Object Detection/Recognition System Using Deep Learning

Progress Report

In fulfillment of the requirements for the

NU 302 R&D Project at

NIIT University

Submitted by

Choudapur Sanjay Reddy: U101115FCS246

Area

NIIT University

Neemrana

Rajasthan

CERTIFICATE

This is to certify that the present research work entitled "Object Detection Using Deep Learning" being submitted to NIIT University, Neemrana, Rajasthan, in the fulfillment of the requirements for the course at NIIT University, Neemrana, embodies authentic and faithful record of original research carried out by Choudapur Sanjay Reddy/s, Sudarshan/s, Saurab Yadav/s, Ujwal/s, Tushar/s, student/s of B Tech (CSE/ECE), at NIIT University, Neemrana,. She /He has worked under our supervision and that the matter embodied in this project work has not been submitted, in part or full, as a project report for any course of NIIT University, Neemrana or any other university.

Mentored by: Mr. Gaurav Sharma

LIST OF FIGURES

- *Figure: Dataset that we created from google images*
- *Figure: Sample images of the Fruit dataset*
- *Figure: Capsule Network Architecture*
- *Figure: Capsule Network Decoder*
- *Figure: Fruit Detection Experiment 1*
- *Figure: Fruit Detection Experiment 2*
- *Figure: Traffic sign detection after 4000 iterations*
- *Figure: Softmax values after 4000 iterations*
- *Figure: Validation dataset accuracies and loss values after 4000 iterations*
- *Figure: Test dataset accuracy values after 4000 iterations*
- *Figure: Softmax values after 5000 iterations*
- *Figure: Traffic sign detection after 6000 iterations*
- *Figure: Softmax values after 6000 iterations*
- *Figure: Validation dataset accuracies and loss values after 6000 iterations*
- *Figure: Test dataset accuracy values after 6000 iterations*
- *Figure: Traffic sign detection after 7000 iterations*
- *Figure: Softmax values after 7000 iterations*

LIST OF TABLES

- *Table 1: Capsule Network graph values*

CONTENTS

Title	Page no.
Certificate	2
List of Figures	3
List of Tables	3
Rational	**5**
Introduction	**6-7**
Literature Review	**8-24**
Objectives	**25**
Methodology	**26-46**
Results	**47-69**
Summary	**70**
Future Work	**71**
References	**72-73**

Rational

- To create a fully functional system which detects objects at higher level precision.
- To provide robustness and high accuracy to the current systems.
- To be able to detect objects at any angle or any orientation with high accuracies.
- To apply our technology in any current fields in which orientation and positioning of the system are given a high priority and in which clear details of instantiation parameters of the object are given a prominent role.

Introduction

Object detection/recognition is one of the most fundamental yet challenging problems in computer vision. A human being instantaneously can gain a huge deal of information via vision and computers can solve a plethora of problems if they were similarly capable. However, while simply capturing information and storing them via cameras is a trivial task, deriving meaningful information from them is not. An object detection algorithm taken an image (information) as input and returns all regions of that image representing the object of a given class in the form of bounding box/pixel level segmentation. It should be fast, robust and accurate in doing so; if the detection algorithm takes longer than the recognition algorithm, there's no point in applying this algorithm in real world scenarios such as self-driving cars, medical imaging, tracking etc.

The concept of deep learning with regards to object recognition means that neural networks are being used derive non-linear features to map input images to their respective classes. To achieve more accuracy, convolutional neural network architectures are preferred. However, the problem with such CNN architectures is that they don't take into account important spatial hierarchies between simple and complex objects. For example, given an image

of a cat's face, the neural network must take into account how features such as the presence of mouth, eyes, ears, whiskers etc, are oriented relative to each other. CNN architectures generally fail at solving this task. However, with the advent of capsule networks that is a fundamental revamp to the concept of a neural network, such orientation problems cease to exist in object recognition problems. The report presented aims to explore current state of the art object recognition algorithm: YOLO, seek to apply pixel level segmentation based on Mask R-CNN architecture from Facebook AI research for YOLO and perform experiments on CapsNet architecture from Google Brain.

Literature Review

1)

Title: Dynamic Routing Between Capsules

Author: Sara Sabour, Nicholas Frosst, Geoffrey E Hinton

Conference: 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA

Abstract method: A capsule is a group of neurons whose activity vector represents the instantiation parameters of a specific type of entity such as an object or object part. They use the length of the activity vector to represent the probability that the entity exists and its orientation to represent the instantiation paramters. Active capsules at one level make predictions, via transformation matrices, for the instantiation parameters of higher-level capsules. When multiple predictions agree, a higher level capsule becomes active. To achieve these results, we use an iterative routing-by-agreement mechanism: A lower-level capsule prefers to send its output to

higher level capsules whose activity vectors have a big scalar product with the prediction coming from the lower-level capsules.

2)

Title: MATRIX CAPSULES WITH EM ROUTING

Author: Sara Sabour, Nicholas Frosst, Geoffrey E Hinton

Conference: Published as a conference paper at ICLR 2018

Abstract Method: Each layer in a capsule network contains many capsules. They describe a version of capsules in which each capsule has a logistic unit to represent the presence of an entity and a 4x4 matrix which could learn to represent the relationship between that entity and the viewer(thepose). Each of these votes is weighted by an assignment coefficient. The transformation matrices are trained discriminatively by backpropagating through the unrolled iterations of EM between each pair of adjacent capsule layers. On the smallNORB benchmark, capsules reduce the number of test errors by 45% compared to the state-of-the-art. Capsules also show far more resistance to

white box adversarial attacks than our baseline convolutional neural network.

3)

Title: Rethinking the Inception Architecture for Computer Vision

Author: Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens

Abstract Method: They are exploring ways to scale up networks in ways that aim at utilizing the added computation as efficiently as possible by suitably factorized convolutions and aggressive regularization. They benchmark their methods on the ILSVRC 2012 classification challenge validation set demonstrate substantial gains over the state of the art: 21.2% top-1 and 5.6% top-5 error for single frame evaluation using a network with a computational cost of 5 billion multiply-adds per inference and with using less than 25 million parameters. With an ensemble of 4 models and multi-crop evaluation, we report 3.5% top-5 error and 17.3% top-1 error.

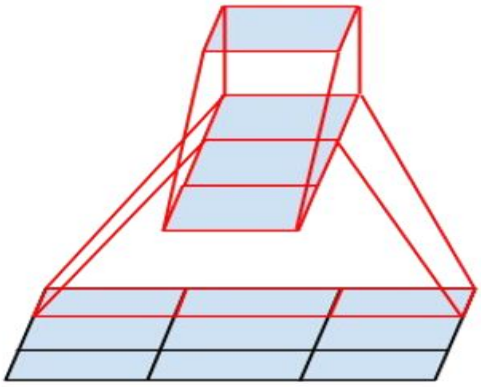


Image Source: http://www.voidcn.com/search/nvthrh

4)

Title: Rich feature hierarchies for accurate object detection and semantic segmentation

Authors: Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik

Abstract method: Published in 2014, this paper proposes a simple and scalable detection algorithm that improves mean average precision (mAP) by more than 30% relative to the previous best result on VOC 2012—achieving a mAP of 53.3%. The approach presented here combines two key insights: (1) applying high-capacity convolutional neural networks (CNNs) to bottom-up region proposals in order to localize and segment objects and (2) when labeled training data is scarce, supervised pre-training for an auxiliary task, followed by domain-specific fine-tuning, yields a significant performance boost. The paper also compares R-CNN to OverFeat which is a sliding-window detector based on a similar CNN architecture. The architecture presented by the authors dubbed as "R-CNN" outperforms OverFeat by a large margin on the 200-class ILSVRC2013 detection dataset.

5)

Title: Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition

Authors: Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun

Abstract method: Published in 2015, the paper aims to equip deep convolutional neural networks with a pooling strategy, “spatial pyramid pooling” to eliminate the requirement for such CNNs to work with fixed size input images. Thus the resultant architecture called SPP-net can generate a fixed length representation regardless of image size/scale. With these advantages, SPP-net should in general improve all CNN-based image classification methods. Using SPP-net, the authors computed the feature maps from the entire image only once, and then pool features in arbitrary regions (sub-images) to generate fixed-length representations for training the detectors. This method avoids repeatedly computing the convolutional features.

6)

Title: Fast R-CNN

Authors: Ross Girshick

Abstract method: Published in 2015, the paper proposes a Fast Region-based Convolutional Network method for object detection. It builds on previous works of R – CNN to efficiently classify object proposals using deep convolutional networks. Comparatively, it employs several innovations to improve training and testing speed while also increasing detection accuracy. Fast R-CNN trains the very deep VGG16 network 9× faster than R-CNN, is 213× faster at test-time, and achieves a higher mAP on PASCAL VOC 2012. Compared to SPPnet, Fast R-CNN trains VGG16 3× faster, tests 10× faster, and is more accurate.

7)

Title: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

Authors: Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun

Abstract method: Published in January 2016, the paper aims to solve the bottleneck faced by SPPnet and Fast R-CNN for regional proposal computation. In this work, a Region Proposal Network (RPN) that shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals is introduced. RPN is a fully convolutional network that simultaneously predicts object bounds and objectness scores at each position. The RPN is trained end-to-end to generate high-quality region proposals, which are used by Fast R-CNN for detection. Further, the authors merged RPN and Fast R-CNN into a single network by sharing their convolutional features—using the recently popular terminology of neural networks with “attention” mechanisms, the RPN component tells the unified network where to look

8)

Title: You Only Look Once: Unified, Real-Time Object Detection

Authors: Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi

Abstract method: Published in May 2016, this algorithm is one of the state of the art architectures for real time object detection. It is one of the algorithms explored in this project. YOLO is a new approach to object detection. While prior work presented earlier repurposes classifiers to perform detection, here object detection is framed as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is less likely to predict false positives on background. Finally, YOLO learns very general representations of objects.

9)

Title: YOLO9000: Better, Faster, Stronger

Authors: Joseph Redmon, Ali Farhadi

Abstract method: Published in December 2016, this paper proposes various improvements to the YOLO detection method, both new as well as drawn from prior work. It is state-of-the-art on standard detection datasets like COCO, Pascal VOC. The algorithm can be run at different sizes offering tradeoff between speed and accuracy. A method is finally proposed in this paper to jointly train on object detection and classification. In conclusion, two new algorithms are introduced in this paper: YOLOv2 and YOLO9000. YOLOv2 is a state-of-the-art algorithm and runs faster than other detection systems across a variety of detection datasets. As mentioned earlier, it can be run at a variety of image sizes offering tradeoff between speed and accuracy. The other algorithm called YOLO9000 is a real time framework that detects more than 9000 object categories by jointly optimizing detection and classification. This is done so by using WordTree to combine data from different from various sources and the joint optimization method to train simultaneously on COCO and ImageNet bridges the dataset size gap between detection and classification.

10)

Title: YOLOv3: An Incremental Improvement

Authors: Joseph Redmon, Ali Farhadi

Abstract method: Published in April, 2018, the authors introduced new design changes that make YOLOv2 better. Such changes include a new network for feature extraction which is a hybrid network of YOLOv2, Darknet and ResNet. At 320 × 320 YOLOv3 runs in 22 ms at 28.2 mAP, as accurate as SSD but three times faster. Looking at the old .5 IOU mAP detection metric YOLOv3 is quite good. It achieves 57.9 AP50 in 51 ms on a Titan X, compared to 57.5 AP50 in 198 ms by RetinaNet, similar performance but 3.8× faster

10)

Title: Deep Residual Learning for Image Recognition

Authors: Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun

Abstract method: The authors in this paper introduce a residual learning framework to ease the training of networks that are substantially deeper than those used previously. The layers learn residual functions with reference to the input layers instead of learning unreferenced functions. Empirical evidence has been presented to show that such residual layers are easier to optimize and can gain accuracy from considerably increased depth. Since depth of representation is of central importance for many visual recognition tasks, it is due to these deep representations that the authors obtained a 28% relative improvement on the COCO object detection dataset. As a future work, the authors will study combining the proposed architecture with strong regularizations which may improve results for 10^3 layered networks.

11)

Title: Object Detection Combining Recognition and Segmentation

Authors: Liming Wang, Jianbo Shi, Gang Song, and I-fan Shen

Abstract method: The authors in this paper develop an object detection method combining top-down recognition with bottom up image segmentation. There are two main steps in this method: a hypothesis generation step and a verification step. In the top-down hypothesis generation step, they design an improved Shape Context feature, which is more robust to object deformation and background clutter. The improved Shape Context is used to generate a set of hypotheses of object locations and figureground masks, which have high recall and low precision rate. In the verification step, they first compute a set of feasible segmentations that are consistent with top-down object hypotheses, then propose a False Positive Pruning(FPP) procedure to prune out false positives. Experiments show that this simple framework is capable of achieving both high recall and high precision with only a few positive training examples and that this method can be generalized to many object classes.

12)

Title: BlitzNet: A Real-Time Deep Network for Scene Understanding

Authors: Nikita Dvornik, Konstantin Shmelkov, Julien Mairal, Cordelia Schmid Inria

Abstract method: Published in 2017, the authors propose a deep architecture called BlitzNet that jointly performs object detection and semantic segmentation in one forward pass, allowing real-time computations. The computational gain of having a single network to perform several tasks, we show that object detection and semantic segmentation benefit from each other in terms of accuracy. . By using a single fully convolutional network to solve both problems at the same time, learning is facilitated by weight sharing between the two tasks, and inference is performed in real time. Moreover, we show that our pipeline is competitive in terms of accuracy, and that the two tasks benefit from each other.

13)

Title: A Survey of Semantic Segmentation

Authors: Martin Thoma

Abstract method: The objective of this paper is to survey different techniques used for pixel-level semantic segmentation. Metrics and datasets for the evaluation of segmentation algorithms and traditional approaches for segmentation such as unsupervised methods, Decision Forests and SVMs are described and pointers to the relevant papers are given. Recently published approaches with convolutional neural networks are mentioned and typical problematic situations for segmentation algorithms are examined. A taxonomy of segmentation algorithms is given.

14)

Title: MaskLab: Instance Segmentation by Refining Object Detection with Semantic and Direction Features

Authors: Liang-Chieh Chen, Alexander Hermans2, George Papandreou, Florian Schroff, Peng Wang3, Hartwig Adam

Abstract method: Published in December 2017, the authors tackle the problem of instance segmentation, the task of simultaneously solving the object detection and semantic segmentation. They present a model called Masklab, which produces three outputs: box detection, semantic segmentation and direction production. Building on top of the Faster-RCNN object detector, the predicted boxes provide accurate localization of object instances. Within each region of interest, MaskLab performs foreground/background segmentation by combining semantic and direction prediction. Semantic segmentation assists the model in distinguishing between objects of different semantic classes including background, while the direction prediction, estimating each pixel's direction towards its corresponding center, allows separating instances of the same semantic class.

