USING OBJECT DETECTION TECHNIQUES TO DETECT AND CLASSIFY ACNE LESIONS

Submitted by

ASTHA GUPTA [Reg No: RA1611003010228]
CHINMAY AGRAWAL [Reg No: RA1611003011199

Under the guidance of
(, Department of)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM UNIVERSITY
(Under section 3 of UGC Act 1956)

S.R.M. Nagar, Kattankulathur, Kancheepuram District

JUNE 2020

SRM UNIVERSITY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report titled "**USING OBJECT DETECTION TECHNIQUES TO DETECT AND CLASSIFY ACNE LESIONS**" is the bonafide work of " **ASTHA GUPTA [Reg No: RA1611003010228], CHINMAY AGRAWAL [Reg No: RA1611003011199**", who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

GUIDE

Dept. of

SIGNATURE

Dr. B. AMUTHA
HEAD OF THE DEPARTMENT
Dept. of Computer Science & Engineering

Signature of the Internal Examiner

Signature of the External Examiner

OWN WORK DECLARATION

Department of Computer Science and Engineering

SRM Institute of Science & Technology

Own Work* Declaration Form

This sheet must be filled in (each box ticked to show that the condition has been met). It must be signed and dated along with your student registration number and included with all assignments you submit – work will not be marked unless this is done.

<u>To be completed by the student for all assessments</u>

Degree/ Course : BTECH

Student Name : ASTHA GUPTA , SUMANTA BANERJEE

Registration Number : RA1611003010228 , RA1611003010052

Title of Work : CLASSIFICATION OF SKIN LESIONS USING TRANSFER LEARNING APPROACH

I / We hereby certify that this assessment compiles with the University's Rules and Regulations relating t Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Educatio Committee guidelines.

I / We confirm that all the work contained in this assessment is my / our own except where indicated, and that I / W have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalised in accordance with the University policies an regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

ACKNOWLEDGEMENTS

We express our humble gratitude to Dr. Sandeep Sancheti, Vice Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.
We extend our sincere thanks to Dr. C. Muthamizhchelvan, Director, Faculty of Engineering and Technology, SRM Institute of Science and Technology, for his invaluable support.

We wish to thank Dr. B. Amutha, Professor Head, Department of Computer Science and Engineering, SRM Institute of Science and Technology, for her valuable suggestions and encouragement throughout the period of the project work.

We are extremely grateful to our Academic Advisor Dr.R.Annie Uthra , Associate Professor, Department of Computer Science and Engineering, SRM Institute of Science and Technology, for her great support at all the stages of project work.

We register our immeasurable thanks to our Faculty Advisor, Mr A.M.J Muthukumaran, Associate Professor and Dr Revathi Venkataraman, Professor, Department of Computer Science and Engineering, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to my guide, Mr U.M Prakash,Associate Professor, Department of Computer Science and Engineering, SRM Institute of

Science and Technology, for providing me an opportunity to pursue my project under his/her mentorship. He provided me the freedom and support to explore the research topics of my interest. His passion for solving the real problems and making a difference in the world has always been inspiring.

We sincerely thank staff and students of the Computer Science and Engineering Department, SRM Institute of Science and Technology, for their help during my research. Finally, we would like to thank my parents, our family members and our friends for their unconditional love, constant support and encouragement.

Sumanta Banerjee

ABSTRACT

The problem of acne is one that affects people all around the world. Mild forms of acne resolve on their own within a few weeks. However, if left untreated, chronic forms of acne lesions could turn into more severe forms resulting in inflammation, swelling, pain and infection. In order to determine the proper medical treatment plan for a patient, a dermatologist has to not only ascertain the severity of acne, but also keep in mind the type of acne lesions developed. This results in a significant amount of time being devoted to the manual task of identifying and classifying the lesions for every patient. In order to help with this issue, an automatic technique could be devised for the purpose of detecting and classifying acne lesions in an image. Through this project, we aim to devise a deep learning-based system to count the number of acne lesions in an image and classify each of them into two categories namely inflammatory and non-inflammatory acne (whiteheads and blackheads). The data set comprises 469 images distributed among both classes. The proposed method relies on the use of widely known object detection algorithms to localise and detect the spots followed by a classification network to predict the labels for each spot. We believe that this system would prove to be very helpful to help to reduce errors and manual effort, while also working as an efficient safety net to help with a patient's diagnosis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

CNN	Convolutional Neural Network
ResNet	Residual Network
FR-CNN	Faster Regional Convolutional Neural Network
YOLO	You Only Look Once
SSD	Single Shot Detector
mAP	mean Average Precision
RMS	Root Mean Squared

CHAPTER 1

INTRODUCTION

1.1 Acne

Diagnosis of acne involves a medical examination of skin by a dermatologist. First step in diagnosis of acne is to determine the severity of acne by grading them in the range of one to four (one being mild and four being severe), then the doctor may examine and determine the types of acne on the patient's skin. The area of skin affected is also taken into account during the diagnosis process. After the skin examination, some questions are asked about the frequency, occurrence, if acne runs in the family.

While doctors are adept at the diagnosis process, there's always a scope for human error during this process. The part of the diagnosis most affected by the human error is the skin examination. Sometimes the doctors are required to quantify the number of acne in an area and a ballpark is taken into account. This is where the intervention of Artificial Intelligence can come handy, to aid the process of diagnosis and consequently resulting in better treatment.

Although the problem of acne is mostly present during the adolescent age period, it can be a chronic problem for many people. Women, especially those suffering from hormonal disorders, can end up with more severe forms of this disease which can lead to permanent scarring. Even though acne is not a life-threatening disease, it can affect one's mental health and lead to more serious mental health issues. Patients have to deal with the social stigma that revolves around this issue and end up constantly being worried about the cosmetic disadvantages that acne problems can result in.

Acne are of several types and they possess certain characteristics which make the types discernible to human beings. For example, plugged pores can be whiteheads or blackheads depending on if they are open or closed. There are two main classes of acne- inflammatory and non-inflammatory.

Just like the name suggests, the former class consists of those acne types which result in the formation of inflammation around the acne lesion and are marked by pain and redness of the skin. They consist of cysts, nodules and pustules and papules. Red bumps which are tender and small can be papules, papules with pus are called pustules or pimples, nodules are larger in size and firm lumps under the surface of skin and pus filled nodules can be classified as cysts. While pustules and papules can be treated with over-the-counter remedies, the former two types require the care of a dermatologist. The second class of acne lesions consist of whiteheads and blackheads. They are usually self-treatable using readily available treatment options.

1.2 Problem Statement

Chronic acne, while affecting the cosmetic appearance of a person, can eventually lead to underlying mental health conditions especially during the adolescent years. It becomes imperative to take some action for the treatment of acne even if it is of the non-inflammatory type.

Furthermore, the transformation of non-inflammatory types to inflammatory types of acne can result in quite painful lesions which would require medical treatment, often in the form of topical drugs. In order to assess the progress of the treatment as well as to study the effects of the prescribed drugs, it is essential for the dermatologists to regularly review and monitor the patient regularly. This can lead to a substantial amount of time and money being involved throughout the treatment on both sides.

By developing a method to help quantify and classify the acne lesions on a facial image, we aim to provide patients with a way to monitor and assess their lesions. Not only would it help them save time and act as a safety net, it could also help dermatologists save time and effort.

Through our project, we wish to develop a deep learning-based object detection and classification technique in the field of dermatology. Using transfer learning to extract weights followed by building a classifier to train the model, the technique would allow the model to detect acne lesions in an image and classify them into one of three classes namely inflammatory, blackheads and whiteheads.

1.3 Deep Learning

A subset of Artificial Intelligence, the field of deep learning involves the use of neural networks in order to build models. Over the years, deep learning has been used majorly in the area of computer vision to detect, localise and classify images. Mimicking the task of neurons in the human body, the deep neural networks consist of multiple layers of nodes which process information as it passes through them.

Traditional techniques for image detection often rely on a domain expert to hand-craft features which are considered to be important for the task at hand. Often in the field of medical imaging, the variability between different classes of images might not be discernable by the human eye. This dependence can eventually result in inefficient results when tested over a large scale of images. In order to deal with this, deep learning offers an automated method to detect features and extract them for the model at hand.

Deep learning techniques involve pre-processing the data before being fed into a neural network. The network, then, learns features from the input images and assigns weights to the selected features. This is followed by a series of training steps for batches of images which help to adjust the weights and reduce the loss of detection. Finally, the trained model is used to detect objects in new images which form the test set.

CHAPTER 2

LITERATURE SURVEY

2.1 Methods Used

In recent years, many dermatologists have worked with computer scientists to help solve the problem of tracking and counting acne lesions. With the advent of computer vision methods, numerous techniques have been proposed to detect and/or classify acne lesions. The studies [1] to [7] use image processing and computer vision techniques to detect the acne lesions in facial images.

Fartash Vasefi et al. propose a color-based detection technique wherein they convert the original images into their CIE L*a*b versions and use a Gaussian filter applied on the a* plane (which depicts redness) to detect the acne spots using Otsu thresholding [1]. Then, they classify that spot as a papule or pustule by detecting if the segmented binary area has a pus-filled center or not, by using the Euler number generated for that particular lesion.

Another acne detection technique has been proposed in [2] wherein the RGB images are converted into their HSV and normalised grayscale versions and the subtraction between the V plane of the former and the latter is performed and the resulting image is converted into its binary threshold-ed form. This has then been used to find the suitable acne lesions.

The study by Thanapha Chantharaphaichi et al. focussed on the reduction of errors in the detection of acne spots uses a Bayesian classifier on top of a blob detector via segmentation, with a histogram-based feature extraction [3]. The training is supervised and the testing is unsupervised. The main idea of this paper is to reduce the error after the extraction phase using a probabilistic method.

Recently, machine learning and deep learning have been used by researchers for the task at hand. A SVM-based classifier has been proposed by C. Chang et al. [4], where the region-of-interest extraction is followed by defect extraction. It involves various stages including the extraction of the region of interest from the input image. The three classes in this study include normal skin, acne lesions and spots.

In [5], Lim ZV et al. uses deep learning to develop a classification model based on severity for three classes. The pre-processed images are augmented and fed to the CNN architecture. A comparison between the three widely known models namely, ResNet, DenseNet and Inception, is performed.

The method proposed by Zhao, Tingting et al. [6] uses five levels of acne severity as the basis of classification. Making the use of transfer learning, wherein the CNN architecture uses features extracted from a base data set, which are then calibrated to suit the domain-focussed data set. A pre-trained ResNet-52 architecture has been chosen as the CNN architecture. The results of the model are compared with those of clinicians to ascertain its performance.

Finally, Shen, X. at al. used a template matching technique with a sliding window to first perform a binary classification between skin and non skin patches, followed by classification into seven different categories of acne type [7]. This technique divides the pre-processed images into patches of fixed sizes and uses the window to classify each of them first as skin or non-skin, followed by the type of acne.

2.2 Conclusion

Having surveyed the literature on the various techniques used for the problem statement, we can conclude that all those studies fell broadly into one of two categories in terms of the methodology used.

The traditional Computer Vision techniques focus on the use of image filtering, edge detection and threshold-based segmentation. The feature extraction is performed on the basis of handcrafted features specified by a domain expert. The papers that use traditional methods perform detection mostly on the basis of a colour-based segmentation. The main problem with using these methods for our problem is that all the different types of acne have a complex set of similar as well as dissimilar features. While both blackheads and whiteheads do not have any redness around them, they have opposite colours in the RGB spectrum. Thus, the use of traditional methods will not prove to be very effective because it will suffer from low-quality feature extraction.

