A Project Report on

Object Detection Using Artificial Neural Networks

Submitted in partial fulfillment of the requirement for the course

CS671 – SYSTEMS PROJECT

SUBMITTED BY

DEEPTANSHU PAUL M 1560319

Under the guidance of

Dr GANESH KUMAR R

Department of Computer Science and Engineering

Faculty of Engineering,

CHRIST(Deemed to be University),
Kumbalgodu, Bangalore - 74.
March 2018.

A Project Report on

Object Detection Using Artificial Neural Networks

Submitted in partial fulfillment of the requirement for the course

CS671 – SYSTEMS PROJECT

SUBMITTED BY

DEEPTANSHU PAUL M 1560319

Faculty-In-Charge Examiner Head of Department

Department of Computer Science and Engineering

Faculty of Engineering,

CHRIST(Deemed to be University),
Kumbalgodu, Bangalore - 74.
March 2018.

Object Detection Using Artificial Neural Networks

Abstract:

In the domain of image classification and object detection, Deep Neural Networks (DNNs) have emerged as a favorable alternative to Cascade Classifiers which were previously the go-to method for object detection. This project explores the various Deep Learning Network Topologies and their accuracy with respect to the CIFAR-10 and COCO datasets, and finally, Real Time Object Detection.

The objectives of this project are:

1. **Implement** the VGGnet 16 and 19 artificial neural network topologies and train them on the CIFAR-10 dataset.
2. **Showcase** the ResNet-50 and MobileNet artificial Neural Network Topologies trained on the COCO dataset (Common Objects in COntext).
3. **Implement** Real-Time Object detection both from the System's webcam as well as a video stream from an Android mobile.

Introduction:

With the advent of video camera surveillance, the statement "Big brother is watching you" seemed quite apt. Video cameras were deployed across the world, and at first, humans were tasked with watching the feeds. However, a human can hardly be expected to monitor every detail in the stream that so many video cameras relayed.

Similarly, in the 2000s, just searching for websites wasn't nearly enough. When one required an image, one couldn't simply search for the keywords of the image and expect viable results from a search engine. Yet, it was something of a need. Hence, computer vision gained footing. Fast forward to today, and we have Amazon Go, a supermarket where there are no cashiers, no checkout counters. With computer vision, Amazon tracks every item that's placed in the customer's bag. This saves time for the customer, and Amazon doesn't have to hire cashiers and build cash counters in their upcoming supermarkets.

Hence, tasking a computer program to make accurate predictions on what an object in a picture was became an excellent problem statement. The traditional approach to this problem taken by the computer vision research community was to hand-craft functions which would look for particular features in the image that were believed to be indicative of certain objects or scenes. For example, hard corners and straight edges might be believed to indicate the presence of manmade objects in the scene. The responses from these feature extraction functions would then be fed into another function which would decide whether to declare a particular object had been detected in the image.

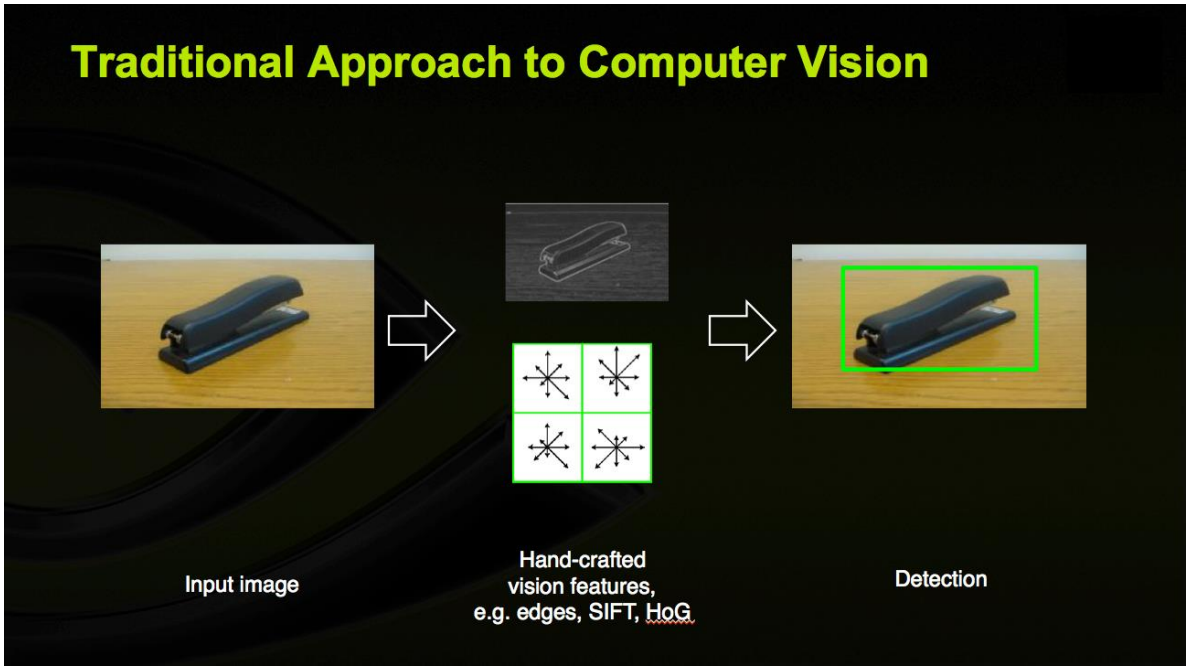


Figure 1: Traditional approach to machine perception: hand-crafted features are extracted from the raw data and is then independently passed to a classification function. This would typically lead to lower accuracy predictions, not to mention hand crafting took time.

Unfortunately, there are a number of problems with this approach. Firstly, it is very hard to think of robust, reliable features which map to specific object types. Secondly, it is a massive task to come up with the right combination of features for every type of object one would want to be able to classify. Thirdly, it is very difficult to design functions that are robust to the translations, rotations and scaling of objects in the image. Together these problems resulted in traditional computer vision struggling to develop high accuracy object detectors and classifiers for a broad range of objects.

Hence, Deep Neural Networks came into the picture. No human knowledge is encoded in the Deep Learning model about the types and combinations of features that are important for labelling different types of object or scene. Instead, a combined feature extraction and classification model was learned by allowing the computer to examine millions of labelled images to discover which features and combinations of features were most discriminative for each of the object classes, and this process is called training the model. Furthermore, this is done in such a way that the model doesn't just learn to classify the specific objects it is trained on; instead, it abstracts out the essence of those objects in such a way that it can recognize previously unseen but visually similar objects. This learning process refined tens of millions of free parameters in Deep Learning models enabling many to be able to accurately classify over 22,000 different object types (ImageNet's competition). The only down-side to this approach, one could say, would be that a lot of data is required to train a model to be accurate enough for the predictions of the model to be relied on, and on a traditional CPU based server this training would take weeks to complete. Networks these days are instead trained in just hours by exploiting GPU acceleration. Almost all Deep Learning Systems now are

trainable in practical amounts of time now. Below is an image depicting how Deep Learning works:

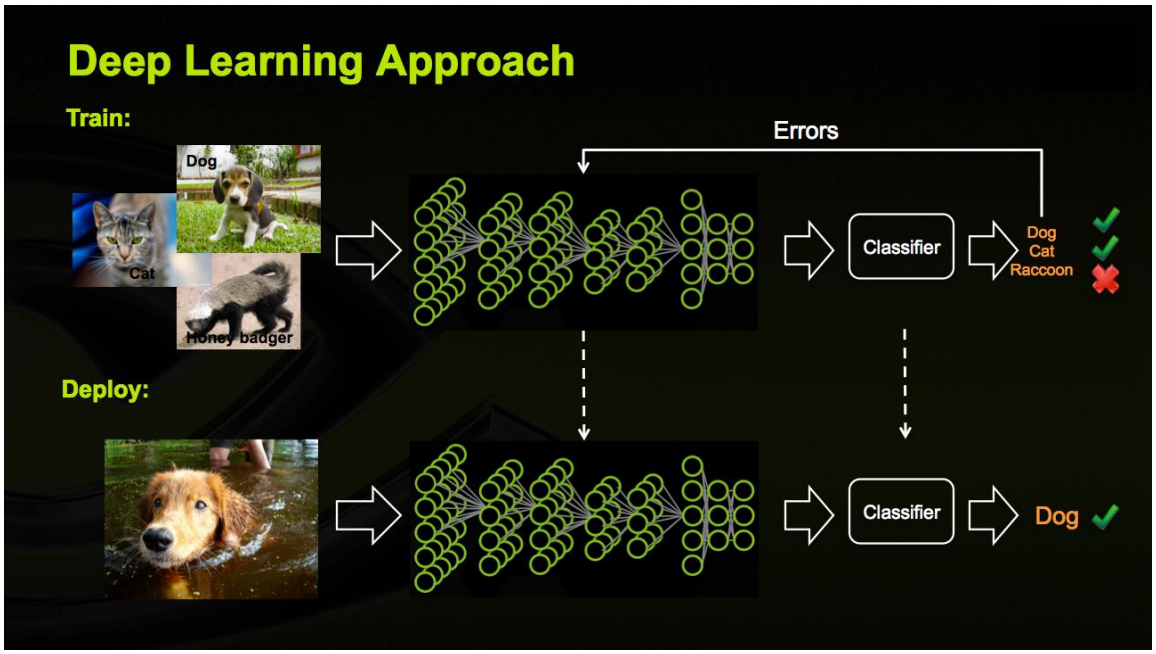


Figure 2: In the Deep Learning approach the feature extraction and classification functions are simultaneously learned using large amounts of training data. The learned model can then be deployed in a new application where previously unseen data samples are classified.

