YOPO - You Only Pose Once

Final Report for CS39440 Major Project

Author: Richard Price-Jones (rij12@aber.ac.uk)
Supervisor: Dr./Prof. Bernie Tiddeman (bpt@aber.ac.uk)

28th April 2018
Version: 1.0 (Release)

This report was submitted as partial fulfilment of a BSc degree in
Computer Science (G401)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name: Richard Price-Jones

Date: 28th April 2018

Consent to share this work

By including my name below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name: Richard Price-Jones

Date: 28th April 2018

Acknowledgements

I would like to thank Bernie Tiddeman for the guidance and mentoring throughout the project. I'm also very grateful to my parents for supporting me throughout my time at university.

Abstract

Recent advances in convolutional neural networks have shown excellent results in a number of computer vision problems. YOPO (You Only Pose Once) attempts to adapt the YOLO (You Only Look Once) algorithm [1] for human pose estimation. YOPO takes YOLO's regression object detection solution which currently uses four parameters to define a bounding box (x, y, w, h), and adds a new parameter to the network, an angle θ. This gives predicted bounding boxes an orientation which when trained on limbs produces an object classification method that can predict bounding boxes with an orientation for each limb.

Changes to the loss function have been made to calculate the area of the intersection between two rotated rectangles, so a much more accurate IOU (Intersection over Union) can be calculated for YOPO. Finally, the network also has a new output for a given bounding box (x, y, w, h, θ, C), where the first five parameters describe the bounding box position and the final parameter C is the conditional class probability, where given an object inside the cell how likely is it to be a given class.

An over-fitted model proved that the software was working and the network outputted bounding boxes with orientation. This model provided input an image with a pose skeleton using bounding boxes. Unfortunately when tested with a large subset of the MPII dataset it failed to produce any meaningful results. I believe additional training of the network with a large dataset is needed for a more accurate assessment of the YOPO network.

CONTENTS

LIST OF FIGURES

Chapter 1

Background & Objectives

1.1 Background

This project adapts an existing piece of research called You Only Look Once (YOLO) [1], with the idea to adapt the network so that as well as object detection, it can also perform human pose estimation. Human pose estimation is defined as the problem of the localisation of human joints [10]. YOLO is a fast object detection network, that splits the input image into a grid, and then predicts a number of bounding boxes for each cell of the grid. The network predicts the centre point, width and height of every bounding box. Finally the best box for each cell that is above a defined threshold is drawn on the image.

Object detection is the process of finding a target object in a given input image and assigning a predicted class such as a face or a dog. The main advantage of YOLO is that it achieves high accuracy while also being able to run in real-time. This gives YOLO many applications such as use with self driving cars, drones and many more. In the YOLO paper it claims to run at 45 FPS(Frames Per Second) on a Nvidia Titan X [1].

Unlike other detection methods such Deformable parts model [11] that uses a sliding window, which means it views an input image many times while moving a window across the image. It also uses static features for detecting objects, this method is very slow and the YOLO paper claims that YOLO's network is faster and more accurate [1].

YOPO adds an extra parameter to the network θ and trains on a set of given limbs with the aim to produce bounding boxes that have angle θ, which describes the pose of the predicted limb. Finally, using the predicted limbs in the image YOPO creates a 2D pose skeleton for each person in the image to estimate their pose.

1.1.1 Convolution Neural Networks

Traditionally a neural network's input neurons are all connected to the hidden layer [12], this is known as a perceptron network, see Figure 1.1. If an RGB image with a height of 300 pixels and a width of 500 pixels was given as input to the network, it would have 450,000($300 * 500 * 3$) input neurons. Additionally, each neuron is connected to every other neuron in the next layer, so with one hidden layer that has 10 neurons in it's layer, it would yield 4,500,000($450000 * 10$) connections between the neurons, which are the weights.

This is a massive amount of inputs which makes the network very computationally expensive to train, and does not scale well with larger images.

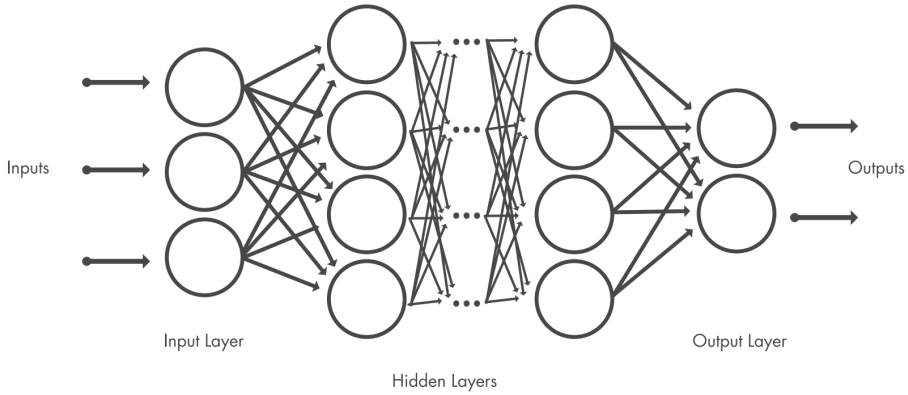


Figure 1.1: Example of a perceptron network [2]

Convolutional neural networks [13] like the perceptron model have an input layer and an output layer with many hidden layers in-between them. Each Layer can perform one of the following operations: convolution, pooling or activation. See Figure 1.2 for an convolutional neural network that can detect a car in an image.

A convolution layer puts the image through a set of convolutional filters, each of which activates certain features from the input image. For example; a filter might activate on a line in an early step of the network, or near the end of the network a filter might activate on a more complex shape. A convolutional neural network uses end to end learning which means that the features that the filters activate on are learned during training, rather than manually set.

Once the image has ran through the network the output of the convolutional layers is a multi-dimensional array (tensor), which is flattened and becomes the input to the fully connected layer. Probabilities are then calculated from the fully connected layer, using a softmax function, which takes each input and squashes it between 0 and 1. It divides each output such that the total sum of the outputs is equal to 1. This has the effect of creating a probability distribution that shows what each class probability scores are.

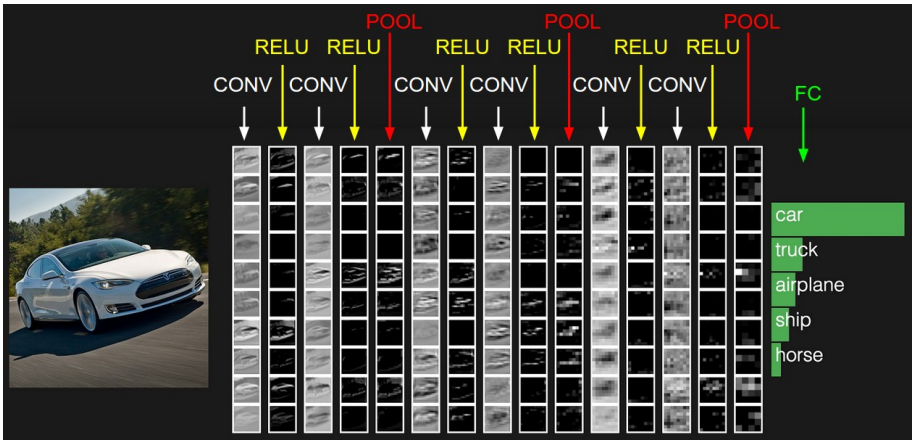


Figure 1.2: Example of a convolutional network detecting a car [3]

1.1.2 Data sets

The MPII human pose dataset [7] is a massive dataset containing 410,000 human activities, 40,000 people and 25,000 images. With each image given an activity label and the position of: right ankle, right knee, right hip, left hip, left knee, left ankle, pelvis, thorax, upper neck, head, right wrist, right elbow, right shoulder, left shoulder, left elbow, left wrist. The coordinates for each joint is given as a centre point of that joint (x,y), which is stored in a MatLab [2] data file format. The documentation of the dataset was full and complete with plentiful metadata should it be needed for the preprocessing such as: scale of the pose in the image, visibility of joints or even activity class if YOPO was expanded from the original requirements of human pose estimation.

The visual Geometry group [14] have a dataset that's based on a popular US TV show that has 499 images which is a small dataset, which they discuss in their paper about evaluation of human pose estimators [15]. The dataset provides a good range of poses and in different settings, however only the upper body was labelled (head, torso, upper and lower arms) which isn't suitable for a complete pose to be estimated.

The Leeds Sports Pose dataset [16] contains 2000 human pose images that have full body annotations which include 14 joint locations: right ankle, right knee, right hip, left hip, left knee, left ankle, right wrist, right elbow, right shoulder, left shoulder, left elbow, left wrist, neck, head top. Like the other datasets the ground truth was in a MatLab data file structure. The ground truth is predefined bounding boxes that describe target classes in an image, for example a ground truth bounding of a person would be a box that surrounds the person, where box is defined as (x, y, w, h). The visibility of the joint is also available in the metadata provided.

The MPII Human Pose dataset has been selected because this dataset had more documentation and metadata than the other datasets reviewed. However, the Leeds Sports Pose dataset would have been suitable as well.

1.1.3 YOLO

YOLO is a fast convolution neural network, it treats object detection as a regression problem. It divides the image into a S by S grid, the default being a 7 by 7 grid that splits the image into cells. It uses a single neural network to predict the bounding boxes and class probabilities directly from the input images, using only a single evaluation for each cell [1].

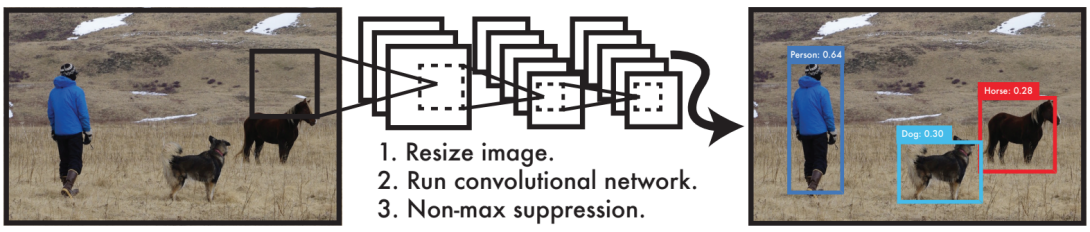


Figure 1.3: YOLO Overview [1]

An example of how YOLO works can be seem in Figure 1.4, the image has been divided into a grid that is shown in red. The centre of the object has fallen into the cell that is highlighted in blue,

that cell is then responsible for predicting that object. The green boxes are the predicted bounding boxes from the network.

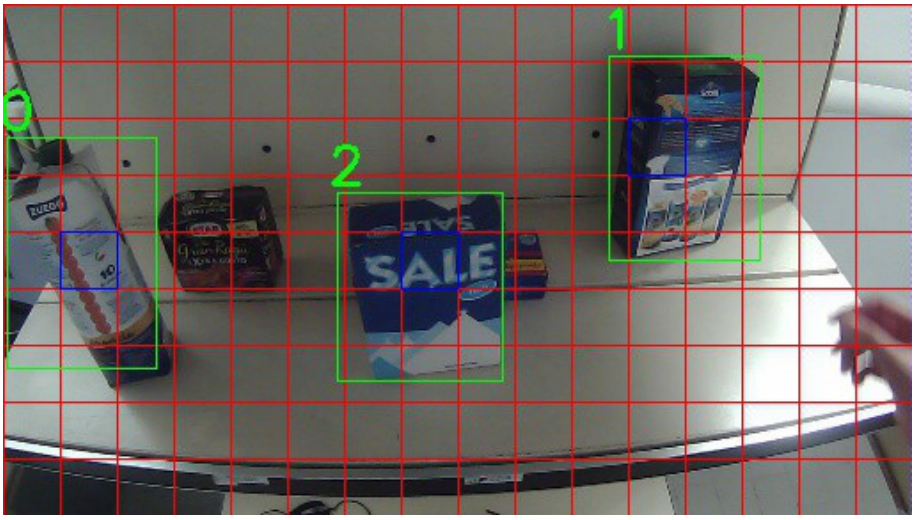


Figure 1.4: YOLO classifying a image [4]

1.2 Analysis

After doing some background research it is clear that before any research and development of the network can begin, a preprocessing step must be completed first. As mentioned above, the MPII dataset only has a centre point of a joint, whereas the YOLO network requires a bounding box for the ground truth of each class. A bounding box is made up of 4 parameters; a centre point (x, y), width (w) and height(h). The original paper [1] discusses that YOLO has a low detection rate of small objects relative to the input image, this is more apparent when there are small objects that are clustered together, such as a flock of birds. This is something that will need to be taken into consideration as human joints are relatively small compared the complete body, and by extension the whole image.

The MPII dataset provides human poses with ground truth describing their joints, chest and head position. YOPO will require the data in a more useable format for the training of the network, i.e the described bounding box format (x, y, w, h).

Currently the YOLO network only works with upright bounding boxes, and as human poses have many different configurations as well as rotations, the network will need to be changed so that it handles the detection of limbs or joints that are not upright/axis aligned. This means potentially there a lot of changes to the current implementation of YOLO that are going to be required.

A technique called transfer learning will be utilised in this project, this takes pre-trained weights from an already trained network. In YOPO's case it will be using the YOLO weights that have been trained using the Imagenet [17] dataset.

