Provectus Robotics Solutions Convolutional Neural Network (PRSNet)

Object Detection Research Paper

By: Liam Gritters

Table of Contents

1.0 Introduction

Provectus Robotics Solutions Convolutional Neural Network (PRSNet) is a framework of classes and functions based off of Darknet/Yolo network by PJ Reddie [1]. It was designed and developed to detect people and vehicles in a video stream.

Convolutional Neural Networks (ConvNets) specialize in processing data where spatial relationships are of importance [2]. Examples of input data would be images, speech recordings, and sentence structure. In each of these examples the sequence and positioning of the data is crucial to understanding differences between the inputs.

For image analysis there are three challenges; classification, object detection, and segmentation [2]. Classification is the ability to state what is in an image but not necessarily where the object is located in that image. Object detection states both where and what is an image. Segmentation determines the shape of the predicted object in the image, and is the most challenging of the 3. In all of these categories a predetermined set of objects are used, this is known as the network's classes. For example in PRSNet, the goal is to detect people and vehicles so the classes would be people, vehicles and other. Prior to processing, the input images are split by row and concatenated to form a tensor. This is done to improve computationally speed and efficiency.

The basic premise behind convolutional neural networks is simple; pass data through the network and adjust parameters to decrease error. This process is iterated thousands of times until the parameters are being adjusted by very small amounts. This end result is known as convergence, and is the overall goal of ConvNets. To achieve convergence or something close to it, typically the structure or architecture of the ConvNet should contain multiple layers, denoted as the depth of the network. These layers usually consist of the following; Convolutional Layer, Activation Layer, Pooling Layer, Softmax Layer, and a Fully Connected Layer [2]. However PRSNet has a few other layers; Batchnorm Layer, Route Layer, Reshape Layer, and Region Layer.

2.0 Prior Research

Before deciding upon using the YOLO convolutional neural network, other object detection methods and other ConvNets were considered.

2.1 Haar Feature Extractor (Viola-Jones Detector)

Commonly used in face detection, this method takes a training set of images of faces and random other images without faces, and extracts features from them [3]. For this extraction process, Haar Features are used (figure 1).

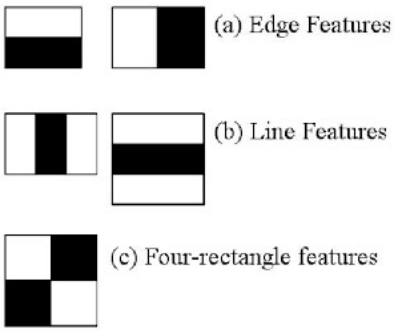


Figure 1: Example of Haar Features

These features pass over the image subtracting the sum of pixels under the white rectangle from the sum of pixels under the black rectangle. This process outputs thousands of features that are then filtered and selected using the Adaboost equation [3]. This equation finds the best threshold which will classify the faces to postive and negative by iterating over the training set and adjusting the features to have the smallest error rate.

Once the Haar Features have been determined they are grouped into different detection stages to filter sections of an image. If that section passes the first stage of filters then then next stage is applied. This speeds up filtering of sections of an image that don't have a face in it [3].

This method of detection works well for images of people facing forward and small variations in brightness. However this method is not very robust with respect to orientation or high contrast of brightness. Moreover, it produces a high amount of false positives.

2.2 HoG-SVM

HoG-SVM aims to be more robust in full body detection and slight changes in orientation and brightness [4]. Like the Haar Feature Extractor, HoG-SVM uses an edge filter to extract features from an image. This edge filter is also called a Sobel operator (figure 2) and consists of two vectors that slide over an image in the x and y directions.

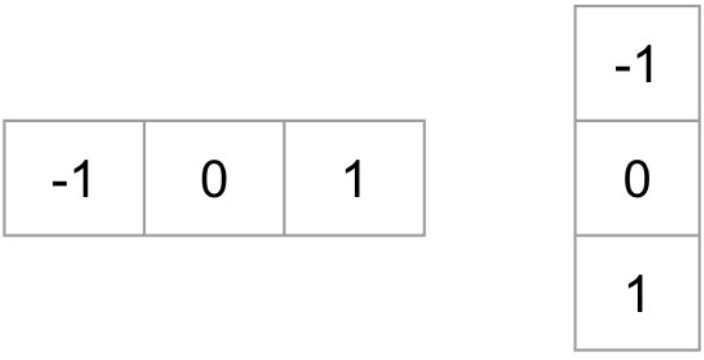


Figure 2: Sobel operator in the x (right) and y (left) directions

The output of these features are then used to calculated the magnitude and direction of the edge or gradient of a line in the image. The gradient and magnitude are then sorted into a histogram of oriented gradients (HoG) that describes the shape of the object. This process is performed on many images of people and other objects and are then trained using a support vector machine (SVM).

A SVM maps these features in higher dimensions and determines a separating boundary [4]. Figure 3 shows a basic concept of SVMs.

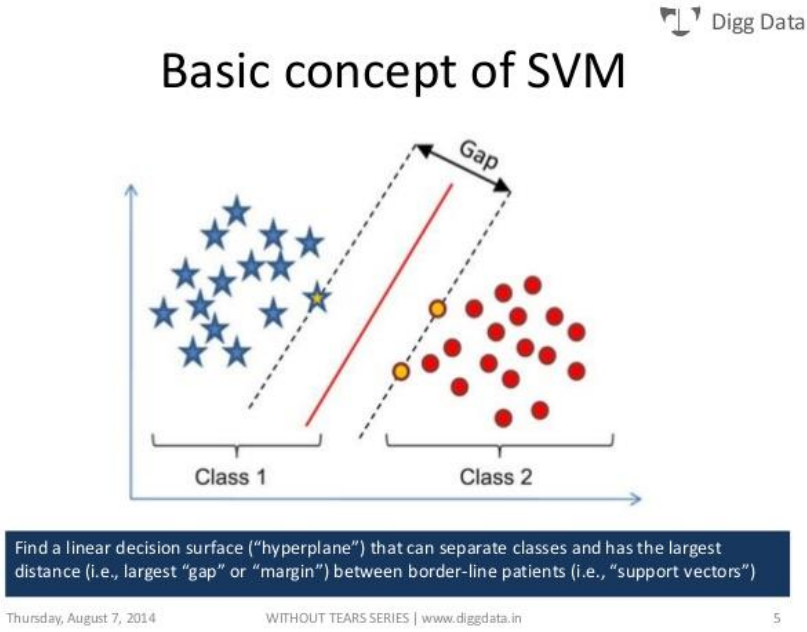


Figure 3: Basic example of support vector machines

Once a boundary has been determined, the detector can be tested on an image containing a person. A section of the image is processed and mapped against the boundary to determine what class the section falls within. Another section a few pixels over is then processed and this continues until the entire image has been evaluated. This procedure of evaluating sections of an image at at time is known as a sliding window.