In the field of object detection in particular, a type of Artificial Neural Networks has shown great promise and great results. These are Convolutional Neural Networks(CNNs). They are comprised of one or more convolutional layers and have other layers such as fully connected layers to supplement them, making multilayered Neural Networks. Since images are essentially two dimensional in structure, CNNs are designed to take advantage of it. A convolutional layer takes an image ($w \times w \times c$ matrix where height and width are the same, and c is the number of channels) and the convolution filter or kernel is a smaller "image" (an $n \times n \times r$ matrix) where n is the height and width of the matrix and $r \leq c$ is the number of channels (RGB in this application). Below are examples of Convolutional Filters. These filters were stitched from the pre-trained model of VGG16 provided by Keras, and a static image was run through the filters leading to the images you see below. However, if a normal image is run through these convolution filters, they give quite a different look to those images, leading to Google dubbing them as "Deep Dream" images. What follows are the 64 best filters of the 5^{th} Convolution Block, 1^{st} layer, when the weights are initialized to the official ImageNet weights.

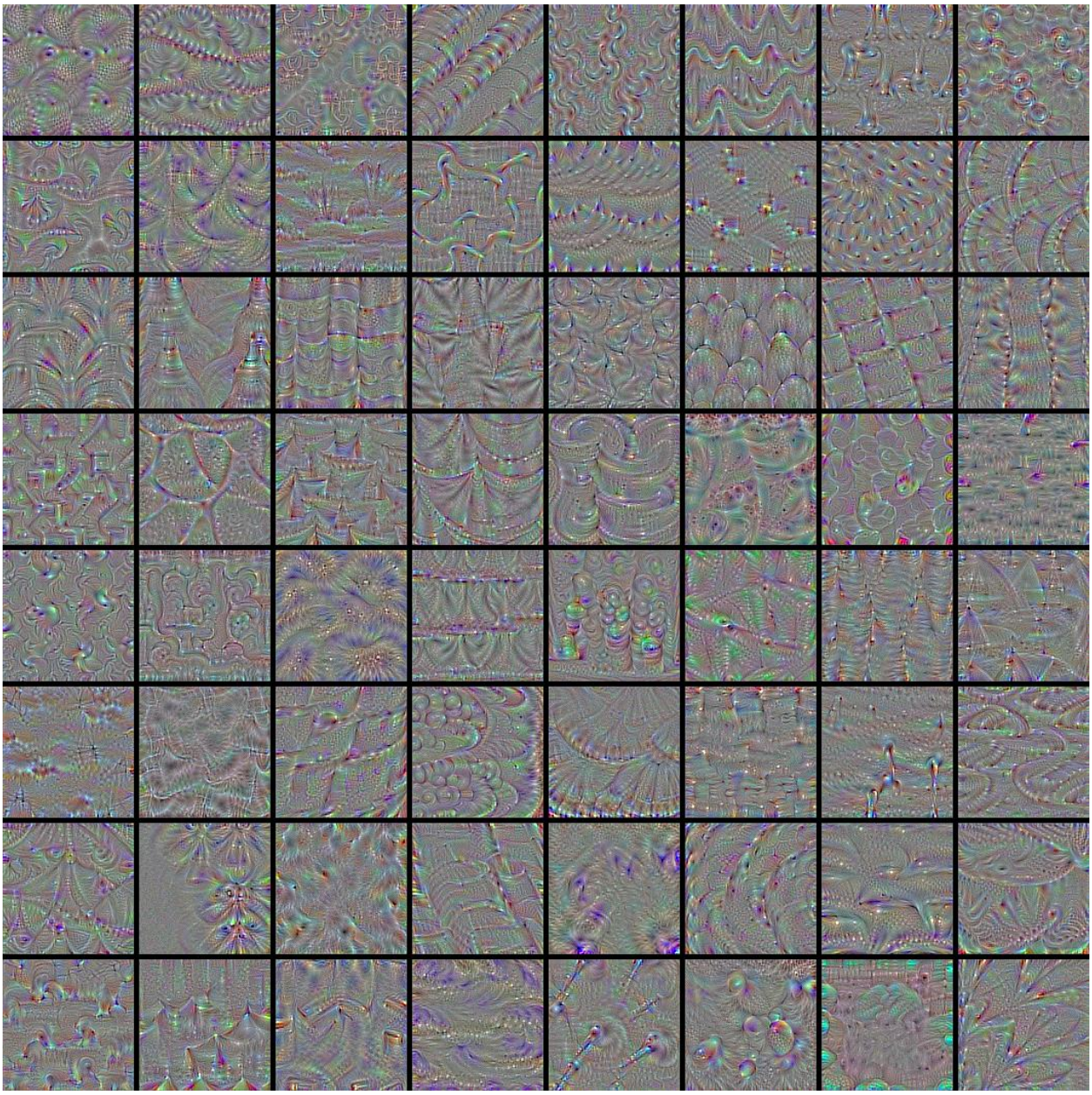


Figure 3: Convolution Filters are essentially miniature images, however, when realized, make no sense to us. They are very accurate, and a single convolution layer in VGGnet has 64 to 512 such filters giving it more versatility in object detection. These are the best 64 out of 512.

The Imagenet Challenge is the biggest annual competition where competitors are provided with 1.2 million natural images from the internet which are labelled with the objects that appear in those images using 22,000 different class labels. A wordnet is essentially combined with the Images to provide an easily processed dataset. Competitors must create a model using this data which will then be tested against a further 100,000 images to see how accurately the model can identify and localize objects within them. This provides a huge dataset to train networks on, and

has led to quite accurate models being developed. This project explores some of those models. Over the past few years CNN based approaches have come to dominate the competition with accuracy in the object identification task recently exceeding 95%, which is comparable with human performance in labelling the objects in the test dataset. However, training the images on the hardware available to a student is hardly capable of such a task and to assist, the weights of previously trained models are available quite readily.

These are, briefly, the concepts required to understand the work done in this project.

Literature Survey:

A total of 3 research papers have been taken for the literature survey portion of this project.

1. **VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION by Karen Simonyan & Andrew Zisserman**,
   Visual Geometry Group, Department of Engineering Science, University of Oxford
   This work was published by the Visual Geometry Group (VGG) from the University of Oxford in 2015, at the International Conference on Learning Representations 2015 that was presided by members from IBM research, Google, University of Oxford and Université de Sherbrooke. It truly is a revolutionary paper, as it brought forth the topology of the VGG16 and VGG19 Convolutional Networks for Image Recognition. This publication was done to publish the work that the Visual Geometry Group at Oxford had done in the ImageNet Challenge of 2014, wherein their work won the First and Second places in the Localization and Classification Tracks respectively. They are affiliated with Google's DeepMind team and the University of Oxford.

   The paper begins with an introduction to what Convolutional Neural Networks are, and they speak of the different improvements done to improve the original architecture of Krizhevsky(2012), such as utilizing smaller receptive window size and smaller strides in the first convolutional layer to modifying training process by first training the network over the entire image and then on multiple scales so that scaling of objects is done in a better fashion. The paper then details the ConvNet configurations, and specifies the one that they utilized, specifying the Architecture, Configurations and then mentioning observations that were done which led to certain changes, such as the dropping of Local Response Normalization layer use that didn't improve performance, in the paper's observation, on the ILSVRC dataset but instead increased the computation time as well as the memory consumption. The configurations they used were five in number, and they labelled them A-E. However, VGG16(D) and VGG19(E) are the winning networks, which is why they have been implemented in this project.

   The paper then specifies the classification framework, starting with how the training procedure was done, and they stress on the importance of the initialization of network weights. This is the reason why a total of 5 networks were made, as the first network, A, has only 11 layers hence random initialization of weights don't adversely

affect the outcome. For the Deeper networks, the first four convolutional layers were initialized to the last three fully connected layers of the network A, with the rest of the layers initialized randomly. The training was done on 224x224 images, and a training scale S was set to 384 and then a multi-scale training was implemented. The paper then talks of the testing phase, and gives the implementation details, wherein they used the Caffe framework and they trained it on multiple GPUs, and to quote, "On a system equipped with four NVIDIA Titan Black GPUs, training a single net took 2–3 weeks depending on the architecture." The rest of the paper details the results of their classification experiments, from single-scale evaluation to multi-scale evaluations and multi-crop evaluations. The paper then details the experiments where they fuse ConvNets and compare with state of the art entries to the ILSCRC-2014 challenge and presents that data as well and presents the conclusion. Finally they have a segment called Localization where detailed configuration data is provided.

This paper is truly a historical paper, as it marked the introduction of CNNs that are more accurate than even humans. The work of the Visual Geometry Group was amazing in many senses, as they improved upon already groundbreaking work and proceeded to set the standard for the networks to come. The VGG networks are single stream in nature and this presents a fairly implementable model for students to replicate as multiple streams lead to the need to configure GPUs/CPUs accordingly.

2. INCEPTION-V4, INCEPTION-RESNET AND THE IMPACT OF RESIDUAL CONNECTIONS ON LEARNING by Christian Szegedy, Sergey Ioffe and Vincent Vanhoucke,

Google Inc. 1600 Amphitheatre Parkway, Mountain View, California.

This work was published in 2016 by Google's team working on the ILSVRC 2015 competition, and they achieved a staggering Top-5 error of 3.08% in the test set of the ImageNet classification challenge with their Inception-v4 network. This paper is yet another historic paper that details the use of residual connections in their network architecture, which accelerated the training of Inception networks significantly.

The introduction acknowledges the achievements of the previous groundbreaking networks, namely Krizhevsky's 'AlexNet' that was also applauded in the VGG's paper. They then compare the two "pure" Inception variants, Inception-v3 and Inception-v4 with the hybrid Inception-ResNet versions. It should be noted that these networks are computationally VERY heavy. ResNet has been implemented in this project. The paper then has related work mentioned, and then mentions the Architectural choices made, from the architecture of the pure inception blocks to the residual inception blocks. Each architectural explanation is supplemented by diagrams to make it easy to understand. However, these networks are multi-stream networks hence the implementation is not easy. Variants of the networks are detailed through diagrams, and then the scaling methods and training methodologies are mentioned. The comparison between the networks is shown in the form of graphs up until 200 epochs of training, when the training curve reaches a near-

plateau phase. The paper then ends with the conclusion, giving the following brief, quoted directly from the paper. “

- Inception-ResNet-v1: a hybrid Inception version that has a similar computational cost to Inception-v3
- Inception-ResNet-v2: a costlier hybrid Inception version with significantly improved recognition performance.
- Inception-v4: a pure Inception variant without residual connections with roughly the same recognition performance as Inception-ResNet-v2.