With deep learning, the task follows an end-to-end approach. The task of pattern detection is performed by detecting similarities between the image of a class on a pixel level. The nodes of the network learn the similarities of images in the same class and use them to perform a probability-based classification. The feature extraction is performed by the algorithm with a focus on even the most minute of details. When it comes to the task of classification of images, deep learning methods have been established to perform much better than traditional techniques.

CHAPTER 3

SYSTEM ANALYSIS

3.1 Dataset

The acne lesion dataset consists of a total of x images,x of which are inflammatory acne images and y are non-inflammatory acne lesions. The non-inflammatory are divided into z images of blackheads and w images of whiteheads. The dataset was obtained by scraping images from Dermnet, the largest independent source for dermatology images. Several scripts were written in Python using the beautifulsoup framework to scrape the images from the website and save them locally on the workstation.

After obtaining the images, they were fed into a preprocessing pipeline which was required before passing it to various object detection algorithms. One of the major parts of the pipeline is annotating the dataset to conform with the implementation of the multiple object detection algorithms. The mostly used data annotation format is PASCAL Visual Object Classes (VOC) format. The dataset was annotated using an open-source data annotation tool known as LabelImg.

3.2 Use of Object Detection Based Methods

Given that the problem statement specifies that we need to build a system which would be able to not only count the acne lesions but also classify it into one of two categories, the decision to use object detection and classification algorithms was not a difficult one. One of the main reasons to not choose traditional methods like image segmentation was due to the variability of the different classes. It would have been very complex to decide for a single parameter of difference between all the classes.

For instance, the inflammatory types are marked by redness around them, but that could not be used for determining the threshold because within the non-inflammatory types the blackheads and the whiteheads are of completely opposite colors in the RGB spectrum. Thus, color-based segmentation was out of the question if we wanted the model to not be biased.

Another important consideration that we kept in mind while choosing deep learning was the high complexity of the data set. The variability between the skin color and the size of the lesion would make it difficult for us to define a fixed set of features. Given that deep learning based methods would break down the image pixel-by-pixel and extract defining features intuitively made it to be a better choice for the problem at hand. It is our belief that while traditional methods do work well in many cases, this is one where convolutional networks would perform better.

3.3 Faster- RCNN

One of the state of the art object detection algorithms used for the particular problem statement was Faster Regional Convolutional Neural Networks. It is the third iteration of Regional Convolutional Neural Networks, Fast Regional Convolutional Neural Networks being the second. It is famously trained using the PASCAL VOC format, hence the dataset was labeled in the PASCAL VOC format.

The typical pipeline of Faster-RCNN includes annotating the dataset after resizing the image maintaining the aspect ratio. After this, the Faster-RCNN model is trained on a base network with data augmentation occurring during the training process. The checkpoints of the model are saved after every fixed set of epochs making sure that the loss is converging.

The architecture of Faster-RCNN consists of two major networks. The first network is the Region Proposal Network (RPN) which outputs proposals for the main networks using a base network and convolution function instead of fully connected layers. This ensures that images of different sizes can be used to train the model. The main network of Faster-RCNN outputs the final coordinates of the objects with a classification score via a softmax function.

Being the third successor of Regional Convolutional Neural Networks, it can test images faster and also do it more accurately when compared to other models in terms of mean average precision(mAP). The advantages of this complex architecture includes being able to detect objects of various sizes in the test image, accepting images of multiple sizes as the input and also the speed at which it tests the images.

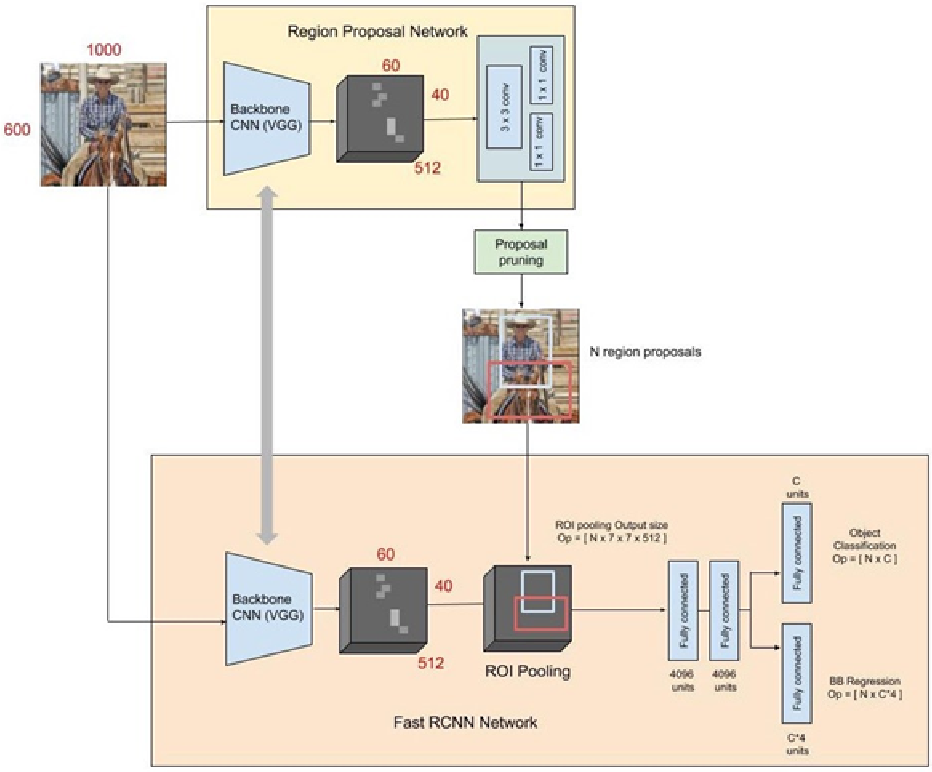

Figure 3.1: Architecture Diagram for FR-CNN [8]

3.4 You Only Look Once

Most object detection algorithms rely on the use of a regression task followed by a classification task. Even though it is a robust technique, it takes up a large amount of time and the underlying process is often complex. To deal with this problem, the YOLO architecture for object detection and classification was proposed.

In YOLO, each image is passed through the convolution layers only once. The image is divided into grids and anchor points for each grid are specified. The task for each grid is to predict if any object is present in its area with its associated probability. The grid responsible for predicting the object is the one in which the centre of the detected object lies. Relying on the probability for the class, YOLO sees the task only as a regression problem.

Over the years, four versions of YOLO have been released. The major benefit of this architecture is the speed it offers, making it the go-to option for real time object detection. Furthermore, unlike region-based detectors, it looks at the whole picture rather than specific regions. Another great advantage that YOLO offers is that it learns not only from the object-specific information, but also considers contextual information about the images. When tested on the COCO dataset for 80 classes, it was evaluated to have an mean Average Precision value of approximately 60 percent, making it at par with its contenders.

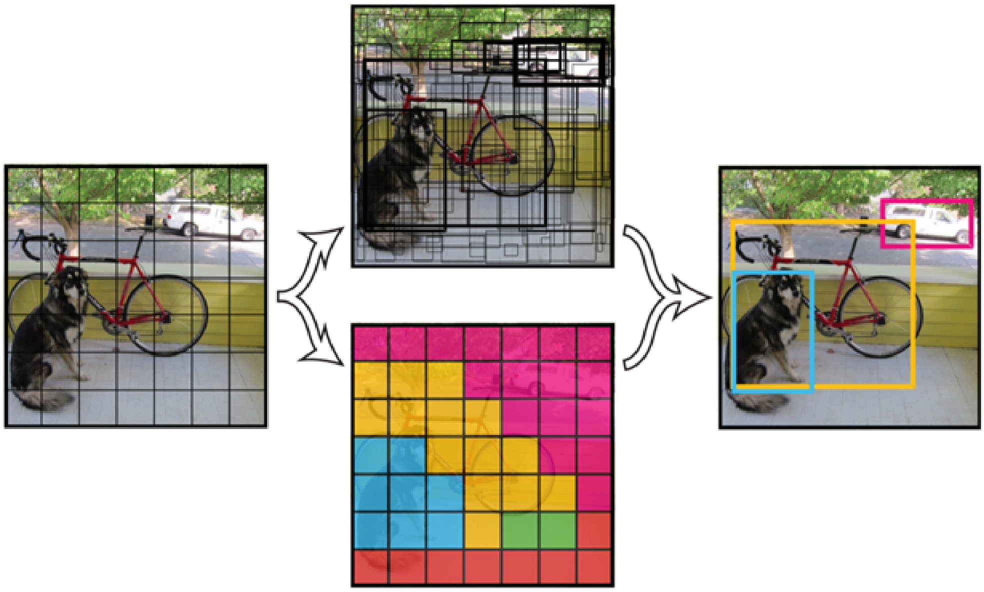

Figure 3.2: Architecture Diagram for YOLO [9]

3.5 Single Shot Detector

Single Shot Detector (SSD) is an object detection algorithm which uses a backbone ConvNet which is pre-trained to extract features. Fully connected final classification layer is removed from the ConvNet, and more convolution layers added to the backbone and the output is received as bounding boxes and their classes. Backbones can be any deep ConvNet trained for image classification, some popular architectures make use of Inception and ResNet networks as backbones.

SSD does not need a sliding window and multiple passes to detect objects, it can detect objects in a single pass, using grid cells. The image is first divided into grids and the resulting cells are responsible for detecting the object inside that cell. Every cell can detect the form and class of the object in their region. Dimension of the grid can also be decided as per the setting requirement.

Anchor boxes are the pre-defined rectangular boxes which are assigned to every cell and are responsible for detecting different sizes and forms of objects. Every anchor box has different dimension ratios and multiple anchor boxes can be assigned to a single cell. These anchor boxes have a specific aspect ratio and zoom level to define them. The size of the object within a cell may be smaller or larger, and to take care of that, zoom can be defined to scale the box with respect to the cell. For example a giraffe and a dog are of different sizes and both would require their bounding boxes to be of different ratios. A giraffe would require a vertically taller box with less width and a dog would have a horizontal box with enough height. The ground truth boxes which are annotated manually before training, are matched with anchor boxes to find out the best overlap during the training. The size of the object within a cell may be smaller or larger, and to take care of that, zoom can be defined to scale the box with respect to the cell.

The most important attribute of SSD is the use of receptive fields. Receptive field is the region which is being used by CNN's feature at that time in the input space. It is known that the different sizes of the region are the result of features at different layers, due to convolution operations. The deeper layers have larger feature size and the size gets smaller as we move to the upper layers. These variable feature maps help the SSD to detect objects of different sizes and give out an accurate bounding box. SSD has more number of convolution layers at the backbone which result in a separate object detection output. Due to having different sizes of receptive fields, earlier layers are able to detect small objects.