1.2.1 Objectives

- FR-1 Preprocess the MPII dataset in the required format for YOPO.
- FR-2 To adapt the YOLO network to perform human pose estimation which shall be known as the YOPO network.
- FR-3 Generate a weight file for the YOPO network, to enable the detection of human poses by the YOPO network.

1.2.2 Deliverables

- Source code of YOPO, preprocessing and YOPO network.
- Project report that discusses the solution, method and critical evaluation.

1.3 Research Method and Software Process

1.3.1 Methodology

An agile approach was chosen for the development and research of this project, mainly because the project is fairly open ended. Although the end objective is clear, the steps to achieve them are very much unclear, it would seem that the agile development process will give more flexibility over a plan driven approach such as waterfall. Due to being a one person team, the agile development process will be adjusted by using the weekly meeting with my supervisor as the sprint retrospective and the sprint planning at the same time. Where the development and current progress will be discussed, and what is going to be to achieved in the next week. Each sprint will be two weeks long. An online tool called JIRA [18] made by Atlassian will be used to manage the agile process, which includes the scrum board along with its ordered backlog.

1.3.2 Development Environment

Git will be used for the version control hosted on GitHub for both my report and software project. This is acting as a remote backup for all my work as well as my version control. While implementing the project and during research code breaking changes will most likely be implemented so it will be very helpful that the source can be committed at a working state and if needed the source code can be restored back to a working state. GitHub also integrates nicely with my continuous integration tool Travis CI [19]. Travis CI builds and runs all unit tests on each commit and pull request, as well as performing a static analysis on the code using a cloud service called Sonarqube [20]. Sonarqube will generate a report based on the quality of the code.

The original framework for YOLO is called Darknet [6] and is written in C and Cuda [21]. Due to a lack of experience in C or Cuda, this project has elected to use an open source TensorFlow [22] implementation called Darkflow (`https://github.com/thtrieu/darkflow`) which is written in Python.

Training a convolutional neural network can require a lot of time and computing resources. The development machine, will also be used for training the network, it has an Intel core i7 with 4

cores with 8 threads clocked at 4.5Ghz, 32GB of RAM (Random Access Memory) and a Nvidia GTX 1070 GPU (Graphics Processing Unit) with 8GB of VRAM. In addition to this Google cloud [10] services will be utilised during the training and development process, as they generously provide $300 free credit to students for the use of their VM (Virtual Machine) instances. The Google VM instance that will be selected has an Intel core i7 with 4 cores with 8 threads, 16GB of RAM, Nvidia Tesla k80 with 12GB of video memory and a 100GB SSD(Solid-State Drive).

1.4 Motivation

I was exposed to computer vision during my internship, where I worked for a company that specialised in image based software, and tackled very large and difficult image based problems. You Only Pose Once looked like a interesting project where I could apply my previous skills and learn a lot about computer vision during the process, furthermore with deep learning being such a hot topic at the moment I thought that doing a deep learning based major project would hopefully give me a entry route into the computer vision and deep learning industry.

Chapter 2

Experiment Methods

2.1 Introduction

This project is attempting to answer the question: with changes to the YOLO network can it detect human poses in a 2D image? This new network shall be known as the 'YOPO Network'.

2.2 Problem Overview

YOLO can only perform object detection in its current implementation. YOPO proposes to adapt the YOLO network so that it can detect human poses, by using new training data supplied of angled human limbs, parameter changes to the network, and significant changes to the training function. A successfully trained network will detect limbs in the image with the correct orientation. A post-processing step then decreases the bounding box's width to a few pixels and then draws the joints as small points on the image that connect the limbs together, this should yield a pose skeleton Figure 2.4. See the following changes to be made:

- Dataset
- Network Configuration
- Tensor sizes and YOLO preprocessing to handle new parameters
- IOU function so that it can calculate the intersection and union areas of two bounding boxes with different sizes and rotations.

2.2.1 Proposed Outcome

With the changes suggested YOPO should be able to perform human pose estimation instead of object detection, see figure Figure 2.1 and Figure 2.5.

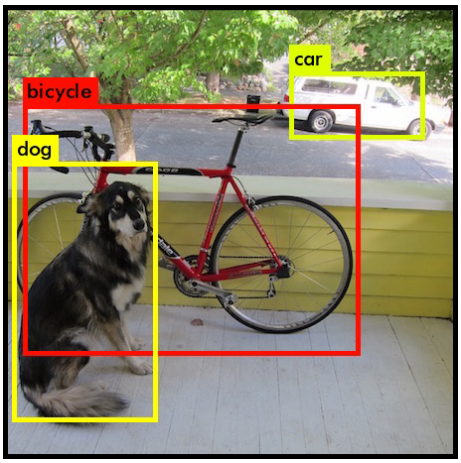


Figure 2.1: YOLO default trained on VOC dataset [5], image [6]

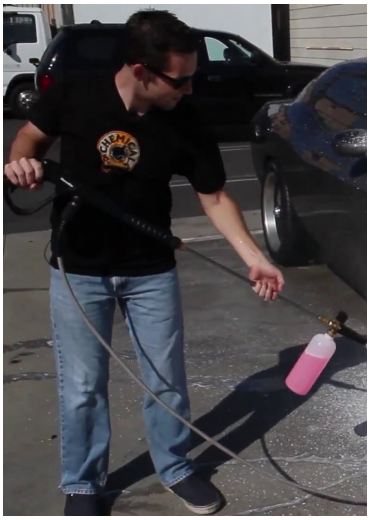


Figure 2.2: Input image

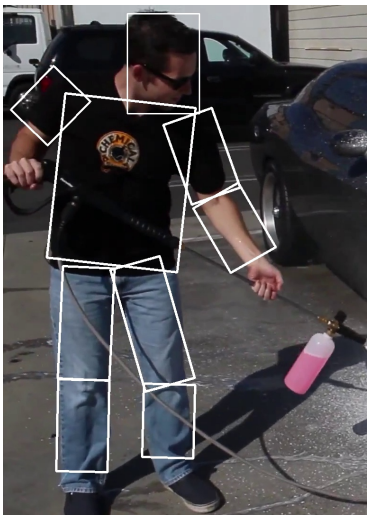


Figure 2.3: Network output

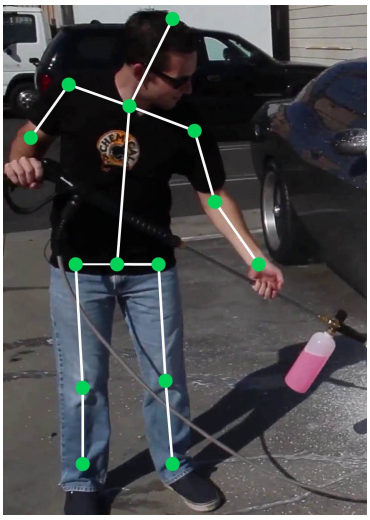


Figure 2.4: Post-processing

Figure 2.5: YOPO target output, image [7]

2.3 YOLO - You Only Look Once

Currently YOLO only outputs bounding boxes that are upright or axis aligned to a grid. This provides no value for trying to estimate a pose, because the orientation of the predicted bounding box needs to be known, so that the pose of the limb can be worked out. See Figure 2.1 for the default network output trained on 20 classes, three of which can be seen in the figure.

2.3.1 TensorFlow

TensorFlow is a numerical computational open source library that builds an abstract graph that has many operations that make up a model. Each operation is a node in the graph and then inputs tensors flow through the graph between each operation, hence the name TensorFlow. Finally, once all the operations have been applied the tensor is returned.

2.3.2 Darkflow

As mentioned in chapter one, a framework called Darkflow written in Python using TensorFlow will be used. It's based on the original framework used in the YOLO paper [1] called Darknet written in C and Cuda. The framework provides a method for running any neural network and in particular using one or many GPUs to decrease training time.

2.3.3 Current Configuration

YOLO's network configuration changes depending on what dataset you would like to train on. The default dataset YOLO uses is called VOC [5] which has 20 objects that YOLO learns to classify. The network has an input tensor of $S * S * (B * 5 * class\_num)$ where S is the grid cell size and B is the amount of bounding boxes per cell, the constant five is the bounding box parameters (x, y, w, h, C) and finally class_num is the number of classes that you want the network to learn.

Currently YOLO works by getting a batch of images as input passing them through each layer in the network, and then finally making a prediction at the fully connected layer. It then evaluates the error in that prediction using a modified version of sum-squared error [1].

The current loss function implemented in Darkflow uses two tensors of size ($S * S$, B, 4), GT the ground truth tensor and then net_out tensor which contains the network predictions for each cell. YOLO then finds the best box available for every cell using a IOU evaluation metric. The IOU is calculated by the area of the intersection divided by the area of the union between the two bounding boxes see Figure 2.6. The IOU score is multiplied with box confidence score C, this confidence score is generated along with the bounding boxes and it simply measures the probability that an object is inside the box and how well it fits inside. This means that if there are no objects inside the cell then the B bounding boxes all have a C of zero.

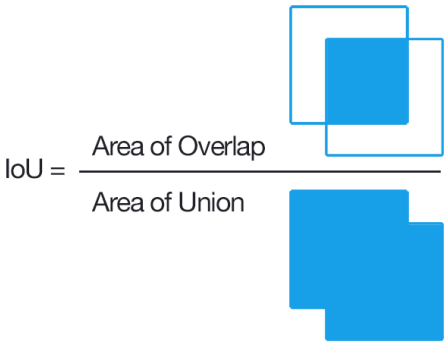


Figure 2.6: Intersection of Union is the area of overlap between the bounding boxes by the area of union [8]

2.4 Proposed Changes

By implementing the changes proposed, the new YOPO network should be able to classify orientated limbs by learning how to predict bounding boxes that have a centre (x, y), width w, height h and angle θ.

2.4.1 Dataset

As mentioned in chapter 1, YOPO uses the MPII dataset, using this the YOPO network is going learn to detect limbs and their orientation. YOPO pre-processes the MPII dataset to give bounding boxes with top left, bottom right and angle. The output of this processing is an XML(Extensible Markup Language) file describing the bounding boxes of each class in the image, along with the images' original height and width, see section 3.2 for an example of an XML ground truth file that is fed into the network.

2.4.2 YOPO Network configuration

YOLO has a configuration file (see section 3.1) that describes the network parameters, and all the layers in the network: convolutional, activation, max pooling and the fully connected. Normally only the fully connected layer requires a change depending on the dataset YOLO is going to be trained on. However because an extra bounding box parameter θ is needed for the orientation of the bounding box, a larger grid at the preprocessing step will be needed so there is less probability that smaller objects such as limbs aren't going to be in the same cell. Cells can only have one classification object type so if multiple objects are in the same cell then the object with the highest IOU with the ground truth will be selected to train on, the other objects will be ignored.

The following changes are proposed to the network configuration:

- classes = 10 (10 limbs)
- coords = 5 (x,y,w,h,θ)
- side = 16 (16^2 cells for 256 total cells)
- Last layer in the network, full connected output = 7168

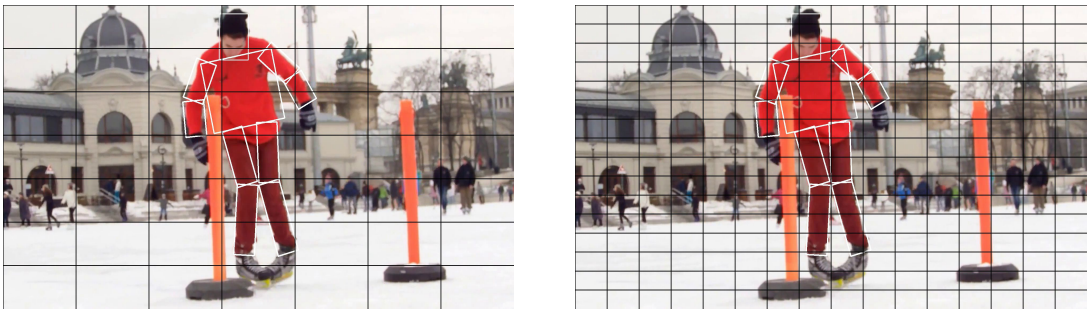
The last layer in the network is the flatten size of the output tensor and is calculated by:

$$S * S * (B * B_p + C_n)$$

Where S is the number of cells of given side of the image, B is the number bounding boxes per cell, B_p is the number of parameters that a bounding box has (x, y, w, h, θ, C), and finally C_n the number of classes you want YOLO to able to classify. Therefore the output tensor size is:

$$16^2 * (3 * 6 + 10) = 7168$$

The 7 by 7 is a grid size normally used for object detection such as a person or a car etc. Human limbs are much smaller than this so a much finer grid is needed, because a single cell predicts B boxes and each cell can only be assigned one class therefore if more than one of the centres of a limb is inside the same cell only the limb with the highest IOU will be picked.



(a) 7 by 7 Grid cell used for object detection

(b) 16 by 16 Grid cell used for limb detection

Figure 2.7: Comparison of grid selection images from MPII [7]

2.4.3 IOU Calculations

YOLO's original IOU calculations will be replaced with a new IOU function that is a TensorFlow Python operation. It will take four tensors:

- image_tensor - Tensor that holds current image metadata
- ground_truth_batch - contains all the ground truth bounding boxes for a batch of images
- net_out_batch - Output from network containing all the predicted bounding boxes for a batch of images
- IOU - with size $(S * S) * B$ holds the IOU score for each box

The new IOU algorithm takes the ground truth and network output tensor mentioned above, performs clipping on both bounding boxes and returns a shape which is defined as a list of points that describe the vertices of the new shape. This is used to find both the intersection and union shapes. Then using Green's theorem [23] we find an area with all the points of the clipped shape. Finally, the IOU is calculated from theses two areas, see Figure 2.6. The IOU scores are added into the IOU tensor which is empty, but they are exactly the same size as the ground truth and network output tensors. This means that the IOU score will have exactly the same index as the two clipping boxes in the ground truth and network output tensors that are used to calculate it. See below (1) for the algorithm overview.