We studied how the introduction of residual connections leads to dramatically improved training speed for the Inception architecture. Also our latest models (with and without residual connections) outperform all our previous networks, just by virtue of the increased model size.”

This paper truly shows revolutionary work, and most certainly may have been implemented in Google’s image recognition algorithms.

3. **MOBILENETS: EFFICIENT CONVOLUTIONAL NEURAL NETWORKS FOR MOBILE VISION APPLICATIONS by Andrew G.Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto and Hartwig Adam,**
Google Inc. 1600 Amphitheatre Parkway, Mountain View, California.

This paper was published in 2017 by Google’s team working on the use of Artificial Neural Networks for object detection on Mobile devices. They present a class of efficient models, wherein accuracy is not prioritized as much as is computational requirements, called MobileNets for mobile and embedded vision applications. They are based on a streamlined architecture which uses separable convolutions.

The paper starts by introducing two global hyperparameters that they say efficiently trades off between latency and accuracy. The introduction first acknowledges the pioneer network in this domain, AlexNet, and talks of prior work, where the need for MobileNets is briefly outlined and then the architecture is presented, where the two hyperparameters are width multiplier and resolution multiplier.

The paper first talks of Depthwise seperable convolutions which are a form of factorized convolutions which factorize a standard convolution into a depthwise convolution, and convolutions of 1x1 are called pointwise convolution. The pointwise convolutions are used to combine the outputs of the depthwise convolutions. This reduces the amount of computation required while not trading off on the accuracy much. Then the network structure and training are specified, with the modifications to the standard layers also mentioned. The models were trained in Tensorflow. Then then paper talks of the two hyperparameters, and their effect. Data is provided to support the assertions made. The accuracy of MobileNets are made with regard to ImageNet accuracy, to other models. Finally the combination of MobileNet with SSD 300 is shown, and shows how multiple

objects can be recognized. The conclusion then states that this is the next step toward Mobile Intelligence, and their assertions are again specified.

Materials and Methods:

To begin with the Materials, first the Hardware Specifications, the Models were built on a System with an i7-7700hq, 8 GB Ram, and the GPU used was an Nvidia GeForce 1050ti with 4 GB of VRAM.

The Datasets used are:

1. **CIFAR-10:**
   The CIFAR-10 dataset consists of 60,000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

   The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. The classes and actual examples are below:

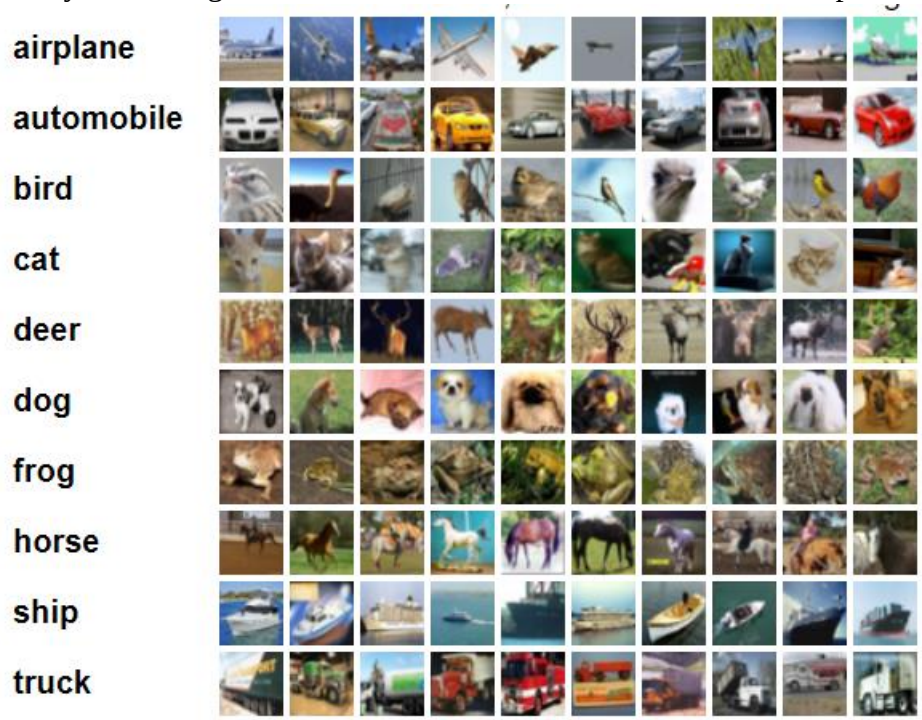


Figure 4: The CIFAR-10 Dataset consists of 60,000 32x32x3 images, between 10 different classes of images. A human being has an average accuracy of 94% on this dataset.

This Dataset is provided by the Canadian Institute For Advanced Research (CIFAR) and is a computationally not too intensive dataset to work with, hence this was chosen as the dataset to work with for this project.

2. **MS-COCO dataset:**

COCO stands for Common Objects in COntext. It is a large-scale object detection, segmentation and captioning dataset. It is provided by Microsoft.The features of COCO include Object Segmentation, Recognition in Context, Superpixel stuff segmentation. It has 330,000 images in which over 200,000 are labeled. There are 1.5 Million object instances and 80 object categories, 91 stuff categories, 5 captions per image and 250,000 people with keypoints. Examples of images in this dataset are:

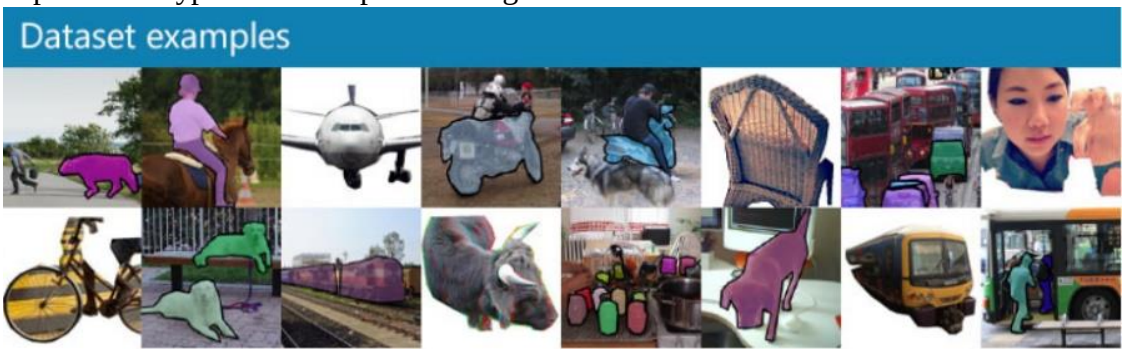


Figure 5: The COCO dataset has masks for the objects for more accurate training for object recognition. The actual picture and the object in context is provided.

In the words of the research paper, "We introduce a new large-scale dataset that addresses three core research problems in scene understanding: detecting non-iconic views (or non-canonical perspectives) of objects, contextual reasoning between objects and the precise 2D localization of objects. For many categories of objects, there exists an iconic view. For example, when performing a web-based image search for the object category "bike," the top-ranked retrieved examples appear in profile, unobstructed near the center of a neatly composed photo". Some more examples of how the dataset is, follows:

(a) PASCAL VOC. (b) MS COCO.

All the models and files made in this project are built in Python. The additional non-inbuilt libraries that are used are linked in the References portion. The models built and used are:

1. **VGG net-16 (VGGnet-D):** This is a 16 layered network consisting of 13 Convolution layers of 64, 128, 256 and 512 Neurons followed by 3 Fully connected layers of 4096 x 2 and 1000 neurons. The output layer is a softmax layer. The total number of parameters for this network is 138 million. The network topology is as follows, for an input of 224x224x3:

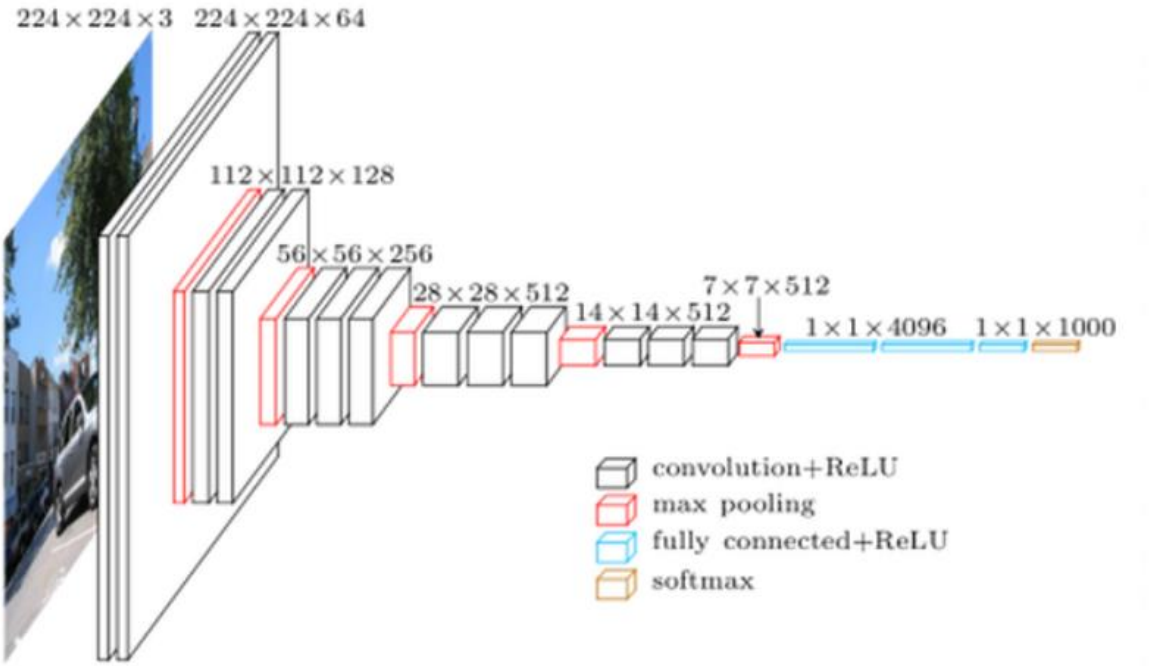
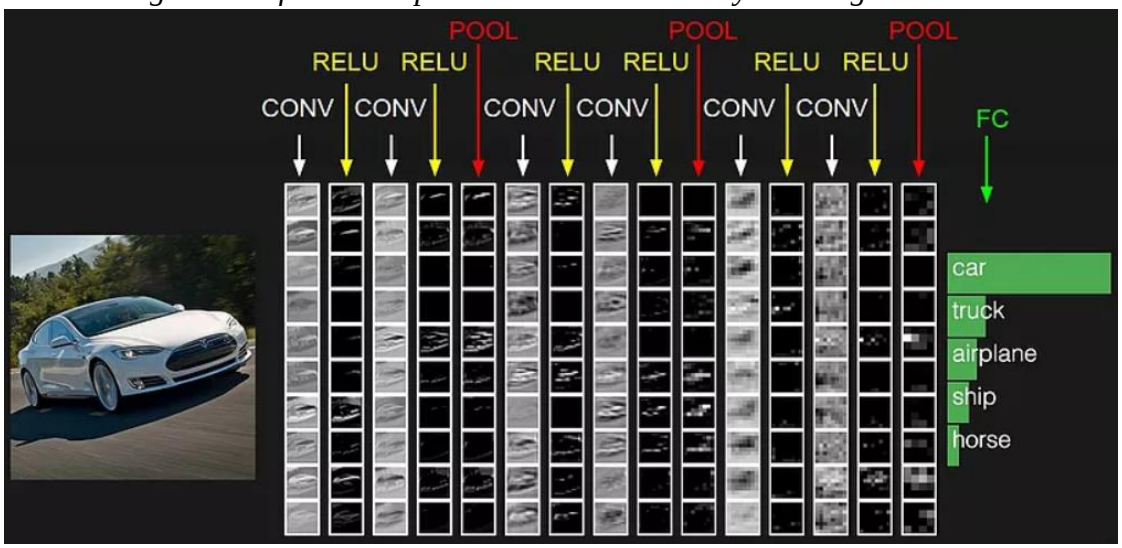