Although HoG-SVMs are good at detecting people in an image, they produce a high amount of false positives and struggle with changing orientations and obscured views of people. Additionally determining the right tuning parameters is very difficult for a dynamic range of backgrounds and weather conditions.

2.3 AlexNet

AlexNet is the one of the first breakthrough convolutional neural networks. It performed incredibly well compared to other traditional computer vision classifiers at the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [5] and began a deep-learning revolution. Compared to previous ConvNets, such as LeNet, AlexNet is much larger and includes convolutional layers stacked on top of each other.

AlexNet and many other convolutional networks since, such as GoogLeNet, VGGNet and ResNet, have all shown to be amazing classifiers even outperforming humans in some instances, but unfortunately are unable to perform object detection with realistic speeds.

2.4 Region-Convolutional Neural Network (R-CNN)

A few years after AlexNet , with the advance in computing capabilities in GPUs, R-CNNs were developed to deal with object detection. They essentially propose regions of interests using lower level detectors prior to passing the information into a ConvNet. There are three papers which build off of each other, improving accuracy and speed; R-CNN, Fast-R-CNN, Faster-R-CNN [6].

However a few problems limited R-CNNs from being used. They are unfortunately too slow at inference time compared to YOLO. Training the data is a difficult process resulting in inconsistencies. Finally, the region proposal process limits the network in detecting complex objects.
This research ultimately led me to the Darknet framework and YOLO convolutional neural network which PRSNet is based off of. YOLO (You Only Look Once) is able to detect objects in real time and scores a high value of 76.8% mean average precision (mAP), a measurement classifiers and object detectors are evaluated by.

3.0 Layers

As stated earlier, a convolutional neural network consists of multiple layers. The arrangement and use of each layer construct the overall architecture of the network. The output of each layer is known as an activation map and acts as the input to the layer after it. In this manner, the initial input data is propagated through the entire network, where its is manipulated to determine its final outcome. Before this is possible, the network must first be trained to learn what objects are. In this process the data is propagated through the network, an error is calculated, which back propagates through the network and updates all of it's parameters. Commonly when discussing the size of convolutional neural networks only the convolutional layers are considered. This is because all the other layers act as supporting layers to the convolutional layer. For instance in PRSNet there are 73 layers but only 19 of those layers are convolutional layers, so the network is considered to have a size or depth of 19. The following sections will delve deeper into what each of these layers do that make this incredible concept possible.

3.1 Convolutional Layer

Convolutional Layers are by far the most important layer. They consist of filters or kernels that pass over the input data, in this case an image, and extract important features from the image [2]. The process of these filters passing over the image is called convolution.

An image is represented by pixels which have values from 0-255. In the figure below this is represented with pixel values of 0-1 for simplicity. A colour image has three channels; blue, green, red all with a matrix of pixels.

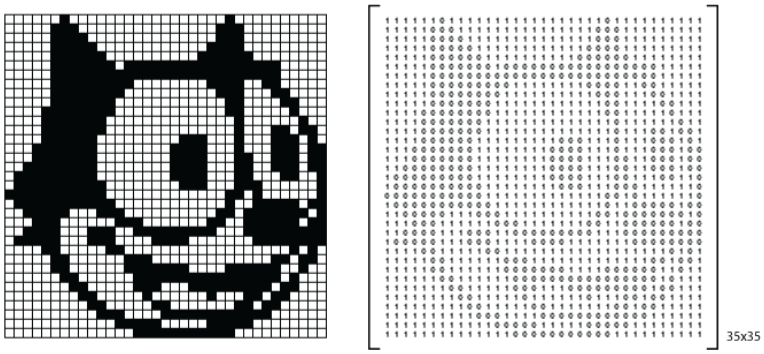


Figure 1: image of cat represent by pixel value of 0-1

Like images, filters are a matrix of values as well. Filters vary in size and stride, but typically remain square. Figure 2 shows the process of an a colour image with three channels being convoluted by two filters. In this example Filter W0 and W1 have a size of 3x3 and a stride of 2. Since there are 3 input channels, each filter has to match it and thus have a depth of 3 channels. For example in PRSNet the first convolutional layer outputs 32 channels, receives 3 channels, has a size of 3 and a stride of 1. There would then be 3 x 32 x 3 x 3 filter values. Additionally before convolution , in order to maintain size between convolutions, zero-padding is added around the edges of the input data. This can be seen in figure 2 as well. Throughout PRSNet, padding is used in convolutional layers.