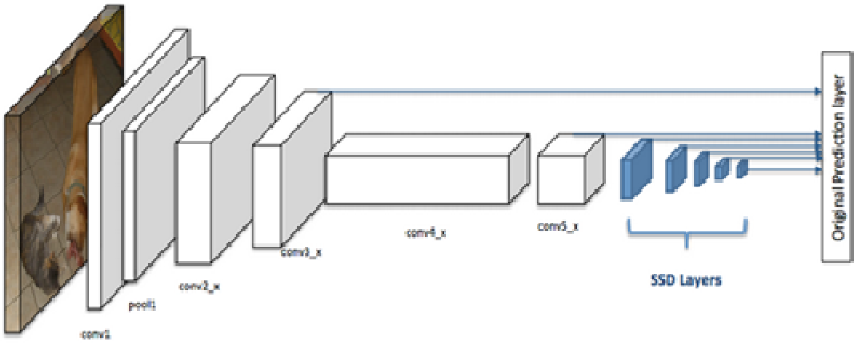

Figure 3.3: Architecture Diagram for SSD [10]

3.6 Hardware Dependencies

The major hardware dependencies include a Graphics Processing Unit consisting of 1xTesla K80 , compute 3.7 having 2496 CUDA cores and 12GB GDDR5 VRAM. It also includes a CPU consisting of 1xsingle core hyper threaded Xeon Processors @2.3Ghz i.e (1 core, 2 threads). These are the configurations of an instance of a Google Colab notebook using which the object detection models are trained.

Apart from the specifications of the workstation, the end product will be developed into a mobile application. The initial development of the application will be for the ios platform requiring an IPhone and will later be developed for android operating systems as well.

3.7 Software Dependencies

The software dependencies includes the programming languages, platform used to train the object detection algorithms and the tools used for data annotation. The programming language used for the project was Python 3.7. Several frameworks in Python were used to solve individual parts of the project. The deep learning frameworks include TensorFlow and Keras. The framework to scrape the acne images from Dermnet was BeautifulSoup. The frameworks for numerical computations include NumPy and Pandas.

The platform to train the object detection algorithms was Google Colab with the hardware specifications as mentioned in the heading above. Google Colab is a platform offered by Google Research. It is used to run code on the cloud in the form of a Jupyter Notebook. It is primarily used to train machine learning and deep learning models, data analysis, etc. The main advantage of using Google Colab is the free resources that are provided by Google which includes free GPU usage.

An open-source tool was used for data annotation known as LabelImg. It is an image annotation tool with a Graphical User Interface. It is used to make bounding boxes across the objects inside the image. It saves the annotations in the form of an XML file with regards to the PASCAL VOC format which was used by Imagenet. It also includes support for YOLO.

CHAPTER 4

SYSTEM DESIGN

A common architecture was decided for all the algorithms chosen. The basic idea was to perform a common pre-processing step for all and use the same evaluation metrics to adjudge their performances. Then the results were compared to determine the best one.

The first step was the extraction of annotations from the dataset of images. This was done in the PASCAL VOC format for FR-CNN and SSD, and a specific format for YOLO algorithm. The total number of images was 496. Data augmentation was also performed while training.

The next step ws to train the model using pre-defined parameters initially. The parameters included the position of anchors, the size of the image batches, the number of epochs (iterations for YOLO) and the optimisers used. The loss function was also selected acording to the specifications of the model. This step also included the use of pre-trained models for feature extraction.

Finally, the models were tested on the test cohort and the mean Average Precision (mAP) was calculated using the Precision vs. Recall Curve for each algorithm.

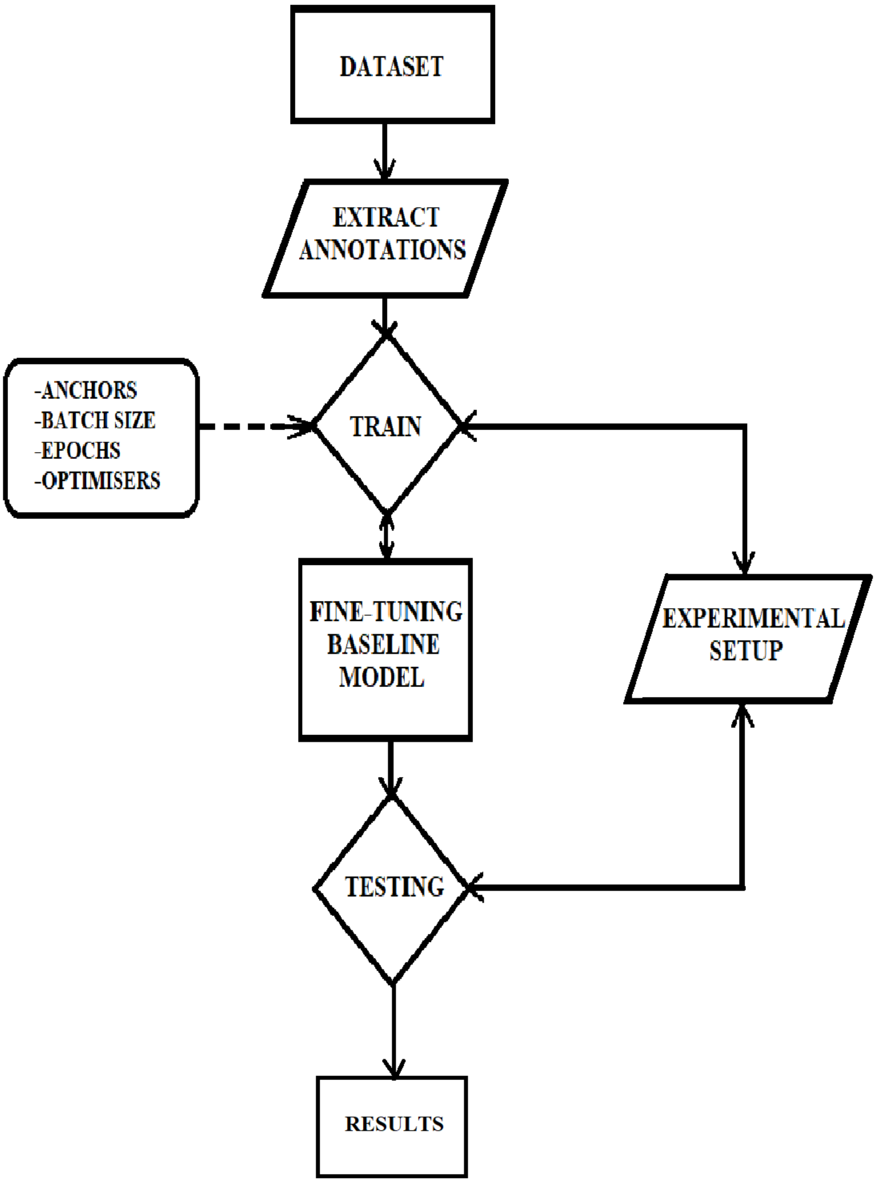

Figure 4.1: Architecture Diagram for all three modules

4.1 YOLO Modules

4.1.1 Pre-processing

YOLO requires a specific form of data annotation that is different from the PASCAL VOC format. The bounding box information for it needs to be in the form of a text file with five features for each object in an image. These five values are the class, xmin, ymin, xmax and ymax values. Furthermore, a separate txt file for each image needs to be created. Additionally, after splitting into train, test and validation sets, a text file needs to be created for each subset which would contain the location of each image file present in the subset. Finally, a yolo.names file and a yolo.data files need to be created in order to define the train, validation and class files. A location to save the last weights during training is also created.

4.1.2 Backbone Architecture

Yolo relies on the use of DarkNet as its base architecture. By means of transfer learning, weights for the base architecture are obtained by training it on the COCO dataset. These weights of the pre-trained model are used as the starting weights for training the YOLO model. DarkNet consists of a network of fully connected layers. Based in C, this architecture provides a high speed alternative to the other models. <diagram>

4.1.3 Training

The YOLO model was trained using initial weights extracted by training the Darknet architecture on the COCO dataset. The adjusted weights were saved after every 1000 iterations and the final weights were saved in the backup folder. For the custom training, the classes were set as three and the filter size was set as 24. The model was trained for 6000 iterations (2000 iterations per class) with a learning rate of 0.001 and a batch size of 64.

4.1.4 Evaluation

In order to evaluate the model, the metric used was the Mean Average Precision. The maximum validation mAP was calculated as 20.04% with 6000 iterations. The confidence threshold was set as 0.5

4.2 SSD Modules

4.2.1 Pre-processing

Preprocessing steps for SSD are similar to FR-CNN. The standard annotation format for object detection algorithms is the PASCAL VOC format. The dataset, after being scraped from the DermNet Server, is contained inside a repository. Using the data annotation tool LabelImg, the images are manually labelled and annotated in the XML format.

The format majorly retains the coordinates of the bounding boxes that are marked on each object inside the image. Other options include marking whether the object in the image is difficult to locate and also if the object is truncated or not. To train using the Tensorflow object detection API, XML annotations need to be converted to TF Records format, for which the scripts are provided as open source by Tensorflow.

4.2.2 Backbone Architecture

Backbone architecture is the pre-trained ConvNet with the final fully connected layer removed. It is used for feature extraction, which helps the model to train on custom image data-set. For the purpose of this experiment we have used the Inception v2 based SSD model, trained on COCO dataset.

4.2.3 Training

For training 90% of images from the data-set were used and model was trained on the following configuration. The batch size was set to 8 with an RMS Prop optimiser. The initial learning rate was set as 0.004 and the decay factor was set as 0.95. The input size of the images was set as 300x300.

4.3 FR-CNN Modules

4.3.1 Pre-processing

The standard annotation format for object detection algorithms is the PASCAL VOC format which is used by Faster-Regional Convolution Neural Networks. The dataset, after being scraped from the DermNet Server, is contained inside a repository. Using the data annotation tool LabelImg, the images are manually labelled and annotated in the XML format. The format majorly retains the coordinates of the bounding boxes that are marked on each object inside the image. Other options include marking whether the object in the image is difficult to locate and also if the object is truncated or not. One of the advantages of Faster-RCNN is being able to train on images with different sizes but ensuring that the smallest side in all the images are of the same size.

4.3.2 Backbone Architecture

The backbone architecture for Regional Convolutional Neural Networks included VGG16. It included the entire network pre-trained on ImageNet without the fully connected layers. The features that were extracted from the last convolutional layer which is conv5, are used as the base for training RCNN. One of the prominent features of object detection algorithms is transfer learning, using the features/weights from one architecture to generalise the model of another architecture. For Faster-RCNN, ResNet(50, 150, 200) are also used as the backbone architecture. ResNet or Residual Networks are also pretrained on ImageNet with deeper layers and includes skip connections which help in extracting better features when compared to VGG16.

4.3.3 Training

Faster-RCNN was trained on VGG16 and ResNet50 with pretrained weights for the backbone architecture. For custom training, three classes were used: inflammatory, whiteheads and blackheads. The weights were saved after every epoch only if the loss reduced after the corresponding epoch. The model was trained with a batch size of 64. The number of epochs were 100 with adam optimiser and a learning rate of 0.0001.

4.3.4 Evaluation

In order to evaluate the model, the metric used was the Mean Average Precision. The maximum validation mAP was calculated as 16.05% with 100 Epochs. The confidence threshold was set as 0.5.

CHAPTER 5

RESULTS

5.1 YOLO

The YOLO algorithm was trained on a dataset of 496 images with a test-train split of 10-90 percent. The backbone architecture used was DarkNet and the batch size was set as the default value of 64.

Data augmentation was used during the training process and all images were resized to 416x416. This was done automatically by the system. The confidence threshold was set at 0.5 for the calculation of precision and recall.

Accuracy was not chosen as the evaluation metric due to the problem of class imbalance. The mean Average Precision value is a more reliable metric for such a scenario. Overall, the model achieved an mAP value of 46.53%.

Certain drawbacks of YOLO are to be responsible for these results. However, the training and testing time for YOLO was lower than its counterparts.

