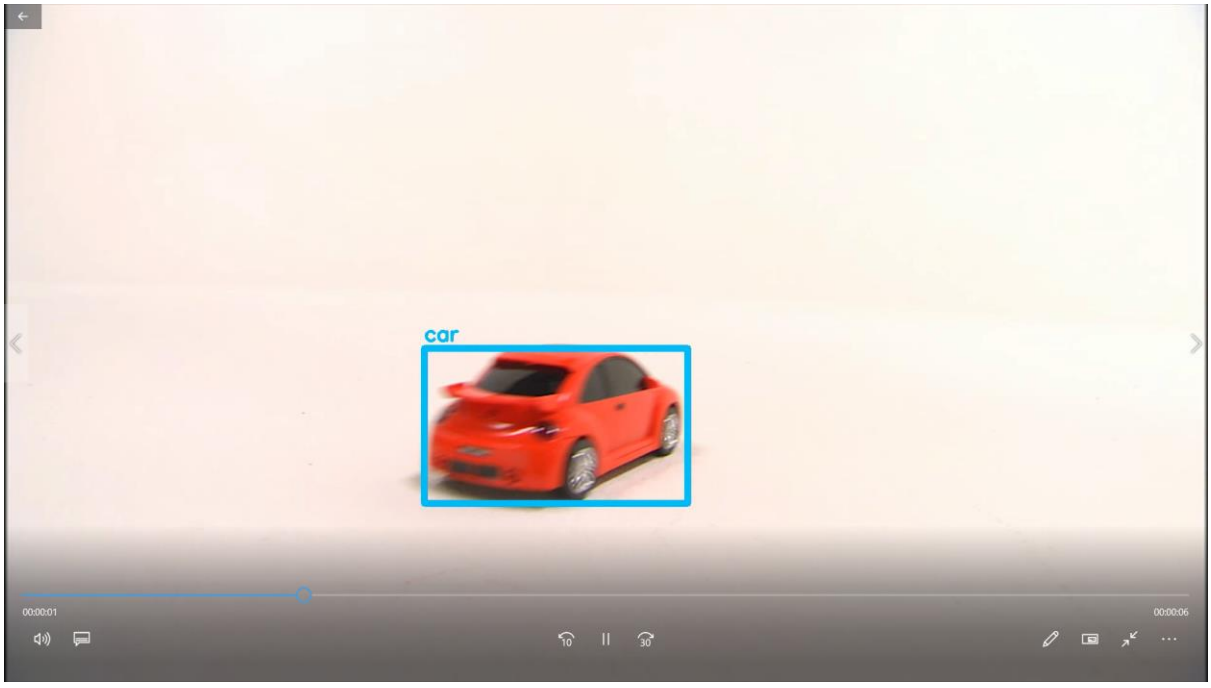
4.1.1. Steps

1) Install tensorflow and opencv.
2) Setup YOLO-v2 (using DarkFlow) and render a video clip with bounding boxes and labels
3) Create a directory of darkflow
4) useBuild the library :Open command prompt in the darkflow folder and use pip install –e.
5) Download the yolo weights and save them in bin folder made in darkfow directory.
6) Save any video in the dark flow directory and open a cmd window and use python flow --model cfg/yolo.cfg --load bin/yolov2.weights --demo Toy_car.mov –saveVideo

4.1.2 Results

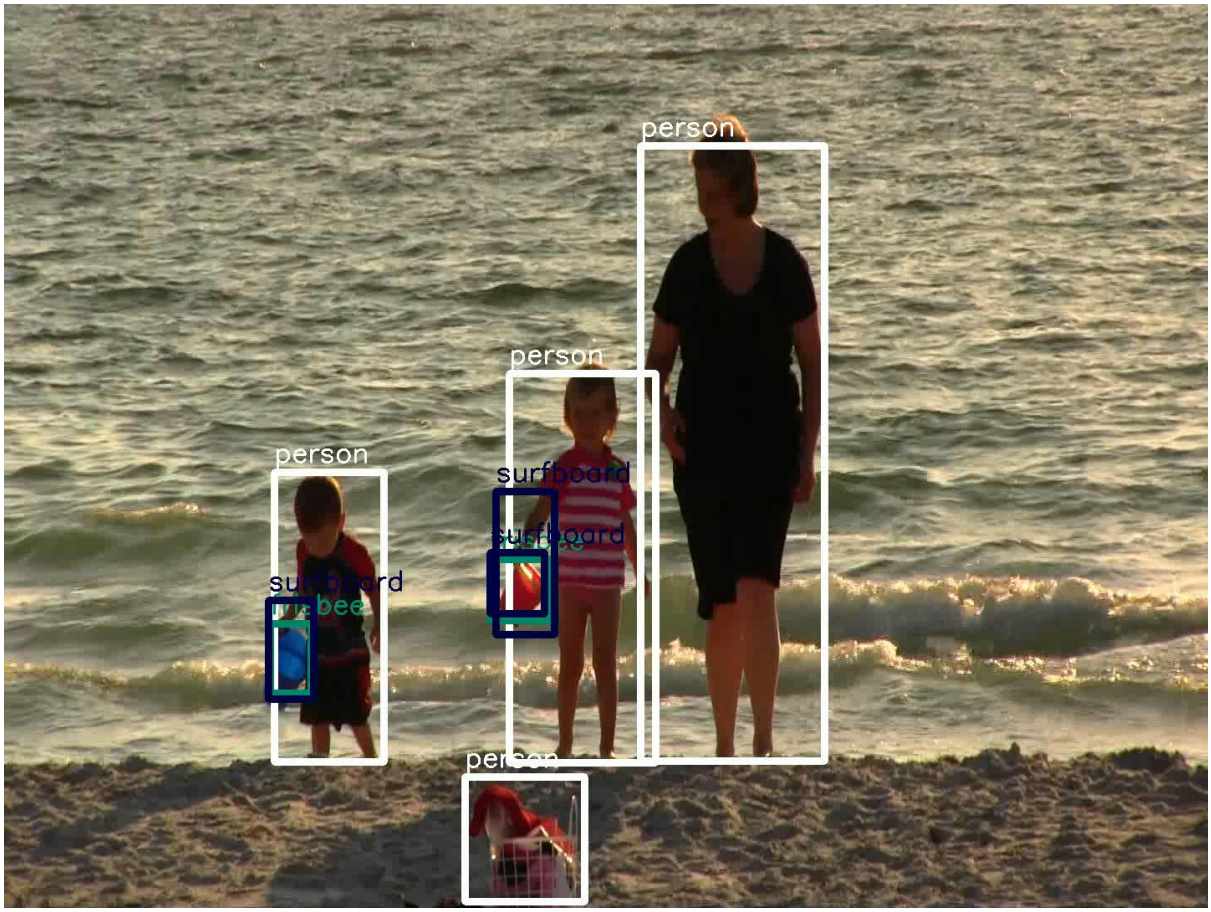






The bounding boxes (rectangles with blue border) show object detection in videos.

(Input Image)



(Output Image)

4.2. Object detection using darkflow

4.2.1 Steps

1) Download darkflow repository in a new folder.

2) Scrap images of an object from google and save it in sub folders.

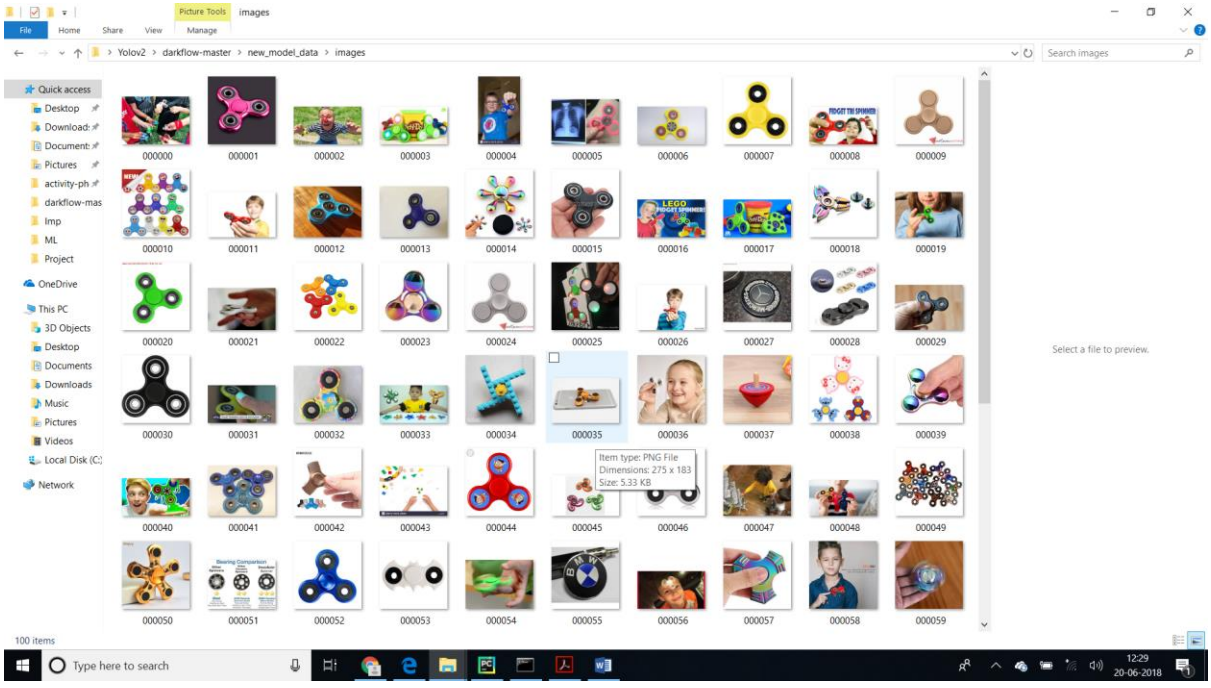
3) Make an image sub folder and integrate all image in it.

4) Make the bounding boxes around them and simultaneously generate the xml files for each image

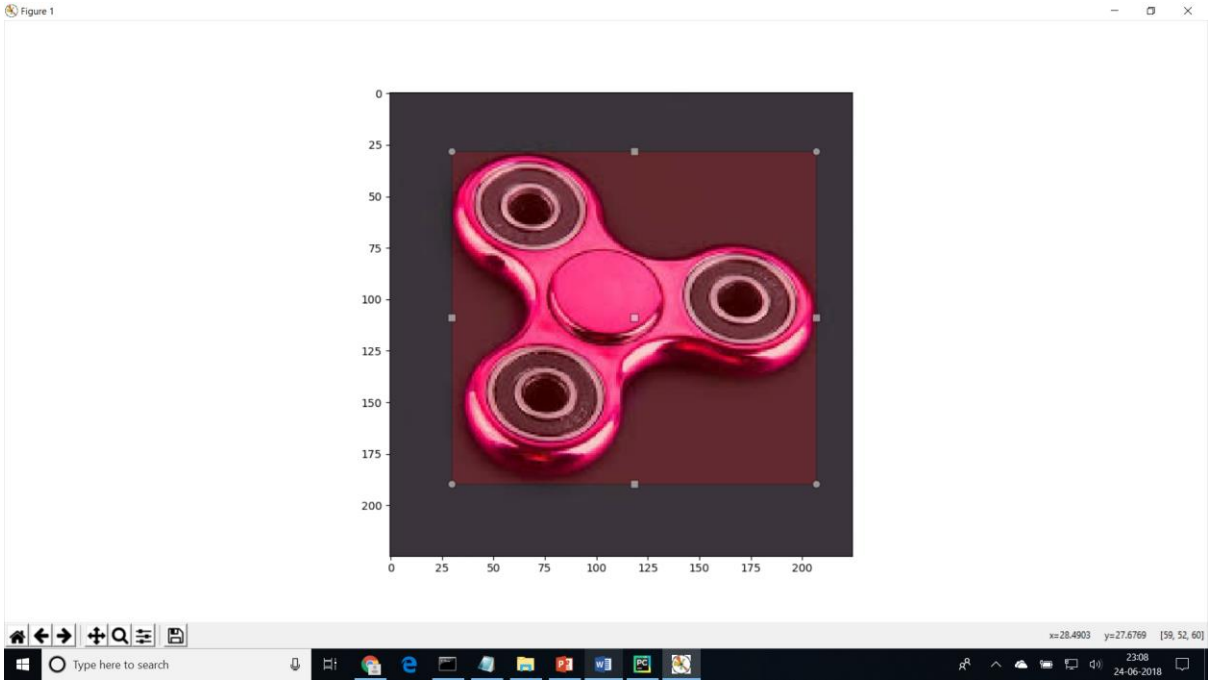
5) Train the model

python flow --model cfg/tiny-yolo-voc-1c.cfg --load bin/tiny-yolo-voc.weights --train --annotation new_model_data/annotations --dataset new_model_data/images --epoch 300

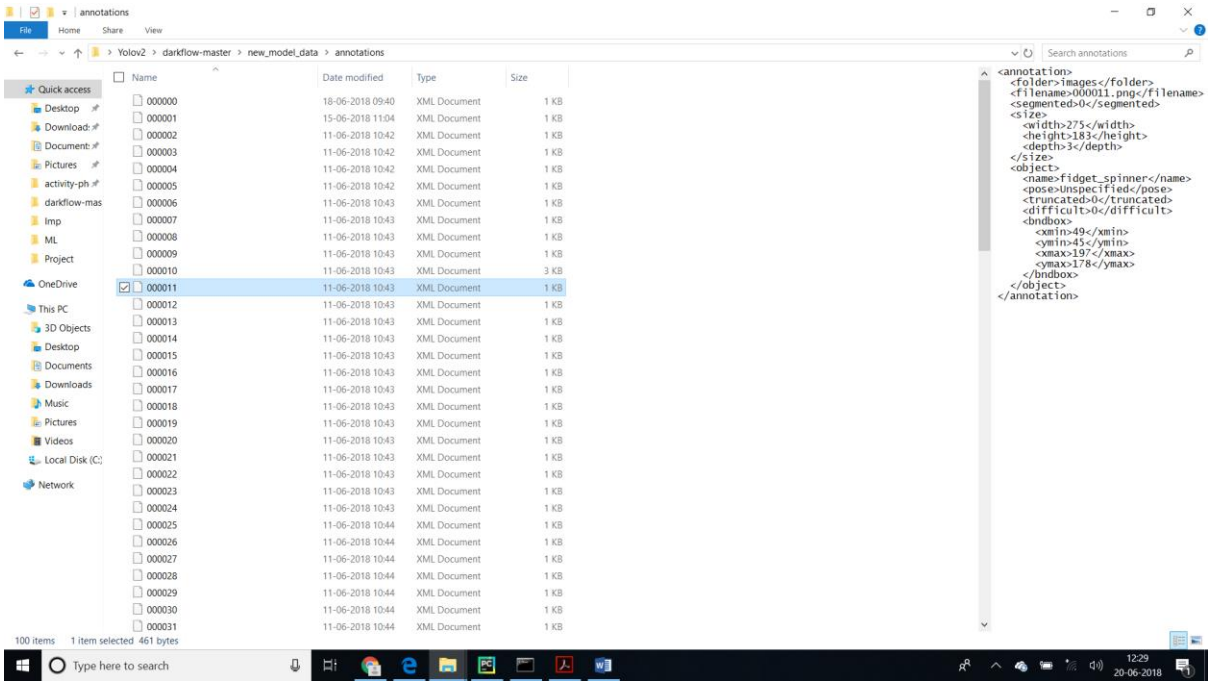
6)After making the required changes in the code like threshold, no. of classes etc, run it to detect the particular object in real time.