Figure 6: A diagrammatic view of the VGG-16 for a 224x224x3 image.

The one that is implemented however uses a 32x32 input. The Kernel Size however for the convolution filters do not change. In the implementation part, the actual network topology with the help of the TensorBoard module is shown. None of these images are from the models implemented in this project. A detection through the layers follows:

Figure 7: A forward input leads to the shown layers being activated.



2. **VGG net-19 (VGGnet-E):** This is the 19-layer network proposed by Oxford's Visual Geometry Group, the changes from the VGG-16 being that it has an extra convolution layer in the third, fourth and fifth convolution blocks. The architecture follows:

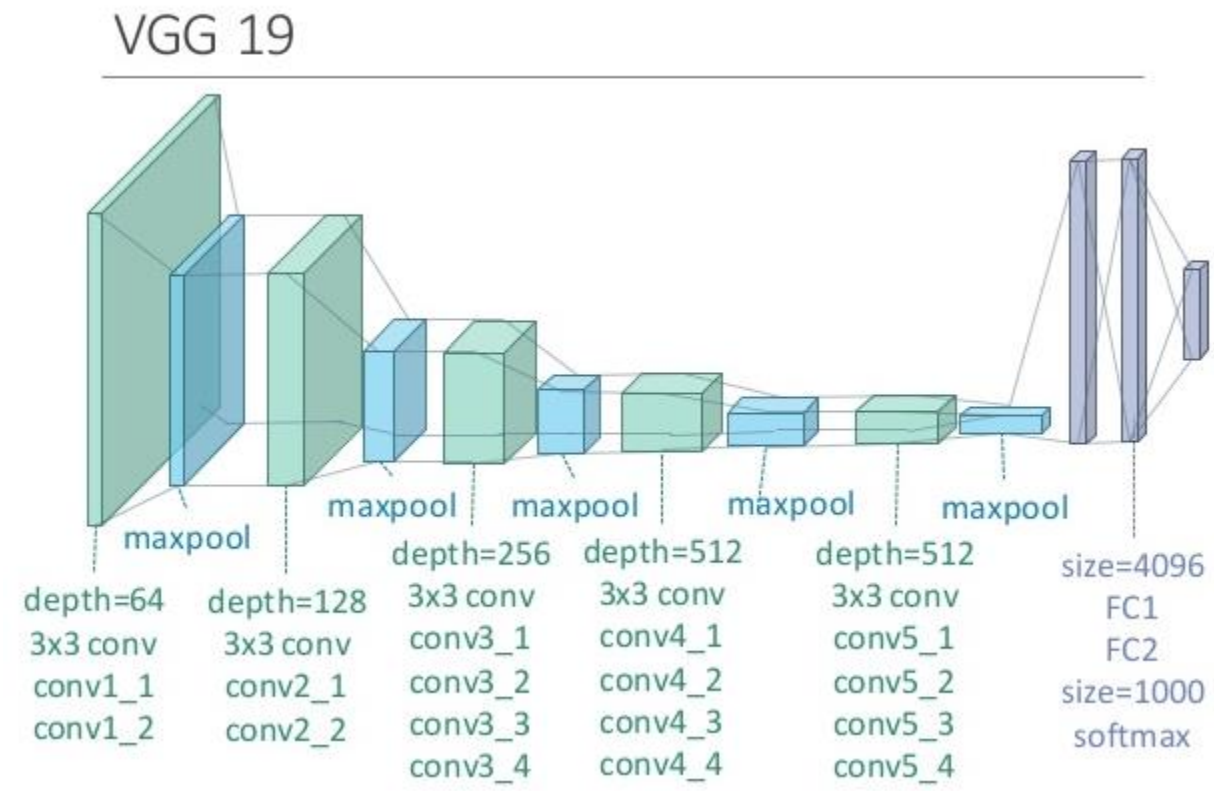


Figure 8: The architecture of the VGGnet-E network has a fourth convolution layer in the 3rd, 4th and 5th convolution blocks, hence making a total of 19 weighted layers.

The VGGnet -19 has both slightly less accuracy as well as is computationally more intensive than the VGGnet -16 (D) which was the sweet spot that the VGG team selected for their submission to the ImageNet competition. Below are some results taken from https://github.com/jcjohnson/ as the benchmark results, run on the ImageNet-2012 version, trained on the Nvidia GeForce GTX1080 GPU which has 8 GB of VRAM.

Network	Layers	Top-1 error	Top-5 error	Speed (ms)
AlexNet	8	42.90	19.80	14.56
Inception-V1	22	-	10.07	39.14
VGG-16	16	27.00	8.80	128.62
VGG-19	19	27.30	9.00	147.32
ResNet-18	18	30.43	10.76	31.54
ResNet-34	34	26.73	8.74	51.59
ResNet-50	50	24.01	7.02	103.58
ResNet-101	101	22.44	6.21	156.44
ResNet-152	152	22.16	6.16	217.91
ResNet-200	200	21.66	5.79	296.51

Figure 9: These results showcase the difference between networks, taking into account even the speed of prediction, as that is an important parameter as well.

The VGG-19, as one can quite clearly see, can easily be replaced by both the VGG-16 as well as the Resnet-34 onwards, both in terms of time taken vs error. AlexNet which was the pioneer has a huge error difference between itself and the other networks, however it is also the fastest by far. ResNet-18 is the next fastest with a fairly decent accuracy, when speed is also a priority.

3. **ResNet-50:** This was one of the models suggested in Google's Inception v4 paper, as a residual Inception network. It has 50 layers, as the name suggests, yet computationally isn't that expensive, although for real time applications one can argue that it's hardly a viable choice, with lower end systems. However, for image classification, ResNet-50 is very widely used, and its use is implemented in this project as well. The ResNet-50 uses the "pure" Inception layers but adds a residual connection that boosts its accuracy. The ResNet is in fact one of the pre-trained models provided by Keras due to its widespread use.

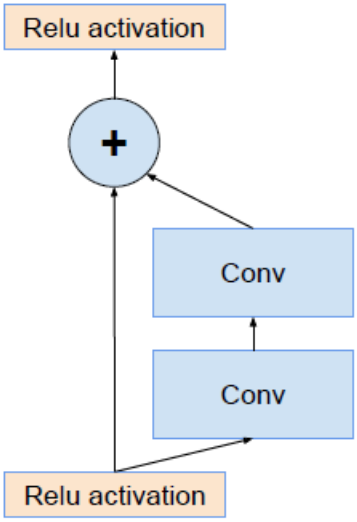


Figure 1. Residual connections as introduced in He et al. [5].

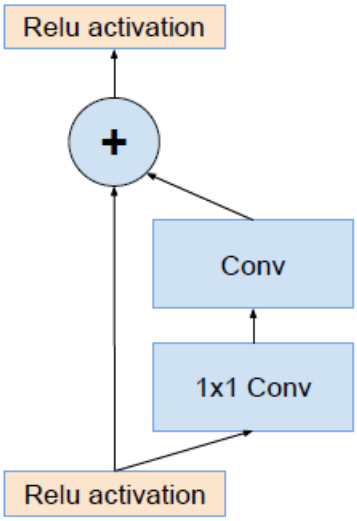


Figure 2. Optimized version of ResNet connections by [5] to shield computation.

Figure 10: The changes to the "pure" Inception layers introduced in ResNet as per the Inception v4 paper. The image was taken from the Research paper.

Pure Inception Layers on the other hand look quite different, but the stem of the network's topology remains more or less the same. The only changes are what are shown above, and the Inception topology is powerful, powerful enough for Google to utilize it in their image search algorithms. The following image shows the "stem" of the Inception v4 network, as per the paper. Again, it should be noted that these images are taken from the research paper itself, which is given in the References Section.