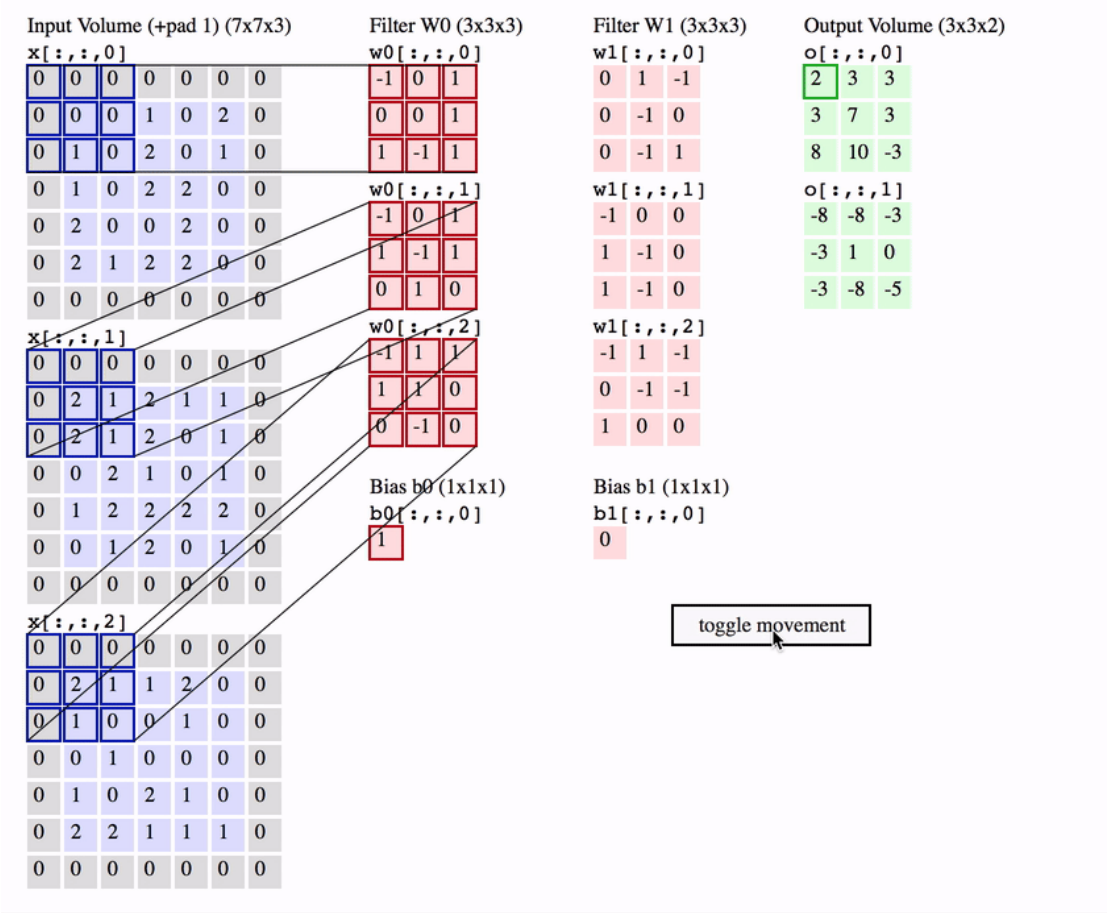


Figure 2: Input image being convoluted by two filters

When the filters pass over the input matrix, they perform the dot product between the filter and region of interest of the input channel. This is shown in figure 3.

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \cdot \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} = a_1b_1 + a_2b_2 + a_3b_3 + a_4b_4$$

Figure 3: Dot product of two matrices

The output of the dot product of each filter channel is then summed together with the bias of that filter. The filter then moves two spaces over (stride of 2) and performs the same process again. The output is represented by the following equation:

$$y_{i,j} = w_k \cdot x_{i,j} + b_k$$

The values inside the filter matrix are known as weights. These values are set to random values at the beginning of training and are adjusted in the correction phase, also known as back-propagation, of the network. As stated earlier the goal is for the network to converge and reduce the amount of error as much as possible. However, this is a very difficult task to achieve, and many steps are taken to prevent divergence [2]. One of them is choosing an appropriate starting range for the weights. For PRSNet, the initialization bounds of the weights is chosen using the "He Normal Initialization Equation" [7]. This initialization practice was made popular in ResNet [8] and helps prevent the weights from either exponentially increasing or decreasing with every correction, which both lead to divergence. In the paper "Delving Deep into Rectifiers," where He Normalization is first announced, the authors state the variance of weights to be:

$$\text{Var}(w(i)) = 2 / ni.$$

Where w(i) are the weights of layer i and n_i is the input of layer i computed as:

$$n_i = K^2 * C$$

With K being the kernel size and C representing the number of input channels.

This equation helps not only reduce divergence but also speed up convergence [8]. Moreover, another significant aspect of "He Normal Initialization" is that it takes into account the use of Rectified Linear Units (ReLU) [7], a function that is used after convolutional layers in PRSNet and is discussed later in this document.

Alternatively, it is also common practice to use pre-trained weights of other networks as a starting point [2]. Regardless of what objects the network was trained on, the first couple of layers of every network are similar, each detecting edges, contrast in colours and brightness. These pre-trained weights will typically replace the first couple of layers of a network, with all other layers being initialized with He Normal Initialization or another weight initialization equation.

3.1.1 Back Propagation

Once the image or input data has passed through the network an error value is calculated at the end of the network. The error or loss function is stated as:

$$J(w) = \frac{1}{2} * (y_i - \sum w_i * x_i)^2$$

This calculation is done in the Region Layer of PRSNet which will be discussed further in the Region Layer section. Once the error is calculated, it is back-propagated through the network so each convolutional layer is able to update their weights and bias values.

Traditionally the updated weights are calculated using Gradient Descent (GD) [9], where the loss represented by J(w) is plotted against the weight value shown in figure 4.

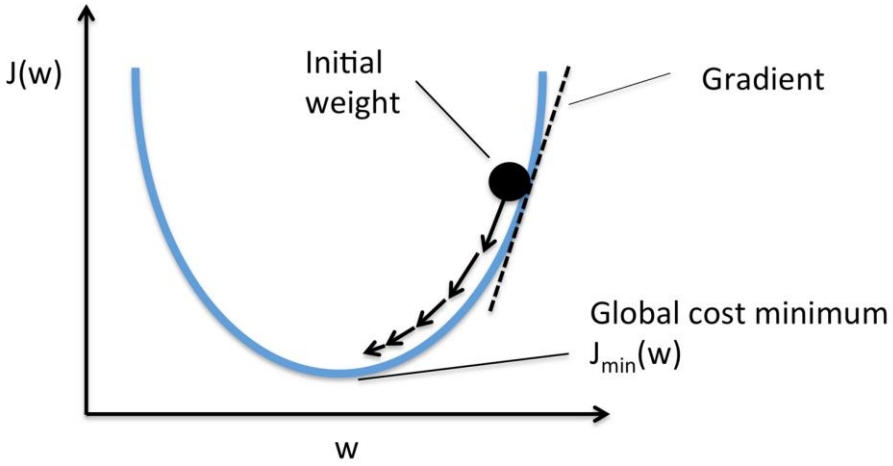


Figure 4: Simple graph representing Gradient Descent

The slope or gradient of J(w) represented as ∇J(w), would then predict the direction in which the weight value should change. The equation for GD is:

$$w_f = w_j - \alpha * \nabla J(w_i)$$

Here, α represents the learning weight, which acts as a scaling factor for each update.