Algorithm 1 YOPO IOU function

```
for each ground_truth_batch, net_out_batch in groundTruth_tensor, netOut_tensor do
    for each ground_truth_cell, net_out_cell in ground_truth_cells, net_out_cells do
        for each truth_box, net_out_box in ground_truth_boxes, net_out_boxes do
            Clip truth_box and net_out_box boxes get intersection shape as a list vertices.
            Use Green's Theorem to calculate a area for intersection shape using vertices.
            Clip truth_box and net_out_box boxes get union shape as a list vertices.
            Use Green's Theorem to calculate a area for union shape using vertices.
            Divide intersection shape area by union area to IOU score
            Set IOU score in IOU Tensor at current index
        end for
    end for
end for
```

Below is the new IOU function, notice the contrast from Figure 2.6

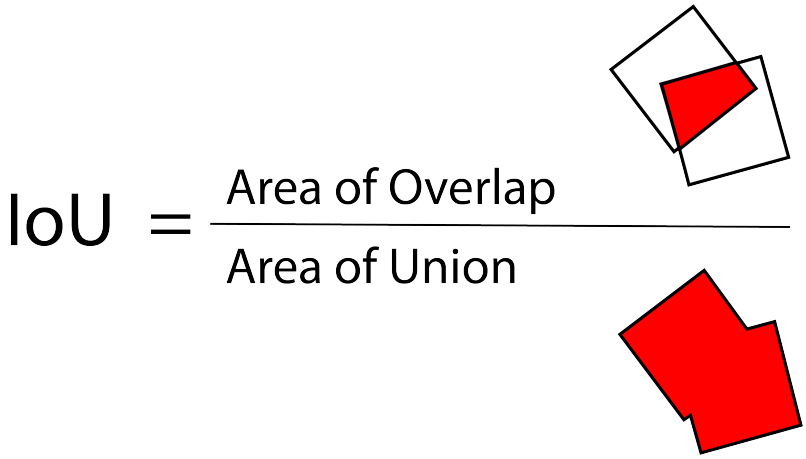


Figure 2.8: YOPO IOU function

Chapter 3

Software Design and Implementation

3.1 Introduction

At the start of this project two open source implementations of YOLO were considered; Darknet and Darkflow, and elected to use Darkflow. The Darkflow GitHub repository was forked and the pre-trained weights were downloaded along with the YOLO default configurations files. This was used as a starting point for the adaptation of YOLO, the YOLO weights were used as a starting point because these weights had already learned how to detect objects and the hope was that YOPO could retrain this weight file to learn angles with different classes.

3.2 Original Darkflow

Darkflow is a Python implementation of darknet using TensorFlow. Darknet [6] is an open source framework written in C and Cuda to run neural networks on a GPU. This means Darkflow could run any Convolutional network that is implemented in TensorFlow not just YOLO, by implementing the framework object.

3.2.1 Original Darkflow Architecture

Darkflow has three core functions: build the TensorFlow Network (TFNet), train the network and test the trained model, the core operations are coloured in grey on the diagram in Appendix D section 4.1. The diagram shows an overview for the whole system, Darkflow takes four inputs; a set of ground truth XML files(Appendix C), training images, a network configuration file and an optional weight file that can be used to speed up the training.

The cliHander parses all the command line arguments and sets a variable called ”Flags” that contains all the options for the Darkflow and provides default values if some non required options are not provided. The TensorFlow network(TFNet) must be created before training or testing can be executed. The TFNet object is very generic, as it does not have a network or framework. A framework is a parent class, that any algorithm that you wish to implement must extend. It defines functions and operations that are needed for this particular algorithm such as how the ground truth data for this algorithm should be parsed and processed. TFNet also requires a network to be

defined using a configuration file (Appendix C section 3.1) and optionally a weight file that will be used to initialise the layers from previous training.

When the network command line training flag is enabled, the train function is executed and uses the framework's shuffle function to get the ground truth and and training images in the correct format for the network. These are then split into batches, a batch is a way of splitting data into manageable chucks for the network to process. Batches are created based the a value called 'batch size', which determines how many images can be inputted into the network in a single step. The batch size is mainly based on what computer resources are available if only a CPU (Computer Processing Unit) is used then it depends how much RAM (Random Access Memory) is available or VRAM (Video Random Access Memory) if a GPU (Graphics Processing Unit) is used. A step is the amount of amount batches required to complete one epoch, and for a one epoch to be completed the whole training set must be processed through the network once. To calculate the amount of steps required to complete training of the network is done by the following:

$$Steps_T = E_n * (\frac{I_n}{B_s})$$

Where $Steps_T$ is the total number of steps required, E_n is the total number of epochs, I_n the image amount of images in the training set, and B_s is the batch size.

Then finally, a TensorFlow session is started, using the framework's custom loss function, a batch of images and ground truth for current batch. The TensorFlow session executes the TensorFlow Graph which is the framework's convolutional neural network, this means that the image passes through the network and a output tensor is returned, which is the input to the framework's loss function. The loss function evaluates how well the network has performed and adjusts the weights accordingly. Once all steps have been completed a weight file is outputted, which is simply a matrix of values that when multiplied by the convolutional layers will optimise the network for detecting a particular set of objects. So in this instance the weight file will make the network hopefully detect the objects that it was just trained on.

Testing the model requires a TFNet to be created, a weight file and at least one test image to be supplied. The images are passed through the network, resulting in an output tensor with size defined in the network configuration file. The framework's post-processing functions are applied with the results being that predicted bounding boxes are displayed on the input image and is written to disk.

Finally, while the selected framework was YOLO, any algorithm and network could be implemented, providing the correct framework and configuration were provided.

3.2.2 Data Processing

Data processing is handled by the TFNet's shuffle and _batch functions. These functions parse the XML ground truth files using the selected frameworks parser, in the architecture diagram (Appendix D) the parser is YOLO. See section 3.2 in the Appendix C of this report for an example XML file. The parser returns a Python list that contains all the ground truth data, and the filenames of the images the data belongs to, see Figure 3.1

```
[
  [image_filename, [image_width, image_height,
    [
      [class_name, xmin, ymin, xmax, ymax],
      [class_name, xmin, ymin, xmax, ymax],
      ...
    ],
  ],
  [image_filename, [image_width, image_height,
    [
      [class_name, xmin, ymin, xmax, ymax],
      [class_name, xmin, ymin, xmax, ymax],
      ...
    ],
   ...
]
```

Figure 3.1: Example of parsed annotation list for a given dataset in darkflow

The $\_batch$ function takes a batch of parsed annotations and creates tensors for the network so that they are ready for the network. It does so in a number of steps:

- Normalise ground truth data
- Create tensors (numpy arrays [24])
- Place the normalised ground truth data into tensors
- Return a Python dictionary with all the tensors.

A YOLO ground truth bounding box is defined as (x, y, w, h), however Darkflow uses ground truth XML annotations that are defined by their top left and bottom right coordinates. So YOLO's batch processing function first converts the ground truth into YOLO's format and then normalises them between 0 and 1. The centre is first calculated by:

$$X = 0.5 * (xmin + xmax)$$
$$Y = 0.5 * (ymin + ymax)$$

X and Y are made relative to a cell, by first calculating the width and height of the cell and then calculating what cell the centre(X_c, Y_c) of the annotation(ground truth box) is located in:

$$cellWidth = \frac{image\_width}{S}$$
$$cellHeight = \frac{image\_height}{S}$$

$$X_c = \frac{X}{cellWidth}$$
$$Y_c = \frac{Y}{cellHeight}$$

Finally we find the local offset in that cell:

$$X_n = X_c - \lfloor X_c \rfloor$$
$$Y_n = Y_c - \lfloor Y_c \rfloor$$

As for the width and height they are normalised with respect to the whole image so:

$$W_n = \frac{(Xmax - Xmin)}{image\_width}$$
$$H_n = \frac{(Ymax - Ymin)}{image\_height}$$

YOLO uses the Sum-Squared error to optimise the network but this treats errors in large and small boxes equally so to partially address this YOLO predicts the square root of the width and height [1]. Therefore the final input width and height is:

$$W_i = \sqrt{W_n}$$
$$H_i = \sqrt{H_n}$$

Now that all the data is correctly normalised it's placed into tensors. They would be the input to the network, see figure Figure 3.2.

```
for obj in allobj:
    probs[obj[5], :] = [0.] * C
    # Sets the cell to the class that the ground truth
    # centre is located in.
    probs[obj[5], labels.index(obj[0])] = 1.
    proid[obj[5], :] = [1] * C
    # Adding the ground truth boxes
    coord[obj[5], :, :] = [obj[1:5]] * B
    # Set the box Confidence score to 1
    # because it's the ground truth.
    confs[obj[5], :] = [1.] * B
```

Figure 3.2: Darkflow adding the normalised ground truth into tensors

3.2.3 Building the network

The YOLO network is made up of 24 convolutional layers, and uses a max pooling layer to reduce the size of the output from the previous layer. See Figure 3.3 for the first convolution and maxpool layer that YOLO uses.

```
[convolutional]
batch_normalize=1
filters=64
size=7
stride=2
pad=1
activation=leaky

[maxpool]
size=2
stride=2
```

Figure 3.3: Example of a convolutional layer and a Max Pooling Layer

The network is built by a Python file called build.py, which creates an object called TFNet. This object creates a framework based on what network is being used in DarkFlow, in our case it's YOLO. It follows these steps:

- Creates a network using the configuration file
- Creates a framework based on the algorithm and network selection.
- performs a forward pass to build network
- Selects and initialises GPUs if any are available.

Darkflow reads the network configuration file line by line and creates the defined network from the file. Each of the layers in the configuration file is created as a TensorFlow operation, this is then added to the TensorFlow graph which is the implementation of the convolutional network. When the tensors flow through TensorFlow graph this is equivalent to an image passing through a convolutional neural network, see Figure 3.4 for YOLO's network architecture.

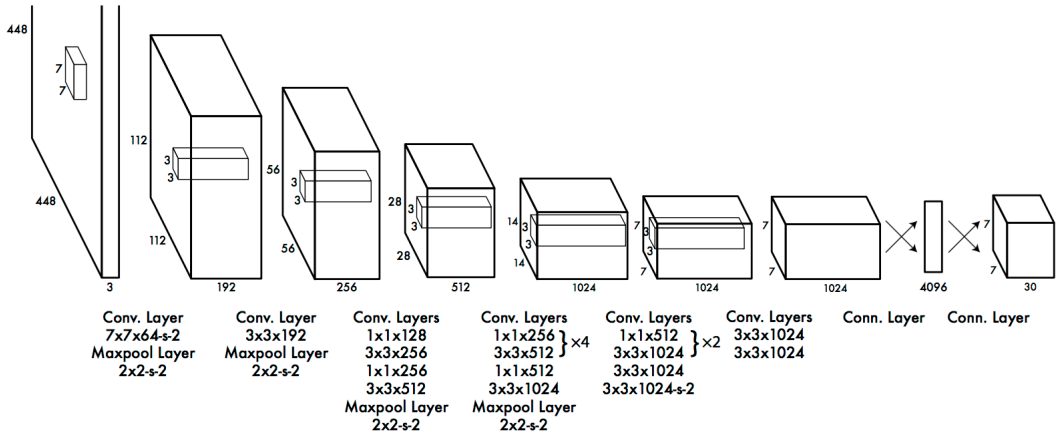


Figure 3.4: YOLO Network taken from YOLO paper [1]

3.2.4 Training the model

The cliHandler object in cli.py calls the train function based on command line arguments given to the program. Only once TFNet is fully initialised the network can be trained. This means the network and all its TensorFlow operations have been created and added to the TensorFlow graph. Also the correct algorithm has been selected and a framework created and added to the TFNet object.

TFNet train function starts a TensorFlow session with a batch of the ground truth tensors and the framework's custom loss operation as parameters. As mentioned in chapter two, TensorFlow defines an abstract graph of operations that make up the network, only when a session is started does the network execute.

3.2.4.1 Framework loss function

YOLO is the selected framework therefore YOLO's custom loss operation is being used to train the network. The operation is defined in the YOLO framework object (section 3.3), which takes the output tensor defined in chapter 2 and the ground truth of the image YOLO's currently training on as parameters.

To evaluate the performance of the network's prediction an IOU metric is used, which is a score that defines how well the predicted box and the ground truth box overlap. The network output tensor is returned as a flatten array (rank one tensor, only one dimension), whereas the ground truth is a multi-dimensional array (rank 3 tensor), so YOLO must first shape the network output tensor to be the same shape as the ground truth tensor. The output tensor coordinates are

converted so they match the Darkflow format (xmin, ymin), (xmax, ymax) so that the area of the predicted box is calculated. This is simply done by $W * H$, however the intersection area is calculated by finding the intersection shape width and height and then calculating the area via $W * H$. Finally the IOU is calculated by:

$$IOU = \frac{intersectio\_area}{groun\_truth\_area + predicted\_area - intersection\_area}$$

The best box is then calculated by the highest IOU score, and the other boxes are given a score of zero. The final score is calculated as per the YOLO paper [1] $confidence * IOU$ of the best box for that cell. The sum squared is applied on the ground truth tensor and the network output tensor, this returns a loss value which is used to update the weights.