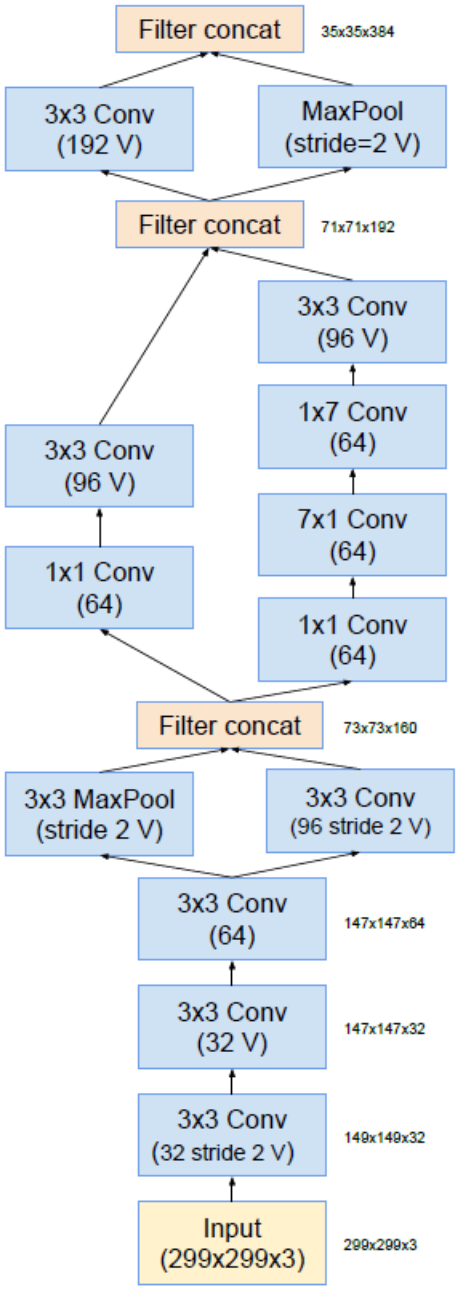


Figure 3. The schema for stem of the pure Inception-v4 and Inception-ResNet-v2 networks. This is the input part of those networks. Cf. Figures 9 and 15

Figure 11: The "stem" of the Inception v4 as well as ResNet Networks

4. **MobileNet SSD:** As mentioned in the MobileNet research paper, MobileNets are built with speed in mind, not accuracy. However, in reality the tradeoff isn't much, and MobileNets give a decent accuracy, which can really be ignored as it provides a good FPS when run on Videos. MobileNet SSD network refers to the Single Shot Detection version of MobileNet. It can be used to analyze frame-by-frame and give box-detections, without specifying

actual object boundaries as is implemented in the ResNet-50 part of this project. The following image is a SSD MobileNet implementation, using VGG16 layers:

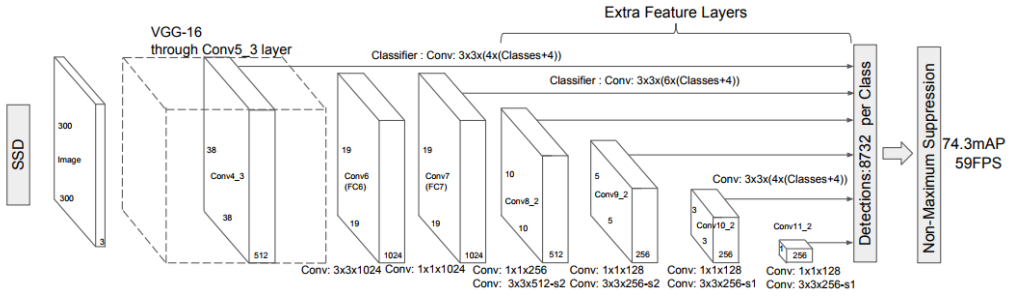


Figure 12: The number of layers is reduced and the blocks are shuffled as well to provide maximum speed without trading off on accuracy too much.

Self-driving cars often use MobileNets to identify the objects around them, to supplement the self-driving algorithm itself.

Actual Work/ Implementation:

The implementations were done in Python 3.5, using Tensorflow 1.6.3, running on the GPU. The different parts of the project that were implemented were:

- VGG-16 with Batch Normalization and Dropout layers added, trained manually on the CIFAR-10 Dataset.
- VGG-19 with Batch Normalization and Dropout layers added, trained manually on the CIFAR-10 Dataset.
- Video detection using ResNet-50, pre-trained on the MS-COCO dataset, with masks to show object detections.
- Real-time object detection using MobileNet SSD trained with 91 classes of objects on the MS-COCO dataset, in TensorFlow, utilizing both the System's inbuilt camera as well as a stream from an Android device.

1. **VGGnet-16 (D):**
   This module was implemented in Python, using Keras with Tensorflow as the backend. The dataset used was the CIFAR-10 dataset, with the training set consisting of 50,000 images and the test set being 10,000 images.

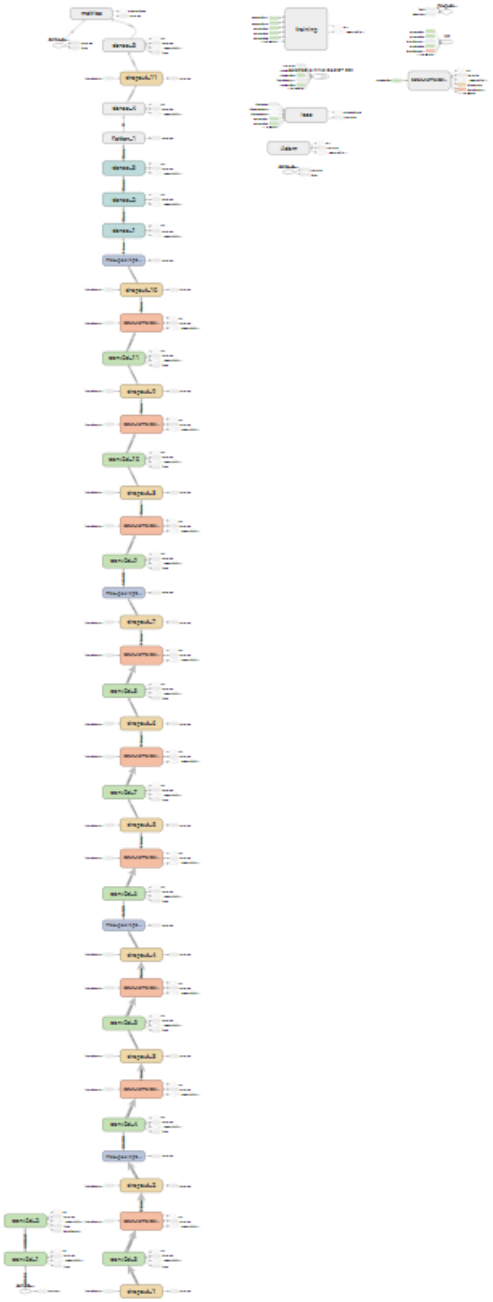


Figure 13: The implemented Architecture, with Batch Normalization and Dropout Layers.

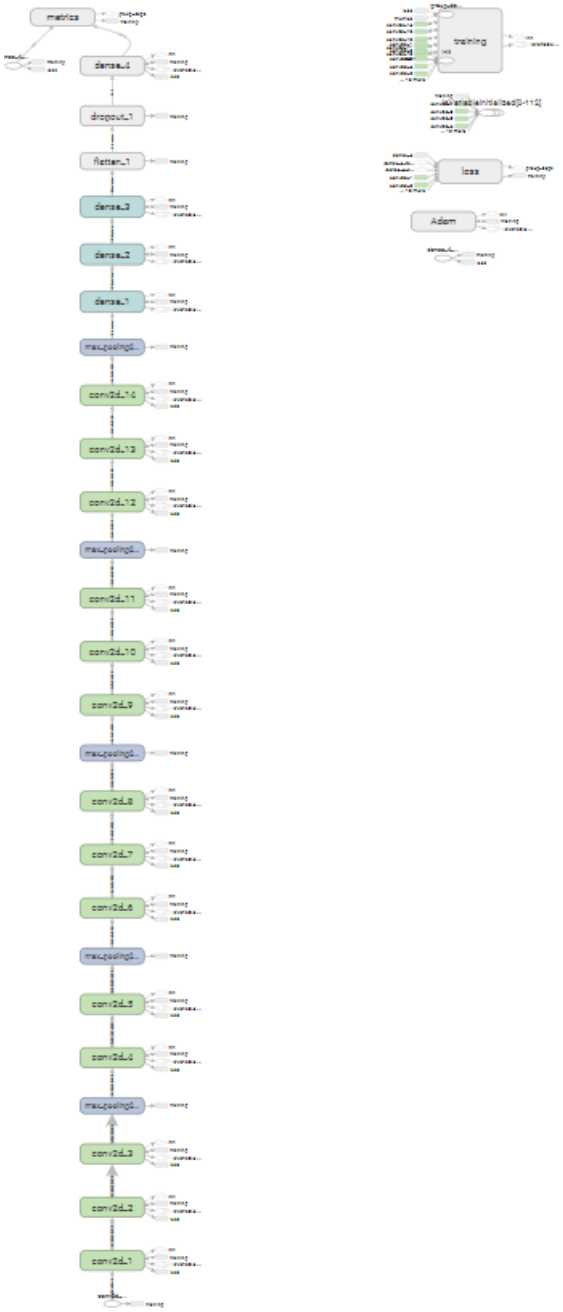


Figure 14: Vanilla VGG-16 with no Batch Normalization.