14)

Title: Squeezenet: Alexnet-Level Accuracy With 50x Fewer Parameters And < 0.5 MB Model Size

Authors: Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, Kurt Keutzer

Abstract method: Published in November 2016, the authors propose a small CNN architecture called SqueezeNet. SqueezeNet achieves AlexNet-level accuracy on ImageNet with 50x fewer parameters. Additionally, with model compression techniques and they are able to compress SqueezeNet to less than 0.5MB (510× smaller than AlexNet). This architecture provides advantages of the smaller CNN architectures such as: (1) Smaller CNNs require less communication across servers during distributed training. (2) Smaller CNNs require less bandwidth to export a new model from the cloud to an autonomous car. (3) Smaller CNNs are more feasible to deploy on FPGAs and other hardware with limited memory.

15)

Title: ImageNet Classification with Deep Convolutional Neural Networks

Authors: Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton

Abstract method: The authors trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, they used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers they employed a recently-developed regularization method called “dropout” that proved to be very effective.

Objectives

- Till now all the applications in these fields used traditional convolutional layers and couple of pooling and non-linearity layers like Relu, but they come with huge drawback which is responsible for the fact that they are not implemented on to critical situations.
- Our aim is to overcome these drawbacks to the extent we can.
- In the process to achieve those goals we have to test and implement various cutting edge technologies in the current field.

Methodology

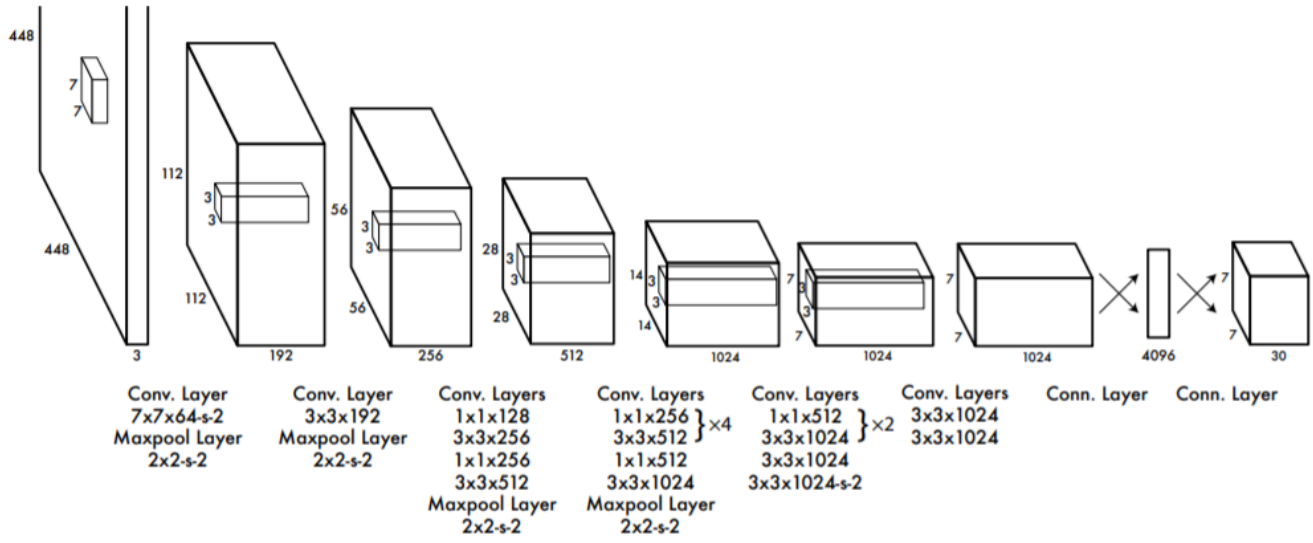
YOLO v1:

Having studied the concepts required for understanding deep learning as a whole, the initial stage is to implement a state of the art algorithm for object detection using bounding box such as the YOLO algorithm and analyze via results, what drawbacks YOLO contains and subsequently investigate YOLO v2 and YOLO v3.

Experiment 1:

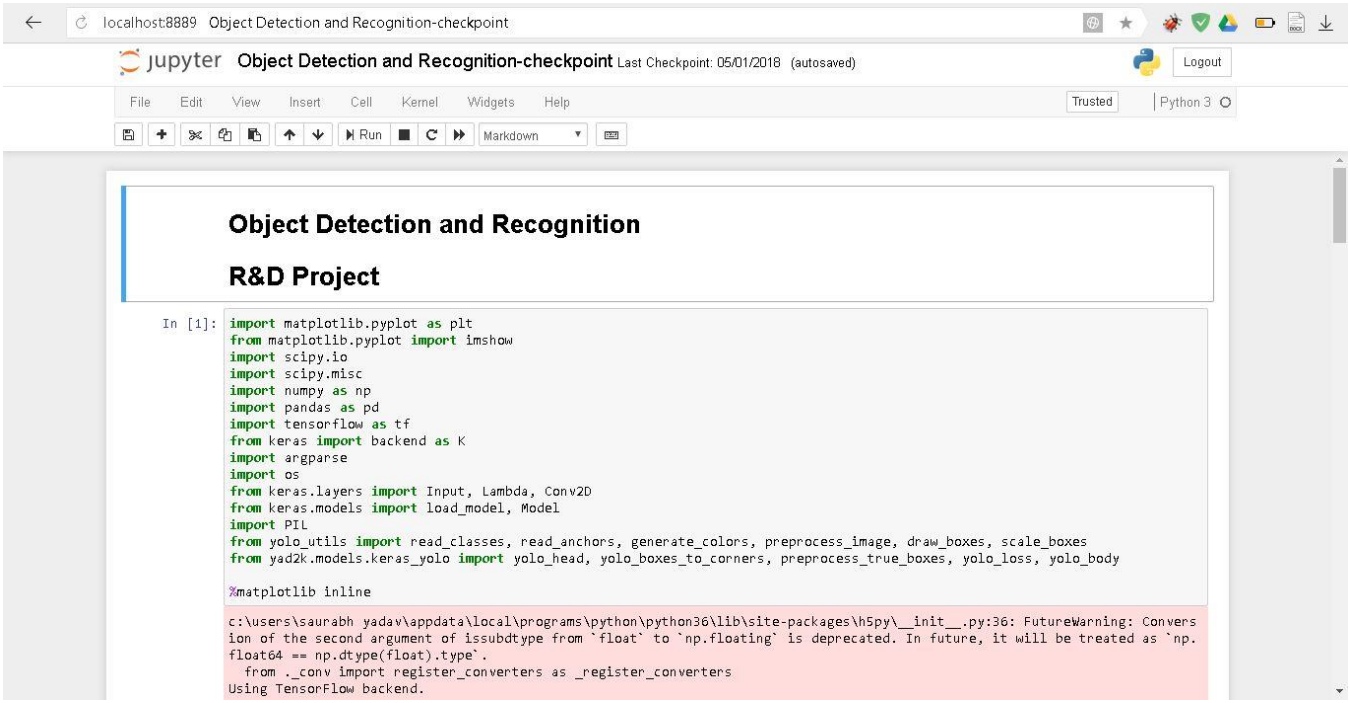
Dataset:
Primarily, to cut down on experimentation time and in order to get the results directly, the ImageNet 2012 dataset is used for training our implementation of the YOLO v1 algorithm, and for testing purposes, the PASCAL VOC 2007 test data set is used in combination with certain portion of the 2012 dataset.

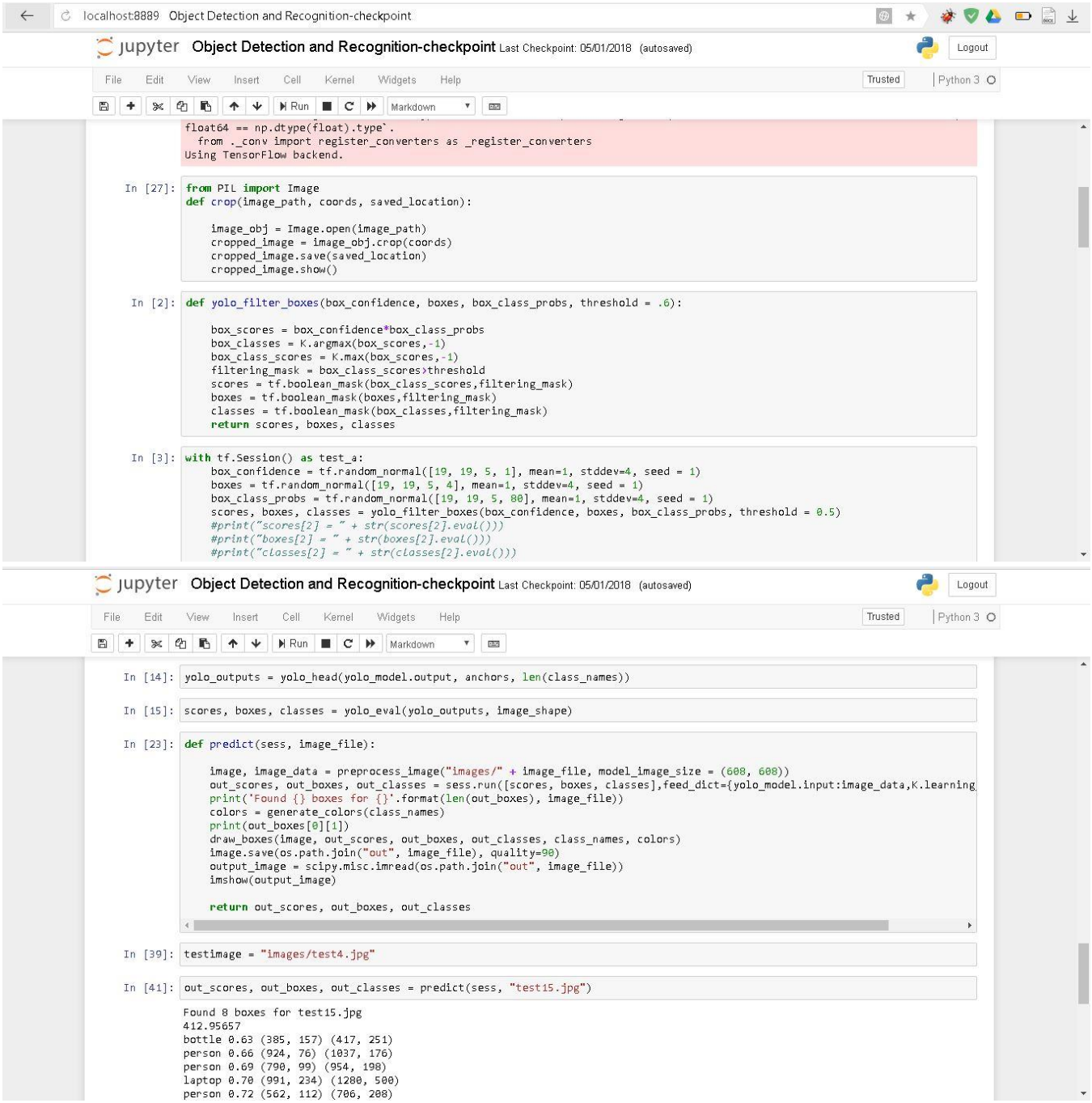
Architecture of YOLO:

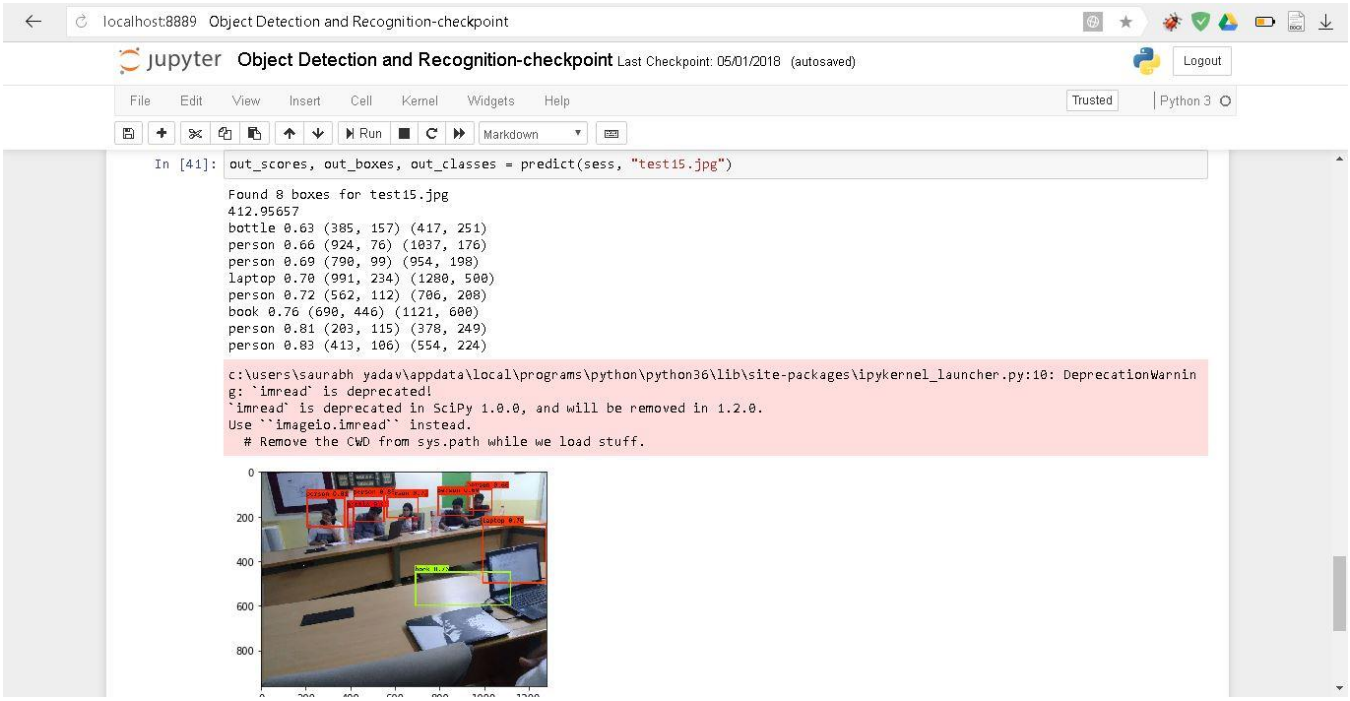


The detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1 × 1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224 × 224 input image) and then double the resolution for detection. This network according to the paper, is inspired by the GoogLeNet model for image classification.