Unlike GD which computes the gradient of the loss function for the entire training data set, Stochastic Gradient Descent (SGD) updates parameters/weights for each training example [9]. The equation for SGD is:

$$w_j = w_{j-1} - \alpha * \nabla J(w_{j-1}, x_i, y_i)$$

where i represents each image and j represents each iteration. This is a much faster and more memory conservative approach. However, since SGD performs frequent updates with high variance, it can cause the objective function to fluctuate heavily and converge at a local minimum as opposed to a global minimum [9], as shown in figure 5.

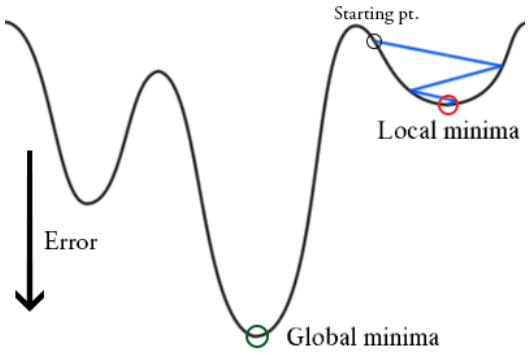


Figure 5: SGD converges at local minimum as opposed to global minimum

To combat this problem, Mini-Batch Gradient Descent (MBGD) takes the positives of both GD and SGD and combines them. MBGD takes a batch of images, 64 in the case of PRSNet, and computes the gradient of the loss function after each batch. This reduces the variance of parameter updates, leading to more stable convergence, while also keeping the efficiency of SGD, and obtaining a global minimum. MBGD is represented as:

$$w_j = w_{j-1} - \alpha * \nabla J(w_j, x_{i:i+n}, y_{i:i+n})$$

where n represents amount of images in a batch.

Adaptive Moment Estimation (Adam) Optimization builds off of these concepts as well as other update optimization equations to speed up convergence and improve global minimum achievement [10]. Adam computes the adaptive learning rates for each parameter. It stores an exponentially decaying average of past gradients and square gradients represented by m(t) and v(t). In other words m(t) is an estimate of the first moment or mean and v(t) is an estimate of the second moment or uncentered variance. These variables are calculates as:

$$m_t = \mathrm{B}_1 * m_{t-1} + (1 - \mathrm{B}_1) * g_t$$
$$v_t = \mathrm{B}_2 * v_{t-1} + (1 - \mathrm{B}_2) * g_t^2$$

where t represents each iteration, g_t represents the gradient of the previous layer, B_1 equals 0.9 and B_2 equals 0.999. m(t) and v(t) are initialized as zero vectors, but because of this, it was observed that they are biased towards zero [10]. This is counteracted by computing bias corrected first and second moment estimates:

$$\widehat{m_t} = \frac{m_t}{1 - \mathrm{B}_1^t}$$
$$\widehat{v_t} = \frac{m_t}{1 - \mathrm{B}_2^t}$$

The parameters are then updated as:

$$w_j = w_{j-1} - \frac{\alpha}{\sqrt{\widehat{v_t}} - \varepsilon} * \widehat{m_t}$$

where ε equals 10^{-8}. Adam Optimization is the algorithm used to update weights and bias values in PRSNet.

3.2 BatchNorm Layer

BatchNorm or batch normalization is a layer that follows the convolutional layer and acts as a replacement to adding the bias values after convolution [11]. BatchNorm normalizes the data in the entire batch; speeding up convergence and reducing covariance shift [12]. Covariance shift is a phenomenon that occurs between training and testing data. For instance, lets say PRSNet was trained mostly on images of black cats and a bunch of images that were not cats. Then when testing on images of orange cats, the network would predict very poorly. This is because the distribution of training images were skewed to mostly detect black cats and would fail when a slightly shifted distribution of cats is used for testing. Batch normalization aims to centre the input distribution. By doing this higher learning rates can be used, because each update wont fluctuate the error so drastically [12]. Moreover, batch normalization helps with over-fitting because it regularizes the data [11].

To achieve this improvement, batch normalization includes four trainable parameters; scale, shift, running mean and running variance. Running mean $\mu(N)$ and running variance $\sigma^2(N)$ are vectors that learn to represent the mean and variance of an entire data set (N) not just each mini-batch (B).

The scale and shift parameters are applied after normalizing. Since normalizing each input of a layer constrains what the data can represent, the scale and shift parameters adjust the normalized output to properly represent the distribution of the input data. For instance if a sigmoid curve (figure 6) were to be normalized, the output would be constrained to the linear section of the curve [12]. To address this, the scale and shift parameters make sure to represent the identity of the data.

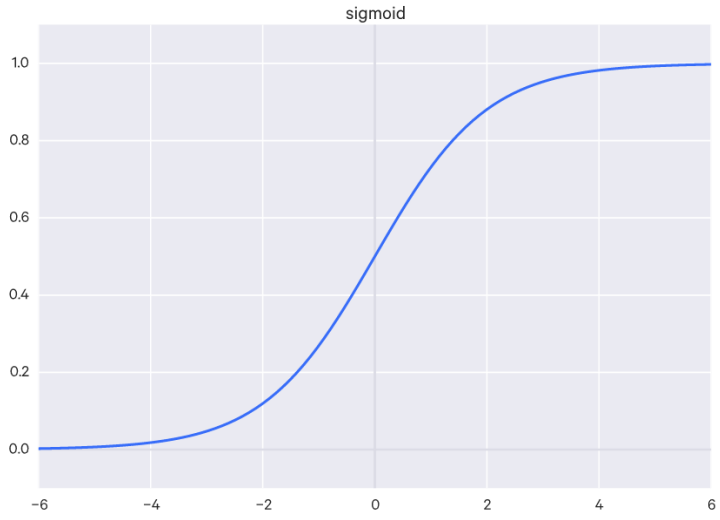


Figure 7: sigmoid curve, the middle section of the curve is practically linear

The process of training a batch normalization layer is shown in figure 8 [11].