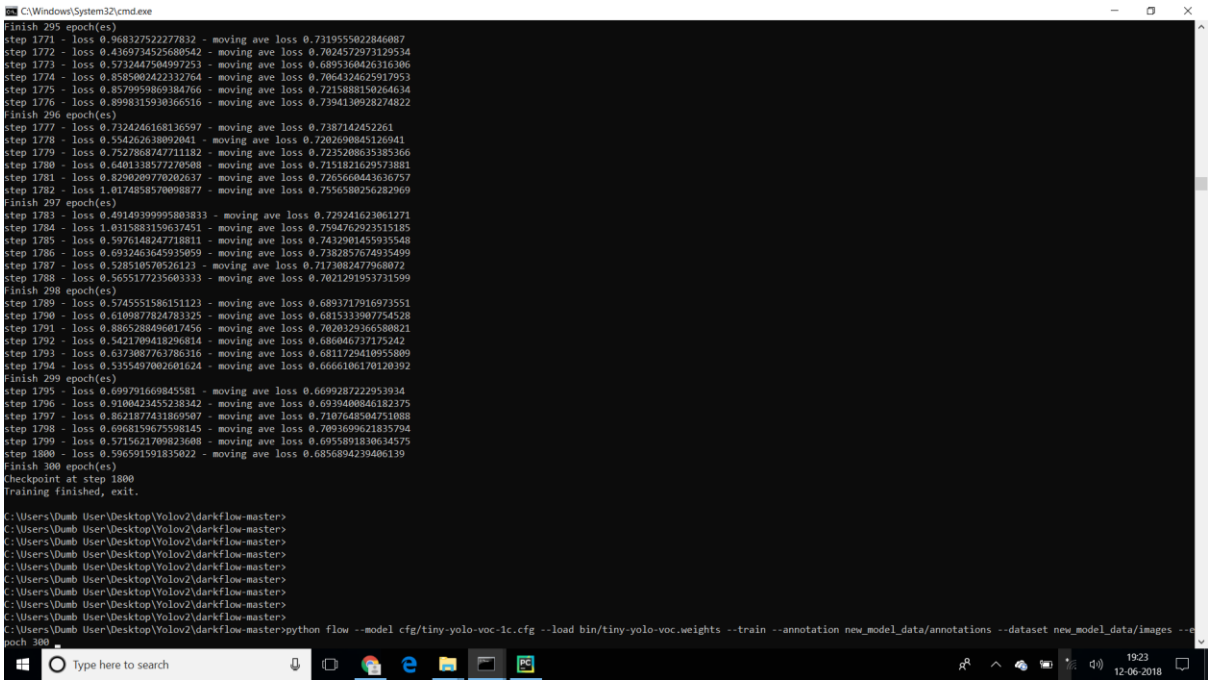
Images folder having images of object ‘spinner’



Drawing bounding boxes for objects



Annotations of each image are saved as xml files



Training the model with 300 epochs

Results:

Objects were detected in videos as well as webcam. The bounding boxes and the labels of objects were also seen.

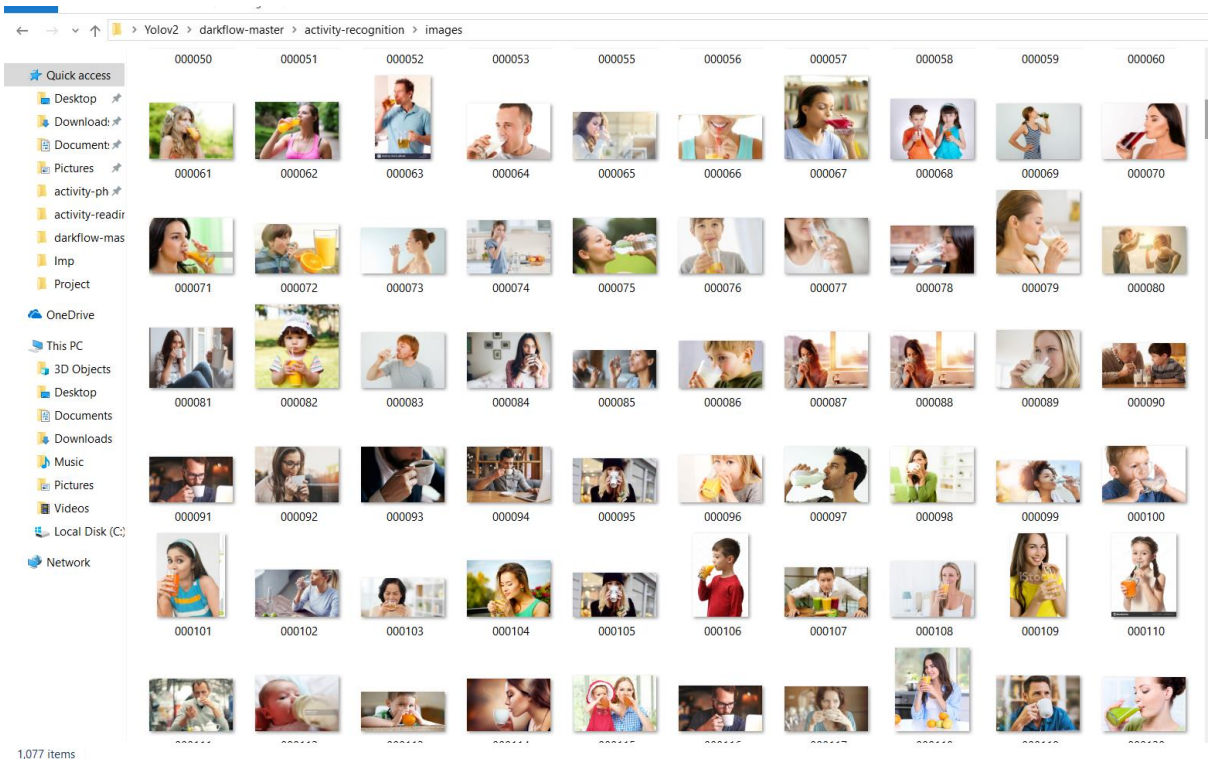
5. Activity Recognition using YOLOv2

5.1. Procedure

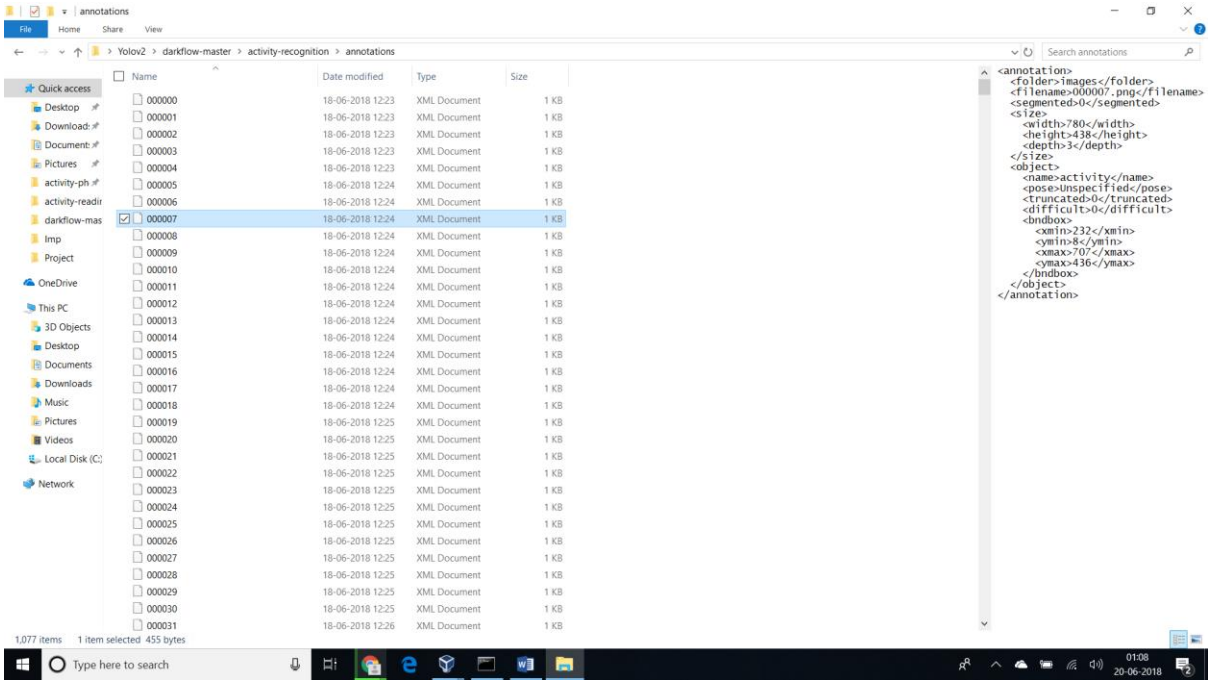
Using a similar approach of custom object detection on our own dataset, we try to detect activities in real time.
We take four activities named **phoning, handshake, reading and drinking**. A new folder in dark flow master is created named activity-recognition. In the folder we make sub-folders of each activity. A code is run to extract images from google for each activity. The code generates 100 images for a particular search name. For each activity try to get around 400 images by using different search names. Then a small code is run to make a new folder ‘images’ to compile images of all four activities and number them. Bounding boxes are made on the images to get trained data. When the bounding boxes are made for a image, subsequently a corresponding xml file is generated in annotations folder.

After the data is ready, labels are changed in labels.txt, a copy of tiny-yolo-voc is made which is named as tiny-volo-vco-4c.cfg. The number of classes in this file is changed to 4 and number of filters is also changed correspondingly.

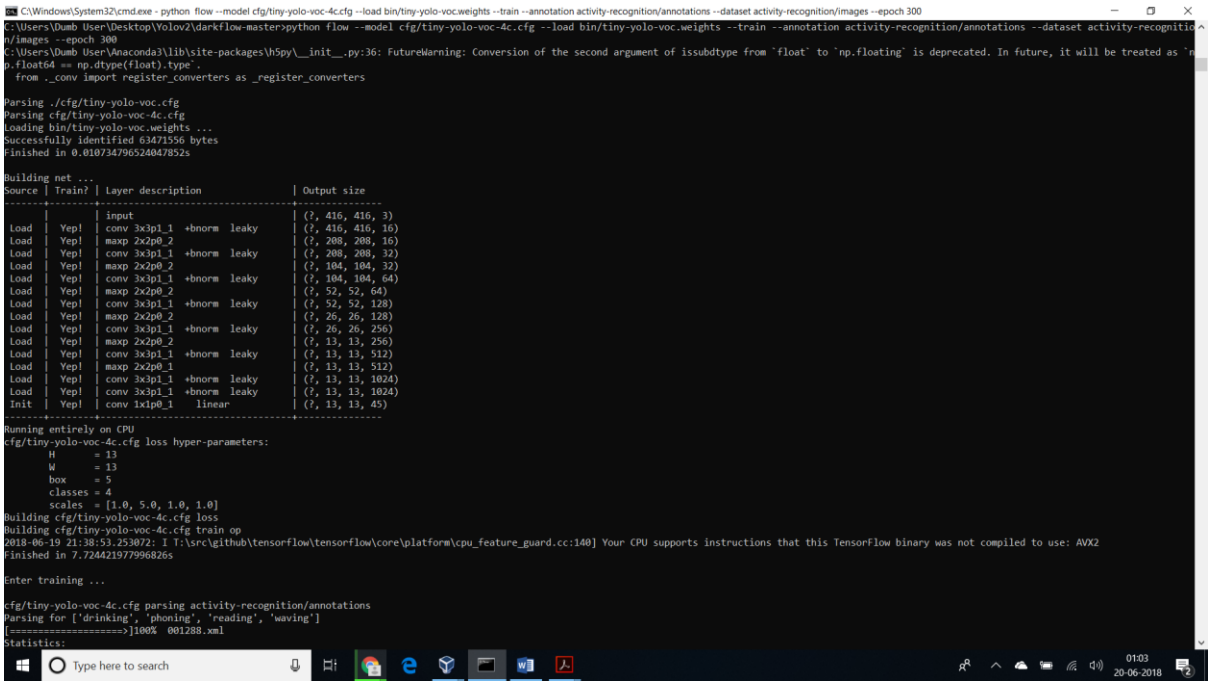
Then training is done with this model 300 epochs.



A set of images of all four classes is compiled



The xml files for each image are saved via code



Training

5.2. Results

The model did not give accurate results for activities. The possible reasons were

1)The dataset used was small that is it had less training examples.
Making bounding boxes was a tedious task requiring manual labour.
2)YOLO is meant for object recognition. It could’nt work for activities.