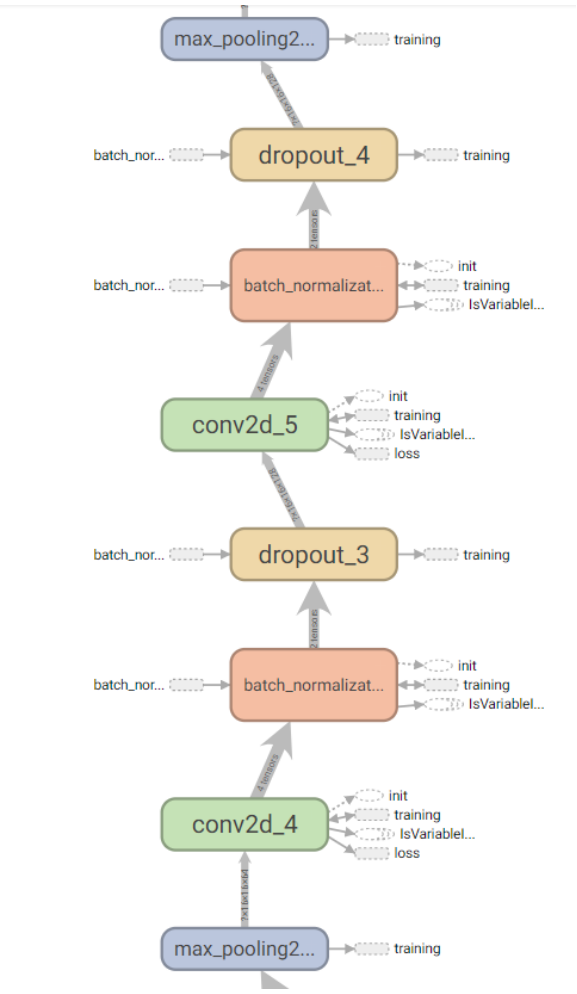


Figure 15: The Modified Convolution blocks implemented in this project.

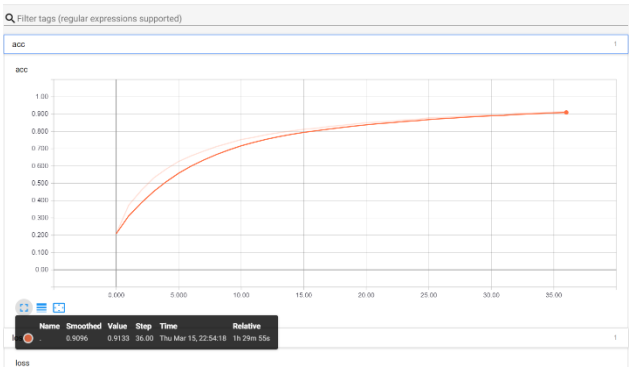


Figure 16: The Accuracy Values over epochs. Ended at 91.33% accuracy.

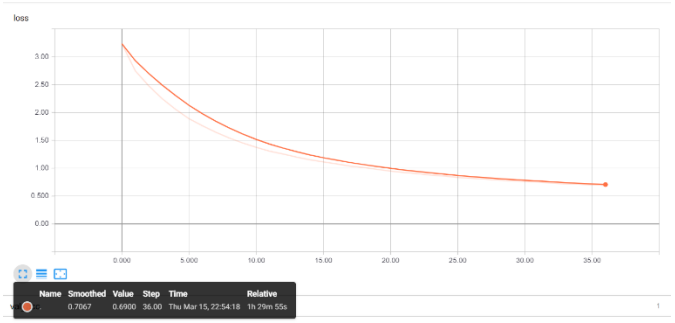


Figure 17: The Loss Values over epochs. Ended at 0.69 loss.

The TensorBoard Modules allow us to visualize the networks that were implemented. The VGG16 network has Batch normalization after each Convolution layer as per the research paper's suggested improvements. The final accuracy values were 91.33% Top 1 accuracy in the CIFAR-10 Database, comparable to a human that averages at 94% Top 1 accuracy. However, CIFAR-10 trained data is not ideal for real time object detection. Hence, just the training data and network topology was observed. The model took 1 hour and 29 minutes to train 36 epochs, which is 149 seconds per epoch, on the Nvidia GeForce 1050ti, running 768 CUDA cores.

2. **VGGnet-19(E):**
   This module was also implemented in Python, using Keras with Tensorflow as the backend. This model was also trained on the CIFAR-10 dataset, using a training set of 50,000 images and a test set of 10,000 images, as is the standard training/test split recommended for this dataset.

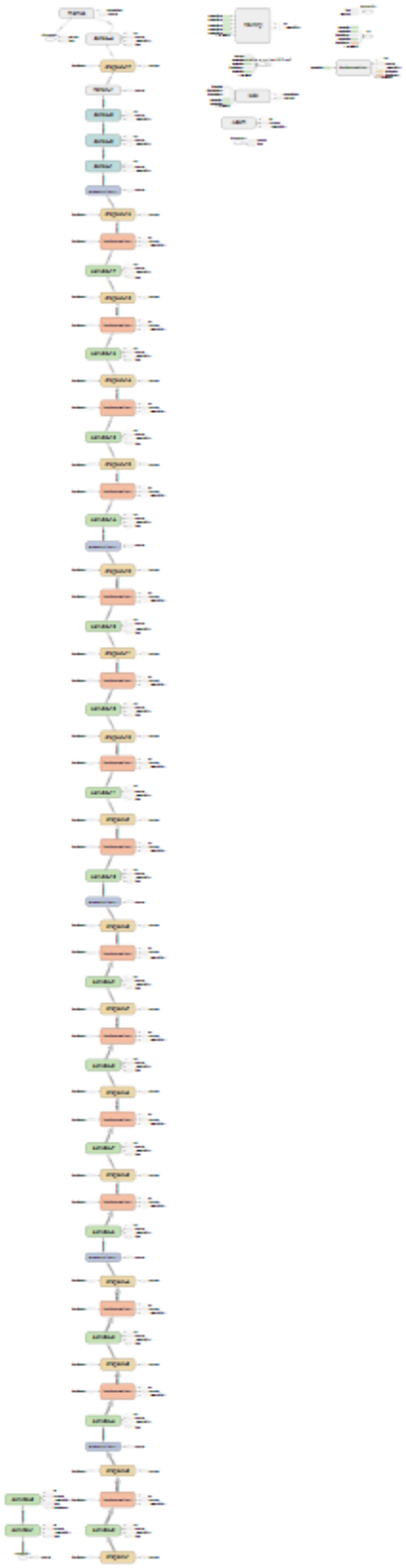


Figure 18: The VGGnet-19 architecture is larger, as each extra layer has in fact, three extra layers as the Batch Normalization and Maxpooling layers are implemented.

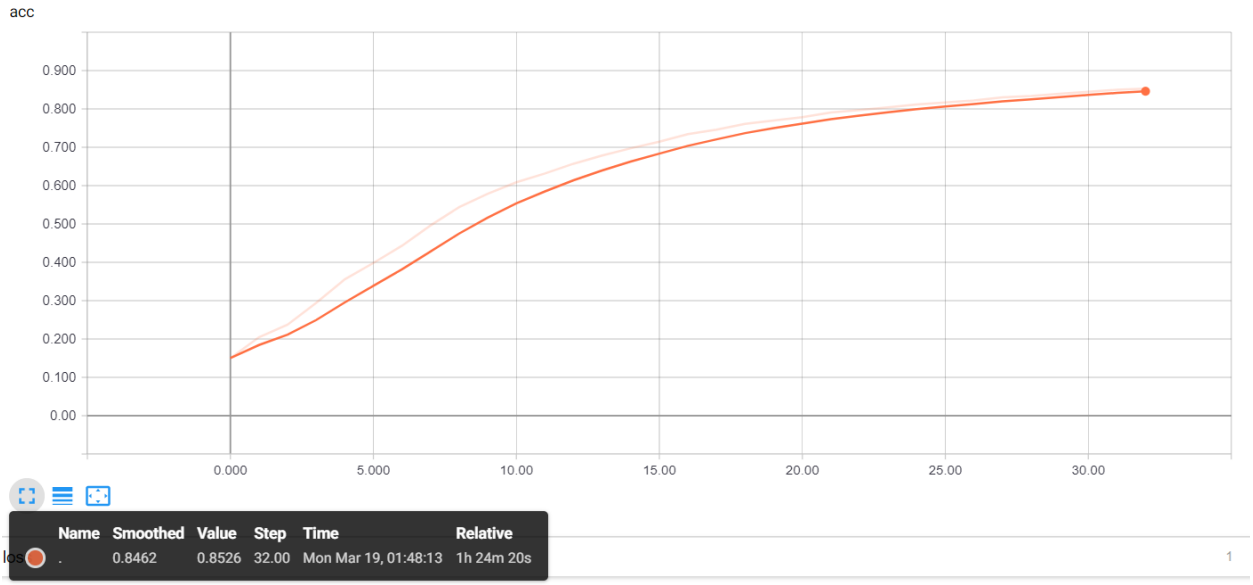


Figure 19: The varying of model accuracy over time is shown. The EarlyStopping module terminated the training at 32 steps and led to an accuracy of 85.26%

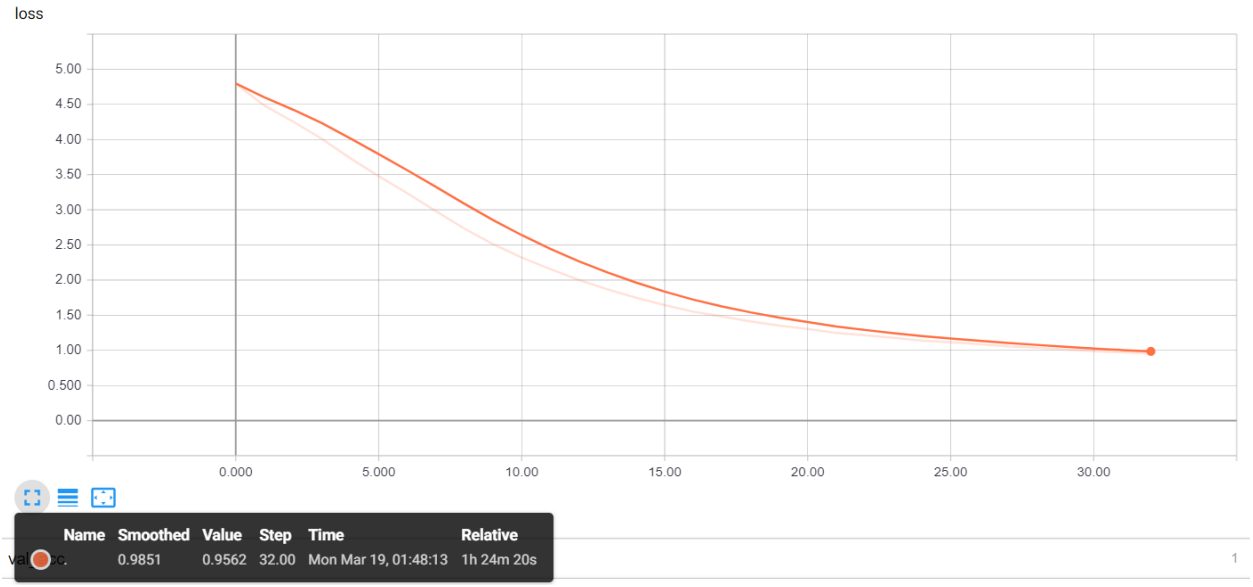


Figure 20: The Loss values of the model are at 0.95 after 32 steps

The model took 1 hour and 24 minutes to train 32 epochs, taking 158 seconds per epoch on average, on the Nvidia GeForce 1050ti. Once again, it should be noted that just the training of the model was done and the implementation not shown due to the fact that the CIFAR-10 does not consist of commonly seen objects, and scaling is hard at 32 pixels of width and height.