See the Figure 3.5 below for the TensorFlow operations implemented in Darkflow.

```
# Extract the coordinate prediction from net.out
coords = net_out[:, SS * (C + B):]
coords = tf.reshape(coords, [-1, SS, B, 4])
wh = tf.pow(coords[:,:,:,2:4], 2) * S
area_pred = wh[:,:,:,0] * wh[:,:,:,1]
centers = coords[:,:,:,0:2] # [batch, SS, B, 2]
floor = centers - (wh * .5) # [batch, SS, B, 2]
ceil  = centers + (wh * .5) # [batch, SS, B, 2]

# Calculate the intersection areas
intersect_upleft   = tf.maximum(floor, _upleft)
intersect_botright = tf.minimum(ceil , _botright)
intersect_wh = intersect_botright - intersect_upleft
intersect_wh = tf.maximum(intersect_wh, 0.0)
intersect = tf.multiply(intersect_wh[:,:,:,0], intersect_wh[:,:,:,1])

# Calculate the best IOU, set 0.0 confidence for worse boxes
iou = tf.truediv(intersect, _areas + area_pred - intersect)
best_box = tf.equal(iou, tf.reduce_max(iou, [2], True))
best_box = tf.to_float(best_box)
confs = tf.multiply(best_box, _confs)
```

Figure 3.5: YOLO loss function calculating IOU between ground truth and network output

3.2.5 Testing the Model

A supplied image is given to the network and the framework's predict function is invoked, which feeds the image input tensor into the network. This then passes through the network and an output tensor is returned with YOLO's prediction for that image. Post processing is then applied, which finds the boxes from the output tensor, using the YOLO box constructor which is a Cython [25] function that takes the output tensor from the network, and a threshold which defines the minimum probability a prediction must have for a bounding box to be created. The bounding boxes are

returned after having non-max suppression applied which removes boxes if they are overlapping each other, and detecting the same class [9] see Figure 3.6.

Finally, the boxes a written to the image using OpenCV's rectangle function to draw the bounding boxes. Alternately, a JSON file can be written to disk, the $(xmin, ymin, xmax, ymax)$, along with the class and probability.

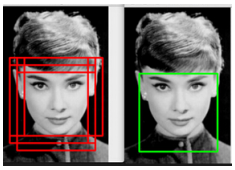


Figure 3.6: Non-maximum suppression applied to an image, [9]

3.3 Modified Darkflow

3.3.1 Darkflow YOPO

YOPO is a new Darkflow framework that is clone of the YOLO framework but with the YOPO adaptations, see Appendix D section 4.2 for an updated Darkflow architecture diagram with YOPO as the selected framework. As you can see in the diagram the framework's boxes in red have changed to suit YOPO's need otherwise it's not been changed from the original YOLO implementation.

3.3.2 YOPO Network Architecture

YOPO uses the same network as YOLO but changes the network hyper parameters and the output layer tensor size. The aim is to use the existing trained convolutional layers to significantly reduce training time. It should be noted that these layers were trained on ImageNet [17] which only has non-rotated boxes in its ground truth as to if this will have a negative impact on YOPO is unclear.

As mentioned in chapter 2, the "side" which is the grid size will be increased to 16, the "classes_num" will be set to 10 which is the number of limbs that YOPO is being trained on, the "coords" will be set to 5 (x, y, w, h, θ), adding the extra parameter θ from the original version.

Then the output layer will be changed to 7168, see chapter 2 for its calculation. This may increase the training speed of the network, and the post processing because the network output tensor size has been changed from 1470 to 7168. This means there are bigger calculations when the IOU is calculated, and in post processing when boxes are being calculated from the output layer which is now about 5 times larger.

In YOLO the default configuration had a 7 by 7 grid size with a total of 49 cells and each cell had a Tensor of size 30, see Figure 3.7. YOPO however has a much larger grid 16 by 16 with a total of 256 cells and each cell has a new tensor of size 28, see Figure 3.8.

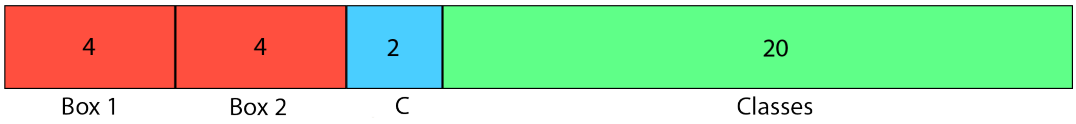


Figure 3.7: YOLO network output tensor per grid cell

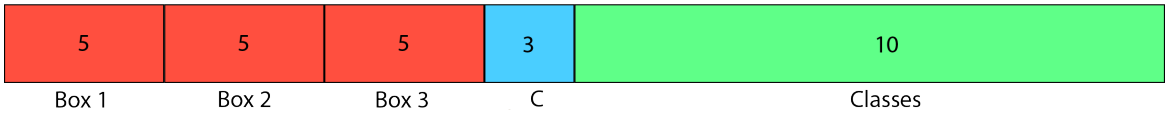


Figure 3.8: YOPO network output tensor per grid cell

3.3.3 YOPO Preprocessing

YOPO uses the MPII dataset, the dataset provides an image folder and a MatLab [26] data file that contains information about each image, in particular the x and y coordinates for all the joints in an image, if the joint is visible in the image, the top left and bottom right rectangle coordinates for the head rectangle. The full list of information can be found in the MPII paper [7]. See figure Figure 3.9 for an example of a ground truth being plotted against an image, with green points showing that the joint is visible, and red points showing that the view of the joint is obstructed and finally the joint id is plotted next to the joint.

Figure 3.9: MPII dataset ground truth plotted on an image

The MatLab data file is parsed into a Python dictionary using SciPy [27], that is ordered via the image filename as the key, for quick access and so that it has a time complexity of $O(1)$. A 'darkflow' boolean flag is checked to decide which framework to process the data for, the pre-processing code can process the MPII dataset for Darknet or Darkflow because of compatibility that might be needed should this be ever used for something other than YOPO.

The limbs data is generated via the 'generate_limb_data' function, it takes the whole dataset of images and the parsed MPII metadata dictionary and creates train and test dataset depending which flags are given to the function. The data by default is given a 80:20 split but this is configurable. Additionally, the amount of data that will be processed can also be changed via a limit flag because the full MPII dataset has 25 thousand images which may not be practical for some models, hardware or time constraints.

It generates the angled bounding boxes by loading the image using OpenCV extracting the width and height information. It then loops through each pose in the image and creates a point object which simply has an x and y value. Each joint has an ID that is provided by the MPII dataset, this is used to know which joints are connected next to each other, for example wrist and elbow with ID 0 and 1 would make up a lower arm. The centre point is calculated via the mid point of two points equation:

$$C = \frac{x_2 + x_1}{2}, \frac{y_2 + y_1}{2}$$

The height of a bounding box is calculated via distance between two points:

$$H = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

The width is calculated heuristically because it's not given in the dataset and a constant would rely on all images being the same size and the pose being the same scale in the image. To partially address this problem the head width is used to calculate a scale factor, during manual inspection over a few hundred images the width looks correct for each of the limbs with the following calculation:

$$W = \frac{50 * Headwidth}{70}$$

Finally, an angle is calculated by applying $\arctan 2$ on the two joints:

$$\theta = \arctan 2(y_2 - y_1, x_2 - x_1)$$

The bounding box is then converted into the DarkFlow format: xmin, xmax, ymin, ymax and with the added YOPO angle. Finally, this is written to an XML file along with its class which is the limb's name, see Figure 3.10. This process is repeated for every limb in every pose for a given set of images.

```
<object>
  <name>right-upper-leg</name>
  <bndbox>
    <xmin>722</xmin>
    <xmax>877</xmax>
    <ymin>454</ymin>
    <ymax>517</ymax>
    <angle>-88.51213247117222</angle>
  </bndbox>
</object>
```

Figure 3.10: A fully Processed YOPO limb

3.3.4 Data Processing

YOPO data processing had a few changes from the original YOPO processing function. The XML parser changed slightly so that it could also parse an angle. B_p is increased by one because of the extra parameter of the angle θ. The angle is normalised between 0 and 1 by:

$$\theta_n = \frac{\theta}{360}$$

An extra tensor is created called 'image_tens' which holds the width and height of the image being processed, this is used for IOU calculations later on in training. Then like YOLO the image and ground truth tensor are returned.

3.3.5 IOU function

YOLO's IOU function is completely replaced for YOPO's IOU function, which is a custom operation in TensorFlow. Meaning that it becomes a node on the graph that tensors will flow through. It's using a function called 'tf.py_func' which wraps a Python function in a TensorFlow operation.

The IOU function takes four tensors: image tensor, ground truth tensor, network output tensor and the IOU tensor. The ground truth and network output tensors both have dimensions $(S^2 * 3 * 5)$ with the YOPO's supplied network configuration. The IOU tensor has shape $(BatchSize * S^2 * B)$ with each image in the current batch having it's own entry in the IOU tensor. We loop through the ground truth and network output tensors in the same loop so that current cell i is the same cell in both the ground truth and network output. To increase performance the ground truth cell is checked to see if any bounding boxes are in $cell_i$, if no ground truth is present then $cell_i$ is skipped because the cell is empty.

Given that current cell $cell_i$ has B boxes, we loop though each box with box_g being the ground truth and box_n being the network output box both with index j. The box coordinates are extracted from each box this includes the angle θ. These values are then de-normalised by applying each parameter's inverse normalisation operations in reverse order.

Once the correct values for each box have been extracted from their respective tensors, two YOPO rectangle objects are created from box_g and box_n. YOPO's custom rectangle constructor requires x, y, w, h, θ as parameters. This object has two main functions calculating an intersection

area, and a union area given another YOPO rectangle as a parameter.

The intersection area is calculated by the rectangle's 'find_intersection_shape_area' function which takes a YOPO rectangle rec_1 and rec_0 and checks if the rectangles intersect. Each rectangle invokes it's "get_vertices_points" function that translates the rectangle from (x, y, w, h, θ) to $(p0, p1, p2, p3)$. Then the Vatti clipping algorithm [28] implemented in pyclipper [29] which is a Cython wrapper for a C++ library called Clipper [30] is used for the clipping of two rectangles. pyclipper returns a list of points that define the intersection shape, see Figure 3.11. The union shape and area are calculated via the same method however the pyclipper is given a 'UNION' flag instead of 'INTERSECTION'.

```
[
    [x0, y0], [x1, y1], [xn, yn]
]
```

Figure 3.11: Python List of points returned by pyclipper

YOPO uses Green's Theorem [23] to calculate the area given a list of shape points, where $p_{n+1} = p_0$ to join the shape.

$$A = \sum_{p=0}^{n} \frac{(x_{p+1} + x_p)(y_{p+1} - y_p)}{2}$$

There are a few situations where the pyclipper will fail: because the network output can give any value for the predicted box parameters. Including a line:

$$W < 1 \vee H < 1$$

or a point:

$$W < 1 \wedge H < 1$$

The function "intersection_over_union" calculates the IOU score with the two areas, the calculations are surrounded in a try catch statement, if edge cases occur that the previous checks do not catch then this is the fail safe and an IOU of zero is returned. The Python documentation says that: 'try/except block is extremely efficient if no exceptions are raised, actually catching an exception is expensive' [31]. When the network is first training the width and height are most of the time less than 1 so to improve performance the if statements are used to catch the common errors because the frequency of exceptions being raised would be expensive, even more so for a function that is executed for both rectangles for every limb in the training set.

Finally the IOU score is set at index $[image\_index\_in\_current\_batch][i][j]$, this processes repeats for B boxes, S^2 cells and with n images. Then tensor is then returned to the network and performs the same as YOLO.

3.3.6 Post Processing

YOPO creates a custom rectangle object from the predicted output and gets the vertices points using the centre (x, y), width w, height h and angle θ of the predicted box. OpenCV draws predicted boxes on the image using the vertices points.

3.4 Development

PyCharm [32] was the editor of choice because it offered good debugging tools and integrated well with SonarQube the static code analysis tool used for this project. It also provides a good set Python coding standards such as enforcing 4 space indentation, automatically reformatting the code on save and on commit.

Git was used for version control, that was hosted on a private GitHub repository. This used Travis CI [19] that runs builds every 24 hours and every commit, executing the automated tests and performing SonarQube static analysis on the source code. SonarQube produces a report of the analysis which is pushed to the SonarQube cloud [33] for review.

DarkFlow was being used as a starting point, which had already been written in Python using TensorFlow, so it only made sense to use Python along with TensorFlow.

Google Cloud was utilised for development and training of the network, as mentioned in chapter 1 $300 free credit was provided for use of the Google Cloud VM instances. A high spec machine was used to train the network during development so that my personal computer could be used for development and testing of small changes to the network.

3.5 Implementation

YOPO was implemented with a single person adjusted scrum, where the project owner was the project supervisor of this project, where each weekly meetings were held to discuss the current progress of the project, and any problems that may have arose from research or development of YOPO. These weekly meeting also acted as sprint retrospective and a weekly stand up.