6. Activity Recognition using CNN

The CNN method on UCF-101 is generally used as a benchmark for other classification methods and is the simplest and most natural video classification method. The CNN method treats these video frames as a single static image, applies CNN to identify each frame, and then averages the prediction results as the final result of the video. However, this method uses incomplete video information, thus making the classifier prone to confusion and resulting in low accuracy. The final accuracy of the CNN method on the test set is top1: 63.0%, top5: 87.5%.

6.1. Basic Idea

The basic idea is to convert the video and classification tags in the data set into images (video frames) and their corresponding classification tags, and then use the CNN network to train and test the images, and transform the video classification problem into a graphics classification problem. The specific steps include:

(1) For each video (training and test video), a certain FPS cut-out video frame (jpegs) is saved as a training set and a test set, and the classification performance of the image is used as the classification performance of the corresponding video: train set has 1 , 788, 425 frames, test
set with 697, 865 frames

(2) Select a pre-trained CNN network architecture and initial weights, and migrate to UCF-101, such as inception v3 with pre-trained on ImageNet
(3) Retrain the CNN network partial layer with the train set to obtain the model.

(4) After the training is completed, the model is checked and verified for all video frames in the test set, and the top1 accuracy rate and top5 accuracy rate output on the whole test set are obtained.

6.2. The operating environment

(1) Server hardware environment: 40 core Xeon cpu, GeForce GTX 1080 8G memory X2, 128G memory, 512G
SSD, 3TB mechanical hard disk

(2) Server software environment: install ubuntu16.04, conda (including python2.7), CUDA, cudnn, tensorflow GPU, keras, etc. python package, SSH service, screen package, vim tool, ffmpeg package

(3) Client: window7 64-bit, pycharm, xshell, xftp

(4) Usage mode: Client remote SSH connection server operation

6.3 101 classes

1 ApplyEyeMakeup
2 ApplyLipstick
3 Archery
4 BabyCrawling
5 BalanceBeam
6 BandMarching
7 BaseballPitch
8 Basketball
9 BasketballDunk
10 BenchPress
11 Biking
12 Billiards
13 BlowDryHair
14 BlowingCandles
15 BodyWeightSquats
16 Bowling
17 BoxingPunchingBag
18 BoxingSpeedBag
19 BreastStroke
20 BrushingTeeth
21 CleanAndJerk
22 CliffDiving
23 CricketBowling
24 CricketShot
25 CuttingInKitchen
26 Diving
27 Drumming
28 Fencing
29 FieldHockeyPenalty
30 FloorGymnastics
31 FrisbeeCatch
32 FrontCrawl
33 GolfSwing
34 Haircut
35 Hammering
36 HammerThrow
37 HandstandPushups
38 HandstandWalking
39 HeadMassage
40 HighJump
41 HorseRace
42 HorseRiding
43 HulaHoop
44 IceDancing

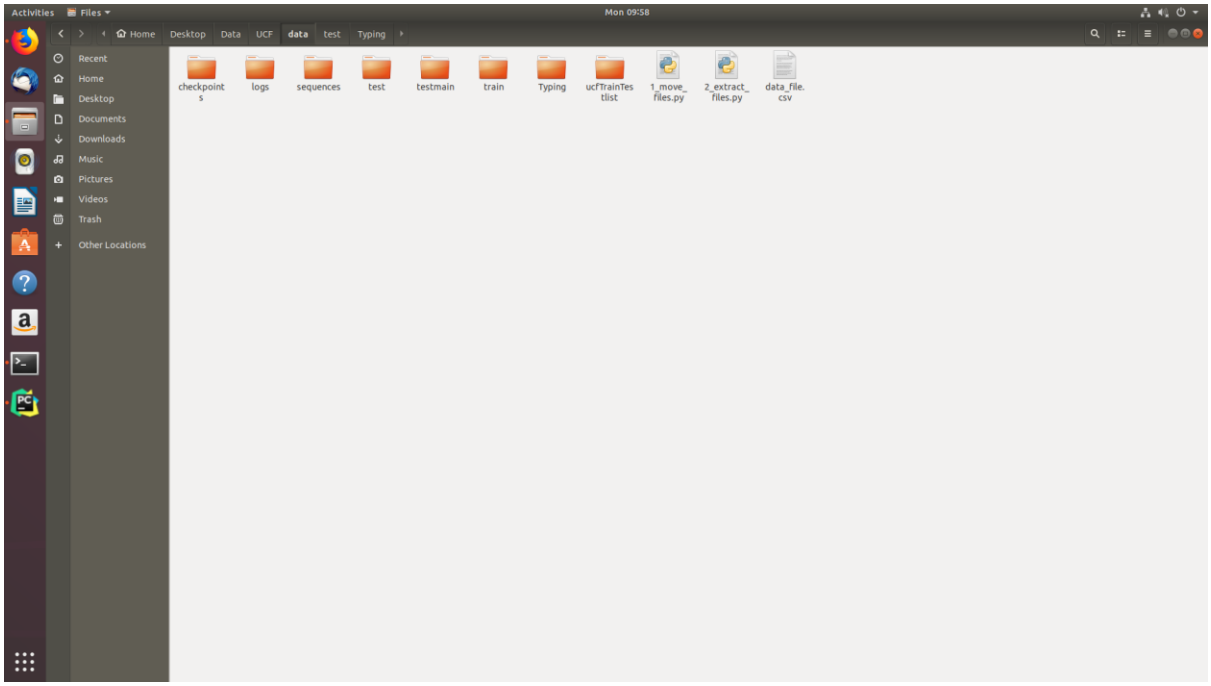
45 JavelinThrow
46 JugglingBalls
47 JumpingJack
48 JumpRope
49 Kayaking
50 Knitting
51 LongJump
52 Lunges
53 MilitaryParade
54 Mixing
55 MoppingFloor
56 Nunchucks
57 ParallelBars
58 PizzaTossing
59 PlayingCello
60 PlayingDaf
61 PlayingDhol
62 PlayingFlute
63 PlayingGuitar
64 PlayingPiano
65 PlayingSitar
66 PlayingTabla
67 PlayingViolin
68 PoleVault
69 PommelHorse
70 PullUps
71 Punch
72 PushUps
73 Rafting
74 RockClimbingIndoor
75 RopeClimbing
76 Rowing
77 SalsaSpin
78 ShavingBeard
79 Shotput
80 SkateBoarding
81 Skiing
82 Skijet
83 SkyDiving
84 SoccerJuggling
85 SoccerPenalty
86 StillRings
87 SumoWrestling
88 Surfing
89 Swing
90 TableTennisShot

91 TaiChi
92 TennisSwing
93 ThrowDiscus
94 TrampolineJumping
95 Typing
96 UnevenBars
97 VolleyballSpiking
98 WalkingWithDog
99 WallPushups
100 WritingOnBoard
101 YoYo

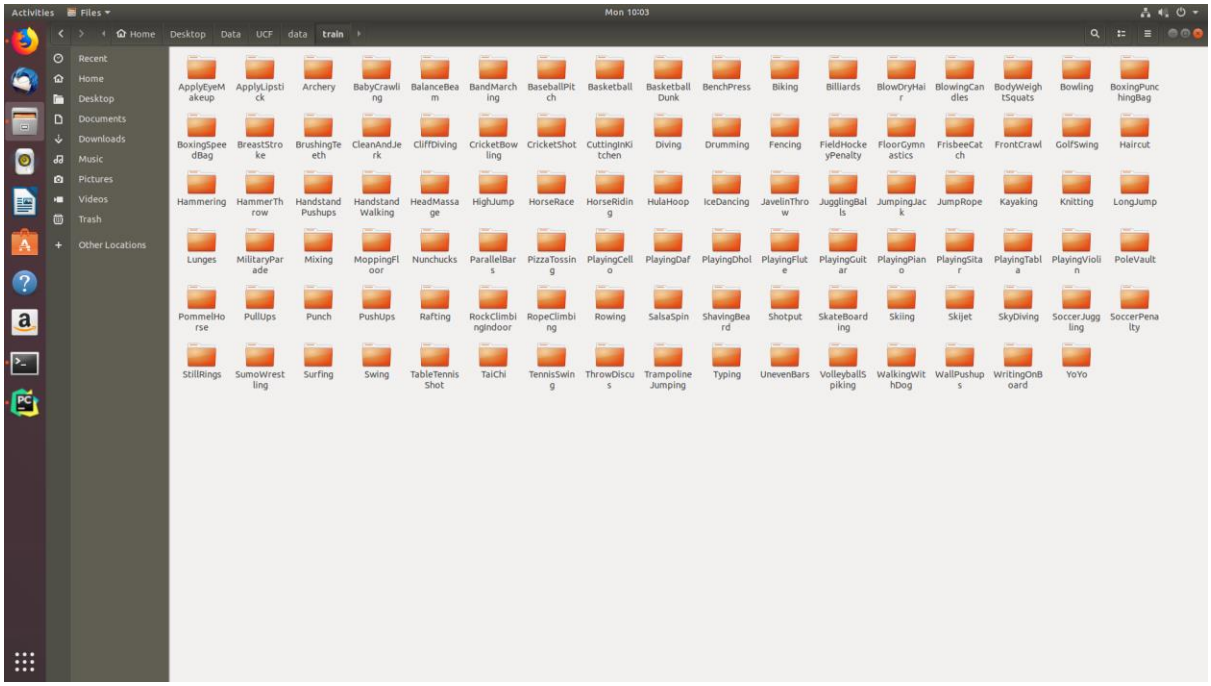
6.4. PROCEDURE

6.4.1 Moving Files

The dataset after it is downloaded is moved into respective folders. 101 folders, one for each class is made both in the training and testing set. The code is run only once.



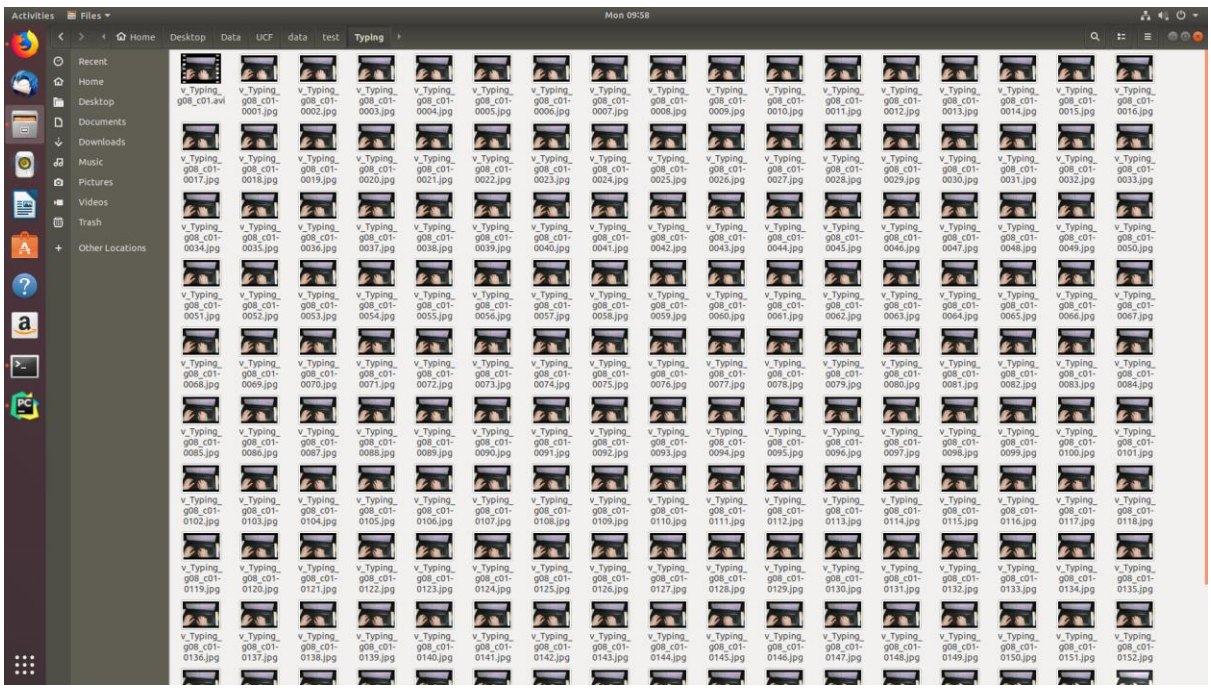
Training and Testing Folders



Both folders contain 101 classes

6.4.2 Extracting Frames

After making the folders, videos have to be converted into frames and the frames of each video have to be stored after the video.



(After the video, all its frames are stored)

6.4.3 Training

Train on images split into directories. Tensorflow 1+ and keras 2+ are used. Training takes 3 to 4 days on CPU, so GPU is recommended. One can decrease the epochs but this would compromise on accuracy. Batch size can also be reduced.

6.4.4. Evaluating Test Set

Thi involves classifying test image set using CNN.Using tensorflow1+ and keras 20+test images our classified through CNN. This make take many hours. This part can even be skipped.

6.4.5. Classifying images

Classify some images through CNN and check accuracy. Ranomly 5 images from test set are taken and their top 5 confidence score are predicted.

6.5 Testing on some videos

Merge some videos of activities and get it in mp4 format. Save it in a folder where this code is present. Name the video as ‘new1.mp4’. When the code is run:

1)Frames are generated and shown as plots.

2)Each plot has a the first confidence score activity as the title.

3)The frames are generated spontaneously and also deleted spontaneously after there prediction is done.