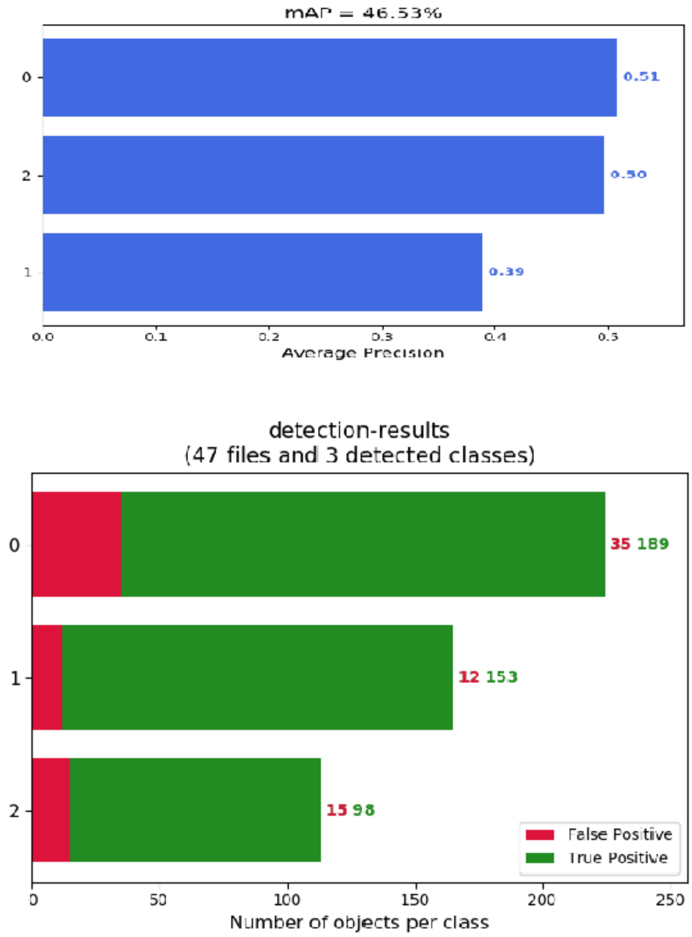

Figure 5.1: Mean Average Precision for YOLO

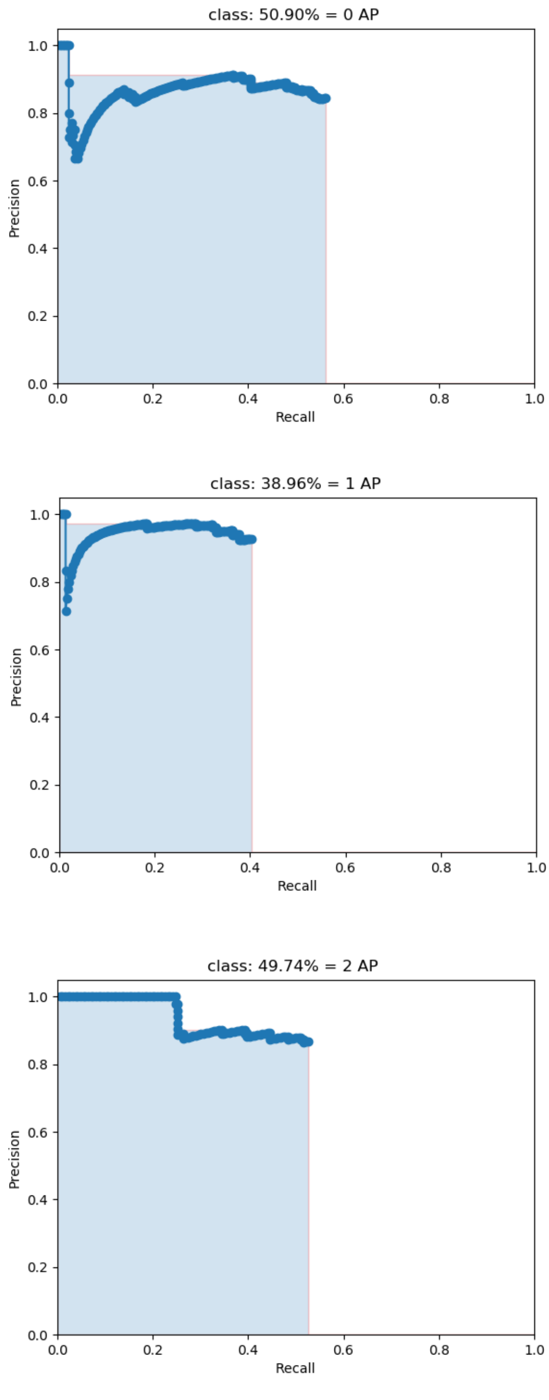

Figure 5.2: Precision vs. Recall Curves for YOLO

5.2 SSD

The SSD algorithm was trained on an NVIDIA 2080Ti workstation with a batch size of 8 for 341 epochs. The base network used was Inception V2. The optimiser used was the Root Mean Squared (RMS) prop optimizer.

Train test split was consistent with other algorithms used in this study. Since the batch size is small, augmentation is not necessary in this model. The performance was measured based on the precision and recall curves for the three classes. It achieved an mAP of 52.04%.

Figure 5.3: An image sample of the detection results

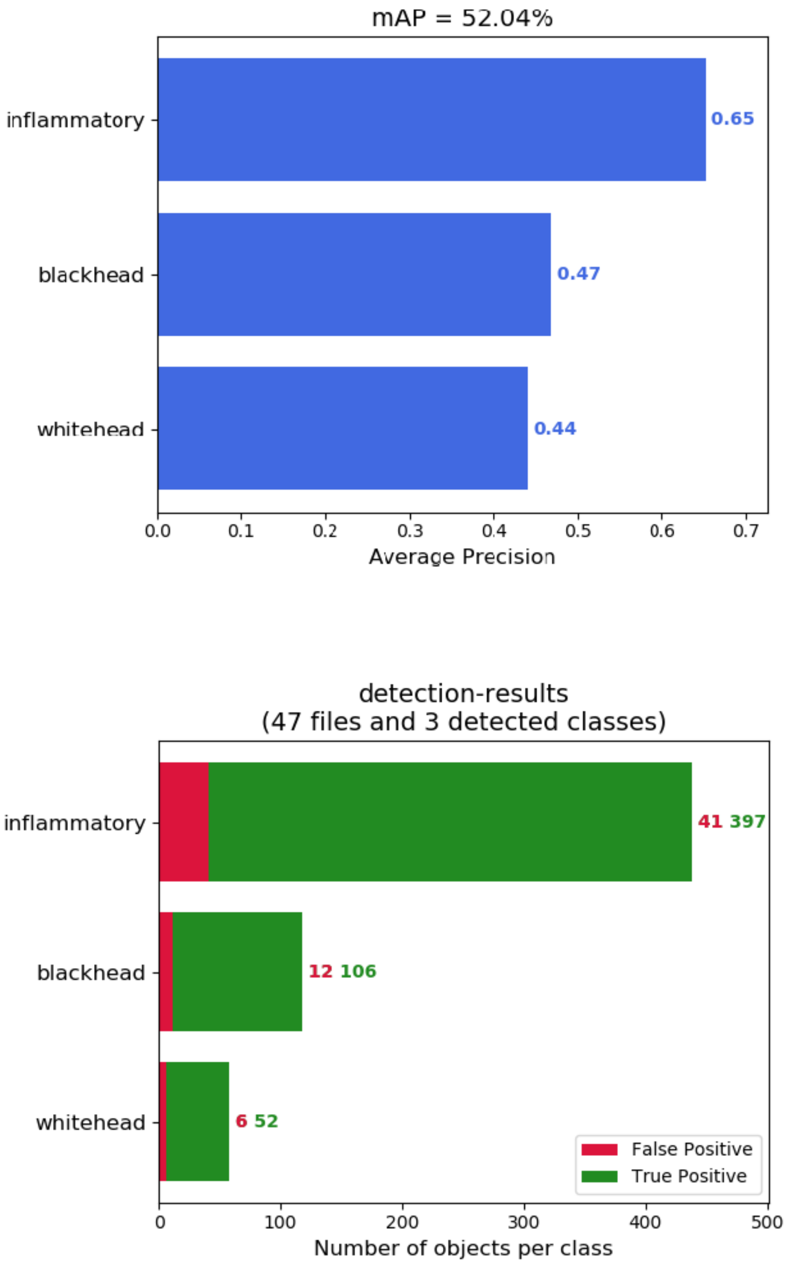

Figure 5.4: Mean Average Precision for SSD

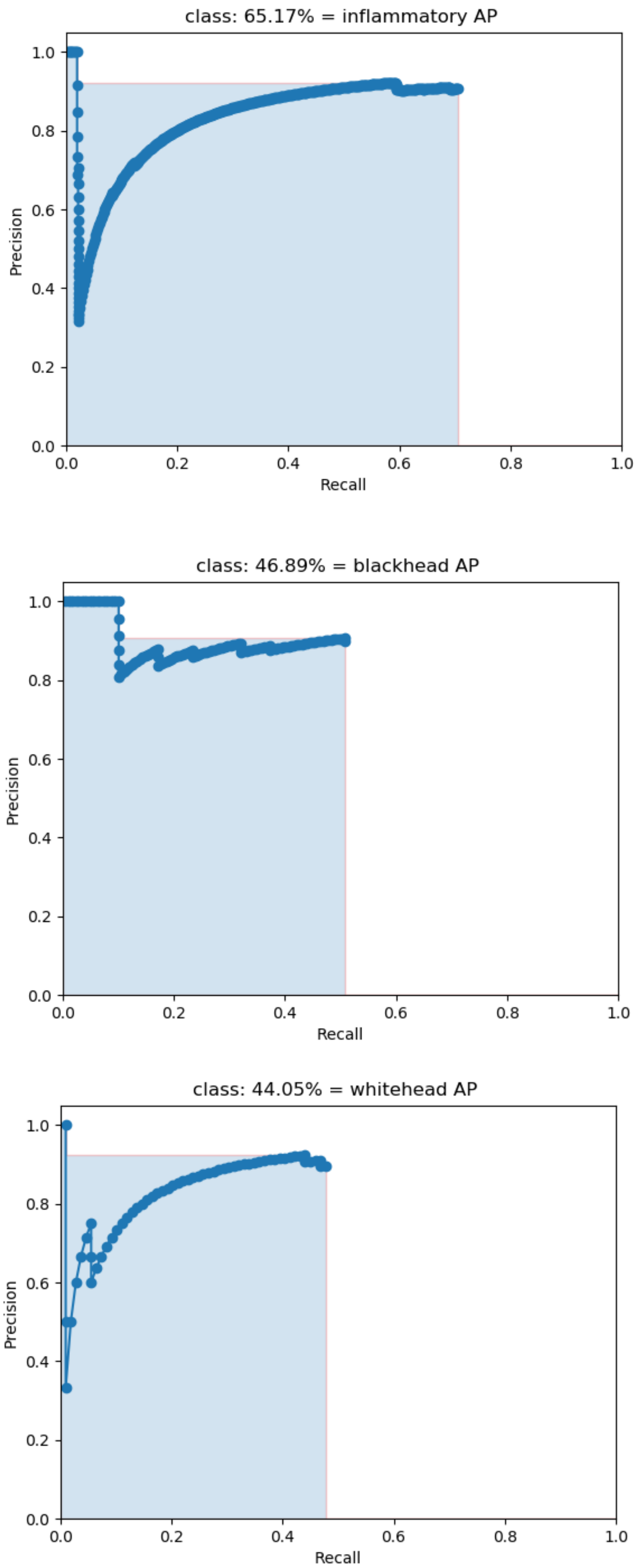

Figure 5.5: Precision vs. Recall Curves for SSD

5.3 FR-CNN

The Faster-RCNN algorithm was trained on an NVIDIA 2080Ti workstation with a batch size of 64 for 100 epochs. The base network used was ResNet50. The optimiser used was Adam with binary cross-entropy loss.