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Input: Network N with trainable parameters Θ;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$
Output: Batch-normalized network for inference, $N_{\mathrm{BN}}^{\mathrm{inf}}$

1: $N_{\mathrm{BN}}^{\mathrm{tr}} \leftarrow N$ // Training BN network
2: **for** $k = 1 \ldots K$ **do**
3: Add transformation $y^{(k)} = \mathrm{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\mathrm{BN}}^{\mathrm{tr}}$ (Alg. 1)
4: Modify each layer in $N_{\mathrm{BN}}^{\mathrm{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
5: **end for**
6: Train $N_{\mathrm{BN}}^{\mathrm{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
7: $N_{\mathrm{BN}}^{\mathrm{inf}} \leftarrow N_{\mathrm{BN}}^{\mathrm{tr}}$ // Inference BN network with frozen // parameters
8: **for** $k = 1 \ldots K$ **do**
9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
10: Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:
$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma^2_{\mathcal{B}}]$$
11: In $N_{\mathrm{BN}}^{\mathrm{inf}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$
12: **end for**

Algorithm 2: Training a Batch-Normalized Network

Figure 8: Equations to train batch normalization parameters

Batch normalization training and inference (testing) vary since during testing, multiple images are not processed but rather just 1 image. This is where the learnt values of the running mean and running variance are used. This is shown in figure 8, Algorithm 2, step 11; with running mean = E[x] and running variance = Var[x].

3.3 Activation Layer

The activation layer follows the batch normalization layer or in the absence of a batch normalization layer, the convolutional layer. An activation is a non-linear function that acts on each input value separately [13]. Activation functions mimic the neurons firing or activating in a brain. They introduce non-linear properties to a network, to help model data more accurately. For instance, without activation functions, the output signal of a network would be a simple linear function, and wouldn't be able to model complex systems [13].

There are 4 different activation functions available in PRSNet.

3.3.1 Logistic or Sigmoid Activation

This activation function takes a number and squashes it into a range of 0 to 1 (figure 9). This activation is commonly used in the final layers to constrain the output to a probability of 0 to 100 percent. There are some drawbacks of using this function however. Since the region of the sigmoid curve is nearly flat at 0 and 1, the derivative is approximately zero, and thus during back propagation the weights don't change significantly in these regions. This problem is known as the vanishing gradient problem for activation functions. Additionally, the exponential function in the sigmoid activation equation is computationally expensive and slows down the network [13].

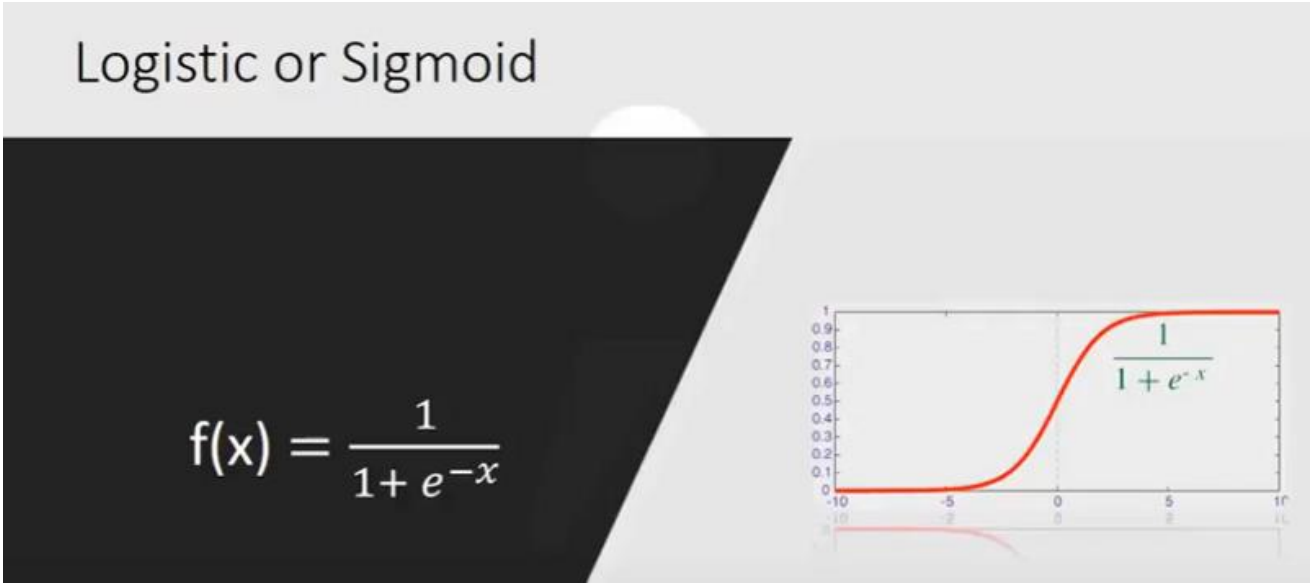


Figure 9: Logistic (sigmoid) activation curve and function

3.3.2 ReLU Activation

Rectified Linear Unit or ReLU is one of the most popular activation functions. It is simply stated as:

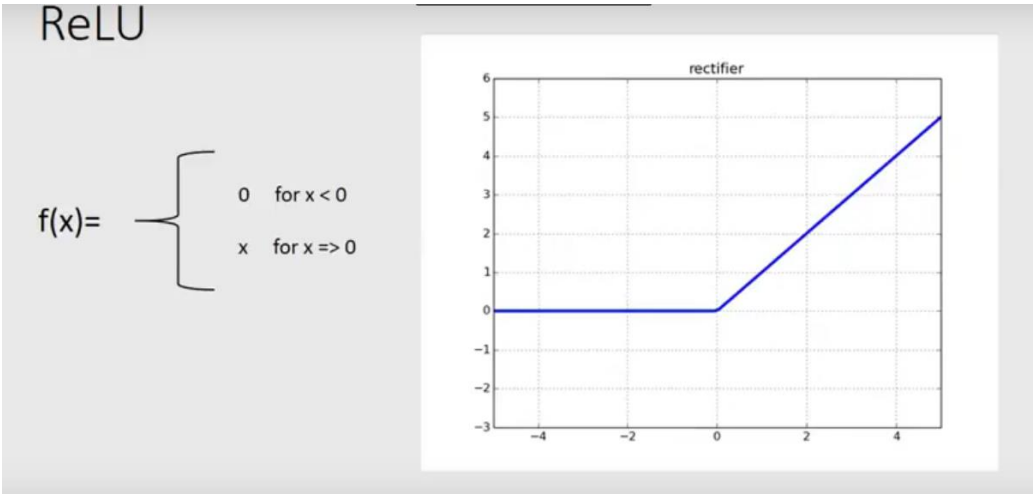


Figure 10: Rectified Linear Unit function and graph

It thresholds negative values and propagates positive values. It has been proven to improve convergence up to 6 times that of the Tanh activation function [14]. The ReLU activation function attempts to solve the vanishing gradient problem in the positive region, while still remaining non-linear due to the threshold bound at zero. Moreover, the function dilutes each activation, speeding up and stabilizing convergence [14]. However there are still drawbacks of the ReLU activation; the outputs are not zero-centred and the vanishing gradient problem still remains when $x < 0$.

3.3.3 Leaky ReLU Activation

This activation function attempts to solve the vanishing gradient problem of ReLU. As shown in figure 11, when x < 0, it is multiplied by 0.01 and propagated through the network.

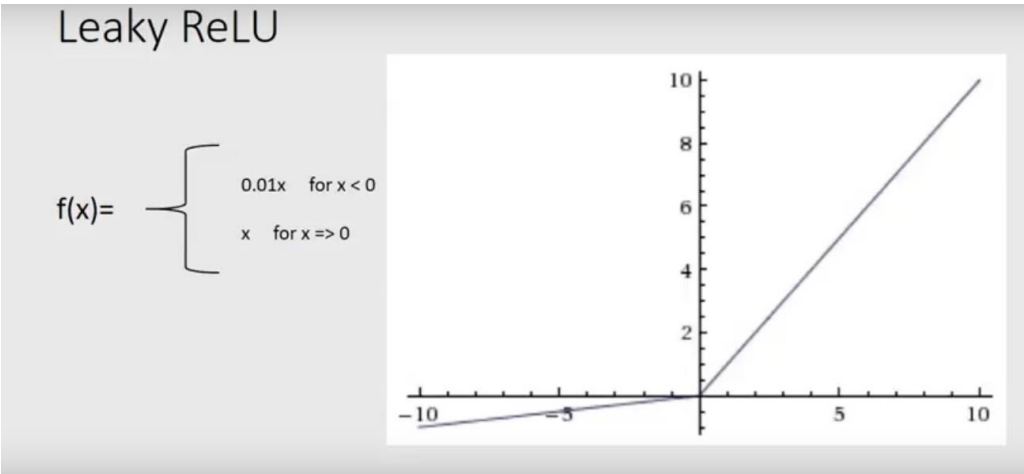


Figure 11: Leaky Rectified Linear Unit function and graph

Leaky ReLU's tend to be inconsistent in their results and are still being researched [13]. This inconsistency is currently unclear. Leaky ReLU's are used in the current PRSNet architecture.

3.3.4 Softmax Activation

The Softmax activation is a generalization of the logistic function that squashes down a set of values to a range of [0, 1]. The function outputs the probability of a value compared to all the other values in the set and is a form of normalization [13]. This is represented by the following equation:

$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

The softmax activation is used at the end of the network as it able to calculate the probability of each class based off of all the class values.

3.4 Pooling Layer

Pooling layers typically follow an activation layer, but are not used after every activation layer. Pooling layers act as a down-sampling layer and are essential in PRSNet. The reasoning behind this layer is to decrease memory size and increase speed of the network [1]. Furthermore, the layer also reduces over-fitting in spatial positioning [2].

There are two different types of Pooling Layers; Average Pooling and Max-Pooling. Average pooling takes the average of all the values in a certain window and propagates that further. Max-pooling, the more commonly used type, takes the maximum value within a certain window. This is shown in figure 12:

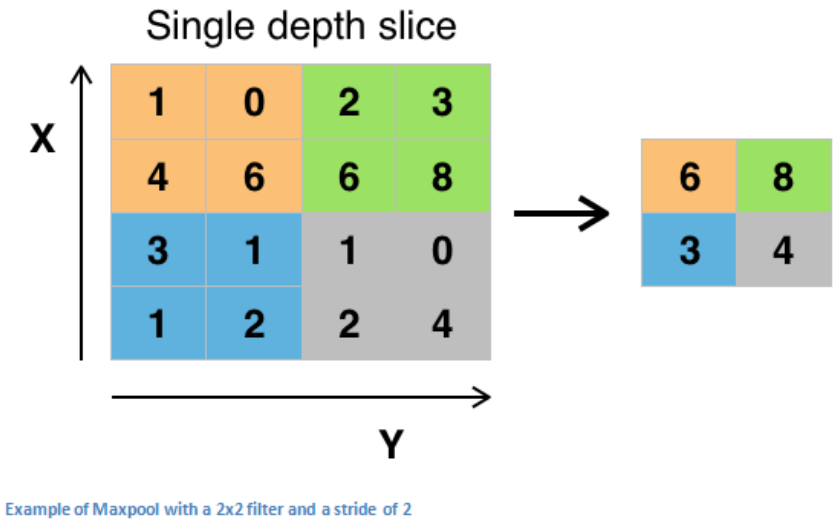


Figure 12: example of Max-pooling window with a size of 2x2 and a stride of 2

In this example, a window size of 2x2 is used, where the largest value in this window is chosen as the output. The window then moves two units over where the process is repeated. In PRSNet Max-pooling layers are used to down-sample a 416x416 image to 13x13. Every Max-pool layer has a size of 2x2 with a stride of 2, thus reducing the input size by a factor of 2.