3.5.1 Sprint 1 - Background Research

3.5.1.1 Overview

Sprint 1 which was two weeks long, started by trying to understand the project, what the objectives of the project were and what were the steps that were going to be taken to achieve the desired outcomes. YOPO presented a lot of new concepts while previously I have had a limited knowledge in the area of computer vision and machine learning from my studies whilst at university, and my previous year in industry. There were still a lot of new concepts such as how convolutional neural networks worked or what tensors were and how they related to images and convolutional neural networks.

I started reading through the YOLO paper and made notes to try and understand how the network and algorithm worked. While doing this I began to realise that my knowledge of machine

learning and convolutional neural networks was somewhat lacking for topic as complex as this. So I decided before I would start working on understanding YOLO, I would try and gain a better understanding of the core concepts that YOLO is built upon. I read through the Stanford computer vision lectures notes [13], I also attended a computer science masters machine learning workshop on convolutional neural networks at the university to get a more personal view point on the topic where I could ask questions.

Once I gained some more knowledge in the subject area, I began to work through the YOLO paper and take notes, and break the network into parts for better understanding how each component of the network worked. Such as to why the network split an image into a grid and why it was faster other traditional methods such as sliding window which had been previously discussed in my computer vision lectures.

3.5.1.2 Retrospective

The objectives this sprint were completed, even through there was some deviation from the main task for background research about the domain area.

3.5.2 Sprint 2 - Research & Implementation Preparation

3.5.2.1 Overview

Sprint 2 lasted two weeks and was mainly focused on deciding which implementation of YOLO YOPO was going use, and setting up DevOps for later use in development. I didn't have much experience in either Python or C, and given the time constraints of the project Python was chosen and therefore Darkflow. I forked the Darkflow GitHub repository and started familiarising myself with the code base. I then used the VOC dataset [5] with the default YOLO configuration file to test the standard YOLO model to see what the network outputted and how to use the network.

A GitHub private repository was setup for YOPO that used continuous integration tools Travis CI and SonarQube to ensure code quality and when unit tests were created, the software could be tested regularly to detect any breaking changes that were made.

The aim of YOPO of was to detect human poses in a 2D image. To achieve this, the rotation of the bounding box had to be calculated. YOLO's bounding boxes already had four parameters that it predicted (x, y, w, h). YOPO suggests that adding a new angle parameter to the bounding box's parameters(x, y, w, h, θ) will allow the new adapted network to detect limbs with orientated bounding boxes that make up the human pose detection.

3.5.2.2 Retrospective

All sprint goals were completed but the SonarQube task. Good progress was made in understand the DarkFlow code base, however there were still some areas that were unclear. Travis-CI was setup correctly and is running the original Darkflow unit tests.

3.5.3 Sprint 3 - Dataset Processing

3.5.3.1 Overview

This sprint was mainly focused on pre-processing the MPII dataset. The ground truth was stored in a MatLab data file. This presented a problem for parsing the ground truth because it was in an unfamiliar format and access to MatLab wasn't available. SciPy was used for the parsing on the MatLab data file, this consisted of looping over the data tables in the file and extracting ground truth for images into an array. Due to the nesting of the data inside the MatLab data file, this task took longer than expected to parse. PyCharm was used because of it's debugger, which was to view inside the data structure and then work out manually how the parser was going to get the data from the file structure. The duration of the task extended from the previously estimated one day to two sprint days. Once the data was loaded into Python, it could then be processed, the "generate_limb_data" function with parameters: array of image files paths and array of ground truth data. This function then loops through the data image array and evaluates if the image ground truths are present in the array. If so then the OpenCV gets the current image and loads into memory while extracting the height and width for later use. A YOPO Darkflow ground truth requires five parameters (x, y, w, h, θ) each one had to be calculated separately from two limb points, the individual calculations for each parameter can be found in subsection 3.3.3. The chest is an edge case because it uses six points: right shoulder, left shoulder, right hip, left hip, thorax and pelvis to create the bounding box. This processes is repeated for each limb, in total there was 10 limbs, once the process completes all bounding boxes along with the height and width of the image are written to a XML file. Finally, the training images and ground truth XML files are sorted into two folders that Darkflow will use, labels and images.

Processing around 12,000 images took around 20 minutes, further investigation into the high process duration was that the use of arrays made the time complexity $O(n^2)$. So YOPO was switched to use a Python dictionary where the index was the filename, which decreased the time complexity to $O(1)$ resulting in a 3 minute processing time.

3.5.3.2 Retrospective

The MPII dataset preprocessing code was fully completed during the sprint although no unit tests where created along with the code base due the time constraints. A preprocessing testing task was added to the backlog to create unit tests that checked the basic bounding box calculations were correct.

3.5.4 Sprint 4 - IOU function implementation

3.5.4.1 Overview

Sprint 4 was attempting to implement a new IOU function for YOPO, that could calculate the intersection and union areas of two rotated bounding boxes. After some research it seemed that clipping the rectangles together was the most appropriate method, however while clipping two axis aligned rectangles is fairly simple, the rotation of these rectangle creates a new layer of complexity for the problem. After more research of clipping the two orientated polygons together, a library called pyclipper was found which provided the required functionality. It was tested with a small

subset know polygons, to investigate the accuracy and a rectangle class was created to provided a structure for a bounding box which contains (x, y, w, h, θ).

The new IOU function integration with Darkflow also had the complexity of TensorFlow to take into account. YOLO's framework was copied and renamed YOPO for a base to start developing YOPO from. After trial and error with the YOLO network and re-reading the paper, some core problems were identified:

- The size of the output layer needed to change to accommodate the new box parameter and different classes to train
- Ground truth tensor needed to be changed to allow for an extra bounding box parameter θ
- The YOLO IOU function needed to be replaced for a new method proposed by YOPO
- The preprocessing of the boxes needed to take θ into account when converting an output tensor to bounding boxes
- Integrating of the YOPO IOU function into the framework

YOPO's network output layer size was changed from 1470 to 7168 to accommodate a grid size of 16 and the new θ parameter for the bounding box. The ground tensors ($S^2 * B * B_p$) were adjusted slightly so that the last dimension B_p was increased by one again for the θ parameter.

YOPO's previous IOU function needed to be completely replaced, for the suggested IOU function. As mentioned above the IOU function used a custom rectangle class and was separate from the network. However the network was written in TensorFlow which only used tensors using an abstract defined work-flow(TensorFlow Graph). This presented a difficult integration problem so two different methods were considered: Writing the Vatti clipping algorithm in TensorFlow and then implementing green's theorem to calculate the area of the Vatti clipping shape, however this was not appropriate for the time constraints of the project. A custom operation written in Python, normally TensorFlow operations are written in C++ with a GPU version of the operation written in Cuda. However, due the time constrains Python was used, the 'py_func' function was used to implement the operation which wraps a Python function in a TensorFlow operation. The new IOU function that takes the network output tensor and ground truth tensor and as discussed in subsection 3.3.5 this creates a custom rectangle object for the new IOU score to be calculated. YOLO's squared mean error function was kept the same, the only difference between YOPO was that the input tensors had different sizes.

Finally 14 unit tests for the IOU function were created to test all the common uses for the function as well as the edge cases that were found on a case by case basis.

3.5.4.2 Retrospective

Sprint 4 was a difficult sprint with a lot of layers of complexity in the integration of the proposed IOU function with TensorFlow. A lot of research had to be done into the best method for implementing the custom TensorFlow operation. All the objectives were completed and in this sprint the core YOPO operation was implemented.

3.5.5 Sprint 5 - Bug fixing & Code Refactoring

3.5.5.1 Overview

This sprint built on the previous sprint focused mainly on stress testing the network and creating unit tests for the IOU function, which is a core part of YOPO. Stress testing the network involved an image and ground truth set which was created using the YOPO preprocessing source code with 100 images. The network's first IOU calculations failed when network output tensor had returned a rectangle with zero width. To address this, a guard statement was added using if statements so that if the width was zero than the predicted rectangle was a line and therefore the IOU score was zero, because lines do not have an area. Further testing indicated that the network could also predict extremely low values during the initial training iterations, and therefore both width and height were very close to zero and therefore IOU calculations gave an area of zero. More guard statements were added to help prevent theses cases. During a large test with 2000 images using the Google cloud VM, the network's IOU calculations would be unable to find an intersection shape and would throw an exception thus crashing the network. This was an extreme edge case that could not always be identified so catch statement was added to act as a fail safe, if both guard statement failed to catch the error. The Python documentation stated the state exceptions are slower that 'if' statements if the program throws a high frequency of exceptions.

Finally, the YOPO code base was re-factored and cleaned a lot of the debugging code up from the source code. The SonarQube report was reviewed and many bugs and code smells were fixed.

3.5.5.2 Retrospective

Sprint 5 was the final development sprint, it focused mainly on testing the network and creating unit tests for any bugs found to ensure that they had been fixed. Ensure the IOU function could handle all possible inputs and ensure that it generate a correct IOU score.

Chapter 4

Testing

4.1 Overall Approach to Testing

Testing is very important in software engineering however this project had time constraints and to manage those effectively only IOU function was unit tested due to it being a critical component of the software. The testing of YOPO is split into two parts: the testing of the software and the model that YOPO's produces.

4.2 Automated Tests

YOPO used Travis CI to automate its testing process, every git commit started a build which built the software and executed the unit tests. This provided real time feedback, because every time a significant change was made to the code base, a commit was made and therefore the code changes could be tested to check if any breaking changes were made. The IOU function had automated tests, that tested known polygons with known areas, and performed the IOU calculations to check that a valid score was being produced.

4.3 Unit Tests

YOPO unit tests covered the core functional part of YOPO which was the IOU function, basic operations were tested such bounding boxes with no orientation to ensure that original method hasn't been broken by the new changes. The union shape area and the intersection area are calculated individually so that if the IOU calculation breaks it can be quickly isolated to which part or even both. As the network was being tested with a small dataset, if a bug was found then once it was fixed a unit test would be created to ensure that it's working correctly. An example would be if the width was zero then the software would throw an exception, once that was fixed then a unit test was created to ensure that bug was completely fixed and any new changes to the software didn't introduce the problem in a different area.

4.4 Manual Tests

The MPII dataset required manual testing for the inspection of the bounding boxes to ensure they fitted correctly. While manual testing can be time consuming, it's sometimes necessary to ensure high quality ground truth which the network heavily depends on.

4.5 Static Code Analysis

SonarQube statically analyses the source code of YOPO and give grades for many areas of the code base. Such as bugs that are found in the code, for example an if statement that assigns a value instead evaluating it, this bug is assigned a rating detailing the severity of the problem. A complete report is generated and can be viewed on SonarCloud, see Figure 4.1 for YOPO's SonarQube report. While all bugs and vulnerabilities of source code were fixed that were found during Travis CI's build and test process, due to time constraints there is a lot of technical debt, code smells, and code duplicated most of which is from the original source code. Unfortunately SonarQube was unable to detect YOPO's test coverage.

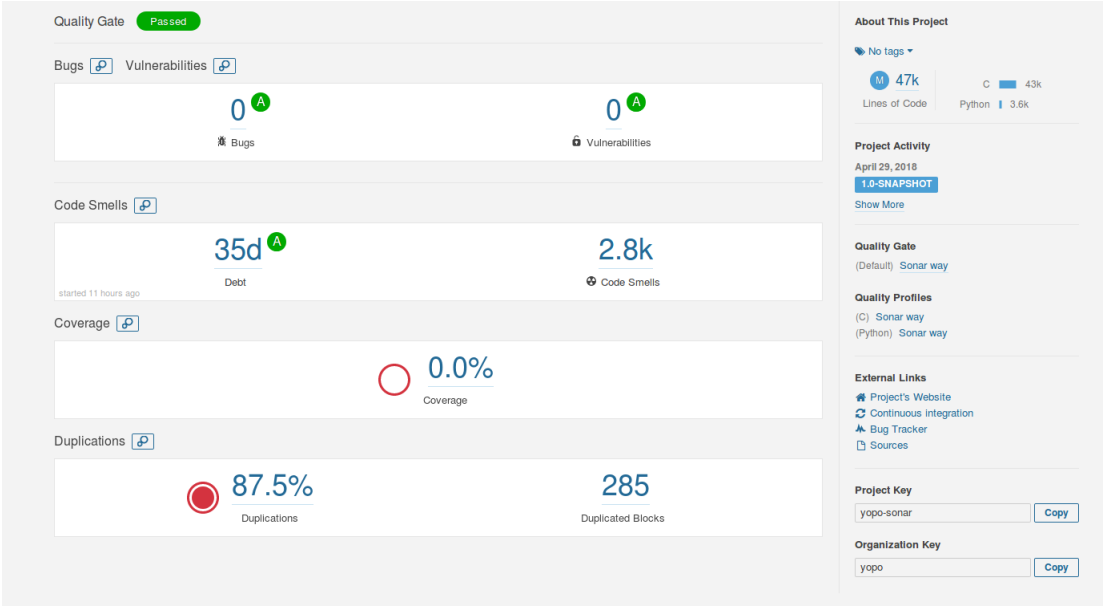


Figure 4.1: YOPO's SonarQube report hosted SonarCloud

Chapter 5

Results and Conclusions

5.1 Overview

The original hypothesis for this project was that by replacing the IOU function and adding extra angle parameter to YOLO it would allow the new adapted YOLO network to estimate human poses. YOPO was made up of two core elements, the software and the outputted model. Given the time constraints of the project three tests were performed, two over-fitted tests and one test using a large subset of the MPII dataset. The initial overfitted test was done to ensure the network and software was functionally correct and to see if a sensible model would created with a low frequency of training epochs. The final overfitted test was primarily to evaluate if the network would output a model that could perform human pose estimation, with a considerable amount training epochs, as side effect it also test the core functionality of the software as well. Finally the large dataset tested if with a large set of images could YOPO output a model that could detect a human pose in 2D images.