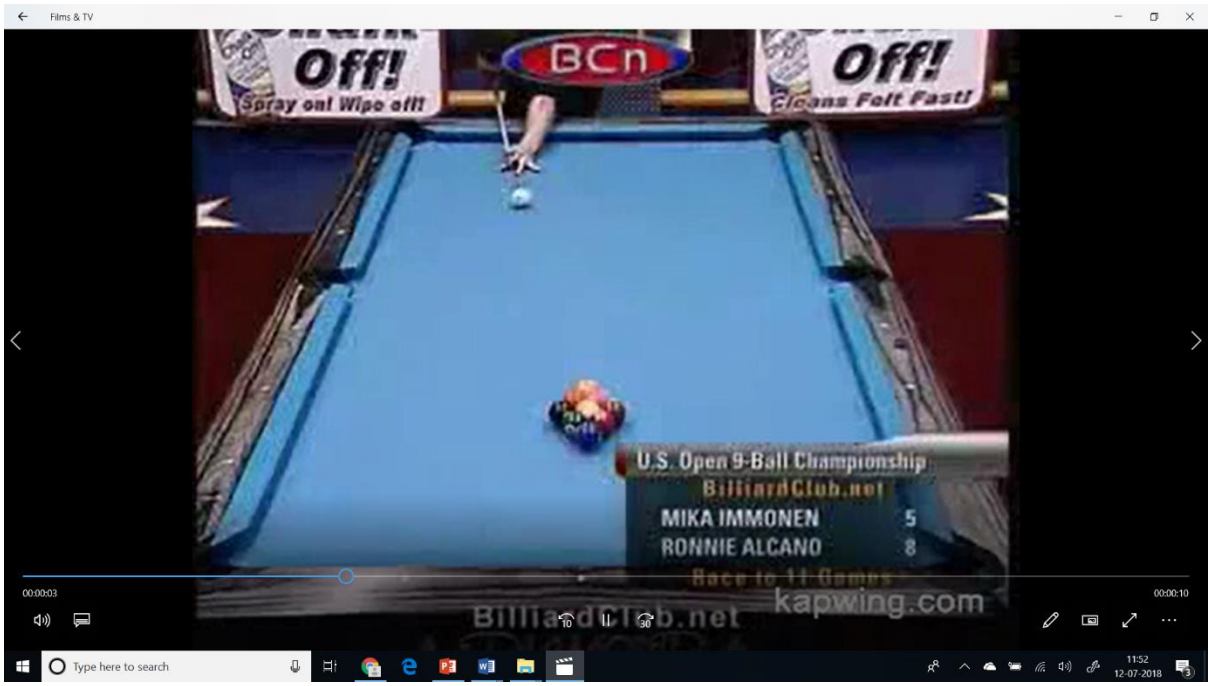
4)The first frame top confidence score is predicted.

5)After the first image, set of 10 frames are taken and a guassian function of the predictions are drawn. The most probable activity is outputted for a series of 10 frames.

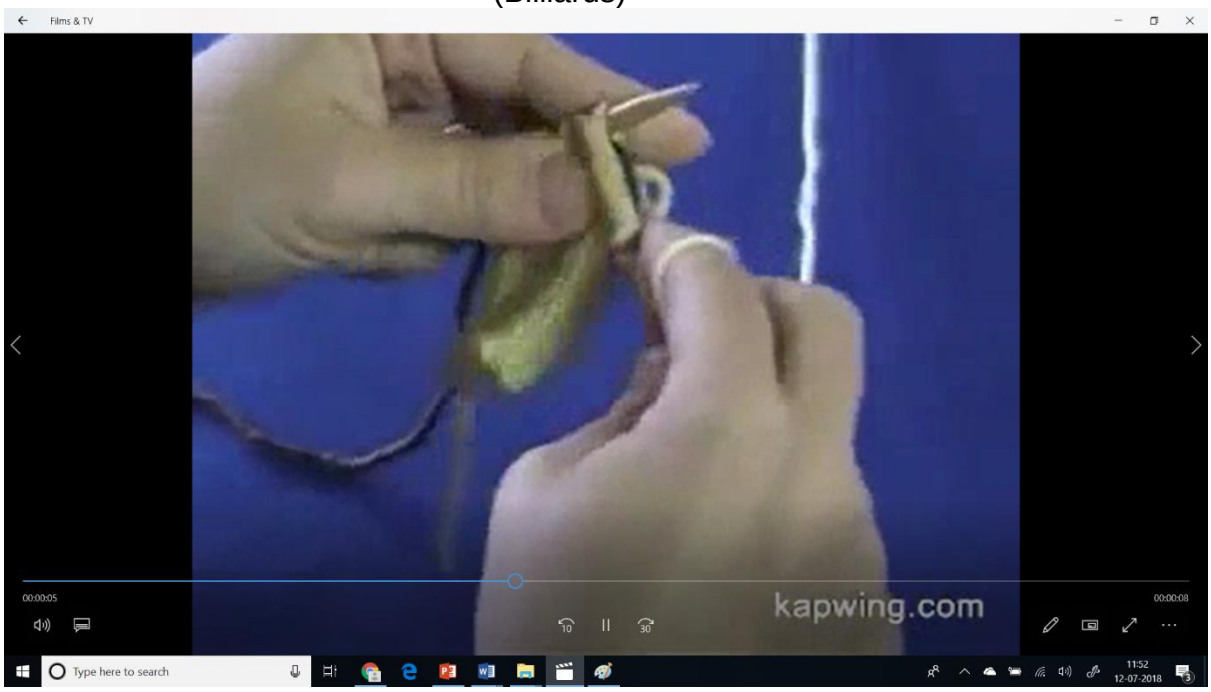
7. Results

The input video has activities like:

(Drumming)

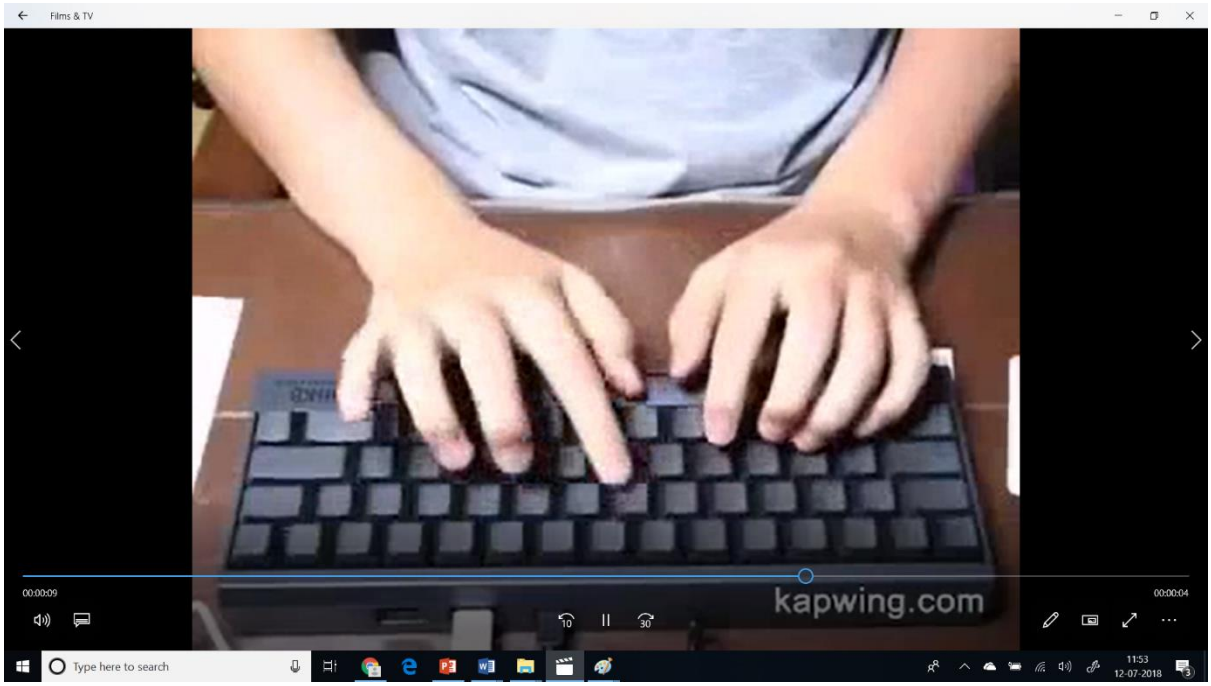


(Billiards)



(Knitting)

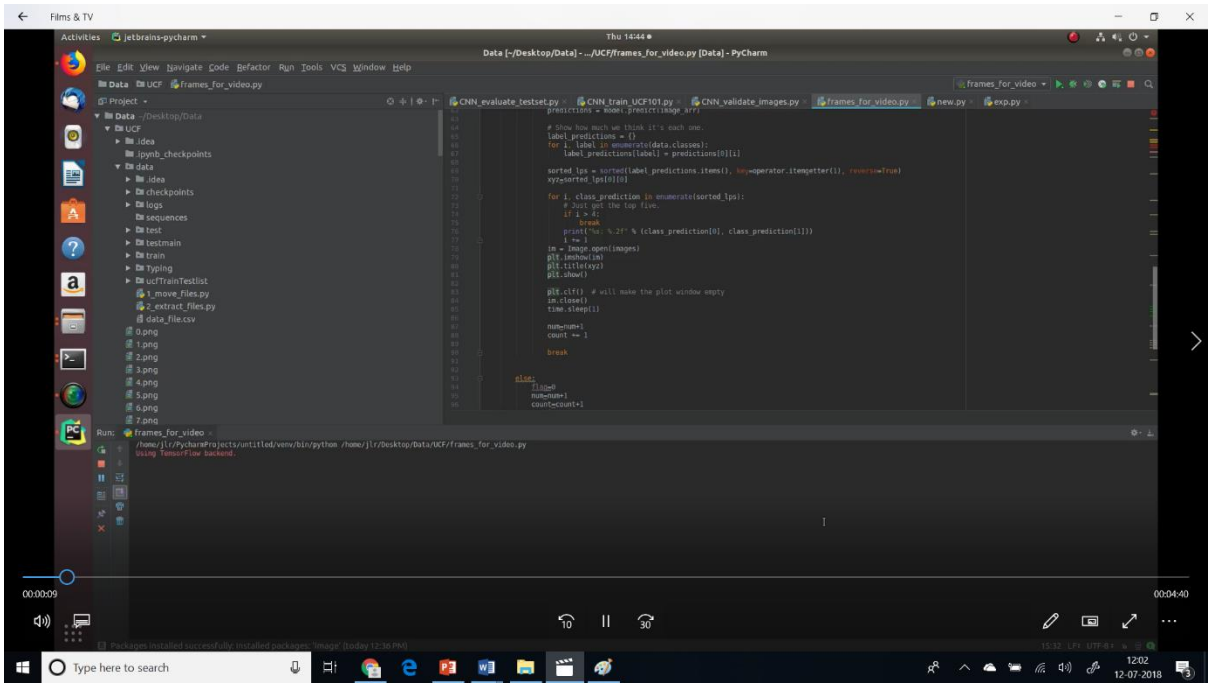
(Table tennis shot)



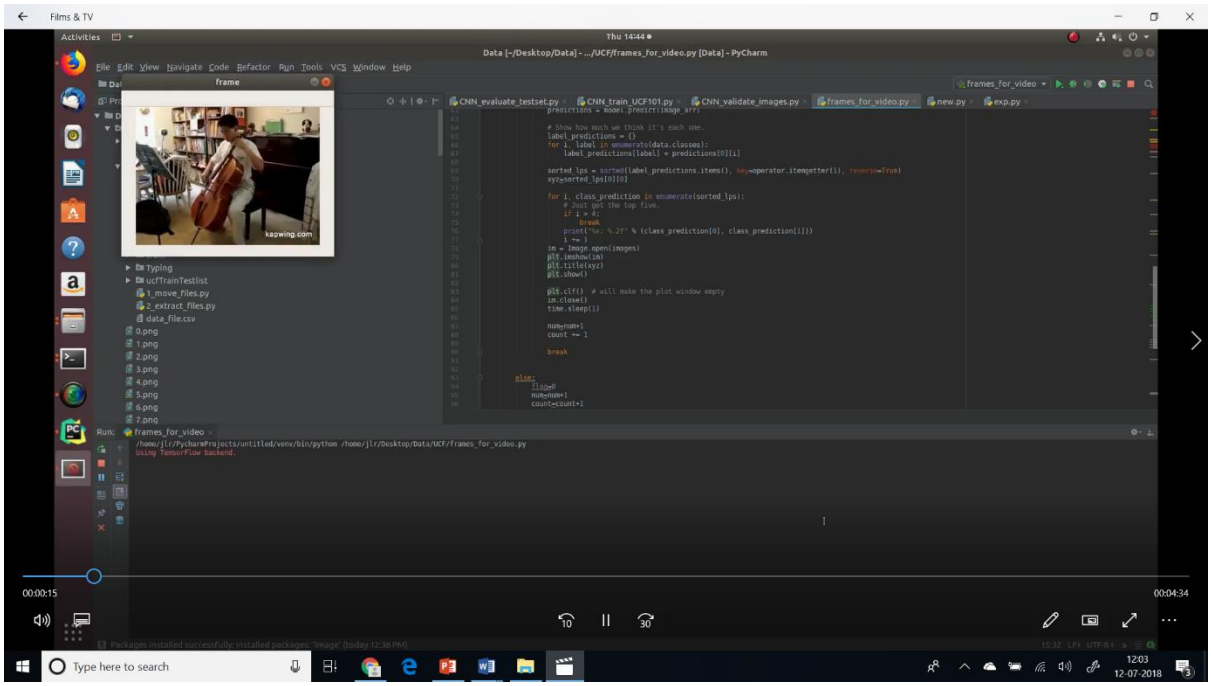
(Typing)

(Playing Cello)