The network has 24 convolutional layers followed by 2 fully connected layers. Instead of the inception modules used by GoogLeNet, we simply use 1 × 1 reduction layers followed by 3 × 3 convolutional layers. The final output of our network is the 7 × 7 × 30 tensor of predictions


localhost:8889 Object Detection and Recognition-checkpoint
jupyter Object Detection and Recognition-checkpoint Last Checkpoint: 05/01/2018 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown
Object Detection and Recognition
R&D Project
In [1]:
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
import scipy.io
import scipy.misc
import numpy as np
import pandas as pd
import tensorflow as tf
from keras import backend as K
import argparse
import os
from keras.layers import Input, Lambda, Conv2D
from keras.models import load_model, Model
import PIL
from yolo_utils import read_classes, read_anchors, generate_colors, preprocess_image, draw_boxes, scale_boxes
from yad2k.models.keras_yolo import yolo_head, yolo_boxes_to_corners, preprocess_true_boxes, yolo_loss, yolo_body
%matplotlib inline
c:\users\saurabh yadav\appdata\local\programs\python\python36\lib\site-packages\h5py__init__.py:36: FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
from ._conv import register_converters as _register_converters
Using TensorFlow backend.


localhost:8889 Object Detection and Recognition-checkpoint
jupyter Object Detection and Recognition-checkpoint Last Checkpoint: 05/01/2018 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown
float64 == np.dtype(float).type`.
from ._conv import register_converters as _register_converters
Using TensorFlow backend.
In [27]:
from PIL import Image
def crop(image_path, coords, saved_location):
image_obj = Image.open(image_path)
cropped_image = image_obj.crop(coords)
cropped_image.save(saved_location)
cropped_image.show()
In [2]:
def yolo_filter_boxes(box_confidence, boxes, box_class_probs, threshold = .6):
box_scores = box_confidence*box_class_probs
box_classes = K.argmax(box_scores,-1)
box_class_scores = K.max(box_scores,-1)
filtering_mask = box_class_scores>threshold
scores = tf.boolean_mask(box_class_scores,filtering_mask)
boxes = tf.boolean_mask(boxes,filtering_mask)
classes = tf.boolean_mask(box_classes,filtering_mask)
return scores, boxes, classes
In [3]:
with tf.Session() as test_a:
box_confidence = tf.random_normal([19, 19, 5, 1], mean=1, stddev=4, seed = 1)
boxes = tf.random_normal([19, 19, 5, 4], mean=1, stddev=4, seed = 1)
box_class_probs = tf.random_normal([19, 19, 5, 80], mean=1, stddev=4, seed = 1)
scores, boxes, classes = yolo_filter_boxes(box_confidence, boxes, box_class_probs, threshold = 0.5)
#print("scores[2] = " + str(scores[2].eval()))
#print("boxes[2] = " + str(boxes[2].eval()))
#print("classes[2] = " + str(classes[2].eval()))
jupyter Object Detection and Recognition-checkpoint Last Checkpoint: 05/01/2018 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown
In [14]:
yolo_outputs = yolo_head(yolo_model.output, anchors, len(class_names))
In [15]:
scores, boxes, classes = yolo_eval(yolo_outputs, image_shape)
In [23]:
def predict(sess, image_file):
image, image_data = preprocess_image("images/" + image_file, model_image_size = (608, 608))
out_scores, out_boxes, out_classes = sess.run([scores, boxes, classes],feed_dict={yolo_model.input:image_data,K.learning
print('Found {} boxes for {}'.format(len(out_boxes), image_file))
colors = generate_colors(class_names)
print(out_boxes[0][1])
draw_boxes(image, out_scores, out_boxes, out_classes, class_names, colors)
image.save(os.path.join("out", image_file), quality=90)
output_image = scipy.misc.imread(os.path.join("out", image_file))
imshow(output_image)
return out_scores, out_boxes, out_classes
In [39]:
testimage = "images/test4.jpg"
In [41]:
out_scores, out_boxes, out_classes = predict(sess, "test15.jpg")
Found 8 boxes for test15.jpg
412.95657
bottle 0.63 (385, 157) (417, 251)
person 0.66 (924, 76) (1037, 176)
person 0.69 (790, 99) (954, 198)
laptop 0.70 (991, 234) (1280, 500)
person 0.72 (562, 112) (706, 208)



YOLO9000:

The objective of the next experiment will be to observe how YOLO v1 base detection system has been improved upon to produce YOLOv2.
Based on previous experiment it has been observed that YOLOv1 makes significant localization errors. This has been made evident in the results shown for the previous experiment.

Experiment 2:

Dataset:
The same dataset that has been used to generate results for experiment 1 has been used here.

YOLOv3:

The objective of the next experiment will be to observe how YOLOv3 outperforms the previous versions of YOLO in the previous experiments.
Based on previous experiment, it has been observed that given complex domains like Open Images Dataset, when there are images that contain overlapping labels (i.e, Person and Woman), the softmax function used for such multilabel class prediction does not lead to good performance. So, in this experiment, independent logistic classifiers are used. Also, during the training stage, binary cross-entropy loss is used for class prediction.

Experiment 3:

Dataset:
The same dataset that has been used to generate results for experiment 2 has been used here.

Architecture of YOLOv3:

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

This network uses successive 3 × 3 and 1 × 1 convolutional layers but now has some shortcut connections as well and is significantly larger.

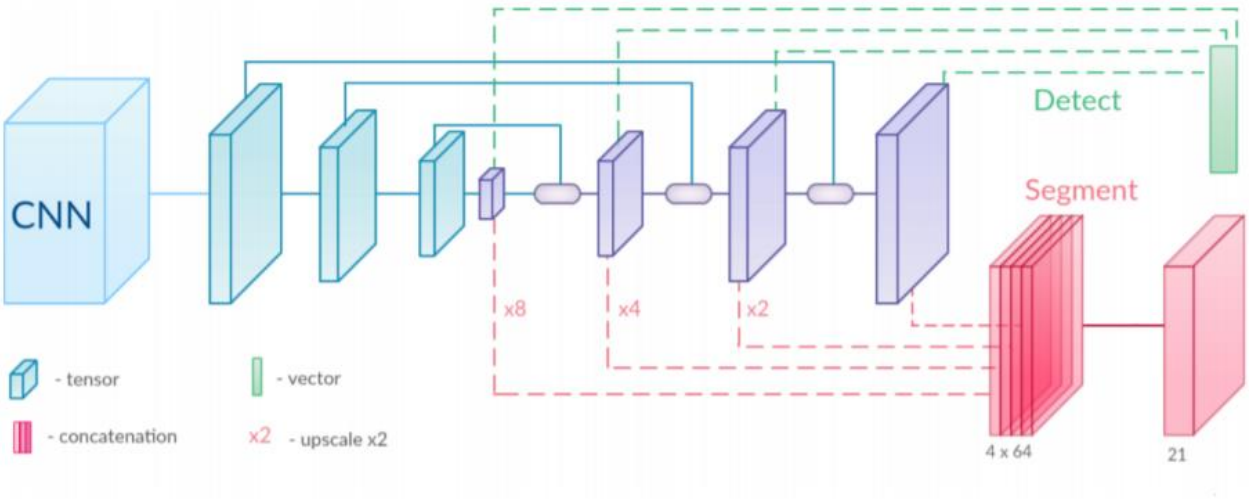
Blitznet:

Given the results of the previous experiments, and based on the objectives, the focus of this experiment is to try implementing a deep architecture that jointly performs object detection and semantic segmentation in one forward pass, allowing real-time computations. This idea of joint semantic segmentation and object detection was investigated first for shallow approaches. Learning both tasks simultaneously could be better than learning them independently.

Experiment 4:

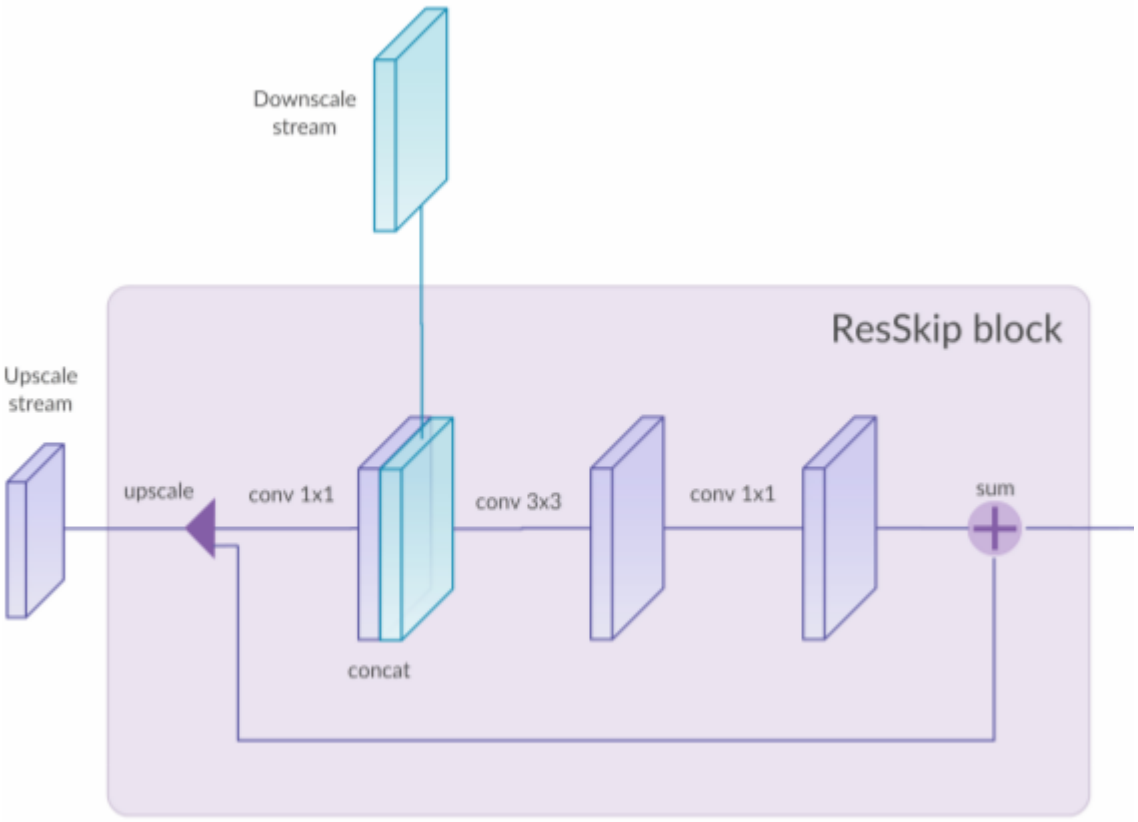
Dataset:
We used COCO dataset which consists of 91 different object classes.

Architecture of BlitzNet:



The BlitzNet architecture, which performs object detection and segmentation with one fully convolutional network. On the left, CNN

denotes a feature extractor, here ResNet-50; it is followed by the downscale-stream (in blue) and the last part of the net is the upscale-stream (in purple), which consists of a sequence of deconvolution layers interleaved with ResSkip blocks (see figure below). The localization and classification of bounding boxes (top) and pixelwise segmentation (bottom) are performed in a multiscale fashion by single convolutional layers operating on the output of deconvolution layers.



ResSkip block integrating feature maps from the upscale and downscale streams, with skip connection.

[Paste code snapshot of BlitzNet]

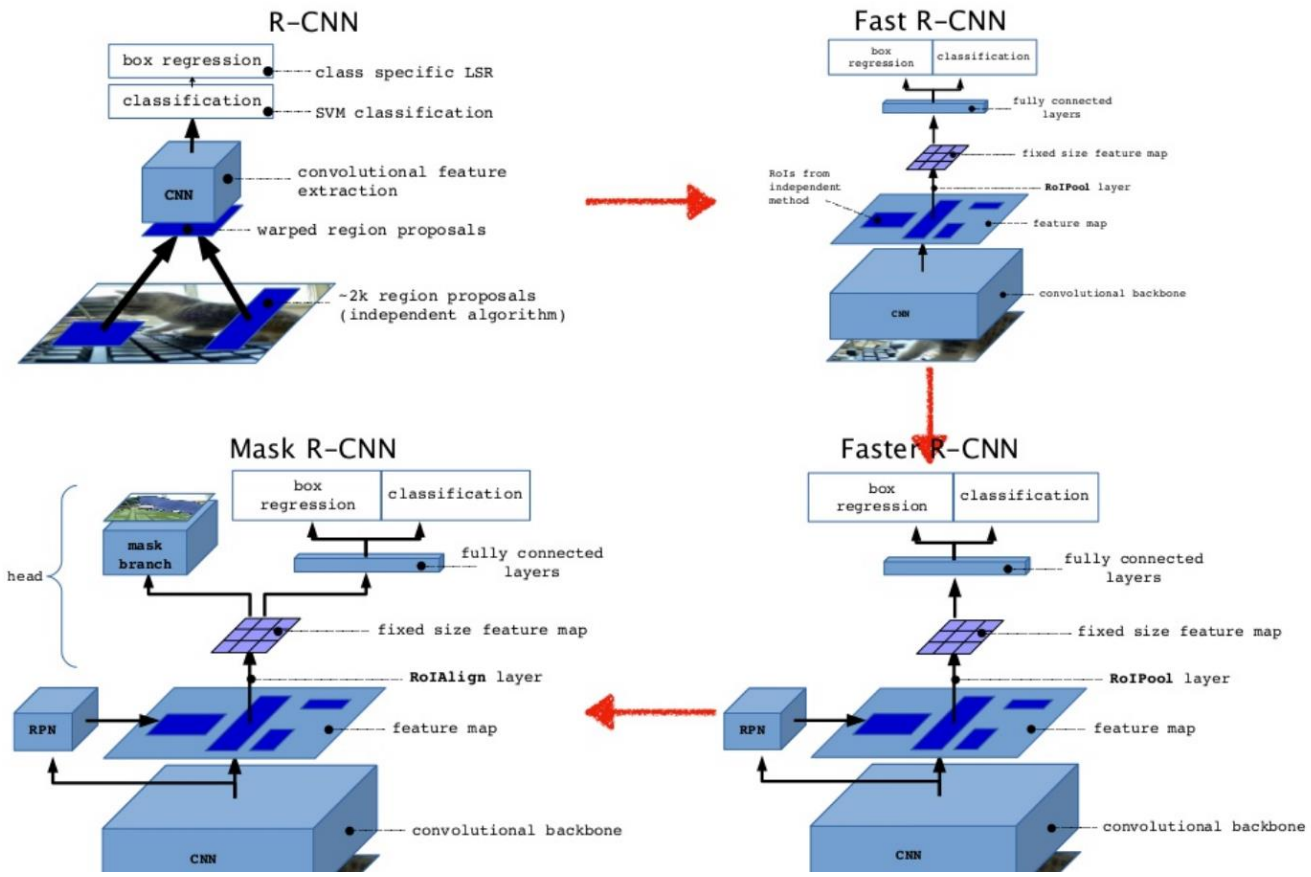
Experiment 5:

Having performed previous experiments, the objective of this experiment will be to tackle the problem of object detection/recognition and instance segmentation. This is based on Mask R-CNN work implemented by the Facebook AI Research team.