3. Resnet-50 with Masks:

The ResNet-50, by Google, is a fairly accurate network. By using a pre-trained model on the MS-COCO dataset, the project has a video stream taken from the videocam and then that is processed and displayed using masks to show object demarcations and detections. Screenshots of the implementation follow:

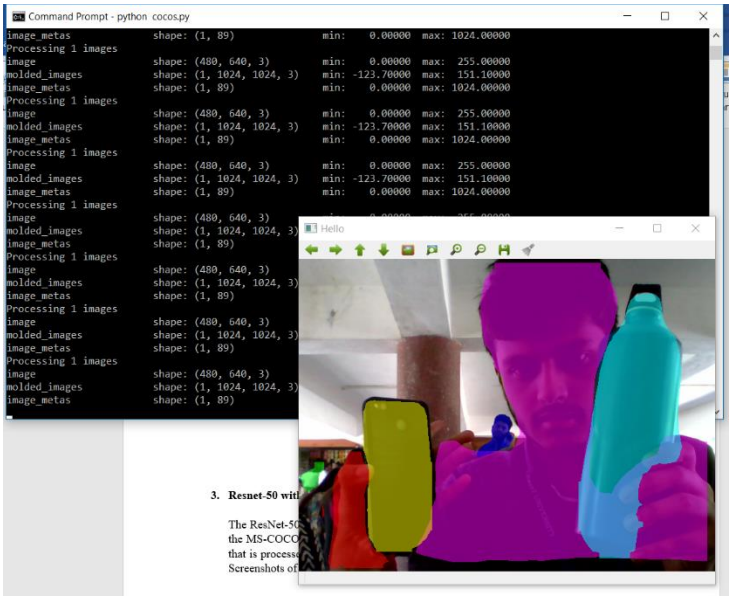


Figure 21: Resnet-50's detections with masks to demarcate object detections. Bottle, Person and Cell phone are some of the detections shown above.

This network has an accuracy of 76% on the COCO dataset, at top-1 error being 24%. The top-5 accuracy is a stunning 92.98% on the COCO dataset, however this was not tested in the project, but the data provided by the author of the pretrained weights. The model is built using Keras with a Tensorflow backend, with the custom layers being built in Tensorflow itself and then imported and run separately. The Residual weights are stored in the RAM, and hence the RAM usage is high in comparison to the other hardware usage. The performance, however was at 2-3 frames per second, running on the CPU, an i7-7700hq with 4 cores; 8 threads. The model did not, however, utilize the GPU for anything other than the detections part of the model, the processing was done on the CPU for most part, as can be seen by the CPU-GPU usage stats below:

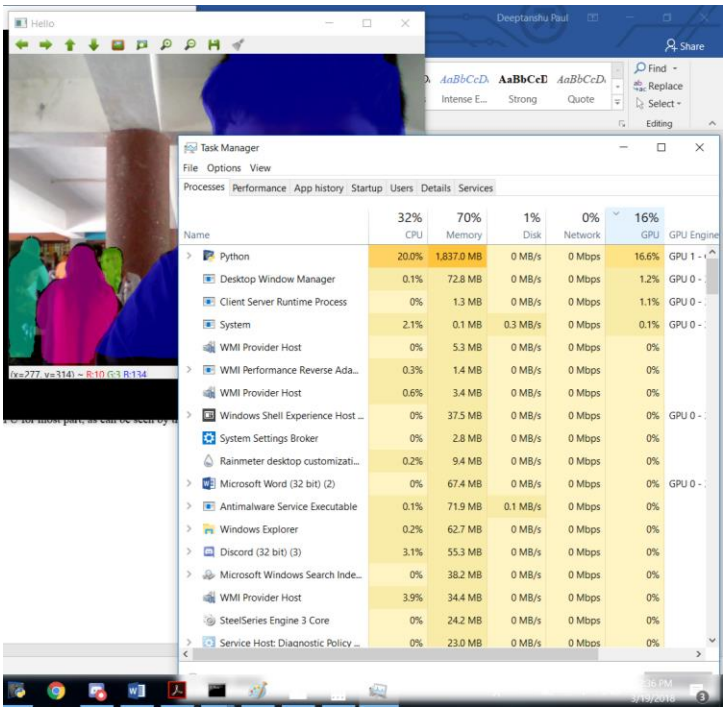


Figure 22: The Task Manager's reported usage of various hardware resources.

4. **Real Time Object Detection using MobileNet SSD on both Webcam as well as through an Android mobile:**

   Real time object detection is the requirement of the hour and hence, MobileNets are used. As the research paper stated, the performance was indeed very impressive, with a framerate of up to 16 FPS achieved when the input stream gives an FPS of 18-20 FPS. These values are not constant, however, and if there are more objects, the FPS does take a hit as well. The implementation was done in Python, using OpenCVs Deep Neural Network library to import the Tensorflow model for MobileNetSSD. The other modules used are imutils, to capture the videostream from the camera, urllib3 to capture the Video Stream from the Android Mobile, whereas the Android device used the IP Webcam application to host the video which the camera captures. Single Shot Detection refers to taking a single frame and processing it. In order to deliver a framerate of 14 FPS with the source being at 20 FPS would require for a detection speed of 10 milliseconds, keeping in mind that the image preprocessing part, such as the scaling of the image, etc, take time too. The Method used is called Multi-box detection, hence in a single frame, the algorithm detects more than one object, and each of the detections are highlighted in the output detections of the network. This does, however, sometimes lead to the same object being labelled as two different things, yet a MobileNet based on an Inception type of topology and structure would yield more accurate results, but the hardware demands would also be more. The images that follow are those taken from the model implemented.

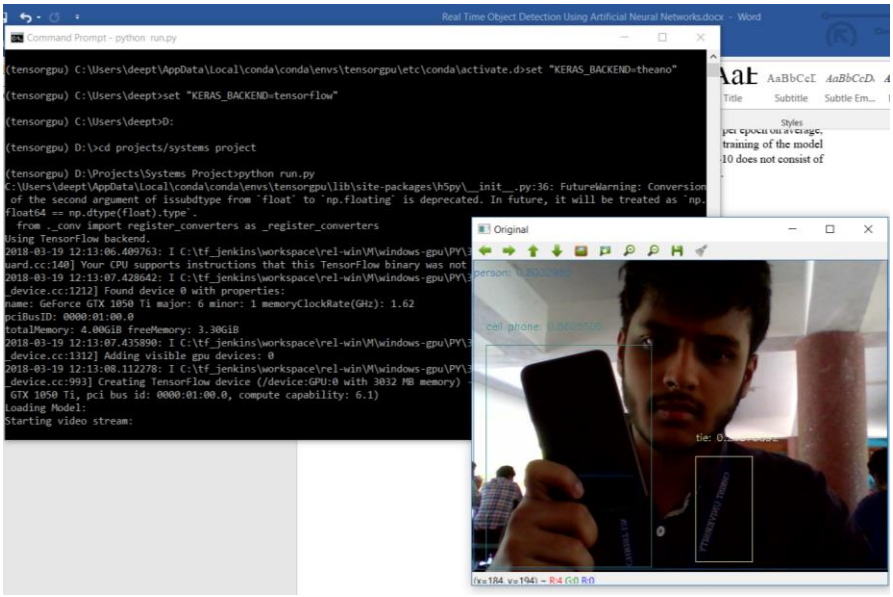


Figure 23: Real Time Object detection from the Computer's Camera. The resolution is 640x480.

The Camera input is 640x480, but since the MobileNet-SSD takes an input of 300x300, the image was first preprocessed down to 300x400 and then scaled to 300x300. The output however, as can be seen, is at the native resolution; the detections are transformed according to the output of the neural network and then shown on the original image.

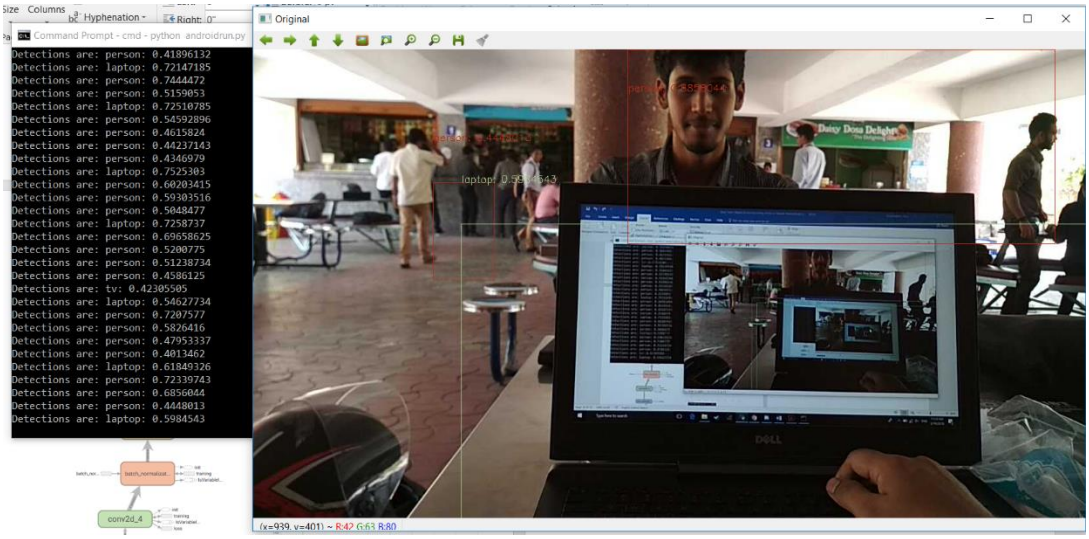


Figure 24: Real time Object detection done on the video stream from the android mobile. The resolution is 1280x720.