The train-test split was 90:10. There was no validation set associated with the model as the size of the dataset was really low. The test set included an equal number of images in which there were a majority of single class lesions.

To tackle the problem of class imbalance, we performed data augmentation while training the algorithms to ensure synthetic data is generated accordingly and can neutralise the problem as much as possible.

As shown in the graphs, the highest mAP achieved was 86.61% with Faster-RCNN. The individual Precision-Recall Curves for each class are also shown below.

Figure 5.6: Sample output image for FR-CNN

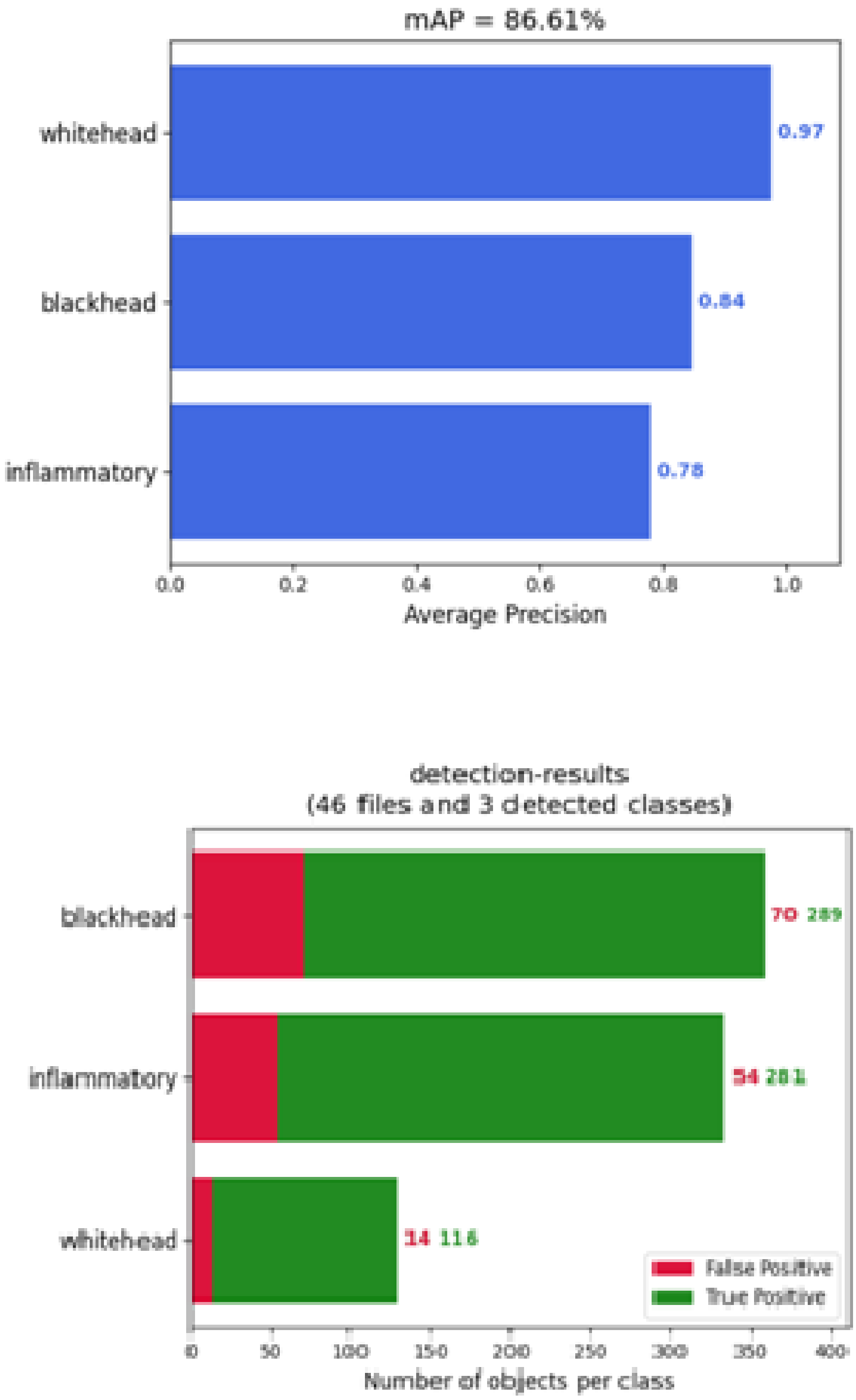

Figure 5.7: Mean Average Precision for SSD

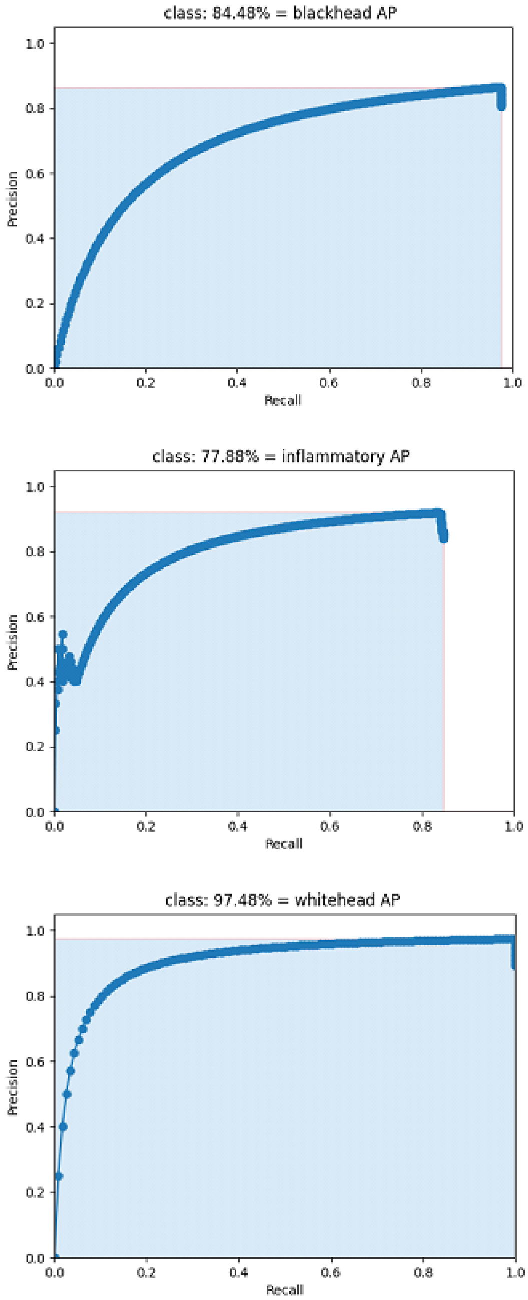

Figure 5.8: Precision vs. Recall Curves for FR-CNN

CHAPTER 6

CONCLUSION

Acne detection methods that specifically localise or classify acne lesions either based on severity or type are being used by dermatologists in day to day lives. A novel deep learning system that can both localise and classify acne lesions has been studied and developed in this project.

As we compare various object detection algorithms that were used to detect acne lesions, we come to the conclusion that Faster-RCNN performs better than YOLO and SSD Multibox in terms of Mean Average Precision obtained by calculating the area under the precision-recall curve for all the classes. One of the reasons for this is the two-network structure of Faster-RCNN. The model first identifies all the objects in the image and then moves further ahead to localise and classify them accurately. By changing the anchor scales in Faster-RCNN from the norm, we were able to successfully detect small lesions such as blackheads and whiteheads as well.

There were several limitations that were encountered along the way which are mentioned in the report. Some of these limitations were resolved using data augmentation techniques and optimisations in the model architecture. These changes helped achieve a better model to detect acne lesions in terms of precision.

This project is aimed to set a benchmark for acne detection models in the future. The images used in the future to develop better models must include high quality images with proper labeling. The study also shows that there is a dire need to maintain a centralised dataset for acne lesion detection to build stronger and more powerful models in the future.

CHAPTER 7

FUTURE ENHANCEMENTS

The research in applications of Artificial Intelligence in the field of healthcare is fast growing and starting to make its presence. Based on the results and work provided in this work, there's a large scope for further development and improvement of current methods.

7.1 Dynamic Region of Interest (ROI) Extraction

Currently, any region-of-interest is extracted manually before being fed into the model. However, in a future enhancement it would be possible to develop a way to extract the regions automatically through image processing techniques. The location of the pupil could serve as a reference point using which the system would be able to obtain cheek, nose and forehead areas to be treated as separate images for the training process. This would remove the background outliers leading to an increase in efficiency and performance.

7.2 Sub-Classification of Inflammatory Class Acne

There are several sub classifications of inflammatory acne such as inflamed comedones, papules, pustules, nodules and cysts. These types can be diagnosed using object detection algorithms if there is enough data available. Improvement of models to detect a wide range of classes can make the model applicable in the real life healthcare setting.

7.3 Severity and classification

Current research work related to acne, focuses on specific problems such as calculating the severity of acne or detecting the classes using image data. After gaining sufficient maturity of work in specific fields, these can be clubbed to provide broader results.

7.4 Forming a benchmark dataset

Currently the research being done on the acne image data, is using multiple sources to extract and collect images from multiple sources. Most of the researchers who have worked on problems related to acne images, have not made the dataset open source. The need of benchmark datasets for AI applications in healthcare is critical for future research and development.

7.5 Mobile application development

To aid dermatologists in the real time diagnosis process, a mobile application can be really helpful. Therefore the development and optimization of lighter object detection models can immensely improve the mobile diagnosis process. These models can run tests on the mobile backend to analyse input data and give the output instantly.

APPENDIX A

CODING

A.1 Pre-processing

```
from os import listdir
from os.path import isfile, join
import os
import cv2
from PIL import Image
import imutils
from xml.dom import minidom
import xml.etree.ElementTree as ET

# change this to your path
annotations_path = "/Users/ronald/Desktop/work/sengupta/acne_deep_learning/keras-frcnn/ACNE/Annotations/"
images_path = "/Users/ronald/Desktop/work/sengupta/acne_deep_learning/keras-frcnn/ACNE/JPEGImages/"

# get all xml files in a list
annotations_files = sorted(
    [
        join(annotations_path, f)
        for f in listdir(annotations_path)
        if isfile(join(annotations_path, f))
    ]
)

# change <path> content to the filename.
```

```python
for f in annotations_files:
    tree = ET.parse(f)
    root = tree.getroot()
    for item in root.iter("path"):
        item.text = f
    tree.write(f)
    print("File -", f, "written.")
```