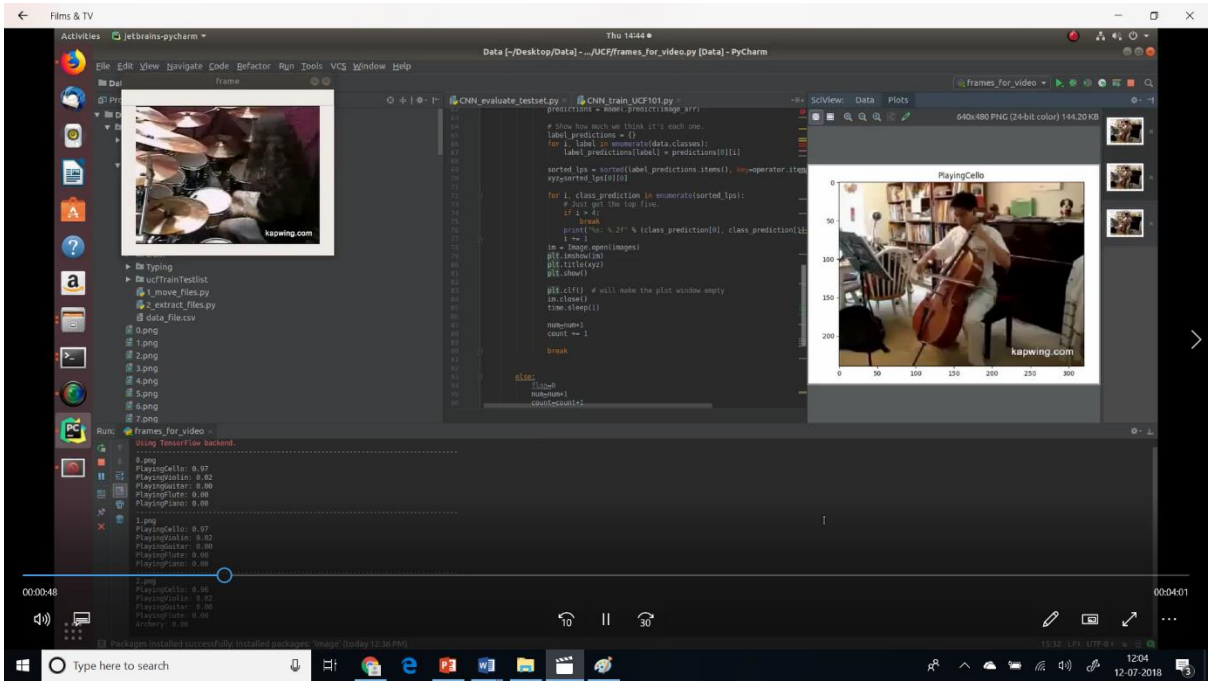
Top 5 confidence scores are shown for each frame. The accuracies vary from 90% to 100% for these activities. The input video is shown before giving outputs.
The scematic images are as follows:



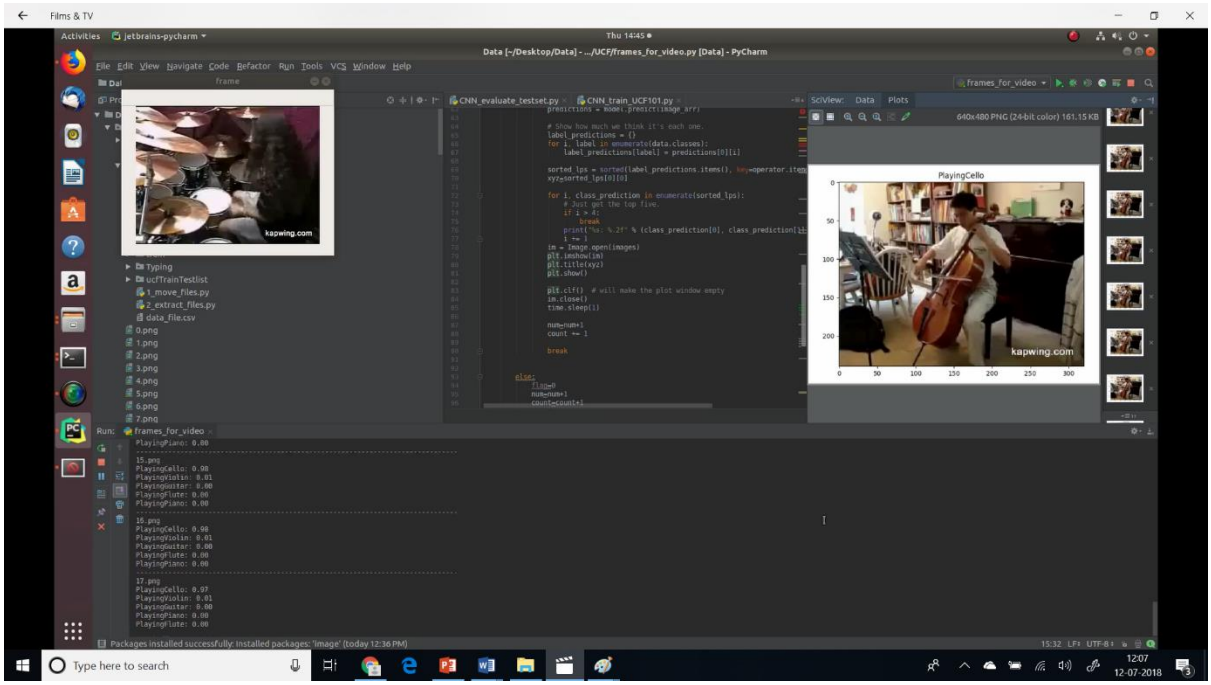
(The code is run)

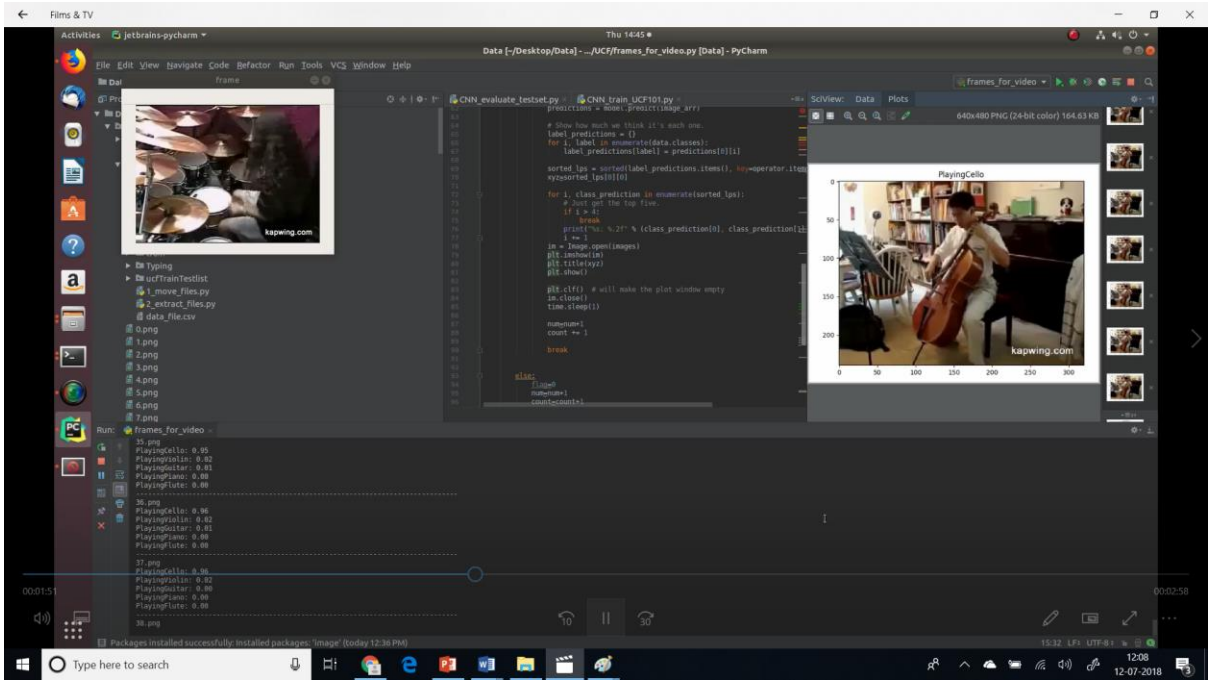


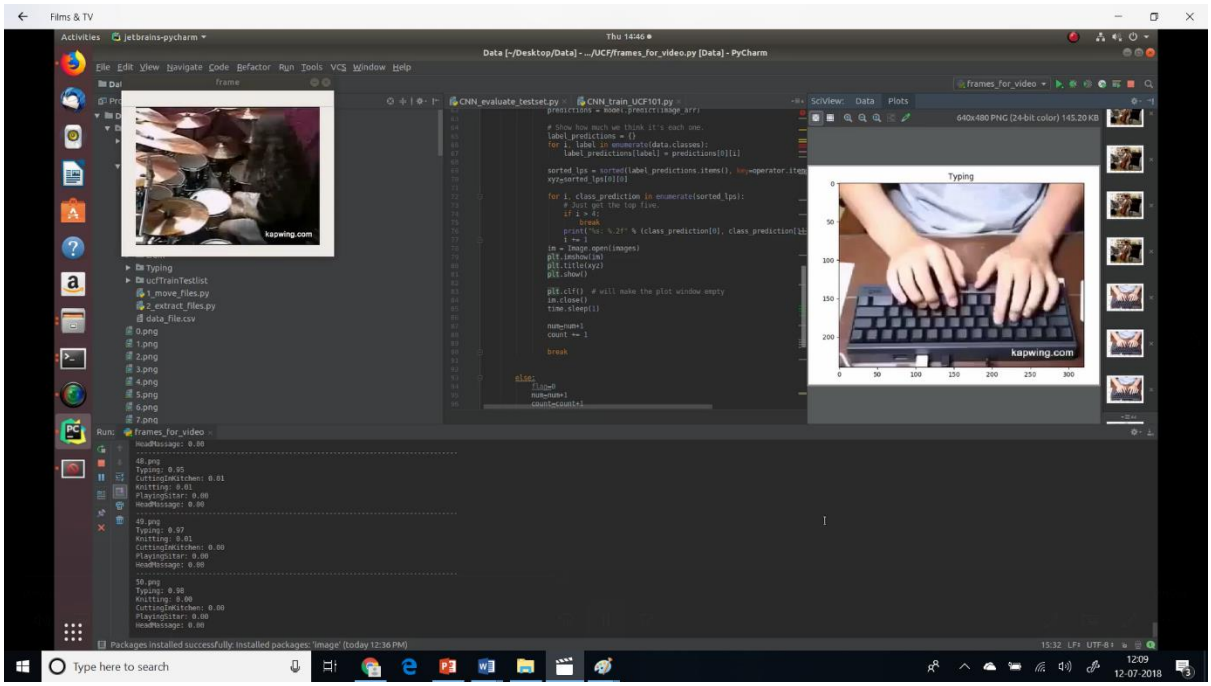
(Video is shown first)

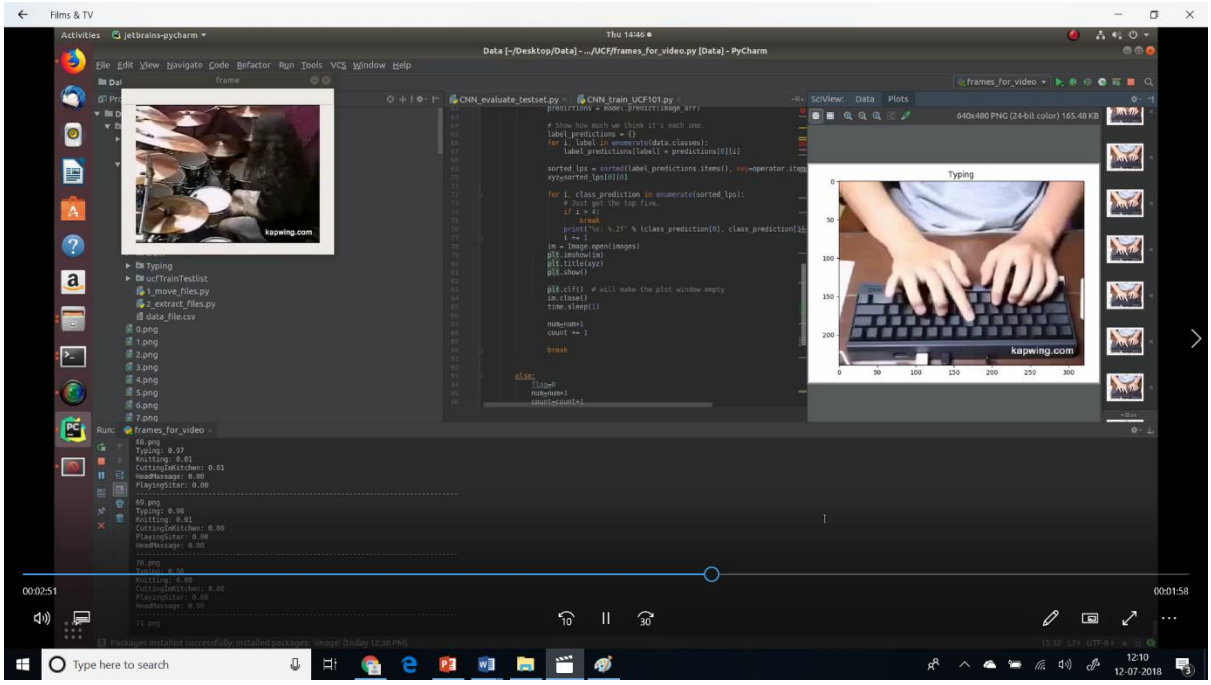


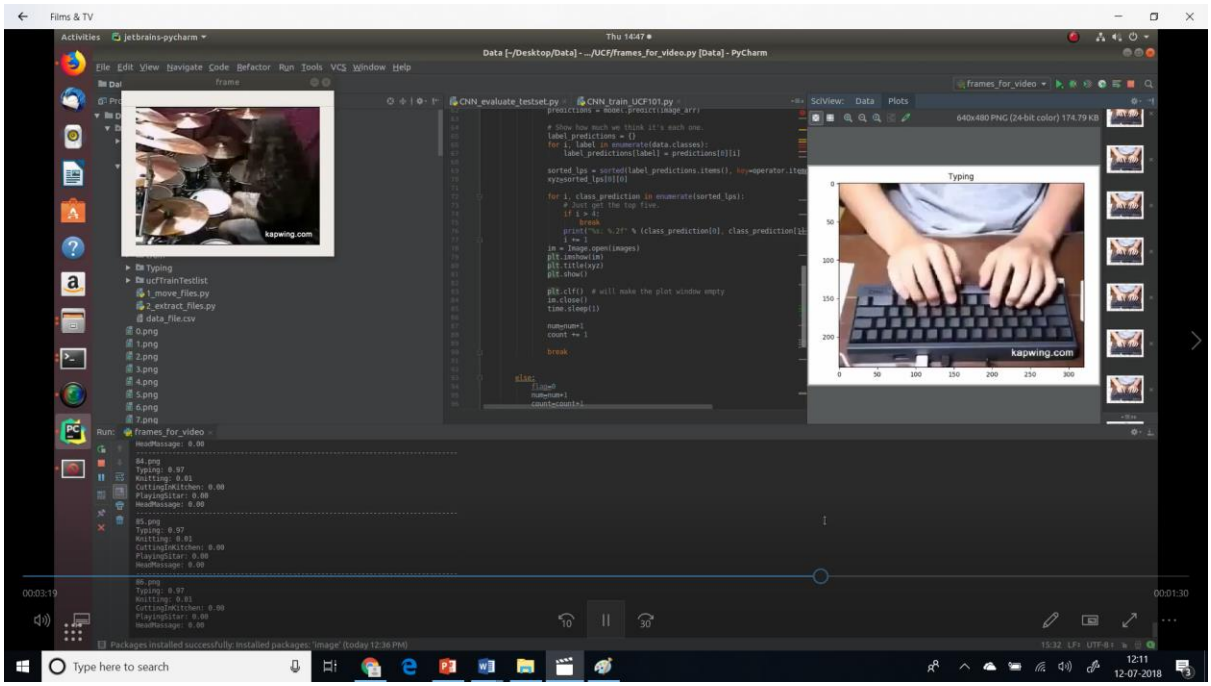
(Frames are generated and shown as plots. The top 5 confidence scores are also predicted. As it is clear the accuracy is 97 % for first frame)