The camera input in this case is 1280x720 and this again is scaled down in the preprocessing part to 300x300 and fed into the MobileNet SSD. The detections are again scaled back up to 1280x720

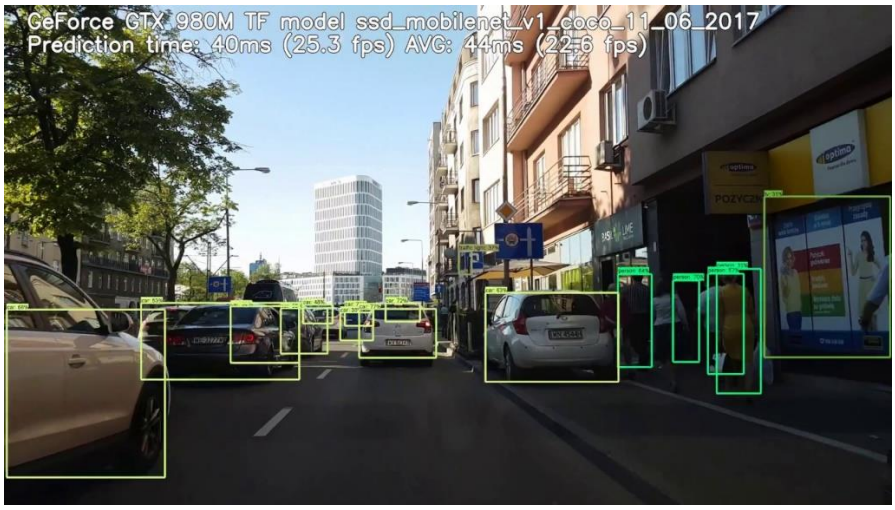
and then displayed. The network isn't extremely accurate, posting scores of 65% on the COCO dataset. All of this, while the drop in FPS from the source to output was hardly a 2 FPS drop. This streaming was done over Wifi, however, if cloud is used, this can be done remotely as well, leading to great amount of potential in the field of Self-driving vehicles.

Results and Discussions:

Artificial Neural Networks, as seen above, have really proven to be quite reliable and useful in the field of object detections. In Self Driving Cars, the Drive PX2 module that's used by Audi and such are built specifically to host a Neural Network on the GPU provided, and it provides detections at an amazing 60 Frames Per Second. Neural networks, compared to Cascades and Feature recognition, are very accurate and versatile, responding to issues such as scaling and angles in a fairly competent fashion. Convolutional Networks have the added advantage of being able to completely ignore the issue of Scaling, as convolution filters of 3x3 size cater to even the smallest possible discernable object instance in an image.

In the future, once GPU computing becomes more refined, the accuracy levels and performance levels can truly reach heights, with self-driving cars already on the road, such as the Tesla. Object detection in images make search engines so much more accurate, as instead of an image being indexed under a single keyword, it can now be indexed under multiple keywords. This way, if a person wishes to search for say, a mountain with a lake under it, it would show more refined results as only pictures indexed under both a mountain and a lake would show up. Once the image is indexed, no more processing needs to be done, and hence, since millions of images can be processed and indexed in seconds by a server consisting of multiple GPUs, this is now a very viable task, and is being implemented.

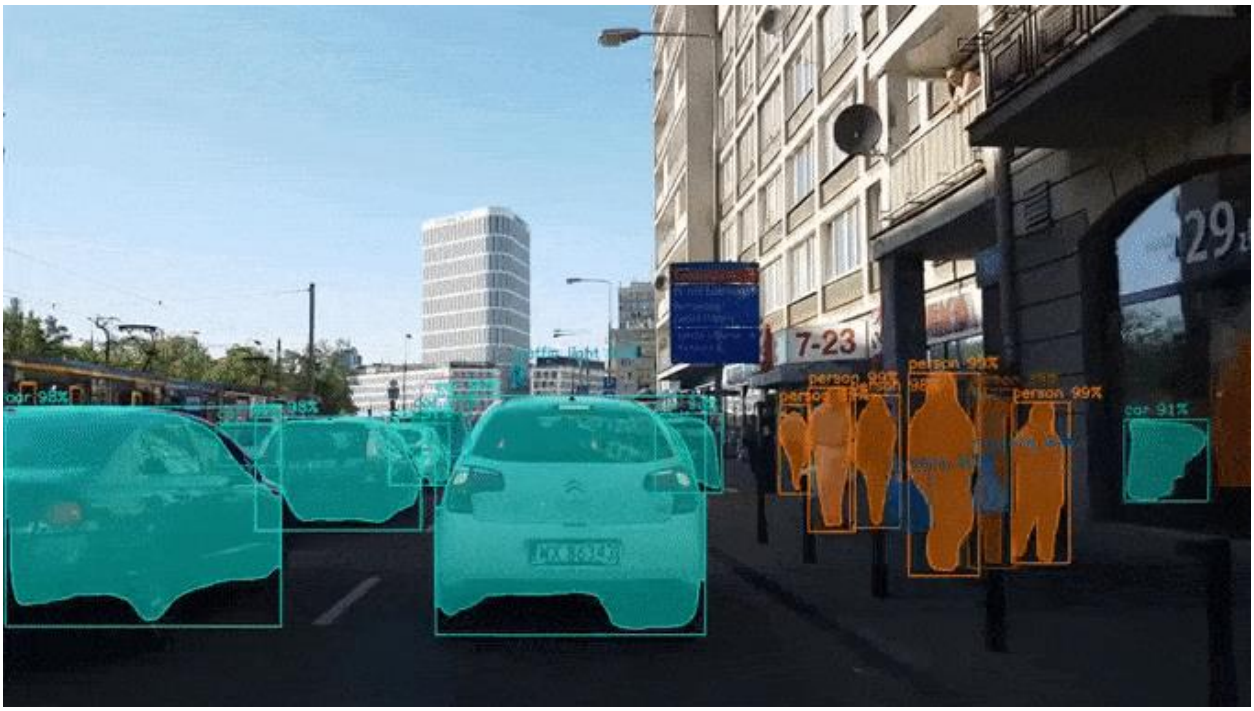
Artificial Neural networks due to their capability to learn, are widely used now, as error parameters once passed can be used to single out mistakes. In fact, there is an AI toolkit in which a Deep Neural Network monitors the training process, correcting the errors of the Network that's being trained to improve accuracy.



As seen in the image, the object detection actually is quite accurate and provides 22.6 FPS, which is good enough to decide the path of a vehicle when utilizing a self-driving module.

Conclusions:

In a day and age where tasks that previously required human input have been taken up by computer systems, Artificial Neural Networks pose a fair alternative to human presence in certain applications. Deep Learning has mimicked human and animal behavior in many cases and has led to many breakthroughs in automation, and with Object detection in particular, has shown prowess and can replace humans, provided the right hardware is available. Deep Learning networks have been deployed in many places, from Google's search algorithms to the self-driving cars of Tesla and Audi too.



This is an actual implementation of ResNet, however it was modified for the use of 4k Video so as to get extremely accurate predictions, that in turn leads to perfect path-plotting.

Both images are taken utilizing different models to analyze the input image.

Hence, in Conclusion, Convolutional Neural Networks, Deep Neural Networks can actually replace human beings in many aspects of object detection related tasks, and this project has demonstrated its use, specifically in the domain of Real Time Object detection, as well as tested accuracy on the CIFAR-10 and MS-COCO datasets.

References

[1]K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", *Arxiv.org*, 2015. [Online]. Available: https://arxiv.org/abs/1409.1556. [Accessed: 15- Mar- 2018].

[2]C. Szegedy, S. Ioffe, V. Vanhoucke and A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning", *Arxiv.org*, 2016. [Online]. Available: https://arxiv.org/abs/1602.07261. [Accessed: 15- Mar- 2018].

[3]Y. Zhang, H. Peng and P. Hu, "CS341 Final Report: Towards Real-time Detection and Camera Triggering - Semantic Scholar", *Semanticscholar.org*, 2017. [Online]. Available: https://www.semanticscholar.org/paper/CS341-Final-Report%3A-Towards-Real-time-Detection-an-Zhang-Peng/01b527641d1f23cddac932d9fe20b44ec39114a3. [Accessed: 15- Mar- 2018].

[4]A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", *Arxiv.org*, 2017. [Online]. Available: https://arxiv.org/abs/1704.04861. [Accessed: 15- Mar- 2018].

[5]T. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. Zitnick and P. Dollár, "Microsoft COCO: Common Objects in Context", *Arxiv.org*, 2017. [Online]. Available: https://arxiv.org/abs/1405.0312. [Accessed: 15- Mar- 2018].

[6]A. Krizhevsky, I. Sutskever and G. Hinton, "ImageNet classification with deep convolutional neural networks", *Dl.acm.org*, 2012. [Online]. Available: https://dl.acm.org/citation.cfm?id=2999257. [Accessed: 19- Mar- 2018].

[7]v. Ozhiganov, "Convolutional Neural Networks vs. Cascade Classifiers for Object Detection - DZone Big Data", *dzone.com*, 2017. [Online]. Available: https://dzone.com/articles/cnn-vs-cascade-classifiers-for-object-detection. [Accessed: 19- Mar- 2018].

[8]"Unsupervised Feature Learning and Deep Learning Tutorial", *Ufldl.stanford.edu*, 2016. [Online]. Available: http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/. [Accessed: 19- Mar- 2018].