Dataset:
Just like the previous experiments, we've used official datasets. For this experiment we use COCO dataset.

Architecture of the Mask R-CNN model:

The diagram below presents a diagrammatic overview of how different CNN based architectures led to the development of Mask R-CNN architecture.

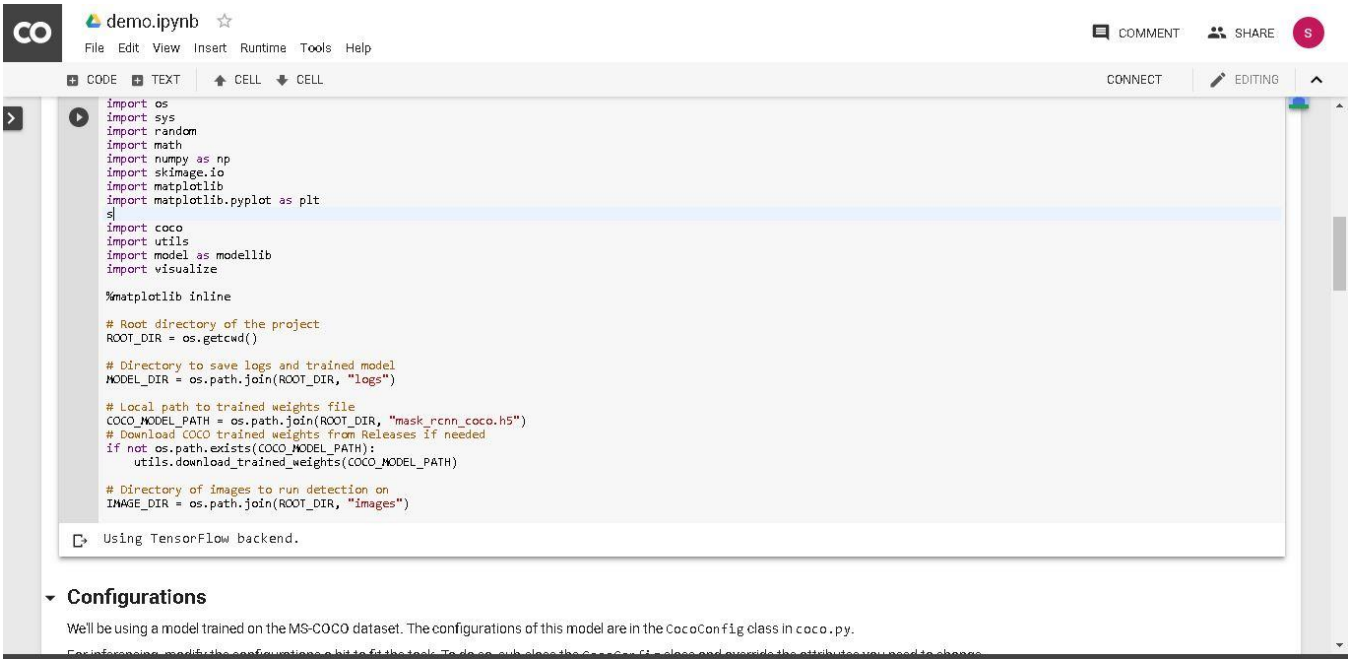


Differences between this experiment and the official paper:

1) **Image Resizing**: To support training multiple images per batch we resize all images to the same size. For example, 1024x1024px on MS COCO. We preserve the aspect ratio, so if an image is not square we pad it with zeros. In the paper the resizing is done such that the smallest side is 800px and the largest is trimmed at 1000px.

2) **Bounding Boxes:** Some datasets provide bounding boxes and some provide masks only. To support training on multiple datasets we opted to ignore the bounding boxes that come with the dataset and generate them on the fly instead. We pick the smallest box that encapsulates all the pixels of the mask as the bounding box. This simplifies the implementation and also makes it easy to apply image augmentations that would otherwise be harder to apply to bounding boxes, such as image rotation.

3) **Learning Rate**: The paper uses a learning rate of 0.02, but we found that to be too high, and often causes the weights to explode, especially when using a small batch size. It might be related to differences between how Caffe and TensorFlow compute gradients (sum vs mean across batches and GPUs).



demo.ipynb

File Edit View Insert Runtime Tools Help

COMMENT SHARE S

CODE TEXT CELL CELL CONNECT EDITING

```
class InferenceConfig(coco.CocoConfig):
    # Set batch size to 1 since we'll be running inference on
    # one image at a time. Batch size = GPU_COUNT * IMAGES_PER_GPU
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1

config = InferenceConfig()
#config.display()
```

Create Model and Load Trained Weights

```
# Create model object in inference mode.
model = modellib.MaskRCNN(mode="inference", model_dir=MODEL_DIR, config=config)

# Load weights trained on MS-COCO
model.load_weights(COCO_MODEL_PATH, by_name=True)
```

```
Tensor("ROI/strided_slice_6:0", shape=(6000, 4), dtype=float32)
Tensor("mrcnn_detection/strided_slice_4:0", shape=(?, ?), dtype=float32)
```

```
# COCO Class names
# Index of the class in the list is its ID. For example, to get ID of
# the teddy bear class, use: class_names.index('teddy bear')
class_names = ['BG', 'person', 'bicycle', 'car', 'motorcycle', 'airplane',
               'bus', 'train', 'truck', 'boat', 'traffic light',
               'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird',
               'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear',
               'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie',
               'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball',
               'kite', 'baseball bat', 'baseball glove', 'skateboard',
               'surfboard', 'tennis racket', 'bottle', 'wine glass', 'cup',
               'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',
               'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza',
               'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed',
```

demo.ipynb

File Edit View Insert Runtime Tools Help

COMMENT SHARE S

CODE TEXT CELL CELL CONNECT EDITING

Visualising the Results

```
visualize.display_instances(image, r['rois'], r['masks'], r['class_ids'],
                            class_names, r['scores'])
```

Experiment 6:

Dataset:

Initially we used a dataset that we created by downloading it from google images and various other sources. The dataset contained 6 classes of fruits and every photo is used directly without a change in the orientation of the object. Each object class contained 250-400 images.

Problem with the data was that, we were not able to prepare a good data dataset which has the images of every orientation of the objects. Samples of dataset that we made are in the next page.

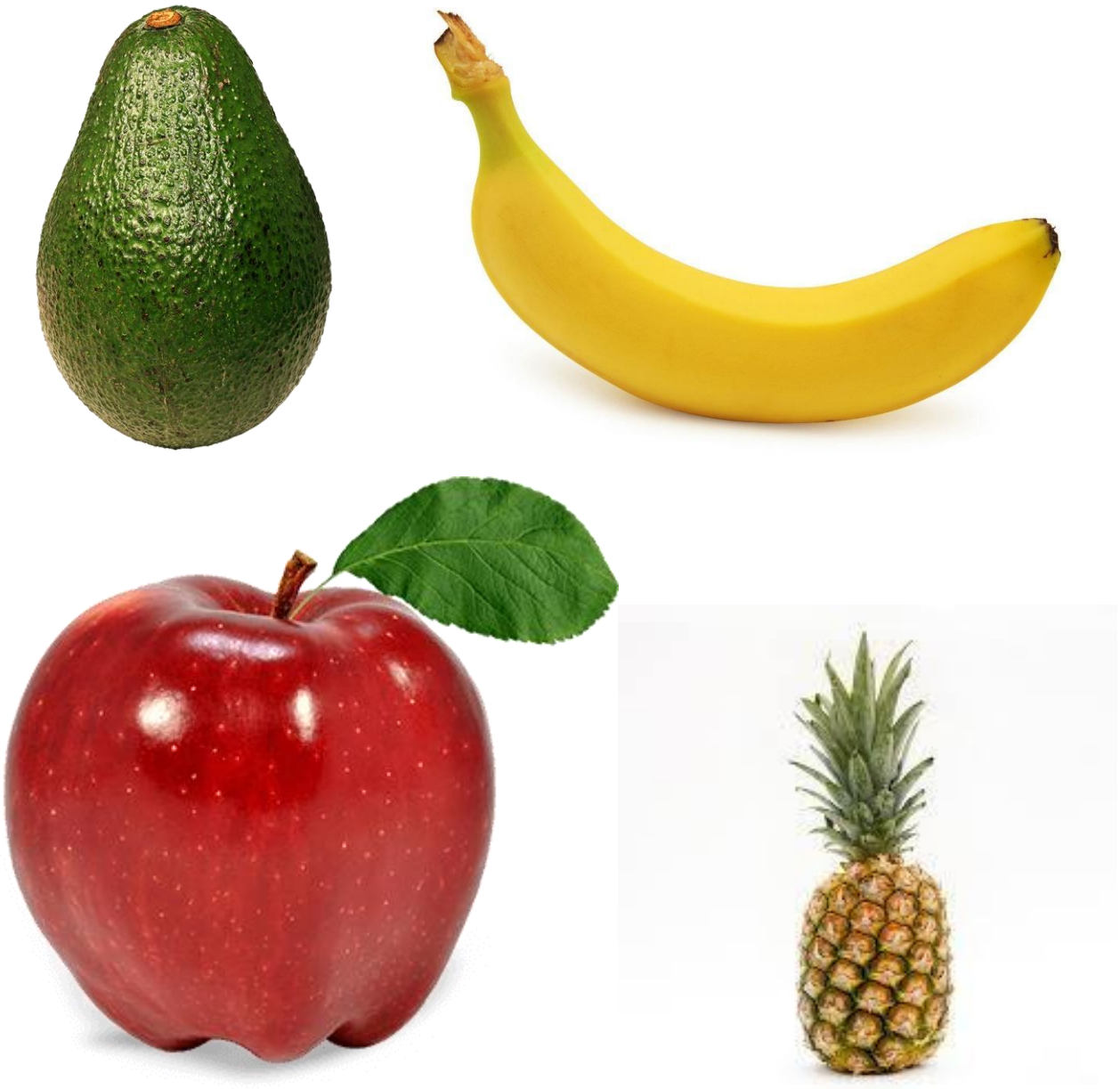


Figure: Dataset that we created from google images

Architecture of the inception model:

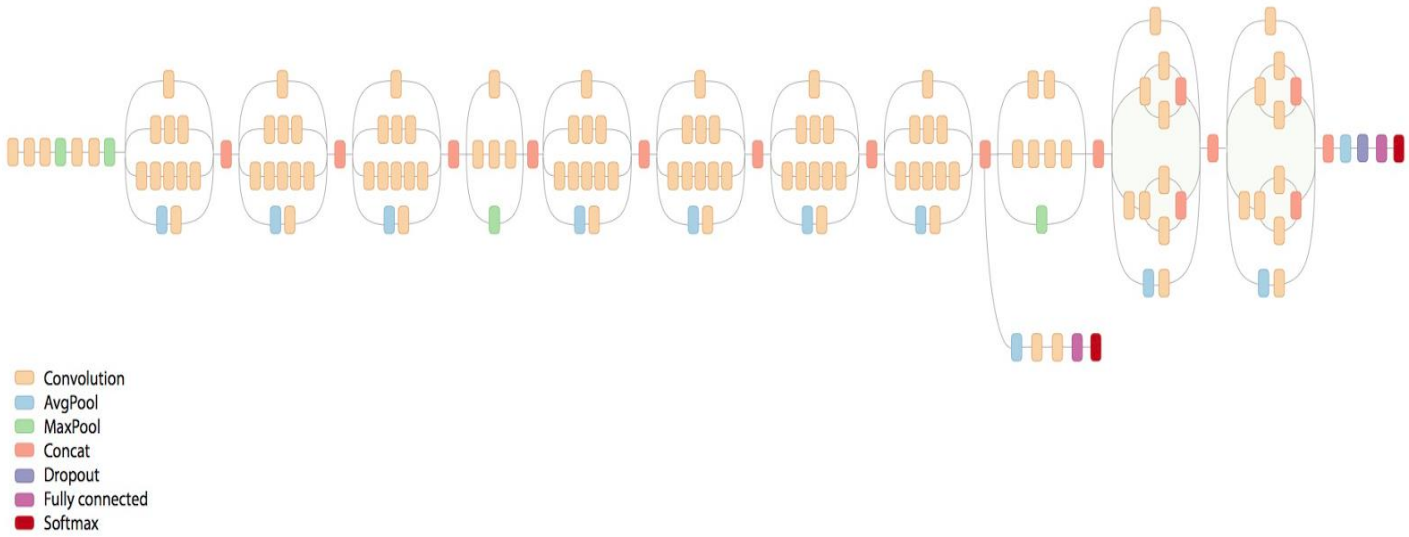


Image Source: https://hacktilldawn.com/2016/09/25/inception-modules-explained-and-implemented/

The architecture is built very cleverly by the google brain team. Let's say we are using 1×1, 3×3, and 5×5 convolutions along with a 3×3 max pooling. Pooling is added to the Inception module for no other reason other than, historically, good networks having pooling. The larger convolutions are more computationally expensive, so the paper (inception paper mentioned in the references) suggests first doing a 1×1 convolution reducing the dimensionality of its feature map, passing the resulting feature map through a relu, and then doing the larger convolution (in this case, 5×5 or 3×3). The 1×1 convolution is key because it will be used to reduce the dimensionality of its feature map we can use 1×1 convolutions to reduce the dimensionality of your input to large convolutions.

Experiment 7:

Dataset:

Instead of creating the dataset on our own, we were able to extract 17 classes of different types of fruits data from a famous fruits dataset which has all possible orientations of any particular fruit.

Figure: Sample images of the Fruit dataset

This quality of the dataset helped us to overcome the problem which was faced in experiment 6 i.e., unable to detect the images with different orientations and angles. The following are the different fruit classes we used.

Apple Golden 2	3/13/2018 1:49 PM	File folder
Apple Red 2	3/13/2018 1:49 PM	File folder
Apple Red Yellow	3/13/2018 1:49 PM	File folder
Avocado	3/13/2018 1:49 PM	File folder
Banana	3/13/2018 1:49 PM	File folder
Cactus fruit	3/13/2018 1:49 PM	File folder
Cherry	3/13/2018 1:49 PM	File folder
Dates	3/13/2018 1:50 PM	File folder
Guava	3/13/2018 1:50 PM	File folder
Kiwi	3/13/2018 1:50 PM	File folder
Lemon	3/13/2018 1:50 PM	File folder
Mango	3/13/2018 1:50 PM	File folder
Orange	3/13/2018 1:50 PM	File folder
Papaya	3/13/2018 1:50 PM	File folder
Pineapple	3/13/2018 1:49 PM	File folder
Raspberry	3/13/2018 1:49 PM	File folder
Strawberry	3/13/2018 1:49 PM	File folder

Figure: Different classes of dataset we used

Motivations for using Capsule Network:

Problem with the Convolutional Neural Network:

- The features (nose, left eye, right eye for the figure below) location information is lost at (Max)Pooling layer of CNN. Hence, there is no spatial relationship between features and orientations of the features.