5.2 Over Fitted Model

An overfitted test was performed to test if the network was working, this used two images, see section 4.3 and section 4.4 in Appendix D for inputs to the network. All limbs in both training images were visible and therefore an even class distribution across all the different classes, see Appendix D section 4.5.

The network was trained for 500 epochs with it performing 1000 training steps(iterations) on the two input images. The training was watched in real-time using TensorBoard [34] which monitored the network for different chosen statistics and in this case it was the average loss of the network over steps executed, see Figure 5.1. Using the Figure 5.1 the model appeared to have converged.

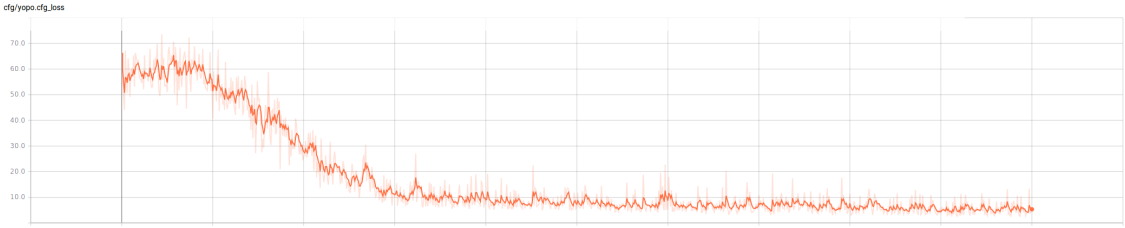


Figure 5.1: TensorBoard Output from 2 training images, Y: average loss and X: number of steps

The output model was then tested yielding the following outputs: Figure 5.2 and Figure 5.3.

Figure 5.2: Predicting bounding boxes from the network for Image A

Figure 5.3: Predicting bounding boxes from the network for Image B

Reviewing Figure 5.2 and Figure 5.3 we see a very poor performance from the network, due to no limbs been detected. However bounding boxes do have a rotation so the network must be predicting angles. The network appears to have converged therefore longer training could be needed, a larger dataset or different hyper-parameters.

5.2.1 Results from 20,000 epochs

Another overfitted test was performed to ensure the network is working correctly before a large dataset was provided to the network for training. This overfitted test was designed to ensure that the overall functionality of the software worked, see Figure 5.4 and Figure 5.5. Figure 5.4 and Figure 5.5 show that the network post processing was working, Figure 5.5 in particular seems have all the limbs correctly labelled. However, both figures ”head” has approximately the correct centre point, width and height, however the orientation of the box doesn't seem accurate. Further training could be required or there is a bug in the software either in the training or the post processing.

Figure 5.4: Predicting bounding boxes from the network for Image A, with 20,000 epochs

Figure 5.5: Predicting bounding boxes from the network for Image B, with 20,000 epochs

5.3 Large Dataset Model

The larger dataset test couldn't use the full set of images from the MPII dataset provided. This was due the time and computational constraints of the project. However, 2879 training instances were processed and prepared for training. The training duration was about 5 days which equated to about 250 epochs. Again the model appeared to converge, when we reviewed the Tensorboard output, see Figure 5.6.

The output model was then tested and didn't yield particularly good results some of which can be seen below in: Figure 5.7 Figure 5.8 Figure 5.9 Figure 5.10. Very few predictions were outputted from the network and they were all incorrect classifications.

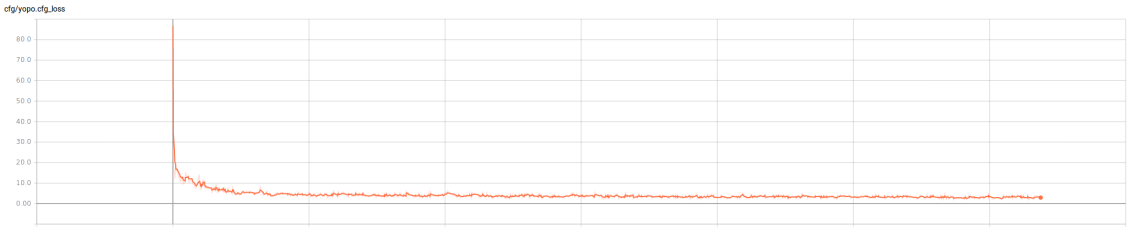


Figure 5.6: TensorBoard Output from 2879 training images, Y: average loss and X: number of steps

Figure 5.7: Predicting bounding boxes from the network

Figure 5.8: Predicting bounding boxes from the network

Figure 5.9: Predicting bounding boxes from the network

Figure 5.10: Predicting bounding boxes from the network, image from training data

5.4 Conclusions

The overfitted models demonstrated that the software itself worked. The network could produce bounding boxes that labelled limbs from a 2D image(Figure 5.5), this demonstrated that the pre-processing, data processing, training, and post processing parts of the software all worked as intended. The outputted model however was unclear if it worked and would potentially require more testing for an accurate assessment. The tests that were performed showed that an over-fitted model worked but it required a tremendous amount of training epochs for a reasonable output. This was unfortunate as part of the original hypothesis was that YOPO would use transfer learning, so that using the original YOLO weights would significantly reduce the amount of training epochs, needed to train the network to an acceptable standard.

Given more time and computing resources, YOPO's network could be tested with different network hyper-parameters such as learning rate, decay possibly even a larger input width and height which was not possible with the current computational power that was available.

Chapter 6

Evaluation

6.1 Project Management & Objectives

Agile was a good choice because it allowed me to adapt the development plan every two weeks, and at the start of the project I didn't have enough knowledge to make an educated plan about how to develop YOPO. Every sprint would start with an ordered backlog for the next sprint however the backlog would change depending on my weekly meeting with my supervisor and the progress of current sprint, a plan based approach would not have allowed for this flexibility. Additionally, the requirements during the start of the project were quite broad, and with how the individual components were going to work, were not clear at the start. This was where the research was mainly involved. However some tasks were longer than originally estimated and therefore the sprint ran over. This then had a cascading effect to other stories in the sprint and as a result testing suffered.

The objectives in subsection 1.2.1 were completed, however the trained model didn't have the performance that I hoped for. The processing code and implementation of the YOPO network was successfully completed however there was a lot of room for improvements that are discussed in the subsections below. Finally, a weight file was outputted however, human pose estimation was very limited to an overfitted model which is discussed in chapter 5.

6.2 Dataset preprocessing

I believe the dataset preprocessing area of the software went well because it processed the data in the correct format that was required for Darkflow and allowed the execution of the experiment. It was well optimised for fast execution with only 3 minutes required to process the whole dataset of 12,000 images. I believe that the box parameters had reasonable calculation methods and only the width parameter had to have assumptions made for it to be calculated. Even so this was partially addressed with a scaling factor based on the head width. However, it did require manually reviewing the outputs to ensure they appear to have accurate bounding boxes for each limb. The data progressing took longer than expected, which was mainly due to my indecisiveness to select a dataset which delayed the development of a parser for the selected dataset. This meant this task took a whole sprint and therefore no automated tests were created.

6.3 Research & Implementation

I believe the research went quite well, considering I started this project with very limited knowledge of computer vision and deep learning. It was quite a steep learning curve to understand all the concepts that the YOLO proposed. A lot of the time during the early sprints were consumed by acquiring the base knowledge, so that I could understand the project.

I found the implementation of the network very difficult and the integration with TensorFlow was particularly difficult. A lot of time was spent understanding TensorFlow and how it worked this consumed a half a sprint. Where it was originally planned for 3 days it took 7 days, this delayed the development of the IOU function which was a core part of the YOPO. Additionally, the tests for IOU function were critical for ensuring the network performed correctly so these test could not be neglected. My main failing in the implementation of the YOPO was that I rushed into developing the components of the software, however I didn't have enough comprehension of TensorFlow which slowed things down while I attempted to debug TesnorFlow. During the development of the IOU function I stopped developing YOPO mid sprint to read and understand the relevant information in the TensorFlow documentation.

However despite these setbacks YOPO can classify human limbs with orientation using an overfitted model, and therefore estimate human poses which was the original hypothesis. It was unfortunate that I was unable to obtain more GPU processing power, however, this was out my control and I believe I did the best I could with the computational resources available to me.

6.4 Testing

The YOPO's testing strategy can be viewed in two parts: software and the outputted model. The software testing could definitely have been improved. The preprocessing had no automated testing, I would have liked at least the limb generation to be tested, as without good ground truth data the network model could be become useless. In terms of the YOPO network it's IOU function had a lot of automated testing, however there were no end to end tests. For example, a test to evaluate whether the network could perform one training epoch with the set of images. Additionally, another end to end test would be to test the network with a known weight file and a known configuration file to evaluate the model, and see if the output of the network is within the expected range. Finally, all the tests would run on the Travis CI build engine.

The model had three tests, the two overfitted tests just proved the software worked. However the third test provided a demonstration into if YOPO could provide human pose estimation as per the original hypothesis. I would have liked to have more tests which used different hyperparameters, possibly different datasets, if I had more time I would train the network longer with the MPII dataset so that the model could be more accurately tested.

6.5 Future Work

YOPO could be improved in a number of ways. For example, the performance of an non-overfitted model for human pose detection could be improved. More training with a large datasets so that as mentioned in the above subsection a more accurate assessment of the model could be obtained. Additionally the translation of a limb bounding boxes into a human pose skeleton could

be implemented as post processing feature. Furthermore the handling of multiple poses in a image, so that each person in the image would have a estimated pose skeleton instead of just bounding box around each of their limbs.

Currently, the IOU calculations are executed on the CPU. The speed of the network could also be improved by the implementation of the IOU function as a custom operation in C++. This would allow a CUDA implementation to be added which would allow the IOU function to executed on the GPU. This would also require the Vatti clipping algorithm and Green's theorem to be re-written in Cuda so it could made into a TensorFlow operation.

A new two part system could also be considered, first using YOLO to detect people in images and then using YOPO for human pose estimation on the YOLO detections.

Appendices

Appendix A

Third-Party Code and Libraries

- **Cython** - Is an optimising static compiler for both the python programming language and the extended Cython programming language. It supports the calling of C functions and declaring C types on variables and class attributes. Allowing Python code to call C or C++ code naively. Cython has a Apache License.
- **pyCLipper** - Pyclipper is a Cython wrapper exposing public functions and classes of the C++ translation of the Angus Johnson's Clipper library. Heavily used in the IOU calculations which is discussed in chapter 2 & 3. pyCLipper has a MIT License.
- **Clipper** - An open source freeware library for clipping and offsetting lines and polygons. The Clipper library performs line & polygon clipping - intersection, union, difference & exclusive-or, and line & polygon offsetting. Clipper has a Boost Software License (Version 1.0)
- **Darkflow** - Is a python implementation of Darknet using the TensorFlow Library. It was used as starting point in the project. Chapter 2 & 3 discuss detail how this project modifies the existing source code. Darkflow has a GNU General Public License v3.0.
- **TensorFlow** - TensorFlow is an open source library for high performance numerical computation that was originally made by Google. Darkflow which the project is built on was using Darkflow, chapter 3 discuss how YOPO uses TensorFlow to creates a custom operation for the network. TensorFlow has a Apache License (Version 2.0).
- **OpenCV** - is an open source library for image progressing and machine learning. It contains many of the most used optimised algorithms for computer vision problems. Chapter 3 discuss how is was used for the processing of the MPII dataset. OpenCV has a 3-clause BSD License.
- **SciPy library** - A collection of numerical algorithms and domain-specific toolboxes, including signal processing, optimisation, statistics. Used for preprocessing the MatLab data file, which is discussed in chapter 3. SciPy is open Source with conditions (https://www.scipy.org/scipylib/license.html).
- **NumPy** - Used as an efficient multi-dimensional container of generic data, used to create tensors in YOPO. NumPy is open Source with conditions (http://www.numpy.org/license.html).

Appendix B

Ethics Submission

AU Status
Undergraduate or PG Taught

Your aber.ac.uk email address
rij12@aber.ac.uk

Full Name
Richard Price-Jones

Please enter the name of the person responsible for reviewing your assessment.
Reyer Zwiggelaar

Please enter the aber.ac.uk email address of the person responsible for reviewing your assessment
rrz@aber.ac.uk

Supervisor or Institute Director of Research Department
cs

Module code (Only enter if you have been asked to do so)

Proposed Study Title
You Only Pose Once - using convolution neural networks for pose estimation

Proposed Start Date
1st February 2018

Proposed Completion Date
4th May 2018

Are you conducting a quantitative or qualitative research project?
Mixed Methods

Does your research require external ethical approval under the Health Research Authority?
No

Does your research involve animals?
No

Are you completing this form for your own research?
Yes

Does your research involve human participants?
No

Institute
IMPACS

Please provide a brief summary of your project (150 word max)
Pose estimation is a difficult and well-studied problem. Recent advances in convolution neural networks have shown excellent results in a number of vision problems. This project would attempt to adapt the You Only Look Once (YOLO) algorithm (https://pjreddie.com/darknet/yolo/) to pose estimation. The project would take the pretrained YOLO data and attempt to retrain it for the problem of pose estimation using one of the many datasets available.

Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?

Appendix C

Code Examples

3.1 YOLO Configuration file

A file that describes the network and its configuration.

```
[net]
batch=64
subdivisions=8
height=256
width=256
channels=3
momentum=0.9
decay=0.0005
saturation=1.5
exposure=1.5
hue=1.5

learning_rate=0.0005
policy=steps
steps=200,400,600,20000,30000
scales=2.5,2,2,.1,.1
max_batches = 40000

[convolutional]
batch_normalize=1
filters=64
size=7
stride=2
pad=1
activation=leaky

[maxpool]
size=2
stride=2

[convolutional]
batch_normalize=1
```

```
filters=192
size=3
stride=1
pad=1
activation=leaky

[maxpool]
size=2
stride=2

[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

[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

[maxpool]
size=2
stride=2

[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

[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

[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

[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

[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

[maxpool]
size=2
stride=2

[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

[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
```

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

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

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

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

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

[dropout]
probability=.5

[connected]
output=7168
activation=linear

[detection]
classes=10
coords=5
rescore=1
side=16
num=3
softmax=0
sqrt=1
```

```
jitter=.2

object_scale=1
noobject_scale=.5
class_scale=1
coord_scale=5
```

3.2 YOPO ground truth file

Shows an example of a ground truth XML file for a given image, describing all the limbs' bounding boxes.

```
<annotation>
  <filename>000065339.jpg</filename>
  <size>
    <width>1280</width>
    <height>720</height>
  </size>
  <object>
    <name>chest</name>
    <bndbox>
      <xmin>632</xmin>
      <xmax>825</xmax>
      <ymin>167</ymin>
      <ymax>414</ymax>
      <angle>-92.08798383272335</angle>
    </bndbox>
  </object>
  <object>
    <name>head</name>
    <bndbox>
      <xmin>679.0</xmin>
      <xmax>767.0</xmax>
      <ymin>52.0</ymin>
      <ymax>190.0</ymax>
      <angle>57.47510971220896</angle>
    </bndbox>
  </object>
  <object>
    <name>left-upper-leg</name>
    <bndbox>
      <xmin>561</xmin>
      <xmax>728</xmax>
      <ymin>425</ymin>
      <ymax>488</ymax>
      <angle>-56.97613244420336</angle>
    </bndbox>
  </object>
  <object>
    <name>right-lower-leg</name>
    <bndbox>
```

```
      <xmin>697</xmin>
      <xmax>874</xmax>
      <ymin>443</ymin>
      <ymax>506</ymax>
      <angle>82.23483398157467</angle>
    </bndbox>
  </object>
  <object>
    <name>right-upper-leg</name>
    <bndbox>
      <xmin>722</xmin>
      <xmax>877</xmax>
      <ymin>454</ymin>
      <ymax>517</ymax>
      <angle>-88.51213247117222</angle>
    </bndbox>
  </object>
  <object>
    <name>left-lower-arm</name>
    <bndbox>
      <xmin>587</xmin>
      <xmax>698</xmax>
      <ymin>360</ymin>
      <ymax>423</ymax>
      <angle>-91.03224469156574</angle>
    </bndbox>
  </object>
  <object>
    <name>left-upper-arm</name>
    <bndbox>
      <xmin>569</xmin>
      <xmax>715</xmax>
      <ymin>232</ymin>
      <ymax>295</ymax>
      <angle>-90.0</angle>
    </bndbox>
  </object>
  <object>
    <name>right-lower-arm</name>
    <bndbox>
      <xmin>701</xmin>
      <xmax>866</xmax>
      <ymin>245</ymin>
      <ymax>308</ymax>
      <angle>108.10376303450663</angle>
    </bndbox>
  </object>
  <object>
    <name>right-upper-arm</name>
    <bndbox>
      <xmin>656</xmin>
      <xmax>783</xmax>
      <ymin>372</ymin>
      <ymax>435</ymax>
```

```
      <angle>128.23382517744693</angle>
    </bndbox>
  </object>
</annotation>
```

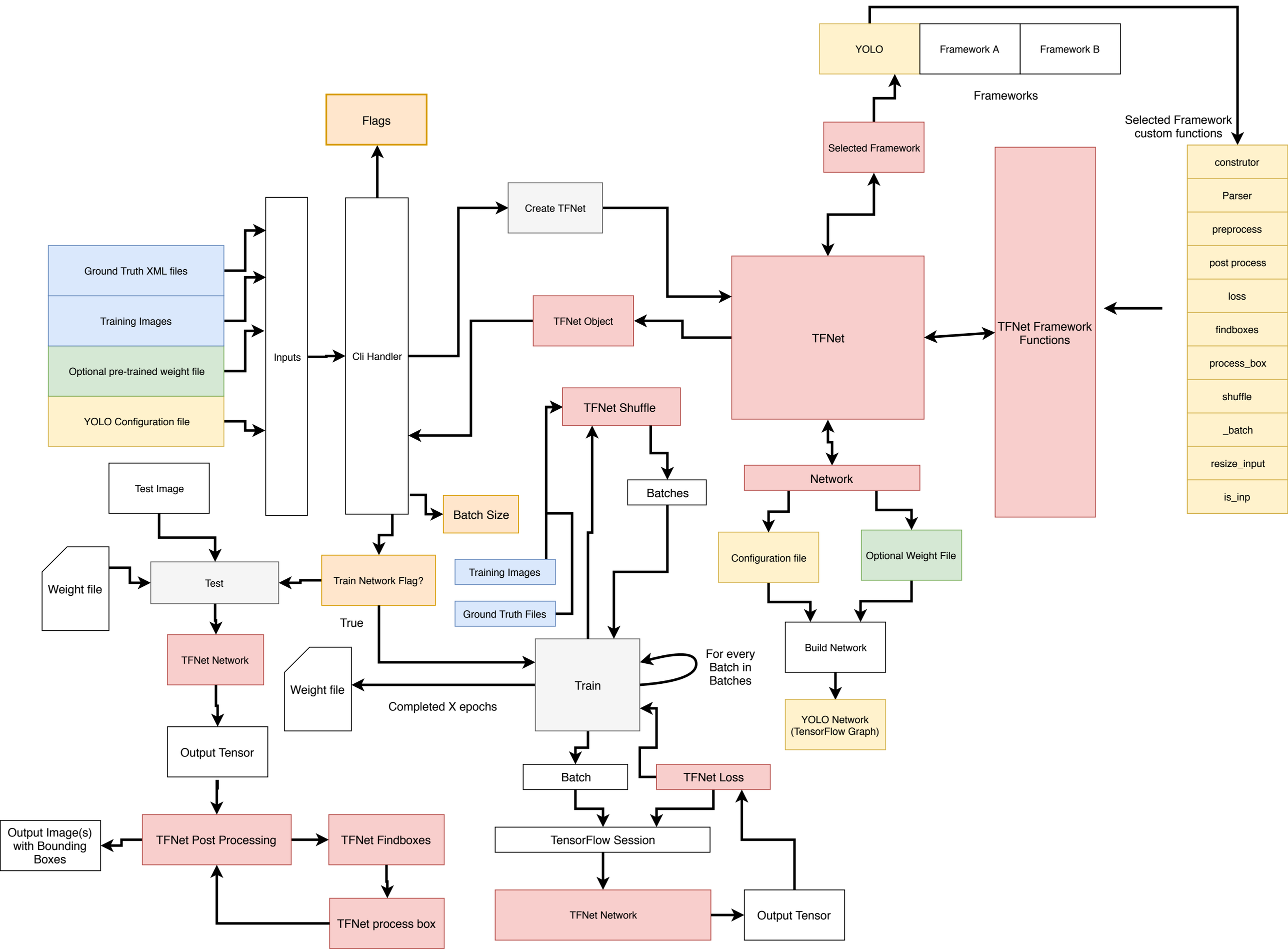
3.3 Darkflow Framework

```
class framework(object):
    constructor = vanilla.constructor
    loss = vanilla.train.loss

    def __init__(self, meta, FLAGS):
        model = basename(meta['model'])
        model = '.'.join(model.split('.')[:-1])
        meta['name'] = model

        self.constructor(meta, FLAGS)

    def is_inp(self, file_name):
        return True

class YOLO(framework):
    constructor = yolo.constructor
    parse = yolo.data.parse
    shuffle = yolo.data.shuffle
    preprocess = yolo.predict.preprocess
    postprocess = yolo.predict.postprocess
    loss = yolo.train.loss
    is_inp = yolo.misc.is_inp
    profile = yolo.misc.profile
    _batch = yolo.data._batch
    resize_input = yolo.predict.resize_input
    findboxes = yolo.predict.findboxes
    process_box = yolo.predict.process_box