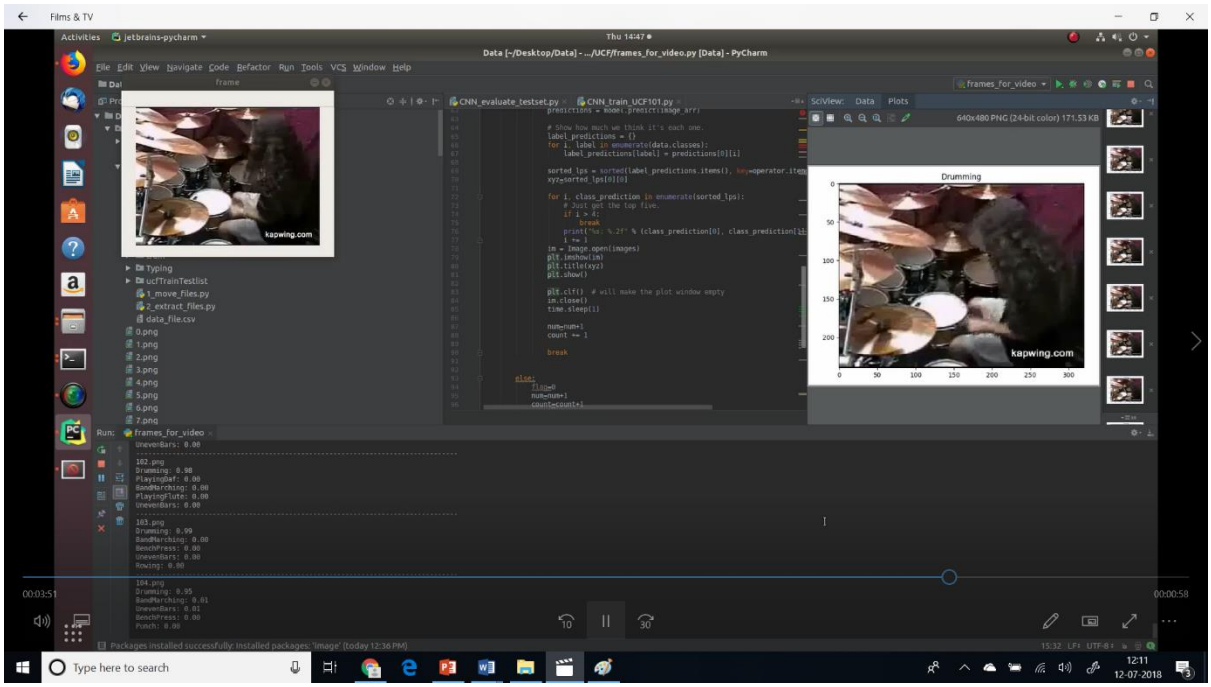

Films & TV
Data [~/Desktop/Data] - .../UCF/frames_for_video.py [Data] - PyCharm
PlayingCello
kapwing.com
Type here to search
12:07
12-07-2018

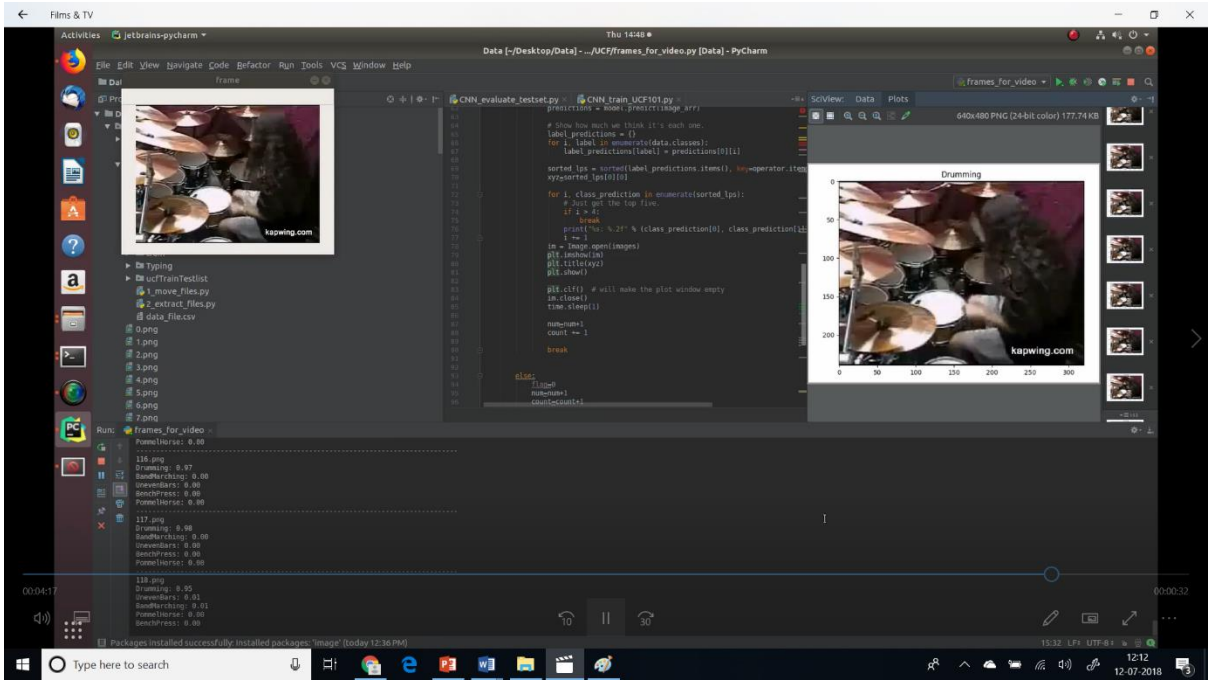

Films & TV
Data [~/Desktop/Data] - .../UCF/frames_for_video.py [Data] - PyCharm
PlayingCello
kapwing.com
Type here to search
12:08
12-07-2018


Films & TV
Data [~/Desktop/Data] - .../UCF/frames_for_video.py [Data] - PyCharm
Typing
kapwing.com
Type here to search
12:09
12-07-2018


Films & TV
Data [~/Desktop/Data] - .../UCF/frames_for_video.py [Data] - PyCharm
Typing
kapwing.com
00:02:51
00:01:58
Type here to search
12:10
12-07-2018

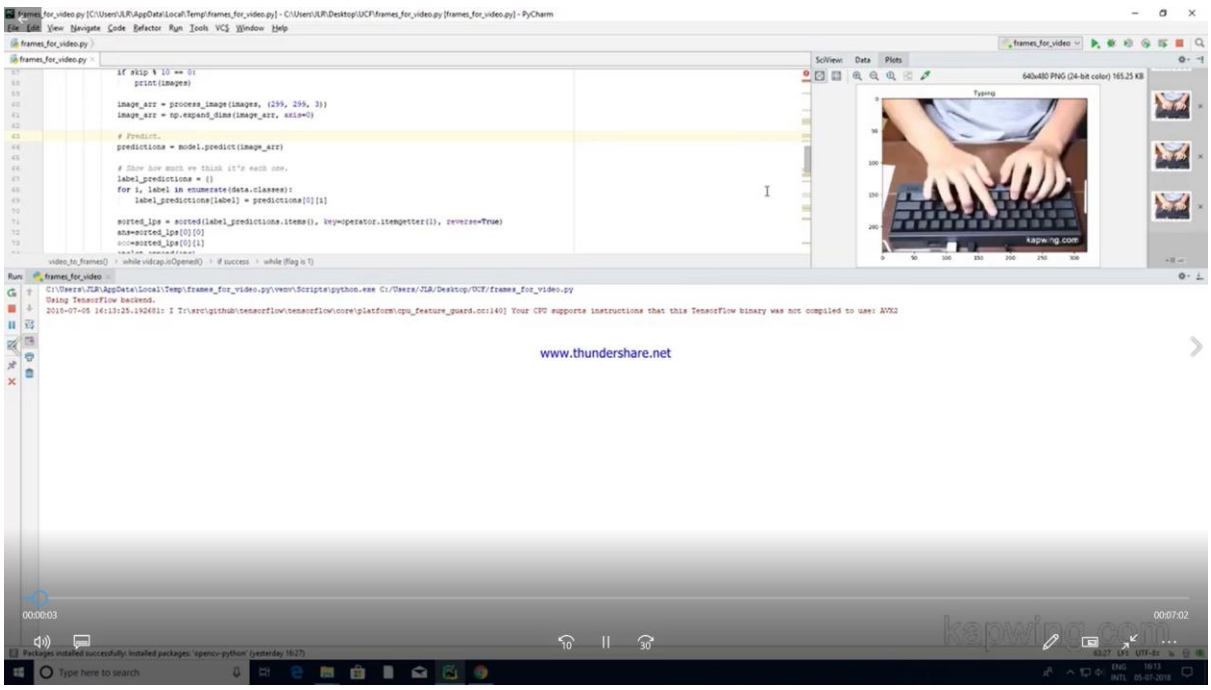

Typing
kapwing.com


Drumming
kapwing.com

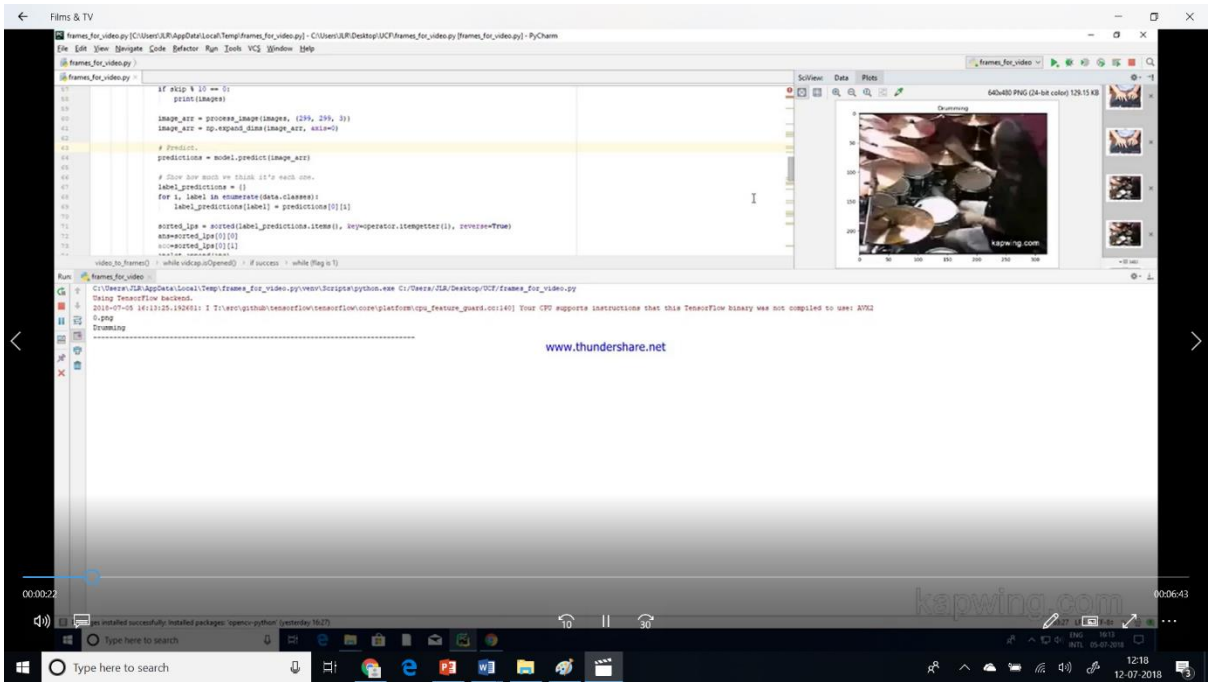


For different frames top 5 activities with most probability are shown. The first activity has accuracy ore than 95% and rest four activities have accuracies less than 1%. For each frame the output activity is shown as title of the frame.