A.2 YOLO Training

A.2.1 Darknet

```python
from ctypes import *
import math
import random

def sample(probs):
    s = sum(probs)
    probs = [a/s for a in probs]
    r = random.uniform(0, 1)
    for i in range(len(probs)):
        r = r - probs[i]
        if r <= 0:
            return i
    return len(probs)-1

def c_array(ctype, values):
    arr = (ctype*len(values))()
    arr[:] = values
    return arr

class BOX(Structure):
    _fields_ = [("x", c_float),
                ("y", c_float),
```

```
                ("w", c_float),
                ("h", c_float)]

class DETECTION(Structure):
    _fields_ = [("bbox", BOX),
                ("classes", c_int),
                ("prob", POINTER(c_float)),
                ("mask", POINTER(c_float)),
                ("objectness", c_float),
                ("sort_class", c_int)]


class IMAGE(Structure):
    _fields_ = [("w", c_int),
                ("h", c_int),
                ("c", c_int),
                ("data", POINTER(c_float))]

class METADATA(Structure):
    _fields_ = [("classes", c_int),
                ("names", POINTER(c_char_p))]



#lib = CDLL("/home/pjreddie/documents/darknet/libdarknet.so", RTLD_GLOBAL)
lib = CDLL("libdarknet.so", RTLD_GLOBAL)
lib.network_width.argtypes = [c_void_p]
lib.network_width.restype = c_int
lib.network_height.argtypes = [c_void_p]
lib.network_height.restype = c_int

predict = lib.network_predict
predict.argtypes = [c_void_p, POINTER(c_float)]
predict.restype = POINTER(c_float)
```

```
set_gpu = lib.cuda_set_device
set_gpu.argtypes = [c_int]

make_image = lib.make_image
make_image.argtypes = [c_int, c_int, c_int]
make_image.restype = IMAGE

get_network_boxes = lib.get_network_boxes
get_network_boxes.argtypes = [c_void_p, c_int, c_int, c_float,
    c_float, POINTER(c_int), c_int, POINTER(c_int)]
get_network_boxes.restype = POINTER(DETECTION)

make_network_boxes = lib.make_network_boxes
make_network_boxes.argtypes = [c_void_p]
make_network_boxes.restype = POINTER(DETECTION)

free_detections = lib.free_detections
free_detections.argtypes = [POINTER(DETECTION), c_int]

free_ptrs = lib.free_ptrs
free_ptrs.argtypes = [POINTER(c_void_p), c_int]

network_predict = lib.network_predict
network_predict.argtypes = [c_void_p, POINTER(c_float)]

reset_rnn = lib.reset_rnn
reset_rnn.argtypes = [c_void_p]

load_net = lib.load_network
load_net.argtypes = [c_char_p, c_char_p, c_int]
load_net.restype = c_void_p

do_nms_obj = lib.do_nms_obj
do_nms_obj.argtypes = [POINTER(DETECTION), c_int, c_int,
```

```
    c_float]

do_nms_sort = lib.do_nms_sort
do_nms_sort.argtypes = [POINTER(DETECTION), c_int, c_int,
    c_float]

free_image = lib.free_image
free_image.argtypes = [IMAGE]

letterbox_image = lib.letterbox_image
letterbox_image.argtypes = [IMAGE, c_int, c_int]
letterbox_image.restype = IMAGE

load_meta = lib.get_metadata
lib.get_metadata.argtypes = [c_char_p]
lib.get_metadata.restype = METADATA

load_image = lib.load_image_color
load_image.argtypes = [c_char_p, c_int, c_int]
load_image.restype = IMAGE

rgbgr_image = lib.rgbgr_image
rgbgr_image.argtypes = [IMAGE]

predict_image = lib.network_predict_image
predict_image.argtypes = [c_void_p, IMAGE]
predict_image.restype = POINTER(c_float)

def classify(net, meta, im):
    out = predict_image(net, im)
    res = []
    for i in range(meta.classes):
        res.append((meta.names[i], out[i]))
    res = sorted(res, key=lambda x: -x[1])
    return res
```

```
def detect(net, meta, image, thresh=.5, hier_thresh=.5, nms=.45):
    im = load_image(image, 0, 0)
    num = c_int(0)
    pnum = pointer(num)
    predict_image(net, im)
    dets = get_network_boxes(net, im.w, im.h, thresh, hier_thresh, None, 0, pnum)
    num = pnum[0]
    if (nms): do_nms_obj(dets, num, meta.classes, nms);

    res = []
    for j in range(num):
        for i in range(meta.classes):
            if dets[j].prob[i] > 0:
                b = dets[j].bbox
                res.append((meta.names[i], dets[j].prob[i], (b.x, b.y, b.w, b.h)))
    res = sorted(res, key=lambda x: -x[1])
    free_image(im)
    free_detections(dets, num)
    return res

if __name__ == "__main__":
    #net = load_net("cfg/densenet201.cfg", "/home/pjreddie/trained/densenet201.weights", 0)
    #im = load_image("data/wolf.jpg", 0, 0)
    #meta = load_meta("cfg/imagenet1k.data")
    #r = classify(net, meta, im)
    #print r[:10]
    net = load_net("cfg/tiny-yolo.cfg", "tiny-yolo.weights", 0)
    meta = load_meta("cfg/coco.data")
    r = detect(net, meta, "data/dog.jpg")
    print r
```

A.2.2 Custom CFG File

```

[net]
#Testing
#batch=1
#subdivisions=1
#Training
batch=64
subdivisions=64
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=100
max_batches = 6000
policy=steps
steps=5400
scales=.1,.1

[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=leaky
```

```
# Downsample

[convolutional]
batch_normalize=1
filters=64
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=32
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=64
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
```

```
filters=128
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=leaky

[convolutional]
```

```
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=256
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
```

```
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
```

```
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=128
```

```
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
```

```
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
```

```
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=512
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
```

```
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
```

```
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
```

```
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear
```

```
[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
```

```
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
```

```
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
```

```
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear
######################
[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky
[convolutional]
```

```
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky
[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky
[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky
[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky
[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky
[convolutional]
```

```
size=1
stride=1
pad=1
filters=24
activation=linear
[yolo]
mask = 6,7,8
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,
    116,90,  156,198,  373,326
classes=3
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
[route]
layers = -4
[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky
[upsample]
stride=2
[route]
layers = -1, 61
[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky
```

```
[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky
[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky
[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky
[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky
[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky
```

```
[convolutional]
size=1
stride=1
pad=1
filters=24
activation=linear
[yolo]
mask = 3,4,5
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
classes=3
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
[route]
layers = -4
[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky
[upsample]
stride=2
[route]
layers = -1, 36
[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
```

```
activation=leaky
[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky
[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky
[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky
[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky
[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
```

```
activation=leaky
[convolutional]
size=1
stride=1
pad=1
filters=24
activation=linear
[yolo]
mask = 0,1,2
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,
    116,90,  156,198,  373,326
classes=3
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

A.3 SSD Training

```
model {
  ssd {
    num_classes: 3
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
```

```
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
      }
    }
    similarity_calculator {
      iou_similarity {
      }
    }
    anchor_generator {
      ssd_anchor_generator {
        num_layers: 6
        min_scale: 0.05
        max_scale: 0.95
        aspect_ratios: 1.0
        aspect_ratios: 2.0
        aspect_ratios: 0.5
        #aspect_ratios: 3.0
        #aspect_ratios: 0.3333
        reduce_boxes_in_lowest_layer: true
      }
    }
    image_resizer {
      fixed_shape_resizer {
        height: 300
        width: 300
      }
    }
    box_predictor {
      convolutional_box_predictor {
        min_depth: 0
        max_depth: 0
        num_layers_before_predictor: 0
        use_dropout: false
```

```
        dropout_keep_probability: 0.8
        kernel_size: 3
        box_code_size: 4
        apply_sigmoid_to_scores: false
        conv_hyperparams {
          activation: RELU_6,
          regularizer {
            l2_regularizer {
              weight: 0.00004
            }
          }
          initializer {
            truncated_normal_initializer {
              stddev: 0.03
              mean: 0.0
            }
          }
        }
      }
    }
    feature_extractor {
      type: 'ssd_inception_v2'
      min_depth: 16
      depth_multiplier: 1.0
      conv_hyperparams {
        activation: RELU_6,
        regularizer {
          l2_regularizer {
            weight: 0.00004
          }
        }
        initializer {
          truncated_normal_initializer {
            stddev: 0.03
            mean: 0.0
```

```
        }
      }
      batch_norm {
        train: true,
        scale: true,
        center: true,
        decay: 0.9997,
        epsilon: 0.001,
      }
    }
    override_base_feature_extractor_hyperparams: true
  }
  loss {
    classification_loss {
      weighted_sigmoid {
      }
    }
    localization_loss {
      weighted_smooth_l1 {
      }
    }
    hard_example_miner {
      num_hard_examples: 3000
      iou_threshold: 0.99
      loss_type: CLASSIFICATION
      max_negatives_per_positive: 3
      min_negatives_per_image: 0
    }
    classification_weight: 1.0
    localization_weight: 1.0
  }
  normalize_loss_by_num_matches: true
  post_processing {
    batch_non_max_suppression {
      score_threshold: 1e-8
```

```
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 100
      }
      score_converter: SIGMOID
    }
  }
}

train_config: {
  batch_size: 8
  optimizer {
    rms_prop_optimizer: {
      learning_rate: {
        exponential_decay_learning_rate {
          initial_learning_rate: 0.004
          decay_steps: 800720
          decay_factor: 0.95
        }
      }
      momentum_optimizer_value: 0.9
      decay: 0.9
      epsilon: 1.0
    }
  }
  fine_tune_checkpoint: "pre-trained-model/model.ckpt"
  from_detection_checkpoint: true
  # Note: The below line limits the training process to 200K steps, which we
  # empirically found to be sufficient enough to train the pets dataset. This
  # effectively bypasses the learning rate schedule (the learning rate will
  # never decay). Remove the below line to train indefinitely.
  num_steps: 200000
```

```
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    ssd_random_crop {
    }
  }
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "annotations/train.record"
  }
  label_map_path: "annotations/label_map.pbtxt"
}

eval_config: {
  #metrics_set: "coco_detection_metrics"
  num_examples: 62
  # Note: The below line limits the evaluation process to 10
    evaluations.
  # Remove the below line to evaluate indefinitely.
  max_evals: 100
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "annotations/test.record"
  }
  label_map_path: "annotations/label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

A.4 FR-CNN Training

```
from __future__ import division
import random
import pprint
import sys
import time
import numpy as np
from optparse import OptionParser
import pickle

from keras import backend as K
from keras.optimizers import Adam, SGD, RMSprop
from keras.layers import Input
from keras.models import Model
from keras_frcnn import config, data_generators
from keras_frcnn import losses as losses
import keras_frcnn.roi_helpers as roi_helpers
from keras.utils import generic_utils

sys.setrecursionlimit(40000)

parser = OptionParser()

parser.add_option("-p", "--path", dest="train_path", help="Path
    to training data.")
parser.add_option(
    "-o",
    "--parser",
    dest="parser",
    help="Parser to use. One of simple or pascal_voc",
    default="pascal_voc",
)
parser.add_option(
```

```
        "-n",
        "--num_rois",
        type="int",
        dest="num_rois",
        help="Number of RoIs to process at once.",
        default=32,
)
parser.add_option(
        "--network",
        dest="network",
        help="Base network to use. Supports vgg or resnet50.",
        default="resnet50",
)
parser.add_option(
        "--hf",
        dest="horizontal_flips",
        help="Augment with horizontal flips in training. (Default=
       false).",
        action="store_true",
        default=False,
)
parser.add_option(
        "--vf",
        dest="vertical_flips",
        help="Augment with vertical flips in training. (Default=
       false).",
        action="store_true",
        default=False,
)
parser.add_option(
        "--rot",
        "--rot_90",
        dest="rot_90",
        help="Augment with 90 degree rotations in training. (
       Default=false).",
```

```
    action="store_true",
    default=False,
)
parser.add_option(
    "--num_epochs",
    type="int",
    dest="num_epochs",
    help="Number of epochs.",
    default=2000,
)
parser.add_option(
    "--config_filename",
    dest="config_filename",
    help="Location to store all the metadata related to the
   training (to be used when testing).",
    default="config.pickle",
)
parser.add_option(
    "--output_weight_path",
    dest="output_weight_path",
    help="Output path for weights.",
    default="./model_frcnn.hdf5",
)
parser.add_option(
    "--input_weight_path",
    dest="input_weight_path",
    help="Input path for weights. If not specified, will try to
    load default weights provided by keras.",
)

(options, args) = parser.parse_args()

if not options.train_path:  # if filename is not given
    parser.error(
        "Error: path to training data must be specified. Pass
```

```
        --path to command line"
    )

if options.parser == "pascal_voc":
    from keras_frcnn.pascal_voc_parser import get_data
elif options.parser == "simple":
    from keras_frcnn.simple_parser import get_data
else:
    raise ValueError(
        "Command line option parser must be one of 'pascal_voc'
     or 'simple'"
    )

# pass the settings from the command line, and persist them in
    the config object
C = config.Config()

C.use_horizontal_flips = bool(options.horizontal_flips)
C.use_vertical_flips = bool(options.vertical_flips)
C.rot_90 = bool(options.rot_90)

C.model_path = options.output_weight_path
C.num_rois = int(options.num_rois)

if options.network == "vgg":
    C.network = "vgg"
    from keras_frcnn import vgg as nn
elif options.network == "resnet50":
    from keras_frcnn import resnet as nn

    C.network = "resnet50"
else:
    print("Not a valid model")
    raise ValueError