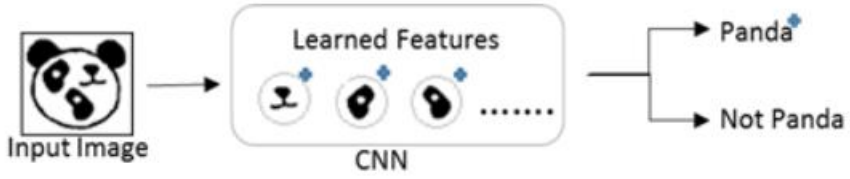


Image Source: http://www.medium.com

- For understanding let's consider the CNN is trained on panda images. If the above picture is the input, it wrongly predicts that the image is panda because of the lack of orientation information

- Following cases are also different scenarios in which CNN fails due to loss of information of orientation and angle.

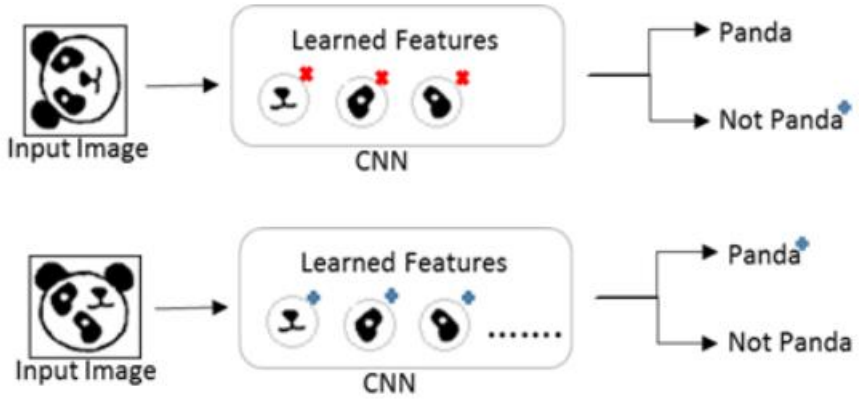
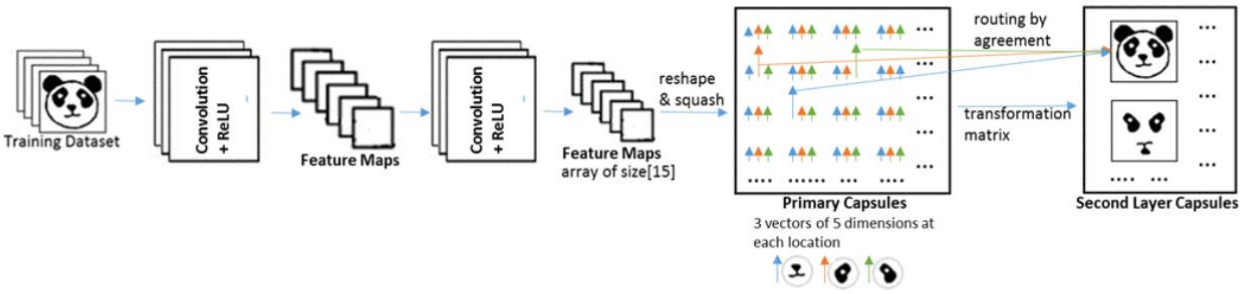


Image Source: http://www.medium.com

The above discussed problems can be solved by capsule network only. The next experiment is specifically aimed at implementing and improvising capsule network.

What is Capsule Network?

CapsNet (Capsule Network) is a neural network that performs inverse graphics. CapsNet is composed of capsules. A capsule comprises of a group of neurons in a layer which performs internal computations to predict the presence and the instantiation parameters (values for feature properties such as orientation, size, velocity, colour) of a particular feature at a given location.



Capsule Network

Image Source: http://www.medium.com

Experiment 8:

- Till now, people were only able to implement CapsNet with MNIST and MNIST-fashion datasets which are black and white small resolutions images and some colored datasets like CIFAR.
- High accuracy is reached on MNIST and for CIFAR 10, results are promising
- Instead of adding a layer after another, the idea is to add several layers inside another one.
- The heart of method is divided into two parts:
  The **PrimaryCapsules** and the **DigitCaps**
- Using an iterative routing process, each active capsule will choose a capsule in the layer above to be its parent in the tree
- Additional reconstruction loss is used to encourage the digit capsules to encode the instantiation parameters of the input digit
- Using a different dataset (Not MNIST) some details in the architecture are different.

- German traffic sign dataset (Colored images with slightly high resolutions)
- Aim is to increase the scope of capsule net in different real-life implementations which involves colored images.
- Tensorflow library is used.
- Instead of training the capsule net with raw images, it is trained with the pickle files of the same.
- The dataset consists of 43 different classes, so that many capsules are present in the final layer of the capsule net.
- In the field of self-driving car and various other applications, detection of traffic signals plays a prominent role.
- Till now all these applications used traditional convolutional layers and couple of pooling and non-linearity layers like Relu.
- They all work fine until they find a critical situation.
- When it comes to a person's life held at stake because of a self-driving car, every rarity should be taken into account

Model Architecture:

- Convolutional layer 1
- Convolutional layer 2
- Primary Capsule layer
- TrafficCapsule layer
- Decoder

Decoder reconstructs the image using convolutions and the nearest neighbor algorithm to scale up images.

```python
def _build_main_network(self, images, conv_2_dropout):
    """
        This method is used to create the two convolutions and the CapsNet on the top
        **input:
            *images: Image PLaceholder
            *conv_2_dropout: Dropout value placeholder
        **return: **
            *Caps1: Output of first Capsule layer
            *Caps2: Output of second Capsule layer
    """
    # First BLock:
    # Layer 1: Convolution.
    shape = (self.h.conv_1_size, self.h.conv_1_size, 3, self.h.conv_1_nb)
    conv1 = self._create_conv(self.tf_images, shape, relu=True, max_pooling=False, padding='VALID')
    # Layer 2: Convolution.
    shape = (self.h.conv_2_size, self.h.conv_2_size, self.h.conv_1_nb, self.h.conv_2_nb)
    conv2 = self._create_conv(conv1, shape, relu=True, max_pooling=False, padding='VALID')
    conv2 = tf.nn.dropout(conv1, keep_prob=conv_2_dropout)

    # Create the first capsules layer
    caps1 = conv_caps_layer(
        input_layer=conv1,
        capsules_size=self.h.caps_1_vec_len,
        nb_filters=self.h.caps_1_nb_filter,
        kernel=self.h.caps_1_size)
    # Create the second capsules layer used to predict the output
    caps2 = fully_connected_caps_layer(
        input_layer=caps1,
        capsules_size=self.h.caps_2_vec_len,
        nb_capsules=self.NB_LABELS,
        iterations=self.h.routing_steps)

    return caps1, caps2
```

Figure: Capsule Network Architecture

Decoder architecture:

```python
def _build_decoder(self, caps2, one_hot_labels, batch_size):
    """
        Build the decoder part from the last capsule layer
        **input:
            *Caps2:  Output of second Capsule layer
            *one_hot_labels
            *batch_size
    """
    labels = tf.reshape(one_hot_labels, (-1, self.NB_LABELS, 1))
    # squeeze(caps2):   [?, len_v_j,    capsules_nb]
    # labels:           [?, NB_LABELS,  1] with capsules_nb == NB_LABELS
    mask = tf.matmul(tf.squeeze(caps2), labels, transpose_a=True)
    # Select the good capsule vector
    capsule_vector = tf.reshape(mask, shape=(batch_size, self.h.caps_2_vec_len))
    # capsule_vector: [?, len_v_j]

    # Reconstruct image
    fc1 = tf.contrib.layers.fully_connected(capsule_vector, num_outputs=400)
    fc1 = tf.reshape(fc1, shape=(batch_size, 5, 5, 16))
    upsample1 = tf.image.resize_nearest_neighbor(fc1, (8, 8))
    conv1 = tf.layers.conv2d(upsample1, 4, (3,3), padding='same', activation=tf.nn.relu)

    upsample2 = tf.image.resize_nearest_neighbor(conv1, (16, 16))
    conv2 = tf.layers.conv2d(upsample2, 8, (3,3), padding='same', activation=tf.nn.relu)

    upsample3 = tf.image.resize_nearest_neighbor(conv2, (32, 32))
    conv6 = tf.layers.conv2d(upsample3, 16, (3,3), padding='same', activation=tf.nn.relu)

    # 3 channel for RGG
    logits = tf.layers.conv2d(conv6, 3, (3,3), padding='same', activation=None)
    decoded = tf.nn.sigmoid(logits, name='decoded')
    tf.summary.image('reconstruction_img', decoded)

    return decoded
```

Figure: Capsule Network Decoder

Softmax function integrated values:

In the results section, the results of the final output is shown on graph with the x axis varying from 1-42. Meanings of those values are stated in the below table.

ClassId	SignName
0	Speed limit (20km/h)
1	Speed limit (30km/h)
2	Speed limit (50km/h)
3	Speed limit (60km/h)
4	Speed limit (70km/h)
5	Speed limit (80km/h)
6	End of speed limit (80km/h)
7	Speed limit (100km/h)
8	Speed limit (120km/h)
9	No passing
10	No passing for vehicles over 3.5 metric tons
11	Right-of-way at the next intersection
12	Priority road
13	Yield
14	Stop
15	No vehicles
16	Vehicles over 3.5 metric tons prohibited
17	No entry
18	General caution
19	Dangerous curve to the left
20	Dangerous curve to the right
21	Double curve
22	Bumpy road
23	Slippery road
24	Road narrows on the right
25	Road work
26	Traffic signals
27	Pedestrians
28	Children crossing
29	Bicycles crossing
30	Beware of ice/snow
31	Wild animals crossing
32	End of all speed and passing limits
33	Turn right ahead
34	Turn left ahead
35	Ahead only
36	Go straight or right
37	Go straight or left
38	Keep right
39	Keep left
40	Roundabout mandatory
41	End of no passing
42	End of no passing by vehicles over 3.5 metric tons

Table 1: Capsule Network graph values

Results

Experiment 1:

Result:
The YOLOv1 model was able to successfully perform bounding box based object detection and it assigns class label to each of the bounding boxes along with a 0 to 1 “objectness” score.

Drawbacks:
This model struggles to detect small number of objects such as flocks of birds. If for example, an enlarged image of a flock of birds is presented, rather than putting a bounding box over the flock of birds, this model makes individual bird predictions.
Also, there are incorrect localizations for certain input images.

How to overcome?
We investigate the next version of YOLO named YOLO9000 rather than seek to modify this version of the algorithm.

[PASTE IMAGE OF INCORRECT LOCALIZATION]

Experiment 2:

Result:

The YOLO9000 model was able to successfully perform bounding box based object detection and it assigns class label to each of the bounding boxes along with a 0 to 1 “objectness” score. It also able to achieve mean average precision value (mAP) of 73.4% based on combination of VOC 2007 and ImageNet 2012 dataset.

Drawbacks:

This model still struggles to detect small number of objects such as flocks of birds. If for example, an enlarged image of a flock of birds is presented, rather than putting a bounding box over the flock of birds, this model makes individual bird predictions.
Also, there are incorrect localizations for certain input images.

How to overcome?

We investigate the next version of YOLO named YOLOv3 rather than seek to modify this version of YOLO.

Experiment 3:

Result:
In the past YOLO struggled with small objects. But in this experiment, we see a reversal in trends. The Average Precision (AP) score on such small objects is relatively high.

Drawbacks:
Unfortunately, it has comparatively worse performance on medium and large size objects. The paper on YOLOv3 doesn't provide any explanation on this.

Experiment 4:

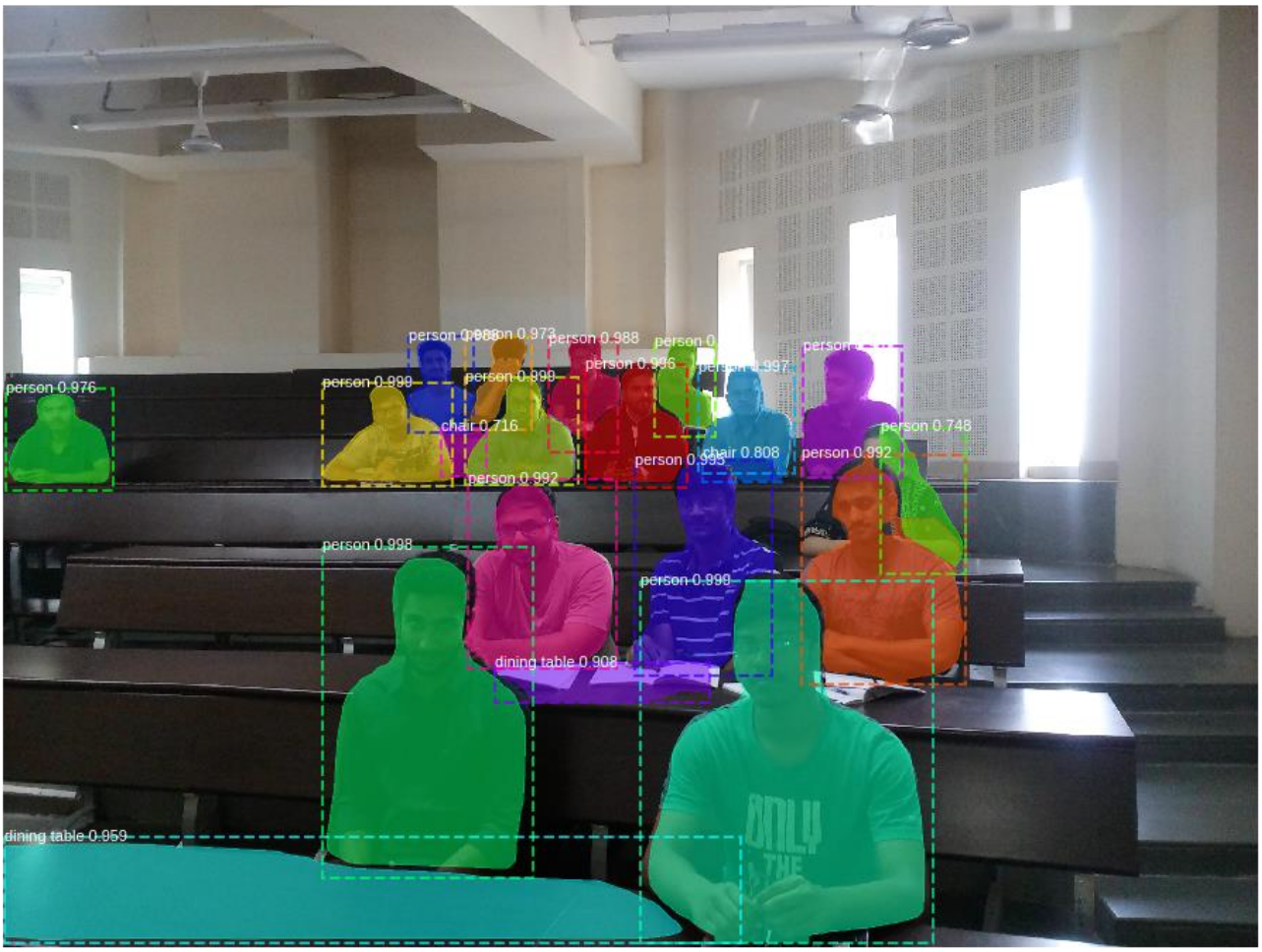
Result:
This implementation of BlitzNet takes an input image and generates two outputs of same dimensions as the input image. One output shows the bounding box based prediction. Second output is the segmentation maps pertaining to identified objects.
The merging of these two outputs gives a crude object detection/recognition with semantic segmentation output.