types = {
    '[detection]': YOLO
}

def create_framework(meta, FLAGS):
    net_type = meta['type']
    this = types.get(net_type, framework)
    return this(meta, FLAGS)
```

Appendix D

Diagrams

4.1 Darkflow Architecture


YOLO
Framework A
Framework B
Frameworks
Flags
Selected Framework
Selected Framework
custom functions
construtor
Parser
preprocess
post process
loss
findboxes
process_box
shuffle
_batch
resize_input
is_inp
Create TFNet
Ground Truth XML files
Training Images
Optional pre-trained weight file
YOLO Configuration file
Inputs
Cli Handler
TFNet Object
TFNet
TFNet Framework
Functions
TFNet Shuffle
Batches
Network
Test Image
Batch Size
Configuration file
Optional Weight File
Weight file
Test
Train Network Flag?
Training Images
Ground Truth Files
True
Build Network
TFNet Network
Weight file
Train
For every
Batch in
Batches
Completed X epochs
YOLO Network
(TensorFlow Graph)
Output Tensor
Batch
TFNet Loss
Output Image(s)
with Bounding
Boxes
TFNet Post Processing
TFNet Findboxes
TensorFlow Session
TFNet process box
TFNet Network
Output Tensor

4.2 Darkflow Architecture with YOPO framework

Frameworks

YOLO | YOPO | Framework B

Selected Framework custom functions

Flags

Selected Framework

construtor
Parser
preprocess
post process
loss
findboxes
process_box
shuffle
_batch
resize_input
is_inp

Ground Truth XML files
Training Images
Optional pre-trained weight file
YOPO Configuration file

Inputs

Cli Handler

Create TFNet

TFNet Object

TFNet

TFNet Framework Functions

TFNet Shuffle

Batches

Network

Configuration file

Optional Weight File

Build Network

YOPO Network (TensorFlow Graph)

Test Image

Batch Size

Weight file

Test

Train Network Flag?

Training Images

Ground Truth Files

True

TFNet Network

Weight file

Completed X epochs

Train

For every Batch in Batches

Output Tensor

Batch

TFNet Loss

Output Image(s) with Bounding Boxes

TFNet Post Processing

TFNet Findboxes

TensorFlow Session

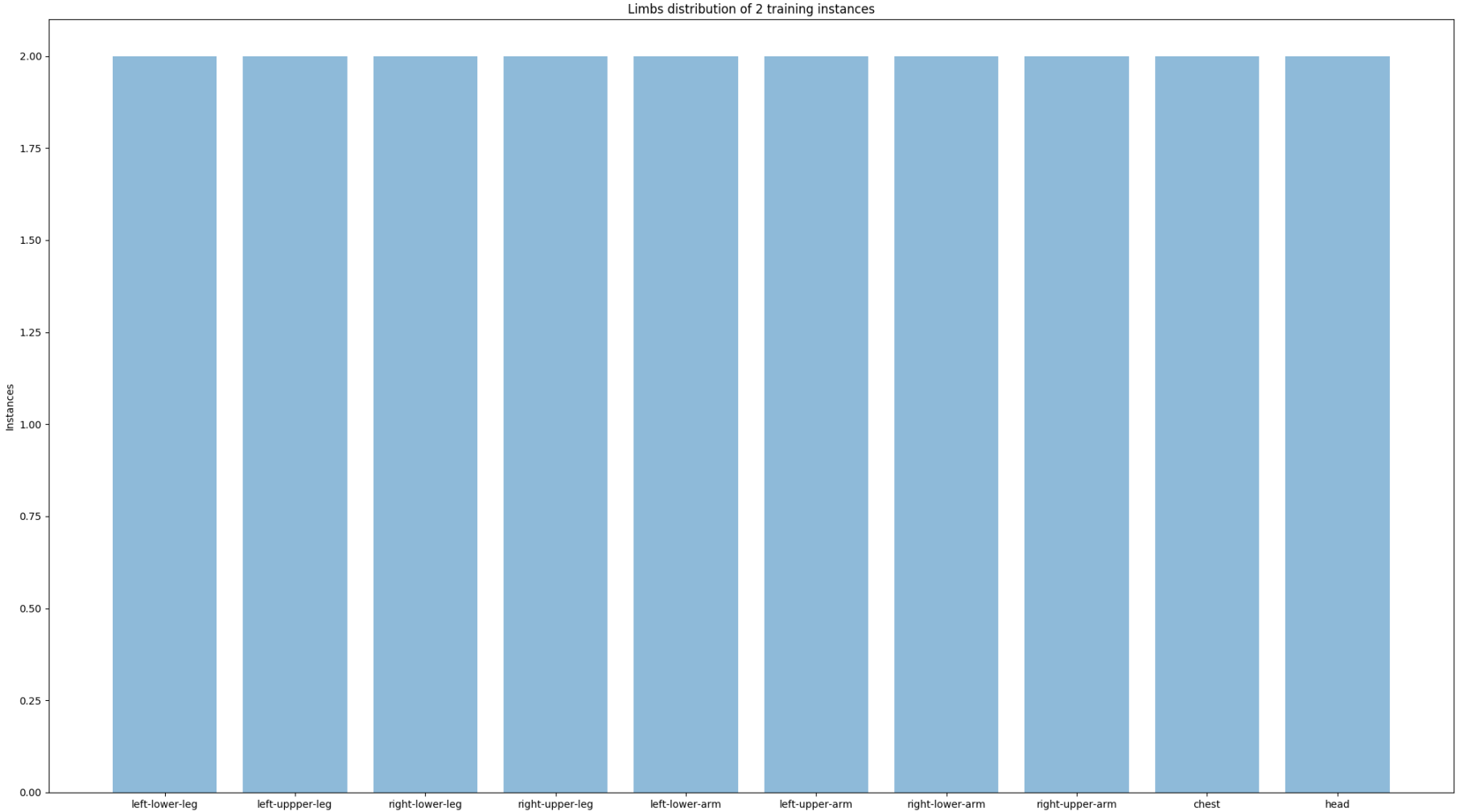
TFNet Network

Output Tensor

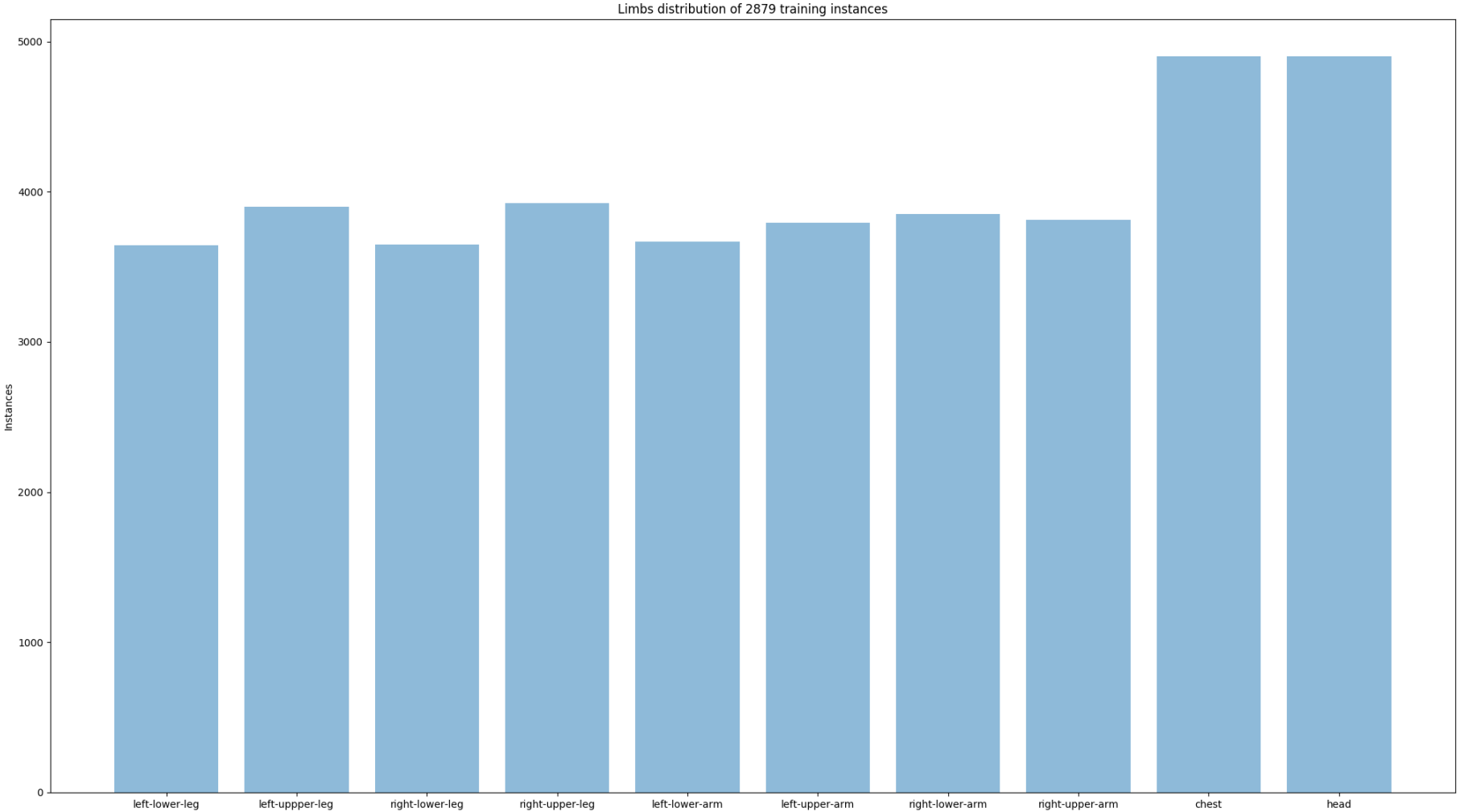
4.3 Input Image A

4.4 Input Image B

4.5 Over-fitted Class Distribution


Limbs distribution of 2 training instances
Instances
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
left-lower-leg
left-uppper-leg
right-lower-leg
right-upper-leg
left-lower-arm
left-upper-arm
right-lower-arm
right-upper-arm
chest
head

4.6 Larger dataset Class Distribution


Limbs distribution of 2879 training instances
Instances
0
1000
2000
3000
4000
5000
left-lower-leg
left-uppper-leg
right-lower-leg
right-upper-leg
left-lower-arm
left-upper-arm
right-lower-arm
right-upper-arm
chest
head

Annotated Bibliography

[1] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. [Online]. Available: http://arxiv.org/abs/1506.02640

The YOLO paper which introduces the network and algorithm of YOLO, which YOPO attempts to adapt for human pose estimation.

[2] Introducing deep learning with matlab. [Online]. Available: https://uk.mathworks.com/campaigns/products/offer/deep-learning-with-matlab.html

A ebook wrriten MathsWorks, introducing convolutional neural networks and deep learning

[3] Convolutional neural networks (cnns / convnets). [Online]. Available: http://cs231n.github.io/convolutional-networks/#convert

An image showing a example convolutional network architecture

[4] Demystify object detection. [Online]. Available: https://medium.com/@madanram/you-only-look-once-aaf841df2c7b

An image was taken from this blog post to describe how YOLO works.

[5] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results," http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

Pascal Visual object classes is annual challenge for computer vision classification and detection. It provideis a labelled dataset which YOLO uses.

[6] J. Redmon, "Darknet: Open source neural networks in c," http://pjreddie.com/darknet/, 2013–2016.

Darknet is the original framework used in the YOLOv1 paper. YOPO uses Darkflow to implement the adapted YOLO network which is an python implementation of Darknet.

[7] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, "2d human pose estimation: New benchmark and state of the art analysis," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

A human pose annotated dataset that YOPO uses to train it's model.

[8] D. A. Rosebrock, "Intersection over union (iou) for object detection," https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/, Nov. 2016, accessed April 2018.

Source for IOU equation image

[9] ——, "Non-maximum suppression in python," https://www.pyimagesearch.com/2015/02/16/faster-non-maximum-suppression-python/, Feb. 2015, accessed April 2018.

Source for NMS(Non-Maximun Supression)

[10] A. Toshev and C. Szegedy, "Deeppose: Human pose estimation via deep neural networks," *CoRR*, vol. abs/1312.4659, 2013. [Online]. Available: http://arxiv.org/abs/1312.4659

A cutting edge pose detection system called DeepPose, where the definitions of human pose estimation is referenced from.

[11] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, Sept 2010.

A sliding method for object detection that YOLO benchmarks it's self against.

[12] S. Marsland, *Machine Learning: An Algorithmic Perspective, Second Edition*, 2nd ed. Chapman & Hall/CRC, 2014, pp. 43–49.

A discussion about the perceptron network

[13] Stanford Computer Science Department. Convolutional Neural Networks (CNNs / ConvNets). http://cs231n.github.io/convolutional-networks/.

Stanford Computer Science Department lecture notes on Convolutional Neural Networks, that were very useful for learning about convolutional neural networks.

[14] Human pose evaluator dataset. [Online]. Available: http://www.robots.ox.ac.uk/~vgg/data/pose_evaluation/

Human Pose dataset based on US TV shows

[15] N. Jammalamadaka, A. Zisserman, M. Eichner, V. Ferrari, and C. V. Jawahar, "Has my algorithm succeeded? an evaluator for human pose estimators," in *European Conference on Computer Vision*, 2012.

A Paper discussing the evaluation of Human Pose Estimators

[16] S. Johnson and M. Everingham, "Clustered pose and nonlinear appearance models for human pose estimation," in *Proceedings of the British Machine Vision Conference*, 2010, doi:10.5244/C.24.12.

A human pose dataset containing 2000 images of a wide range of poses

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[18] Jira — issue & project tracking software — atlassian. [Online]. Available: https://www.atlassian.com/software/jira

Atlassian JIRA - the online Agile Scrum platform I used for managing this project.

[19] Travis ci - test and deploy with confidence. [Online]. Available: https://travis-ci.com/

Travis CI. Hosted continuous integration solution with Github integration. Used by this project.

[20] Sonarqube. [Online]. Available: https://www.sonarqube.org/

Continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, code smells and security vulnerabilities

[21] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," in *ACM SIGGRAPH 2008 Classes*, ser. SIGGRAPH '08. New York, NY, USA: ACM, 2008, pp. 16:1–16:14. [Online]. Available: http://doi.acm.org/10.1145/1401132.1401152

[22] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

A machine learning library that Darkflow used and it was also used to build YOPO.

[23] "Tolaso", "Greens theorem and area of polygons," https://www.math.tolaso.com.gr/?p=1451, Dec. 2017, accessed April 2018.

A blog post describing the proof and applications of Green's Theorem, that was used for the area calculations in YOPO's IOU function.

[24] T. E. Oliphant, "Python for scientific computing," *Computing in Science Engineering*, vol. 9, no. 3, pp. 10–20, May 2007.

Python library that makes it easier and more efficient to use multidimensional arrays

[25] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, "Cython: The best of both worlds," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 31–39, 2011. [Online]. Available: https://aip.scitation.org/doi/abs/10.1109/MCSE.2010.118

The Cython language is a superset of the Python language that additionally supports calling C functions and declaring C types on variables and class attributes. This allows the compiler to generate very efficient C code from Cython code.

[26] C. Thompson and L. Shure, *Image Processing Toolbox: For Use with MATLAB;[user's Guide]*. MathWorks, 1995.

A tool kit for machine learning, analysing data and creating models. The ground Truth data for each of the purposed dataset were in a MatLab data file format.

[27] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for Python," 2001–, [Online; accessed May 2018]. [Online]. Available: http://www.scipy.org/

SciPy provides many user-friendly and efficient numerical routines such as routines for numerical integration and optimization. This is used in the preprocessing of the datasets.

[28] *Commun. ACM*, vol. 35, no. 7, 1992.

A clipping algorithm that PyClipper and therefore Clipper use to find intersection and union shapes.

[29] A. Johnson, M. Chalton, L. Treyer, and G. Ratajc. (2015, Mar) pyclipper. [Online]. Available: https://pypi.org/project/pyclipper/

Pyclipper is a Cython wrapper exposing public functions and classes of the C translation of the Angus Johnsons Clipper library (ver. 6.4.2)

[30] A. Johnson. [Online]. Available: http://www.angusj.com/delphi/clipper.php

The Clipper library performs line polygon clipping - intersection, union, difference exclusive-or, and line polygon offsetting. The library is based on Vatti's clipping algorithm.

[31] P. S. Foundation. How fast are exceptions? [Online]. Available: https://docs.python.org/2/faq/design.html#how-fast-are-exceptions

Python documentation detailing the speed of exceptions.

[32] Pycharm: A cross-platform ide for python by jetbrains. [Online]. Available: https://www.jetbrains.com/pycharm/

Pycharm integrated development environment. By code editor and debugging environment of choice, used throughout this project.

[33] Sonarqube cloud. [Online]. Available: https://about.sonarcloud.io/

Cloud platform that hosts sonarqube reports of analysis of source code.

[34] Human pose evaluator dataset. [Online]. Available: https://www.tensorflow.org/programmers_guide/graph_viz

TensorBoard is a TensorFlow add-on that allows the network to view in real-time and monitor different statics of the network. In this project the loss function was graph and this was used for further analysis of the performance of the network.

[35] Google cloud. https://cloud.google.com/.

Google Cloud VM instances, used for parts of the development and training process

[36] M.-C. Popescu, V. E. Balas, L. Perescu-Popescu, and N. Mastorakis, "Multilayer perceptron and neural networks," *WSEAS Trans. Cir. and Sys.*, vol. 8, no. 7, pp. 579–588, July 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1639537.1639542

A useful book that discusses Multilayer perceptron neural networks

[37] A gentle introduction to transfer learning for deep learning. [Online]. Available: https://machinelearningmastery.com/transfer-learning-for-deep-learning/

A blog post that discusses transfer learning and which YOPO uses to training it's network.