3.5 Routing Layer

The Routing Layer acts as a transport for finer grained features. The layer essentially copies the output of a layer earlier in the network and adds it later to the network. This layer helps emphasize earlier features later in the network and helps prevent the propagated values from decaying [1][8]. However, this pass-through feature takes a considerable amount of memory as the whole activation map of a previous layer is copied.

3.6 Reshape Layer

Reshape or Reorganize layer is typically used along side the routing layer. Like Max-pooling, the Reshape Layer reduces the spatial size, but unlike max-pooling it maintains the extra data as channels. This process is show in figure 13.

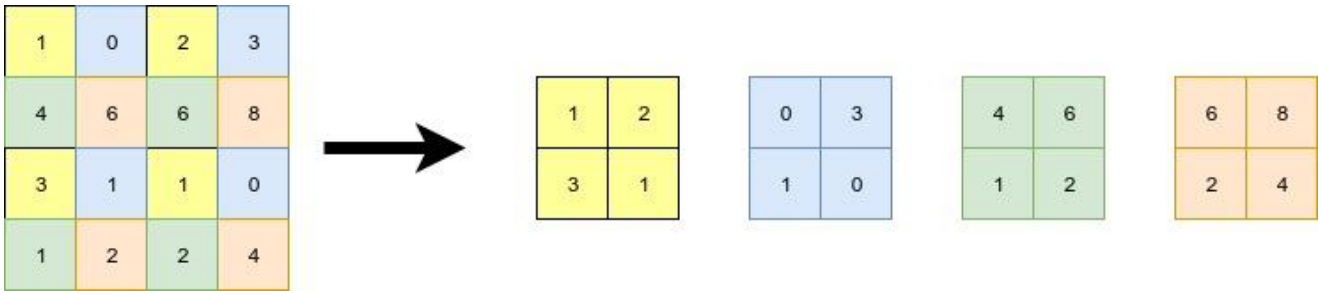


Figure 13: Example of a Reshape Layer

As shown in figure 13, the input matrix is a single channel of size 4x4, and is restructured as 4 channels with a size of 2x2. In this example the reshape layer has a stride of 2.

3.7 Region Layer

The region layer is the final layer of PRSNet and is based off of the region layer in the YOLO object detector [1]. This layer shapes the neural network from being a classier to being an object detector. As stated earlier in this document the loss function is calculated in this layer but in a more complex way than a typical convolutional neural network classifier.

The convolutional neural network down-samples a 416 x 416 image to 13 x 13 pixels. This can be represented as dividing the input image into 13 x 13 feature boxes as shown in figure 14.

Figure 14: Representation of input image divided into 13 x 13 feature boxes

Each of these feature boxes predicts 5 bounding boxes just for their cell. Each of those bounding boxes predict five coordinates and a confidence value, p(C), for each class. The five coordinates consist of a centroid position (bx, by), bounding box size (bw, bh), and object probability (p(O)). The object probability states how certain it is that an object is within the feature box regardless of what object it is. The class probability or confidence states how certain that that specific object is within that cell if an object exists there. The overall layout is shown in figure 15.

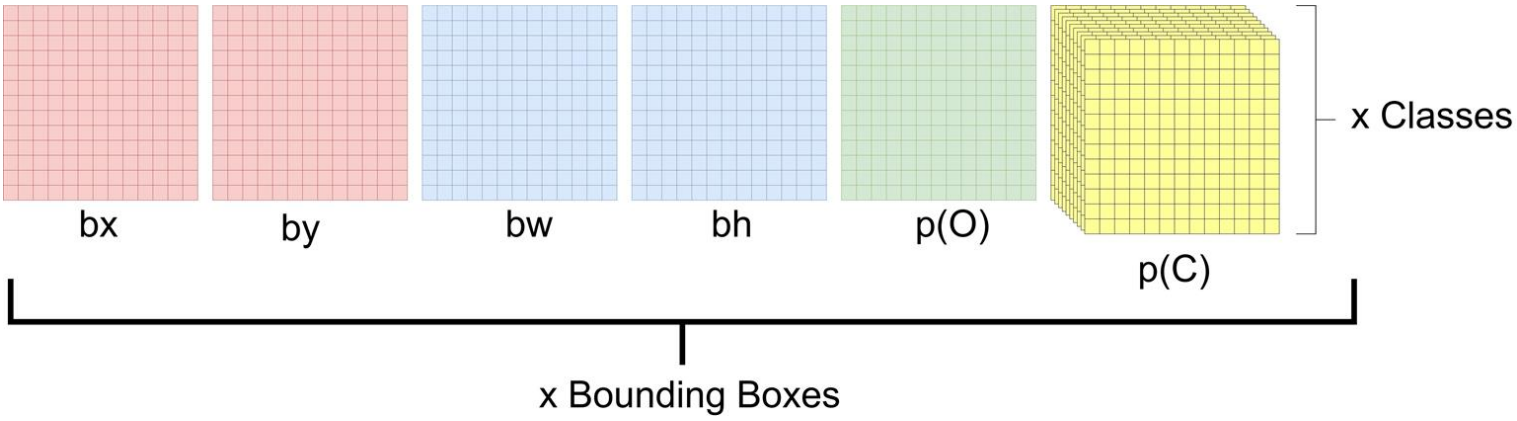


Figure 15: Region Layer channel structure

In the PRSNet architecture the input dimensions of the region layer is 13 pixel width x 13 pixel height x 5 bounding boxes x (5 coordinates + 20 classes) for an overall size of 13 x 13 x 125. The region layer applies a logistic activation to bx,by to constrain them to a range of [0,1]. A logistic activation is also applied to the object probability in order to constrain the values to a probability of 0 to 100

percent. Moreover a softmax activation is applied to the class probabilities to determine which class is most likely within the bounding box. This process is the same for inference and training.