Drawbacks:
There are far many input images where the BlitzNet model fails to detect the object and thus generate the associated segmentation mask. Due to this reason, one cannot rely on such an architecture

for building an Android app using this model for real time object detection/recognition with semantic segmentation as output.

How to overcome?

The next experiment based on Mask R-CNN will overcome problems faced here.



[Paste BlitzNet merged output where a lot of objects fail to be detected by it]

Experiment 5:

Result:

This implementation of Mask R-CNN with our modifications to it (as explained in methodology) overcomes drawbacks faced in the previous experiments. To validate our approach to "bounding boxes" as explained in the methodology, we compared our computed bounding boxes to those provided by the COCO dataset. We found that ~2% of bounding boxes differed by 1px or more, ~0.05% differed by 5px or more, and only 0.01% differed by 10px or more.

Drawbacks:

1) Occlusion: Condition where certain objects aren't visible due to presence of other objects. Lighting conditions affects detection of certain objects. In Figure AB presented next page, a chair is not being detected due to presence of undetected bottle in front of it.
2) Rotation: In Figure AC, if an object in an image is changed with respect to orientation, there are times when the model fails to recognize the object due to the CNN nature of the model.

How to overcome?

This is the basis of experiment 7 and 8 wherein we try to solve the problem of object detection invariant of orientation and try to take into account important spatial hierarchies between simple and complex objects.

(Figure AB)

(Figure AC)

Inception Model:

Experiment 6:

Result:

The model was able to successfully detect many fruit images with a high level precision. The figure in the next page is the screenshot of the implementation of the inception architecture. The model is able to differentiate between those six different datasets without any ambiguity.

Drawbacks:

Once the model is given to detect the images with the objects in a change with orientation, it is giving an answer which is close to the image but not the exact image. So, this cannot be implemented in critical fields which use object detection.

How to overcome?

The problem can be overcome by training the image by a larger dataset which has all possible orientations of the images. This is the basis of the next experiment.

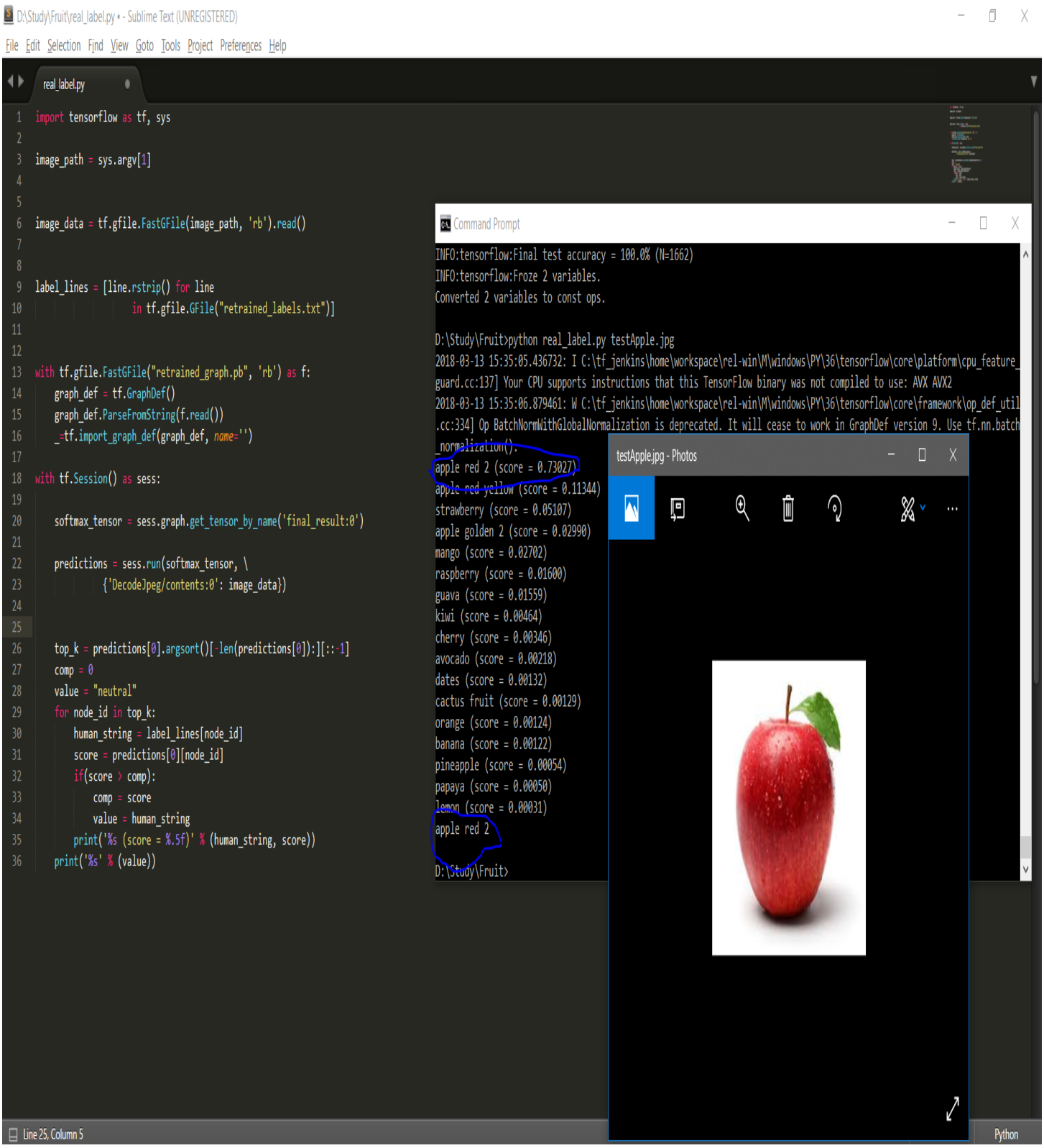


Figure: Fruit Detection Experiment 1

Experiment 7:

Result:

The model was able to successfully detect and distinguish between 17 different types of fruit classes. Even when the image is rotated into different orientation and different angle the model is able to detect with high accuracies without any ambiguity. The results are shown in the next page.

Drawbacks:

Though the model is able to detect the images with different orientations and angles, the time and effort it took to train and learn is not at all acceptable. All credits of the results is attributed to the datasets which had a numerous number of images in all possible angles rather than a particular orientation as was done in the previous experiment.

How to overcome?

The problem can be overcome by using the newly emerging concept of capsule neural networks which is a fundamentally different than a standard convolutional neural network.

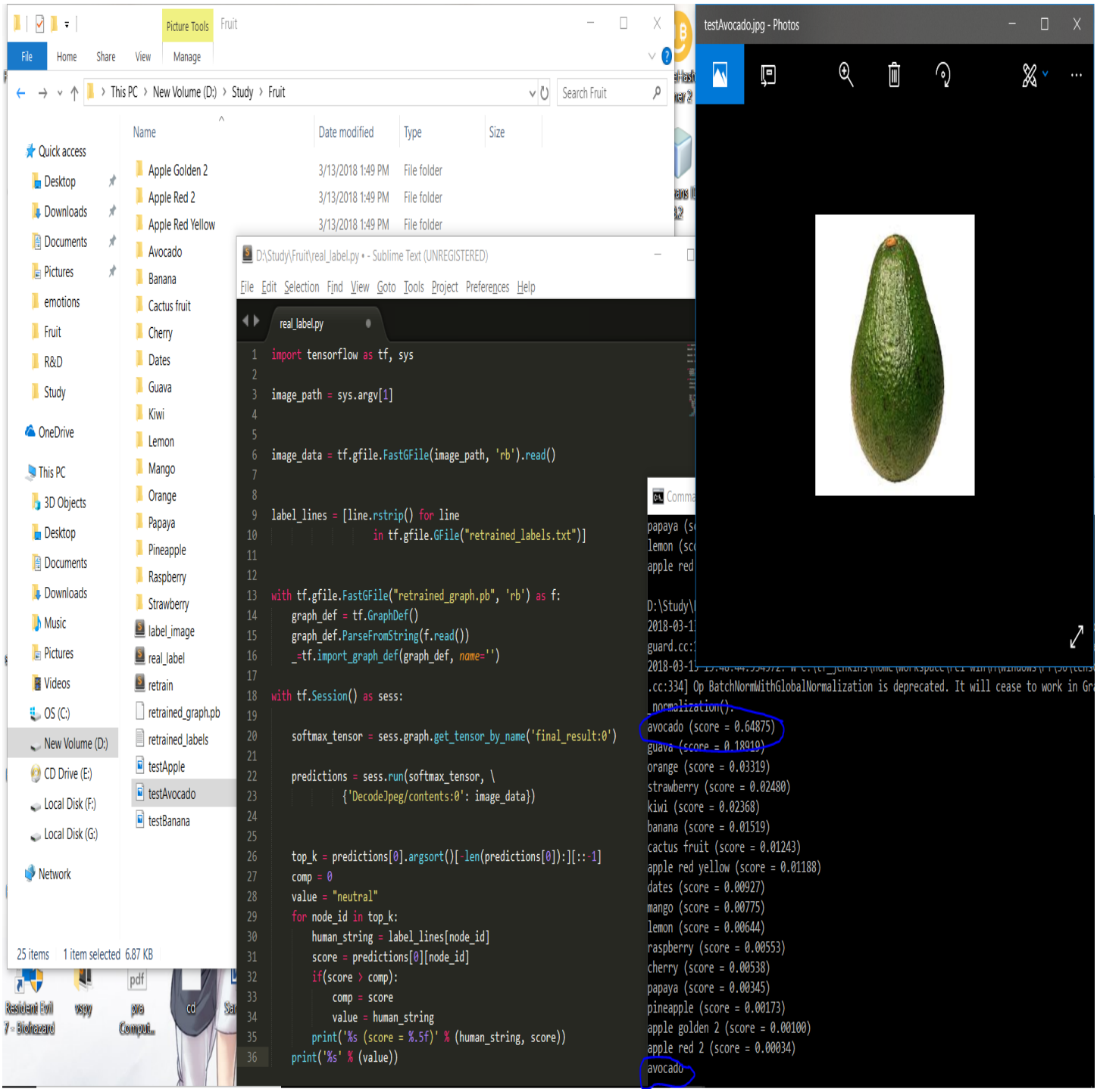


Figure: Fruit Detection Experiment 2

Capsule Networks:

Experiment 8:

Result:

The capsule network was able to clear all the drawbacks mentioned in the above experiments. It clearly gave an outstanding performance. It is able to detect and differentiate between any number of classes which it trained upon without any ambiguity. It clearly reconstructs the input image with the decoder even in the last stages of the neural net. This depicts that even there is a change in the orientation and angle of the object in the image, the capsule net can detect without any ambiguity. This implementation has outperformed all other implementations.

We trained it for 7000 iterations which took more than 150 hours on Russian traffic dataset. We exhibited the results at each level by testing the model with some random images of traffic signs.

NOTE: Each iteration has an inner loop which trains internally because of the dynamic routing algorithm used.

After 4000 iterations:

Results of the classification of test images:

The following images are reconstructed by the neural net in its final layer after 4000 iterations.

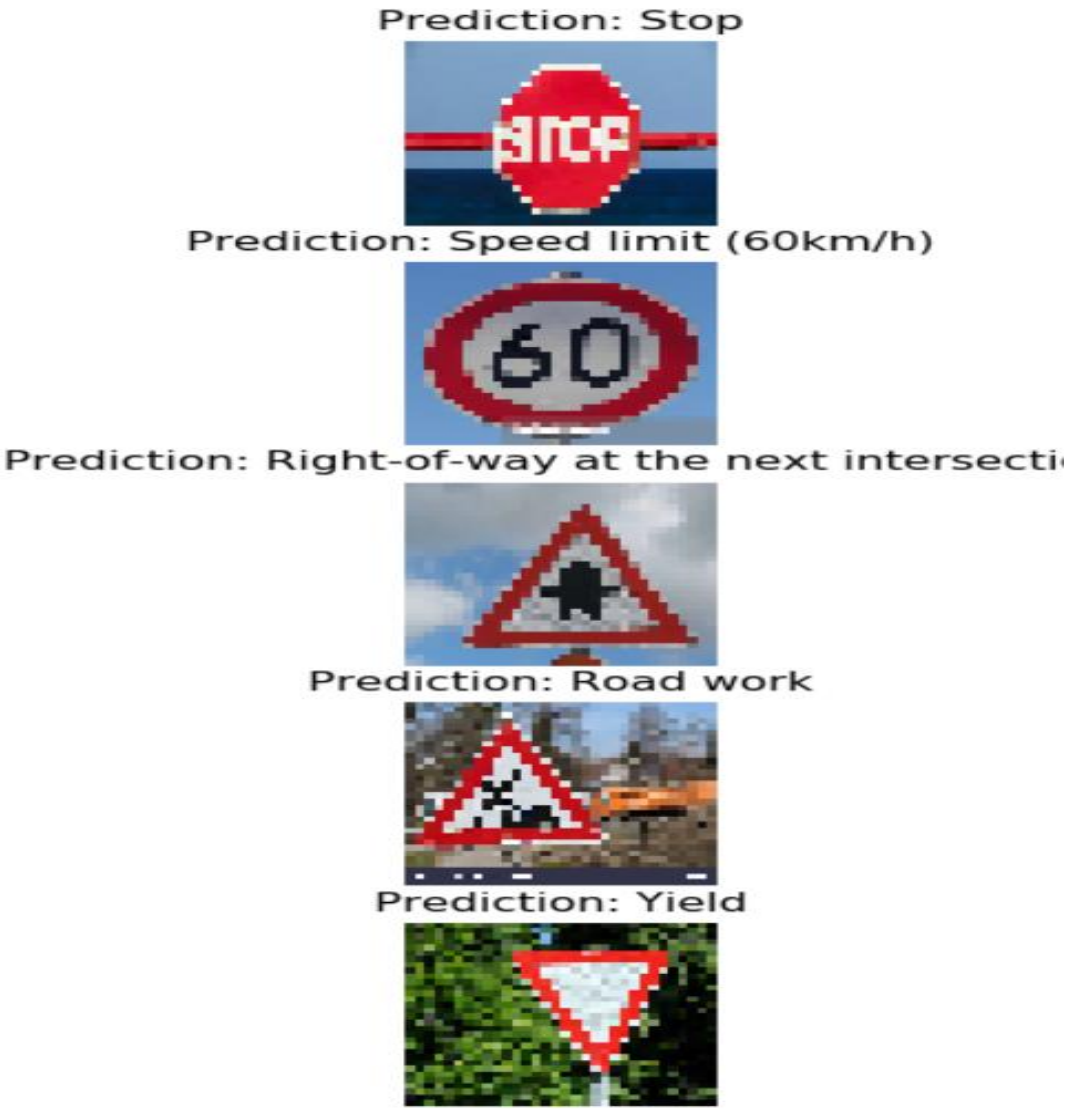


Figure: Traffic sign detection after 4000 iterations

When the values of the softmax function of the results achieved in the final layers are plotted on graph:

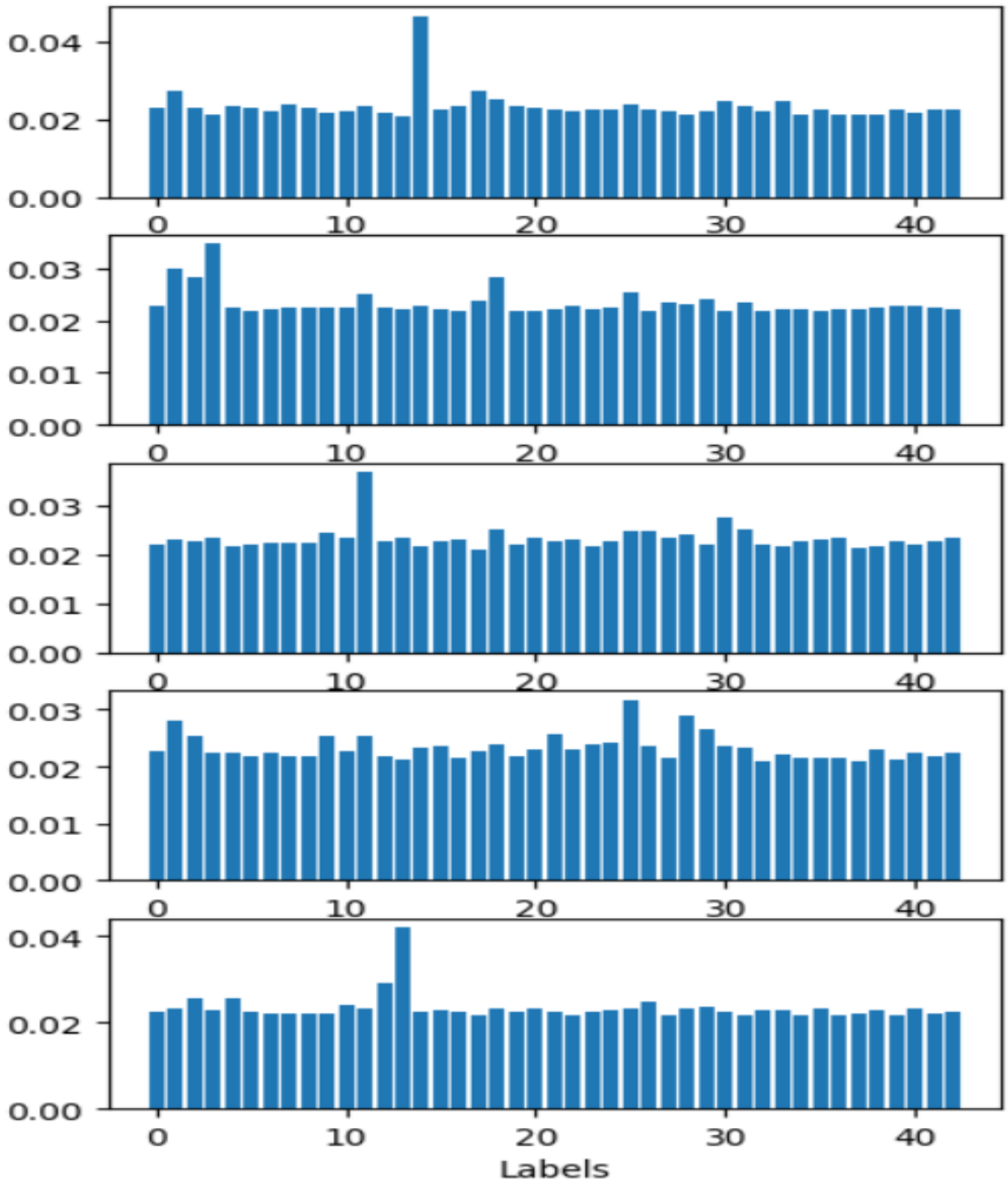


Figure: Softmax values after 4000 iterations

Validation accuracies and loss value:

```
2018-05-01 01:35:28.058077: W T:\src\github\tensorflow\tensorflow\core\framework\allocator.cc:101] Alloc
ation of 440320000 exceeds 10% of system memory.
[Train] Batch ID = 4000, loss = 0.395663, acc = 0.48
[Validation] Batch ID = 4000, loss = 0.145658, acc = 0.82
Evaluate full validation dataset ...
Current loss: 0.151662 Best loss: 0.199272
[TOTAL Validation] Batch ID = 4000, loss = 0.151662, acc = 0.897278911565
ModelBase::Saving model ...
ModelBase::Model successfully saved here: C:\Users\Sanju\capsnet_traffic_sign_classifier\outputs\checkpo
ints\c1s_9_c1n_256_c2s_6_c2n_64_c2d_0.7_c1vl_16_c1s_5_c1nf_16_c2vl_32_lr_0.0001_rs_1--TrafficSign--15250
```

Figure: Validation dataset accuracies and loss values after 4000 iterations

Test accuracy value:

```
C:\Users\Sanju\capsnet_traffic_sign_classifier>python test.py outputs\checkpoints\c1s_9_c1n_256_c2s_6_c2n_64_c2d_0.7_c1vl_16_c1s_5_c1nf_16_c2vl_32_lr_0.0001_rs_1--TrafficSign--1525037231.3690584 dataset\
ModelBase::Loading ckpt ...
2018-05-01 03:19:59.725299: I T:\src\github\tensorflow\tensorflow\core\platform\cpu_feature_guard.cc:140] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
INFO:tensorflow:Restoring parameters from outputs\checkpoints\c1s_9_c1n_256_c2s_6_c2n_64_c2d_0.7_c1vl_16_c1s_5_c1nf_16_c2vl_32_lr_0.0001_rs_1--TrafficSign--1525037231.3690584
Restoring parameters from outputs\checkpoints\c1s_9_c1n_256_c2s_6_c2n_64_c2d_0.7_c1vl_16_c1s_5_c1nf_16_c2vl_32_lr_0.0001_rs_1--TrafficSign--1525037231.3690584
ModelBase::Ckpt ready
Accuracy =  0.888756927949
Loss =  0.153424
Normalized confusion matrix
[[ 0.        0.88333333  0.       ...,  0.        0.        0.      ]
 [ 0.        0.94444444  0.02638889 ...,  0.        0.        0.      ]
 [ 0.        0.02933333  0.91066667 ...,  0.        0.        0.      ]
 ...,
 [ 0.        0.        0.       ...,  0.85555556  0.        0.      ]
 [ 0.        0.        0.       ...,  0.        0.58333333
   0.06666667]
 [ 0.        0.        0.       ...,  0.        0.03333333
   0.83333333]]
findfont: Matching :family=sans-serif:style=normal:variant=normal:weight=normal:stretch=normal:size=12.0 to DejaVu Sans ('C:\\Users\\Sanju\\AppData\\Local\\Programs\\Python\\Python36\\lib\\site-packages\\matplot
lib\\mpl-data\\fonts\\ttf\\DejaVuSans.ttf') with score of 0.050000
findfont: Matching :family=sans-serif:style=normal:variant=normal:weight=normal:stretch=normal:size=10.0 to DejaVu Sans ('C:\\Users\\Sanju\\AppData\\Local\\Programs\\Python\\Python36\\lib\\site-packages\\matplot
lib\\mpl-data\\fonts\\ttf\\DejaVuSans.ttf') with score of 0.050000
```

Figure: Test dataset accuracy values after 4000 iterations

After 5000 iterations:

Results of the classification of test images:

- The images that are reconstructed by the neural net in its final layer after 5000 iterations are almost similar to the one that are reconstructed after 4000 iterations. Same with the accuracies.
- When the values of the softmax function of the results achieved in the final layers are plotted on graph (shown in next page):

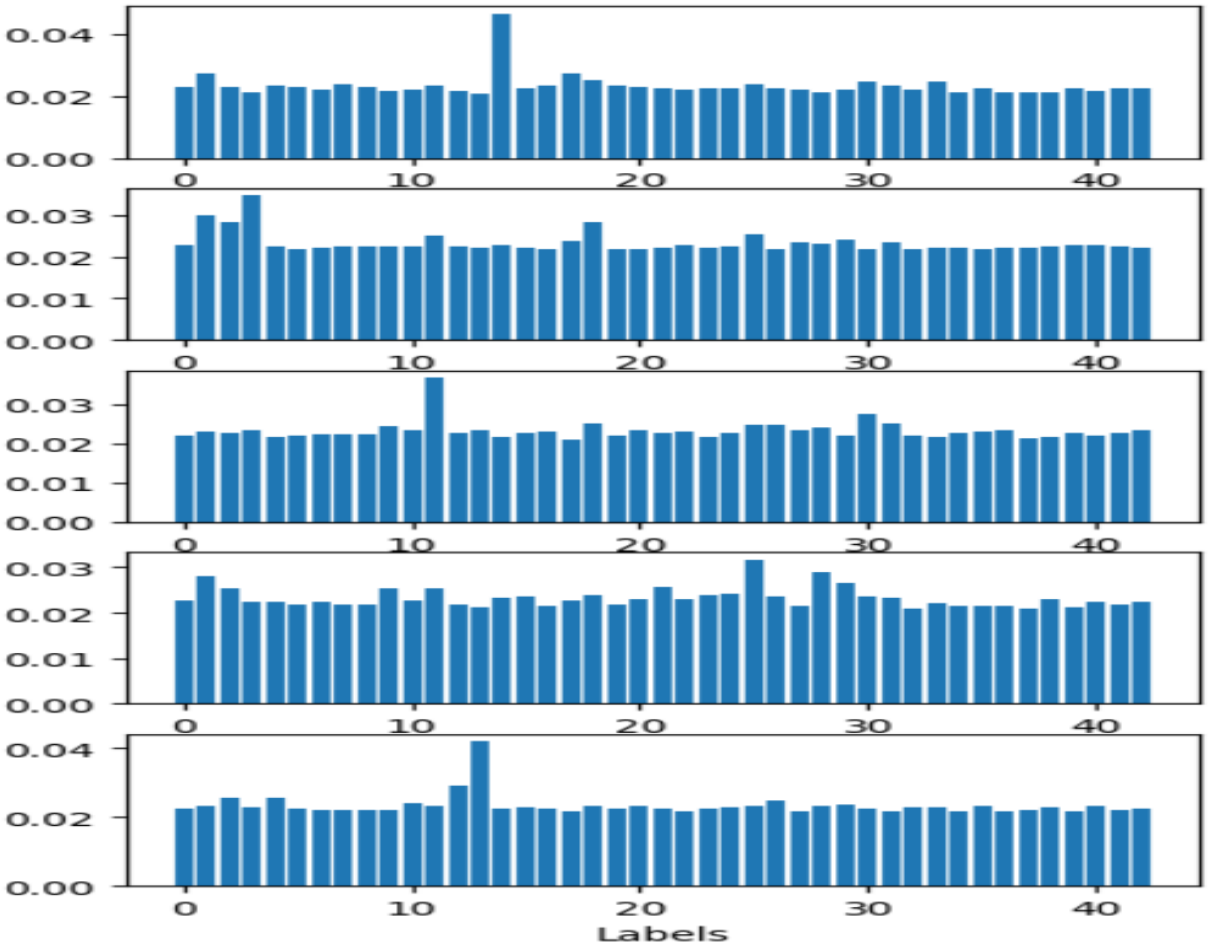


Figure: Softmax values after 5000 iterations

After 6000 iterations:

Results of the classification of test images:

The following images are reconstructed by the neural net in its final layer after 6000 iterations. There is a slight improvement in quality.