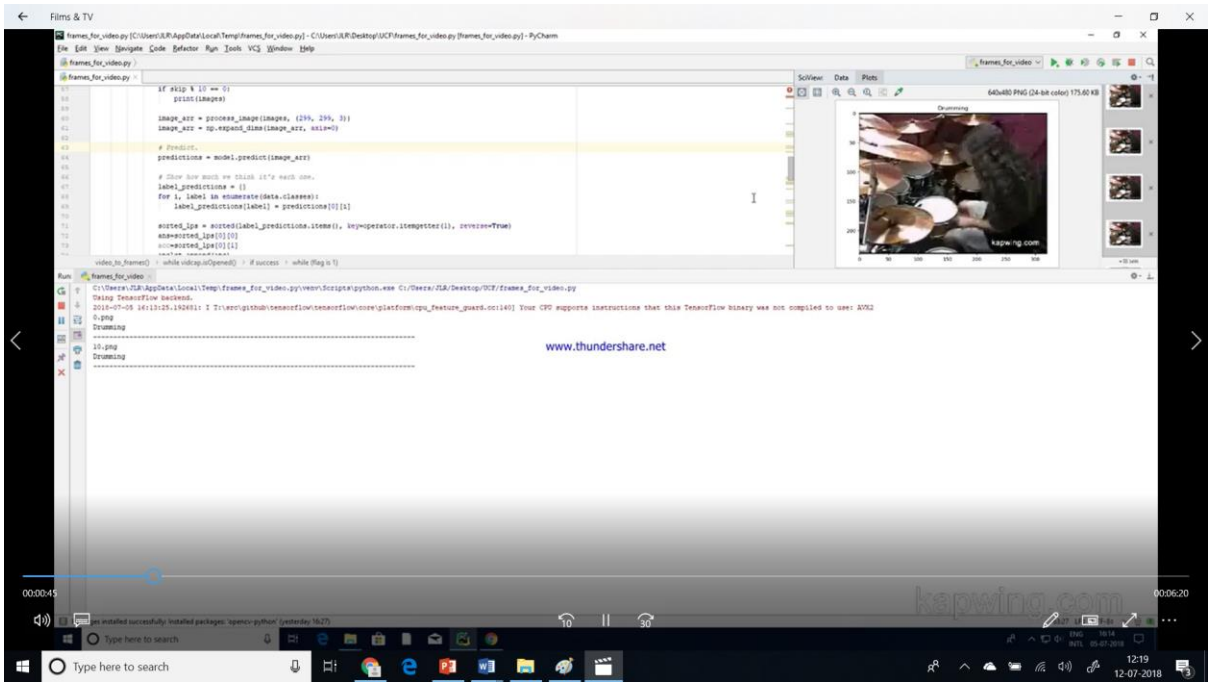
Now Gaussian function of top confidence activities for 10 frames is made and one output activity is printed without percentage.



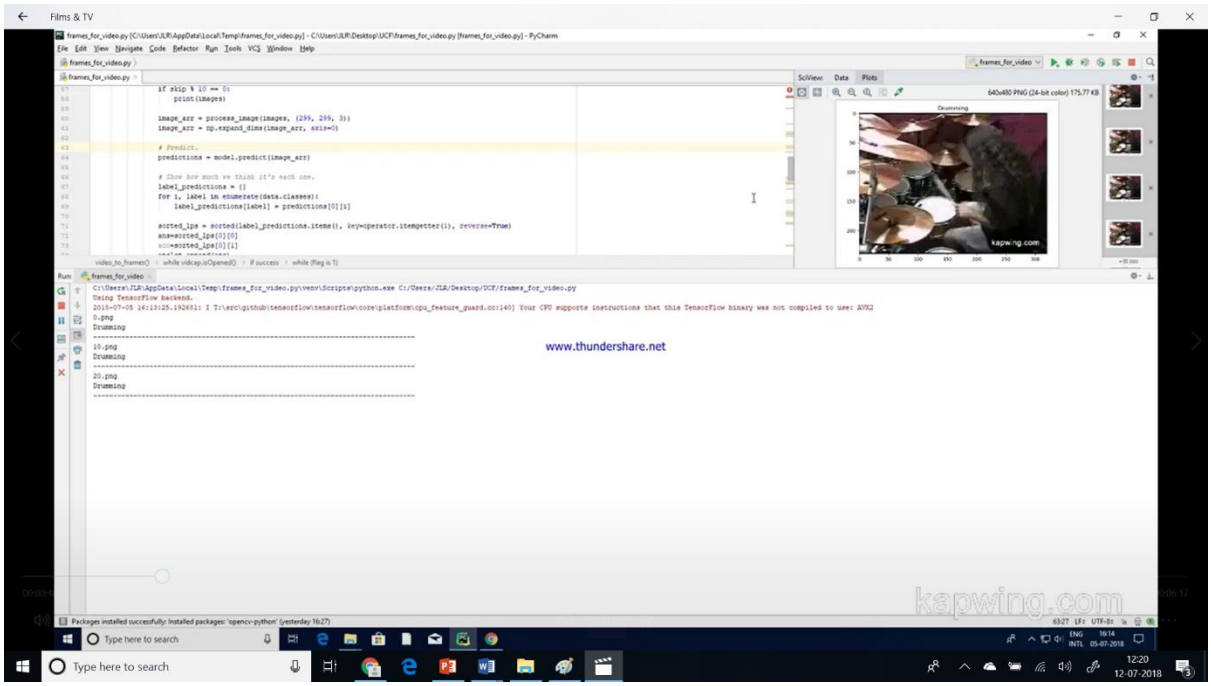
(The execution starts)

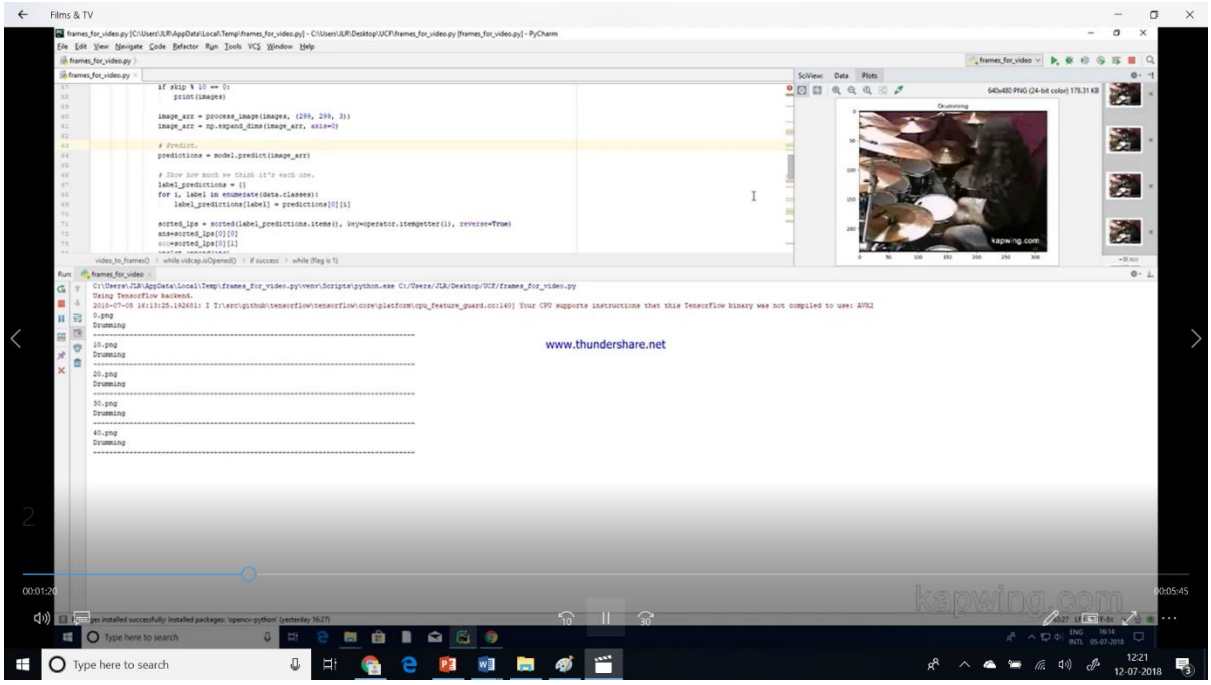


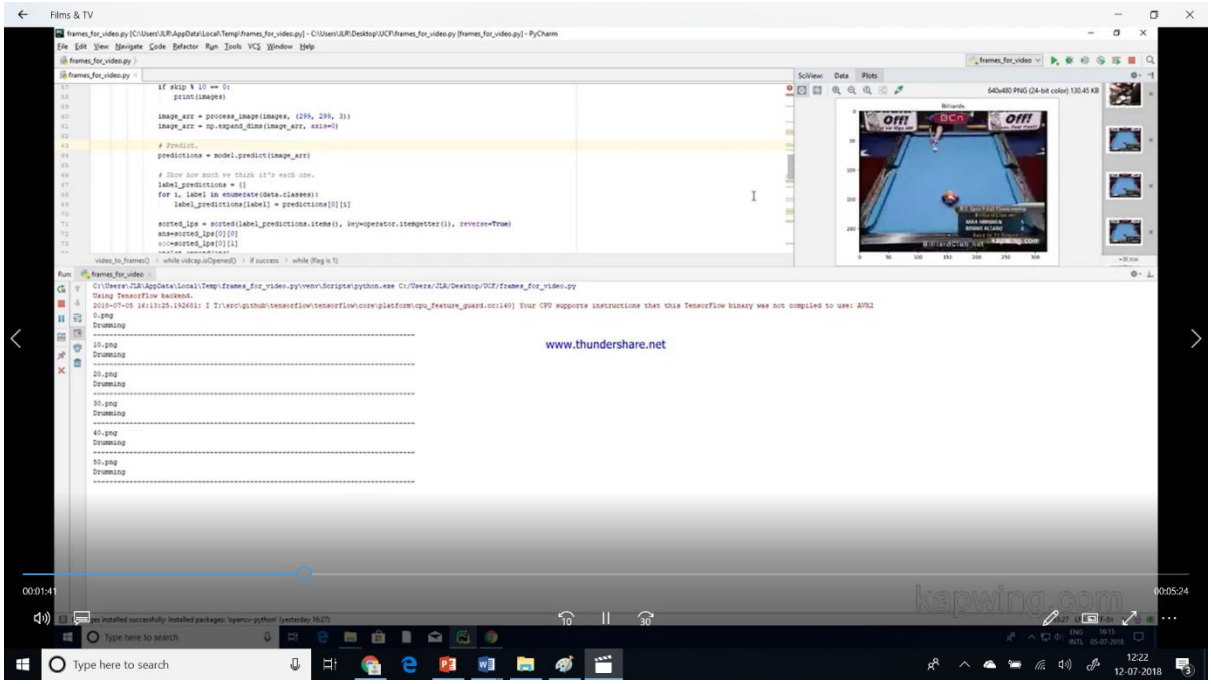
(The result of first frame is shown)



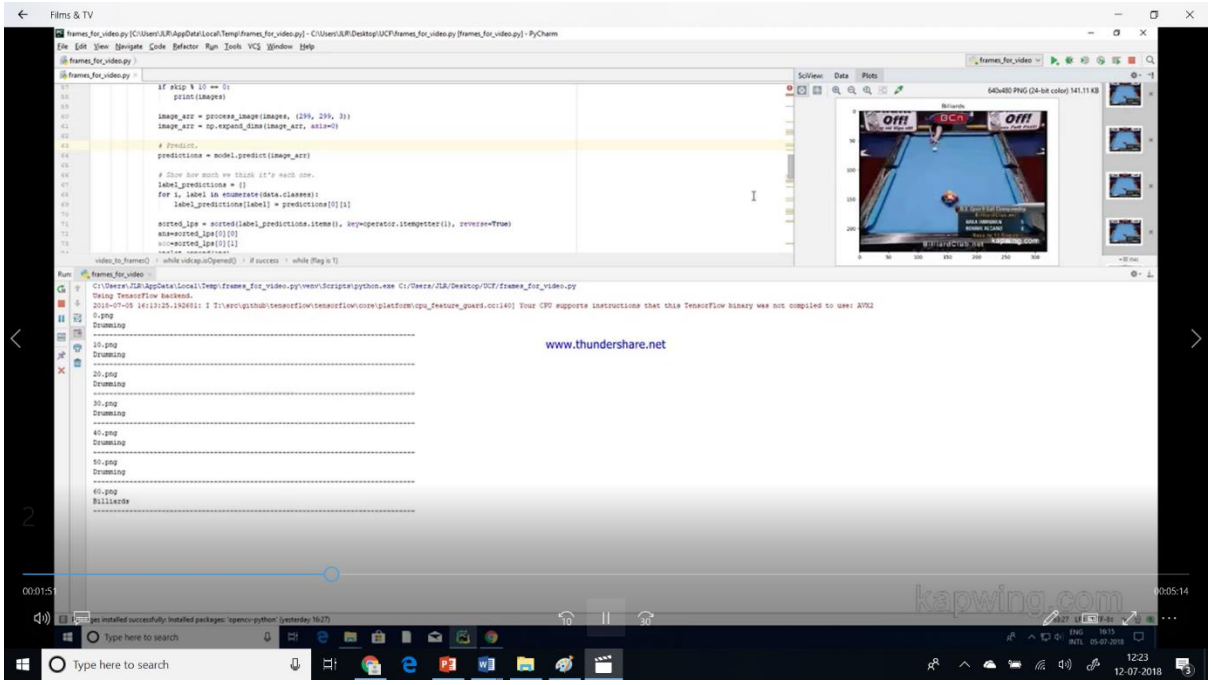
(Only one activity is predicted for next 10 frames. For each frame the output activity is aso shown in the plot)