```

```python

# check if weight path was passed via command line
if options.input_weight_path:
    C.base_net_weights = options.input_weight_path
else:
    # set the path to weights based on backend and model
    C.base_net_weights = nn.get_weight_path()

all_imgs, classes_count, class_mapping = get_data(options.
    train_path)

if "bg" not in classes_count:
    classes_count["bg"] = 0
    class_mapping["bg"] = len(class_mapping)

C.class_mapping = class_mapping

inv_map = {v: k for k, v in class_mapping.items()}

print("Training images per class:")
pprint.pprint(classes_count)
print("Num classes (including bg) = {}".format(len(
    classes_count)))

config_output_filename = options.config_filename

with open(config_output_filename, "wb") as config_f:
    pickle.dump(C, config_f)
    print(
        "Config has been written to {}, and can be loaded when
    testing to ensure correct results".format(
            config_output_filename
        )
    )

```

```
random.shuffle(all_imgs)

num_imgs = len(all_imgs)

train_imgs = [s for s in all_imgs if s["imageset"] == "trainval
    "]
val_imgs = [s for s in all_imgs if s["imageset"] == "test"]

print("Num train samples {}".format(len(train_imgs)))
print("Num val samples {}".format(len(val_imgs)))


data_gen_train = data_generators.get_anchor_gt(
    train_imgs,
    classes_count,
    C,
    nn.get_img_output_length,
    K.image_dim_ordering(),
    mode="train",
)
data_gen_val = data_generators.get_anchor_gt(
    val_imgs,
    classes_count,
    C,
    nn.get_img_output_length,
    K.image_dim_ordering(),
    mode="val",
)

if K.image_dim_ordering() == "th":
    input_shape_img = (3, None, None)
else:
    input_shape_img = (None, None, 3)

img_input = Input(shape=input_shape_img)
```

```
roi_input = Input(shape=(None, 4))

# define the base network (resnet here, can be VGG, Inception,
    etc)
shared_layers = nn.nn_base(img_input, trainable=True)

# define the RPN, built on the base layers
num_anchors = len(C.anchor_box_scales) * len(C.
    anchor_box_ratios)
rpn = nn.rpn(shared_layers, num_anchors)

classifier = nn.classifier(
     shared_layers, roi_input, C.num_rois, nb_classes=len(
    classes_count), trainable=True
)

model_rpn = Model(img_input, rpn[:2])
model_classifier = Model([img_input, roi_input], classifier)

# this is a model that holds both the RPN and the classifier,
    used to load/save weights for the models
model_all = Model([img_input, roi_input], rpn[:2] + classifier)

try:
     print("loading weights from {}".format(C.base_net_weights))
     model_rpn.load_weights(C.base_net_weights, by_name=True)
     model_classifier.load_weights(C.base_net_weights, by_name=
    True)
except:
     print(
          "Could not load pretrained model weights. Weights can
    be found in the keras application folder \
     https://github.com/fchollet/keras/tree/master/keras/
    applications"
     )
```

```

optimizer = Adam(lr=1e-5)
optimizer_classifier = Adam(lr=1e-5)
model_rpn.compile(
    optimizer=optimizer,
    loss=[losses.rpn_loss_cls(num_anchors), losses.rpn_loss_regr(num_anchors)],
)
model_classifier.compile(
    optimizer=optimizer_classifier,
    loss=[losses.class_loss_cls, losses.class_loss_regr(len(classes_count) - 1)],
    metrics={"dense_class_{}".format(len(classes_count)): "accuracy"},
)
model_all.compile(optimizer="sgd", loss="mae")

epoch_length = 1000
num_epochs = int(options.num_epochs)
iter_num = 0

losses = np.zeros((epoch_length, 5))
rpn_accuracy_rpn_monitor = []
rpn_accuracy_for_epoch = []
start_time = time.time()

best_loss = np.Inf

class_mapping_inv = {v: k for k, v in class_mapping.items()}
print("Starting training")

vis = True

for epoch_num in range(num_epochs):

```

```
    progbar = generic_utils.Progbar(epoch_length)
    print("Epoch {}/{}".format(epoch_num + 1, num_epochs))

    while True:
        try:

            if len(rpn_accuracy_rpn_monitor) == epoch_length and C.verbose:
                mean_overlapping_bboxes = float(sum(rpn_accuracy_rpn_monitor)) / len(
                    rpn_accuracy_rpn_monitor
                )
                rpn_accuracy_rpn_monitor = []
                print(
                    "Average number of overlapping bounding boxes from RPN = {} for {} previous iterations".format(
                        mean_overlapping_bboxes, epoch_length
                    )
                )
                if mean_overlapping_bboxes == 0:
                    print(
                        "RPN is not producing bounding boxes that overlap the ground truth boxes. Check RPN settings or keep training."
                    )

            X, Y, img_data = next(data_gen_train)

            loss_rpn = model_rpn.train_on_batch(X, Y)

            P_rpn = model_rpn.predict_on_batch(X)

            R = roi_helpers.rpn_to_roi(
                P_rpn[0],
                P_rpn[1],
```

```
                C,
                K.image_dim_ordering(),
                use_regr=True,
                overlap_thresh=0.7,
                max_boxes=300,
            )
            # note: calc_iou converts from (x1,y1,x2,y2) to (x,y,w,h) format
            X2, Y1, Y2, IouS = roi_helpers.calc_iou(R, img_data, C, class_mapping)

            if X2 is None:
                rpn_accuracy_rpn_monitor.append(0)
                rpn_accuracy_for_epoch.append(0)
                continue

            neg_samples = np.where(Y1[0, :, -1] == 1)
            pos_samples = np.where(Y1[0, :, -1] == 0)

            if len(neg_samples) > 0:
                neg_samples = neg_samples[0]
            else:
                neg_samples = []

            if len(pos_samples) > 0:
                pos_samples = pos_samples[0]
            else:
                pos_samples = []

            rpn_accuracy_rpn_monitor.append(len(pos_samples))
            rpn_accuracy_for_epoch.append((len(pos_samples)))

            if C.num_rois > 1:
                if len(pos_samples) < C.num_rois // 2:
                    selected_pos_samples = pos_samples.tolist()
```

```
            else:
                selected_pos_samples = np.random.choice(
                    pos_samples, C.num_rois // 2, replace=False
                ).tolist()
            try:
                selected_neg_samples = np.random.choice(
                    neg_samples,
                    C.num_rois - len(selected_pos_samples),
                    replace=False,
                ).tolist()
            except:
                selected_neg_samples = np.random.choice(
                    neg_samples,
                    C.num_rois - len(selected_pos_samples),
                    replace=True,
                ).tolist()

            sel_samples = selected_pos_samples + selected_neg_samples
        else:
            # in the extreme case where num_rois = 1, we pick a random pos or neg sample
            selected_pos_samples = pos_samples.tolist()
            selected_neg_samples = neg_samples.tolist()
            if np.random.randint(0, 2):
                sel_samples = random.choice(neg_samples)
            else:
                sel_samples = random.choice(pos_samples)

        loss_class = model_classifier.train_on_batch(
            [X, X2[:, sel_samples, :]],
            [Y1[:, sel_samples, :], Y2[:, sel_samples, :]],
        )

```

```
            losses[iter_num, 0] = loss_rpn[1]
            losses[iter_num, 1] = loss_rpn[2]

            losses[iter_num, 2] = loss_class[1]
            losses[iter_num, 3] = loss_class[2]
            losses[iter_num, 4] = loss_class[3]

            progbar.update(
                iter_num + 1,
                [
                    ("rpn_cls", losses[iter_num, 0]),
                    ("rpn_regr", losses[iter_num, 1]),
                    ("detector_cls", losses[iter_num, 2]),
                    ("detector_regr", losses[iter_num, 3]),
                ],
            )

            iter_num += 1

            if iter_num == epoch_length:
                loss_rpn_cls = np.mean(losses[:, 0])
                loss_rpn_regr = np.mean(losses[:, 1])
                loss_class_cls = np.mean(losses[:, 2])
                loss_class_regr = np.mean(losses[:, 3])
                class_acc = np.mean(losses[:, 4])

                mean_overlapping_bboxes = float(sum(rpn_accuracy_for_epoch)) / len(
                    rpn_accuracy_for_epoch
                )
                rpn_accuracy_for_epoch = []

                if C.verbose:
                    print(
                        "Mean number of bounding boxes from RPN
```

```
 overlapping ground truth boxes: {}".format(
                            mean_overlapping_bboxes
                        )
                    )
                    print(
                        "Classifier accuracy for bounding boxes from RPN: {}".format(
                            class_acc
                        )
                    )
                    print("Loss RPN classifier: {}".format(loss_rpn_cls))
                    print("Loss RPN regression: {}".format(loss_rpn_regr))
                    print("Loss Detector classifier: {}".format(loss_class_cls))
                    print("Loss Detector regression: {}".format(loss_class_regr))
                    print("Elapsed time: {}".format(time.time() - start_time))

                curr_loss = (
                    loss_rpn_cls + loss_rpn_regr + loss_class_cls + loss_class_regr
                )
                iter_num = 0
                start_time = time.time()

                if curr_loss < best_loss:
                    if C.verbose:
                        print(
                            "Total loss decreased from {} to {}, saving weights".format(
                                best_loss, curr_loss
                            )
```

```
                        )
                    best_loss = curr_loss
                    model_all.save_weights(C.model_path)

                break

        except Exception as e:
            print("Exception: {}".format(e))
            continue

print("Training complete, exiting.")
```