Figure: Traffic sign detection after 6000 iterations

When the values of the softmax function of the results achieved in the final layers are plotted on graph:

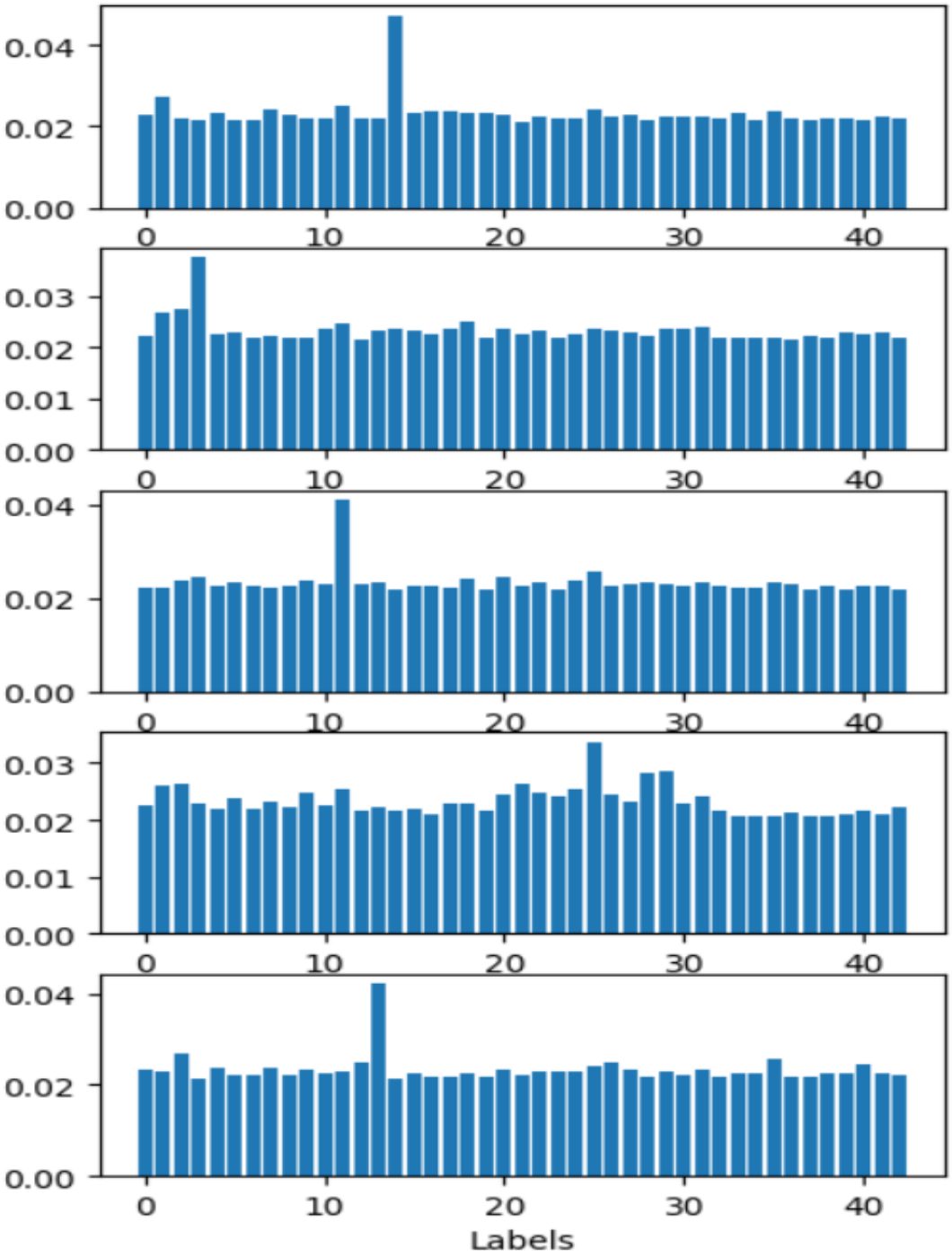


Figure: Softmax values after 6000 iterations

Validation accuracies and loss value:

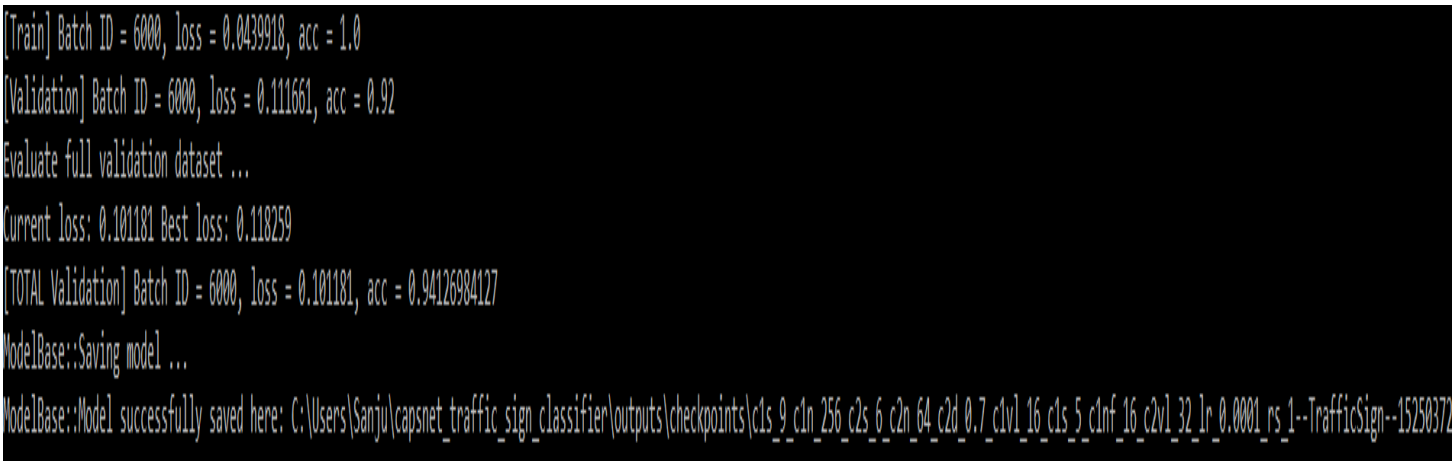


Figure: Validation dataset accuracies and loss values after 6000 iterations

Test accuracy value:

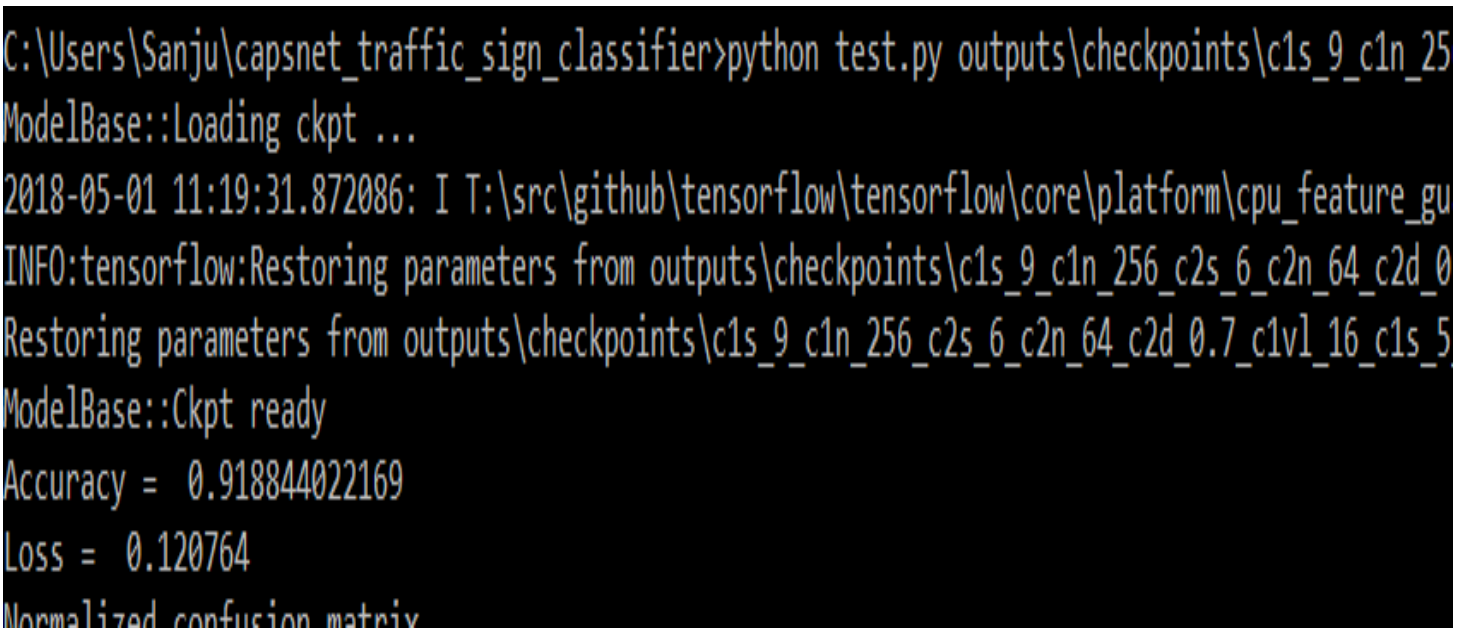


Figure: Test dataset accuracy values after 6000 iterations

After 7000 iterations:

Results of the classification of test images:

After 7000 iterations we stopped training because the reconstructed images had enough clarity for many applications and values of prediction produced softmax function were also appreciable.

Figure: Traffic sign detection after 7000 iterations

When the values of the softmax function of the results achieved in the final layers are plotted on graph:

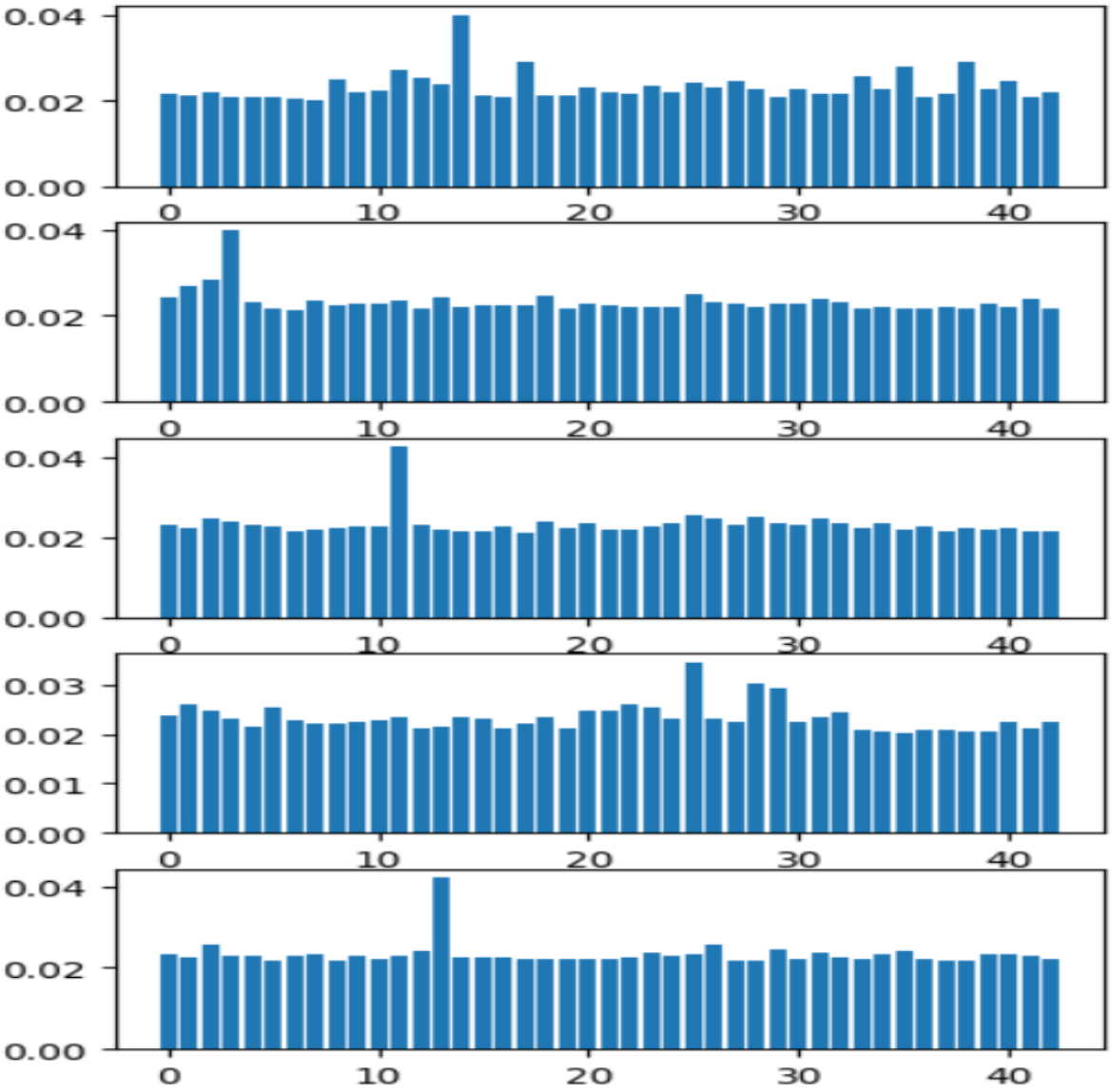


Figure: Softmax values after 7000 iterations

- **Training: 99% (best case: 100%)**
- **Validation: 94%**
- **Test: 93.8%**

Lowest loss attained is: 0.1011

- The result is could not be obtained with a “classic” convolutional network
- But it stands among the implementation of CapsNet done so far.
- Major research and development is done on CapsNet part and less on hyper-parameters tuning and images processing.
- The accuracies would undoubtedly increase with the improvement in image processing techniques used and with the correct hyper-parameters

Drawbacks:

As the training of capsule network requires an inner loop along with the normal epochs, it takes a huge amount of time to train it. For us it took almost a week to get the results we anticipated.

Overall Difficulties Faced:

- Due to the internal loop in capsule network it is harder to train the data and is a huge time consuming process to train.
- So, hardware used is no match to the CapsNet and need to be trained on more powerful hardware.
- The need for checking the accuracy of the CapsNet with different hyper-parameters which was not done for the above same reason.

Summary

We have created and tested various implementations of neural networks and machine learning in to the field of object detection of various objects (fruits, traffic signals) and even human beings with a clear segmentation. We tried and successfully implemented different architectures and systematically built in order to reduce the amount of data that can be used for training. We began by experimenting with many numerous types of combinations of the hyper-parameters. By the end of the last experiment, we were able to detect every image without losing the information of its orientation till the end and were successfully able to reconstruct the input image with high accurate information of its orientation, size, angle and color.

Future Work

- Along with normal iteration in the training process, there is an extra internal loop in capsule network which makes it harder to train.
- So, hardware used is no match to the CapsNet and need to be trained on more powerful hardware.
- There is a need for checking the accuracy of the CapsNet with different hyper-parameters as discussed in the difficulties.
- So, we will try to somehow manage to work on powerful hardware with higher GPU CPU power, which will dramatically improve our performances.
- Our next checkpoint is to increase the accuracy higher than ever attained by the capsule network.
- And also to decrease the over-all lose caused by the marginal and reconstruction loss.
- We will experiment with the different hyper-parameters to attain better values.
- We will eventually implement the same technology in to a product which can then outstand other cutting edge applications in this field.

References

- 'Dynamic Routing Between Capsules', submitted on 25th Oct 2017)

  (Geoffrey Hinton, Sara Sabour, Nicholas Frost)
- Tensorflow Implementation of Capsule Networks on MNIST by Naturomics.
- Pytorch Implementation of Capsule Networks on MNIST
- Keras Implementation of Capsule Networks on MNIST
- MATRIX CAPSULES WITH EM ROUTING (Geoffrey Hinton, Sara Sabour, Nicholas Frost)
- Rethinking the Inception Architecture for Computer Vision

  (Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens)
- Becominghuman.ai
- Medium.ai
- Blitznet: A Real-Time Deep Network for Scene Understanding

  - Nikita Dvornik, Konstantin Shmelkov, Julien Mairal, Cordelia Schmid

- You Look Only Once: Unified, Real-Time Object Detection - Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi
- A Survey of Semantic Segmentation - Martin Thomas