Films & TV
www.thundershare.net
kapwing.com
Type here to search
12:20
12-07-2018

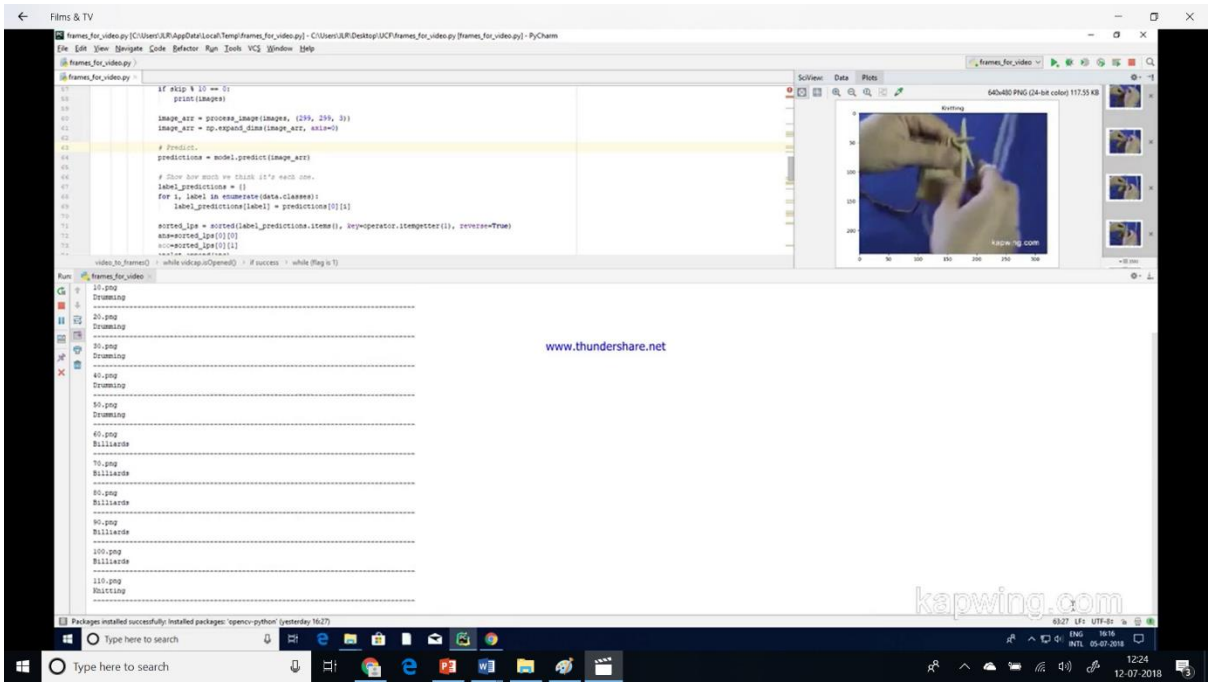

Films & TV
www.thundershare.net
kapwing.com
Type here to search
12:21
12-07-2018

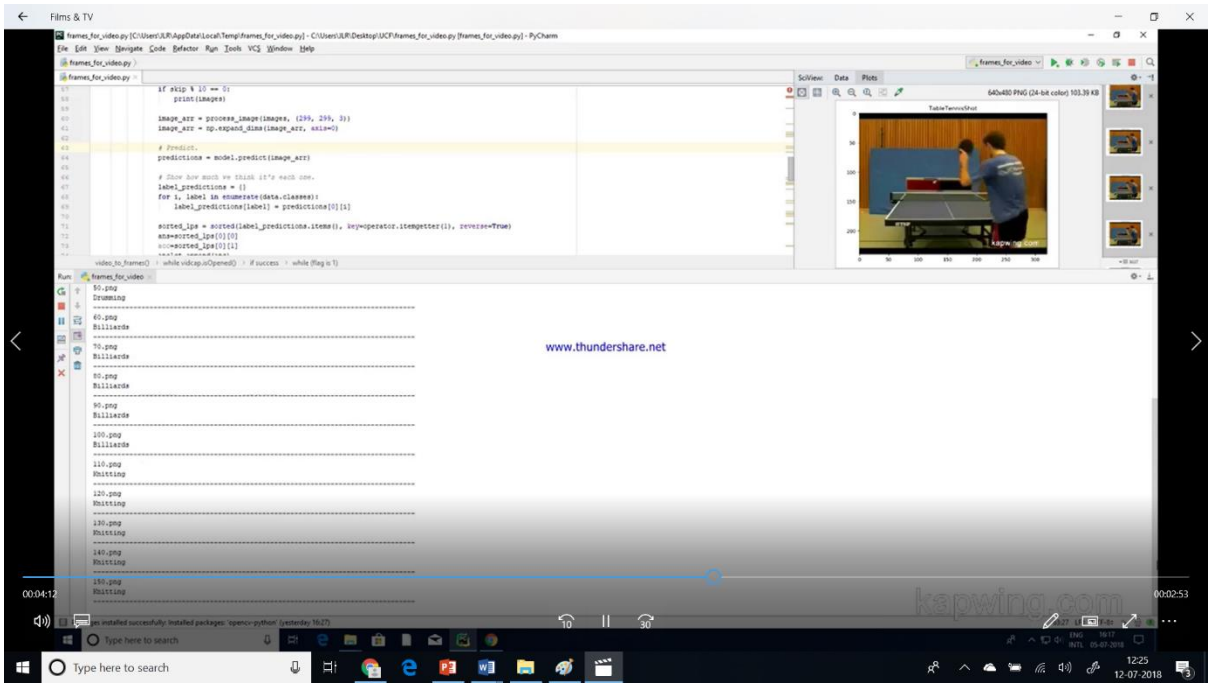


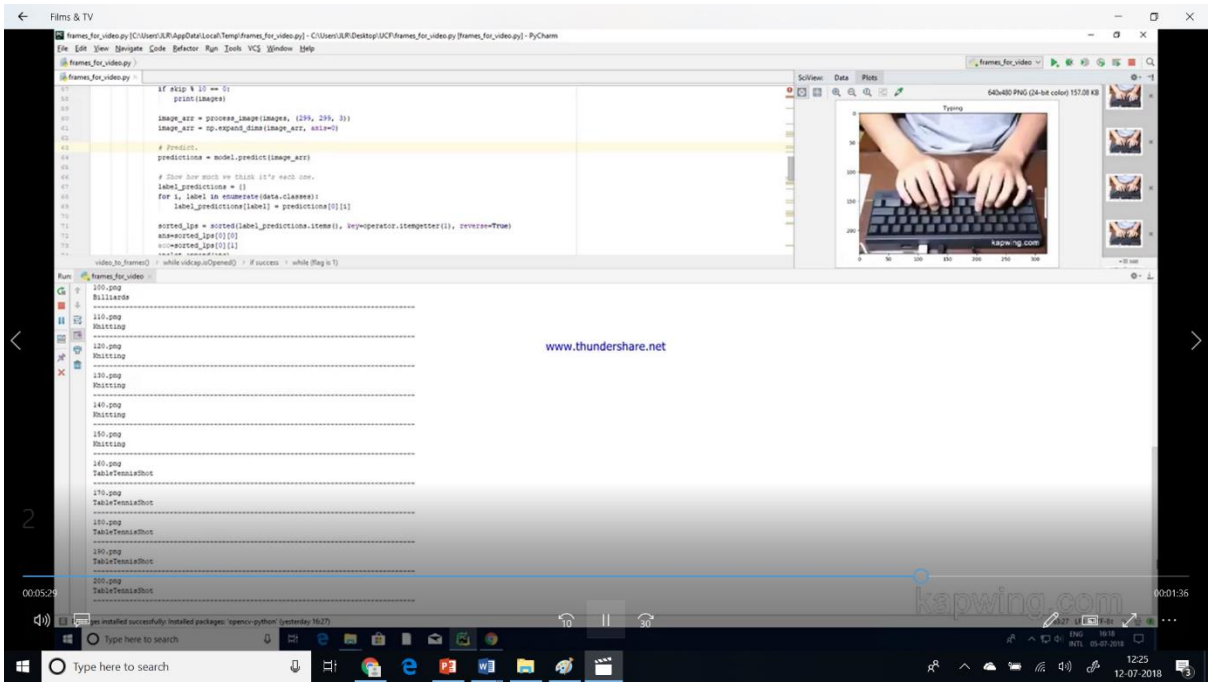
(As the activity has changed to Billiards the next output should be Billiards)



(Yes it is)


Films & TV
www.thundershare.net
kapwing.com
Type here to search
12:24
12-07-2018


Films & TV
www.thundershare.net
kapwing.com
Type here to search
12:25
12-07-2018



So for all these activities this model makes prediction which are accurate. The model is tested in real time also.

8.Errors

1)Only some activities given high accuracy.

2)Some activities like brushing, applying lipstick, eye-makeup, shaving are often confused. The answers are wrong for such activities.

3)For low accuracies the top confidence score keeps on varying and the accuracy percentage can be as low as 12%.

4)The predictions are slow and so it lags a bit in real time. GPU is highly recommended.

5)Training was done for only 20 epochs and so it is a major the reason for the above errors.

6)The setup, model and UCF dataset requires much space and memory and so it can be done on 8GB PC. Also linux aids the process.

9. Scope of Improvement

1)Training was done only for 20 epochs, increasing epochs will give better resultswith accuracy.

2)Similar activities should be avoided. For example only one of these activities should be taken in the dataset: brushing, shaving, applying lipstick, eye-makeup.

3)GPU with high memory should be used for training and testing.

4)Fast-RCNN with LSTM can be used instead of CNN.

5)The pixel resolution is poor for frames of the videos(Video quality is'nt good enough.The code should be trained with other datasets too.

6)In some frames of the videos, activities are not done but still they are trained as a part of particular activity. This lead to erroneous dataset.

7)Also a class no activity should be added for instaces when no activity is done.

10. References

1) https://github.com/sujiongming/UCF-101_video_classification

2) https://zhuanlan.zhihu.com/p/28307781[Online]

3) Charles Han, YOLO-based Adaptive Window Two-stream Convolutional Neural Network for Video Classification.

4) Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., & Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition* (pp. 1725-1732).

5) Geoffrey Hinton, N Srivastava, and Kevin Swersky. 2012. Lecture 6a overview of mini–batch gradient descent. Coursera Lecture slides https://class. coursera. org/neuralnets-2012-001/lecture,[Online].

6) https://github.com/thtrieu/darkflow[Online]

7) L. Wang, Y. Qiao, and X. Tang, “Video action detection with relational dynamic-poselets,” in ECCV, 2014.

8) S. Saha, G. Singh, M. Sapienza, P. H. S. Torr, and F. Cuzzolin, “Deep learning for detecting multiple space-time action tubes in videos,” in BMVC, 2016.

9) X. Peng and C. Schmid, “Multi-region two-stream R-CNN for action detection,” in ECCV, 2016. [Online].Available:https://hal.inria.fr/hal-01349107v3

10) M. Marian Puscas, E. Sangineto, D. Culibrk, and N. Sebe, “Unsupervise tube extraction using transductive learning and dense

11) Sujiongming, https://github.com/sujiongming/UCF-101_video_classification[Online]

11. Glossary

1) epochs: Using all the batches once is 1 epoch. If there are 10 epochs it mean that the data is used 10 times (split in batches).

2) annotation: the labels the data, and annotation is the process of generating them.

3) keras: an open source neural network library written in Python. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

4) YOLO: You Only Look Once algorithm

5) activation function: A function in an artificial neuron that delivers an output based on inputs.

6) UCF101: A Dataset of 101 Human Actions Classes From Videos

7) Tensorflow: an open-source software library for dataflow programming across a range of tasks.

8) Softmax: In probability theory, the output of the softmax function can be used to represent a categorical distribution.

12. Appendix

This is the code which predicts generates frames and predict the outputs.

```
#The code generates frames and predicts confidence score for frames
import cv2
import os
import time

from PIL import Image
import numpy as np
import operator
import random
import glob
from UCFdata import DataSet
from processor import process_image
from keras.models import load_model
import matplotlib.pyplot as plt
def video(video_name):#generating frames
    cap = cv2.VideoCapture(video_name)

    while True:

        ret, frame = cap.read()

        if ret == True:

            cv2.imshow('frame', frame)
            if cv2.waitKey(30) & 0xFF == ord('q'):
                break

        else:
            break



def video_to_frames(video_name):
    # extract frames from a video and save to directory as 'x.png' where
    # x is the frame index

    vidcap = cv2.VideoCapture(video_name)
    count = 0
    num=0
    skip=-1
    anslst=[]
    acclst=[]
    data = DataSet()
    model = load_model('data/checkpoints/inception.017-2.46.hdf5')  # replaced by
your model name
    while vidcap.isOpened():
        success, image = vidcap.read()
        if success:
            cv2.imwrite(os.path.join(os.getcwd(), '%d.png') % num, image)
            flag=1
            while(flag is 1):
                if skip % 10 == 0:#display output for 10 images together
                    print('-' * 80)
                skip=skip+1

                images = '{}.png'.format(num)
                if skip % 10 == 0:
                    print(images)#output as 10,20,30.png
```

```
                image_arr = process_image(images, (299, 299, 3))
                image_arr = np.expand_dims(image_arr, axis=0)

                # Predict.
                predictions = model.predict(image_arr)

                # Show how much we think it's each one.
                label_predictions = {}
                for i, label in enumerate(data.classes):
                    label_predictions[label] = predictions[0][i]

                sorted_lps = sorted(label_predictions.items(),
key=operator.itemgetter(1), reverse=True)
                ans=sorted_lps[0][0]#top activity
                acc=sorted_lps[0][1]#percentage of top activity
                if acc>0.4:#threshold is 40 percent
                    anslst.append(ans)#add the first confidence score to the list
only when it is more than 40 percent
                if skip%10==0:
                    print(max(anslst,key=anslst.count))#print the most frequent
activity
                    anslst=['0']#clear the list

                im = Image.open(images)
                plt.imshow(im)#plot frames
                plt.title(ans)#show the top activity as the title
                plt.show()

                plt.clf()  # will make the plot window empty
                im.close()
                os.remove(images)#remove images
                # time.sleep(0.001)

                num=num+1
                count += 1

                break


        else:
            flag=0
            num=num+1
            count=count+1
            continue
    cv2.destroyAllWindows()
    vidcap.release()
    print("end")

#video('result.mp4')
print("start")
video_to_frames('new1.mp4')#new1 should be in the same folder as this code
```