A.5 Evaluation

```
import os
import cv2
import numpy as np
import sys
import pickle
from optparse import OptionParser
import time
from keras_frcnn import config
import keras_frcnn.resnet as nn
from keras import backend as K
from keras.layers import Input
from keras.models import Model
from keras_frcnn import roi_helpers
from keras_frcnn import data_generators
from sklearn.metrics import average_precision_score


def get_map(pred, gt, f):
  T = {}
  P = {}
```

```
fx, fy = f

for bbox in gt:
  bbox['bbox_matched'] = False

pred_probs = np.array([s['prob'] for s in pred])
box_idx_sorted_by_prob = np.argsort(pred_probs)[::-1]

for box_idx in box_idx_sorted_by_prob:
  pred_box = pred[box_idx]
  pred_class = pred_box['class']
  pred_x1 = pred_box['x1']
  pred_x2 = pred_box['x2']
  pred_y1 = pred_box['y1']
  pred_y2 = pred_box['y2']
  pred_prob = pred_box['prob']
  if pred_class not in P:
    P[pred_class] = []
    T[pred_class] = []
  P[pred_class].append(pred_prob)
  found_match = False

  for gt_box in gt:
    gt_class = gt_box['class']
    gt_x1 = gt_box['x1']/fx
    gt_x2 = gt_box['x2']/fx
    gt_y1 = gt_box['y1']/fy
    gt_y2 = gt_box['y2']/fy
    gt_seen = gt_box['bbox_matched']
    if gt_class != pred_class:
      continue
    if gt_seen:
      continue
    iou = data_generators.iou((pred_x1, pred_y1, pred_x2,
 pred_y2), (gt_x1, gt_y1, gt_x2, gt_y2))
```

```
      if iou >= 0.5:
        found_match = True
        gt_box['bbox_matched'] = True
        break
      else:
        continue

    T[pred_class].append(int(found_match))

  for gt_box in gt:
    if not gt_box['bbox_matched'] and not gt_box['difficult']:
      if gt_box['class'] not in P:
        P[gt_box['class']] = []
        T[gt_box['class']] = []

      T[gt_box['class']].append(1)
      P[gt_box['class']].append(0)

  #import pdb
  #pdb.set_trace()
  return T, P

sys.setrecursionlimit(40000)

parser = OptionParser()

parser.add_option("-p", "--path", dest="test_path", help="Path to test data.")
parser.add_option("-n", "--num_rois", dest="num_rois",
        help="Number of ROIs per iteration. Higher means more memory use.", default=32)
parser.add_option("--config_filename", dest="config_filename", help=
        "Location to read the metadata related to the training (generated when training).",
```

```
        default="config.pickle")
parser.add_option("-o", "--parser", dest="parser", help="Parser
    to use. One of simple or pascal_voc",
        default="pascal_voc"),

(options, args) = parser.parse_args()

if not options.test_path:   # if filename is not given
  parser.error('Error: path to test data must be specified.
   Pass --path to command line')


if options.parser == 'pascal_voc':
  from keras_frcnn.pascal_voc_parser import get_data
elif options.parser == 'simple':
  from keras_frcnn.simple_parser import get_data
else:
  raise ValueError("Command line option parser must be one of '
   pascal_voc' or 'simple'")

config_output_filename = options.config_filename

with open(config_output_filename, 'r') as f_in:
  C = pickle.load(f_in)

# turn off any data augmentation at test time
C.use_horizontal_flips = False
C.use_vertical_flips = False
C.rot_90 = False

img_path = options.test_path


def format_img(img, C):
  img_min_side = float(C.im_size)
```

```
  (height,width,_) = img.shape

  if width <= height:
    f = img_min_side/width
    new_height = int(f * height)
    new_width = int(img_min_side)
  else:
    f = img_min_side/height
    new_width = int(f * width)
    new_height = int(img_min_side)
  fx = width/float(new_width)
  fy = height/float(new_height)
  img = cv2.resize(img, (new_width, new_height), interpolation=cv2.INTER_CUBIC)
  img = img[:, :, (2, 1, 0)]
  img = img.astype(np.float32)
  img[:, :, 0] -= C.img_channel_mean[0]
  img[:, :, 1] -= C.img_channel_mean[1]
  img[:, :, 2] -= C.img_channel_mean[2]
  img /= C.img_scaling_factor
  img = np.transpose(img, (2, 0, 1))
  img = np.expand_dims(img, axis=0)
  return img, fx, fy


class_mapping = C.class_mapping

if ’bg’ not in class_mapping:
  class_mapping[’bg’] = len(class_mapping)

class_mapping = {v: k for k, v in class_mapping.iteritems()}
print(class_mapping)
class_to_color = {class_mapping[v]: np.random.randint(0, 255, 3) for v in class_mapping}
C.num_rois = int(options.num_rois)
```

```

if K.image_dim_ordering() == 'th':
  input_shape_img = (3, None, None)
  input_shape_features = (1024, None, None)
else:
  input_shape_img = (None, None, 3)
  input_shape_features = (None, None, 1024)


img_input = Input(shape=input_shape_img)
roi_input = Input(shape=(C.num_rois, 4))
feature_map_input = Input(shape=input_shape_features)

# define the base network (resnet here, can be VGG, Inception,
    etc)
shared_layers = nn.nn_base(img_input, trainable=True)

# define the RPN, built on the base layers
num_anchors = len(C.anchor_box_scales) * len(C.
    anchor_box_ratios)
rpn_layers = nn.rpn(shared_layers, num_anchors)

classifier = nn.classifier(feature_map_input, roi_input, C.
    num_rois, nb_classes=len(class_mapping), trainable=True)

model_rpn = Model(img_input, rpn_layers)
model_classifier_only = Model([feature_map_input, roi_input],
    classifier)

model_classifier = Model([feature_map_input, roi_input],
    classifier)

model_rpn.load_weights(C.model_path, by_name=True)
model_classifier.load_weights(C.model_path, by_name=True)

```

```
model_rpn.compile(optimizer='sgd', loss='mse')
model_classifier.compile(optimizer='sgd', loss='mse')

all_imgs, _, _ = get_data(options.test_path)
test_imgs = [s for s in all_imgs if s['imageset'] == 'test']


T = {}
P = {}
for idx, img_data in enumerate(test_imgs):
  print('{}/{}'.format(idx,len(test_imgs)))
  st = time.time()
  filepath = img_data['filepath']

  img = cv2.imread(filepath)

  X, fx, fy = format_img(img, C)

  if K.image_dim_ordering() == 'tf':
    X = np.transpose(X, (0, 2, 3, 1))

  # get the feature maps and output from the RPN
  [Y1, Y2, F] = model_rpn.predict(X)

  R = roi_helpers.rpn_to_roi(Y1, Y2, C, K.image_dim_ordering(),
    overlap_thresh=0.7)

  # convert from (x1,y1,x2,y2) to (x,y,w,h)
  R[:, 2] -= R[:, 0]
  R[:, 3] -= R[:, 1]

  # apply the spatial pyramid pooling to the proposed regions
  bboxes = {}
  probs = {}

```

```
for jk in range(R.shape[0] // C.num_rois + 1):
  ROIs = np.expand_dims(R[C.num_rois * jk:C.num_rois * (jk +
 1), :], axis=0)
  if ROIs.shape[1] == 0:
    break

  if jk == R.shape[0] // C.num_rois:
    # pad R
    curr_shape = ROIs.shape
    target_shape = (curr_shape[0], C.num_rois, curr_shape[2])
    ROIs_padded = np.zeros(target_shape).astype(ROIs.dtype)
    ROIs_padded[:, :curr_shape[1], :] = ROIs
    ROIs_padded[0, curr_shape[1]:, :] = ROIs[0, 0, :]
    ROIs = ROIs_padded

  [P_cls, P_regr] = model_classifier_only.predict([F, ROIs])

  for ii in range(P_cls.shape[1]):

    if np.argmax(P_cls[0, ii, :]) == (P_cls.shape[2] - 1):
      continue

    cls_name = class_mapping[np.argmax(P_cls[0, ii, :])]

    if cls_name not in bboxes:
      bboxes[cls_name] = []
      probs[cls_name] = []

    (x, y, w, h) = ROIs[0, ii, :]

    cls_num = np.argmax(P_cls[0, ii, :])
    try:
      (tx, ty, tw, th) = P_regr[0, ii, 4 * cls_num:4 * (
 cls_num + 1)]
      tx /= C.classifier_regr_std[0]
```

```
      ty /= C.classifier_regr_std[1]
      tw /= C.classifier_regr_std[2]
      th /= C.classifier_regr_std[3]
      x, y, w, h = roi_helpers.apply_regr(x, y, w, h, tx, ty, tw, th)
    except:
      pass
    bboxes[cls_name].append([16 * x, 16 * y, 16 * (x + w), 16 * (y + h)])
    probs[cls_name].append(np.max(P_cls[0, ii, :]))

all_dets = []

for key in bboxes:
  bbox = np.array(bboxes[key])

  new_boxes, new_probs = roi_helpers.non_max_suppression_fast(bbox, np.array(probs[key]), overlap_thresh=0.5)
  for jk in range(new_boxes.shape[0]):
    (x1, y1, x2, y2) = new_boxes[jk, :]
    det = {'x1': x1, 'x2': x2, 'y1': y1, 'y2': y2, 'class': key, 'prob': new_probs[jk]}
    all_dets.append(det)


print('Elapsed time = {}'.format(time.time() - st))
t, p = get_map(all_dets, img_data['bboxes'], (fx, fy))
for key in t.keys():
  if key not in T:
    T[key] = []
    P[key] = []
  T[key].extend(t[key])
  P[key].extend(p[key])
all_aps = []
for key in T.keys():
```

```
    ap = average_precision_score(T[key], P[key])
    print('{} AP: {}'.format(key, ap))
    all_aps.append(ap)
  print('mAP = {}'.format(np.mean(np.array(all_aps))))
  #print(T)
  #print(P)
measure_map.py
Displaying train_frcnn.py.
```

REFERENCES

[1] Vasefi, Fartash Kemp, William MacKinnon, Nicholas Amini, Mohammad Valdebran, Manuel Huang, Kevin Zhang, Haomiao. (2018). “Automated facial acne assessment from smartphone images.” 22. 10.1117/12.2292506.

[2] T. Chantharaphaichi, B. Uyyanonvara, C. Sinthanayothin and A. Nishihara, "Automatic acne detection for medical treatment," 2015 6th International Conference of Information and Communication Technology for Embedded Systems (IC-ICTES), Hua-Hin, 2015, pp. 1-6, doi: 10.1109/ICTEmSys.2015.7110813.5

[3] Thanapha Chantharaphaichi, Bunyarit Uyyanonvara et al., “Automatic Acne Detection With Featured Bayesian Classifier For Medical Treatment”, International MultiConference BIM 2015

[4] C. Chang and H. Liao, "Automatic Facial Spots and Acnes Detection System," Journal of Cosmetics, Dermatological Sciences and Applications, Vol. 3 No. 1A, 2013, pp. 28-35. doi: 10.4236/jcdsa.2013.31A006.

[5] Lim ZV, Akram F, Ngo CP, et al., “Automated grading of acne vulgaris by deep learning with convolutional neural networks.”, Skin Res Technol. 2020;26(2):187-192. doi:10.1111/srt.12794

[6] Zhao, Tingting Zhang, Hang Spoelstra, Jacob. (2019). “A Computer Vision Application for Assessing Facial Acne Severity from Selfie Images.” arXiv:1907.07901

[7] Shen, X., Zhang, J., Yan, C. et al. “An Automatic Diagnosis Method of Facial Acne Vulgaris Based on Convolutional Neural Network”. Sci Rep 8, 5839 (2018). https://doi.org/10.1038/s41598-018-24204-6

[8] Ren S., He K., Girshick R. and Sun J. 2015 Faster R-CNN: Towards real-time object detection with region proposal networks Advances in neural information processing systems 91-99

[9] Redmon J., Divvala S., Girshick R. and Farhadi A. 2016 You only look once: Unified, real-time object detection Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 779-788

[10] Liu, Wei et al. "SSD: Single Shot MultiBox Detector." Lecture Notes in Computer Science (2016): 21–37. Crossref. Web.