3.7.1 Training in the Region Layer

The region layer is what makes object detection possible in one forward pass and why YOLO is so unique [1]. However, you aren't able to just add a region layer at the end of a trained convolutional network and expect it to detect objects. This is because object detection is trained into the network, by the way errors are determined and back propagated.

The 125 channels that are taken as input are terrible at detection at the beginning of the training and are each taught to perform their predicting tasks. To speed up this process and maintain stability of the network, the first layers of the network are pre-trained on a classifier and are then trained on the object detector network [1]. A prediction of an object is calculated for each cell in each of the 5 bounding boxes and is checked against a labelled truth. If the prediction is wrong or if the network was not able to properly determine the location of the object, an error is calculated and stored in a error vector that is then back-propagated through the network.

A good location is determined by having a high intersection over union (IOU) value (figure 16) and an accurate centroid prediction.

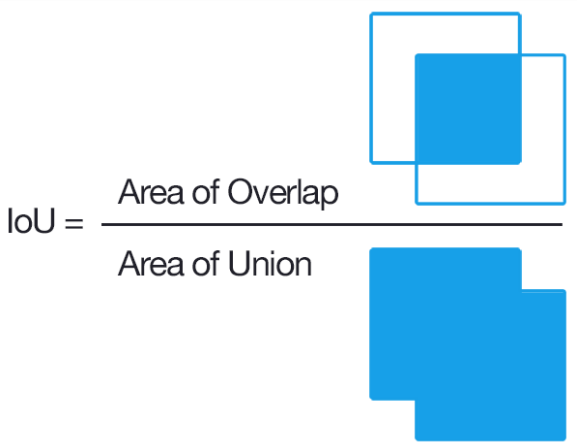


Figure 16: Graphic demonstrating intersection over union calculation

To improve IOU predicting, dimension clusters are used. Dimension clusters are essentially an estimate of common bounding boxes sizes within the entire training set. These sizes are determined prior to training by running k-means clustering on all the labelled bounding boxes [1]. K-means clustering is a type of unsupervised learning algorithm that relates data points to each other [15]. K represents the amount of clusters used to relate a data set. Each cluster acts as a centroid with respect to the other data points and the algorithm tries to minimize the distances from each centroid to all the other data points. This is demonstrated in figure 17.

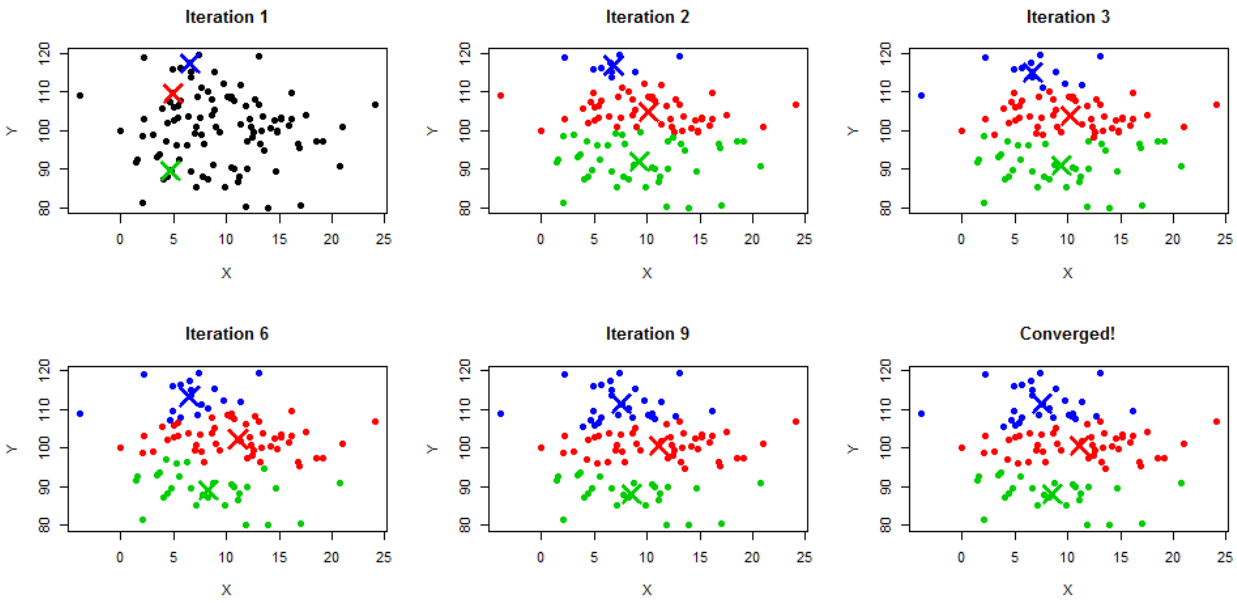


Figure 17: Example of K-Means Clustering with K = 3

The algorithm follows the following steps:
Let X = {x_1,x_2,x_3,……..,x_n} be the set of data points and V = {v_1,v_2,…….,v_c} be the set of centres.

1) Randomly select '*c*' cluster centres.

2) Calculate the distance between each data point and cluster centres.

3) Assign the data point to the cluster centre whose distance from the cluster centre is minimum of all the cluster centres..

4) Recalculate the new cluster centre using:

$$v_i = (1/c_i) \sum_{j=1}^{c_i} x_i$$

where, 'c_i' represents the number of data points in i^{th} cluster.

5) Recalculate the distance between each data point and newly obtained cluster centres.

6) If no data point was reassigned then stop, otherwise repeat from step 3).

The bx and by coordinates predict the centroid of a predicted object. With dimension clusters acting as good initial estimates, the bw and bh predictions adjust these dimension clusters to more precisely fit the predicted object. The object probability, p(O), then states how probable that this predicted object in this bounding box is actually an object. If this predicted object is not actually an object an error is calculated. If there is an object in that location but the bounding box does not match up with the labelled bounding box an error value is calculated for the coordinate parameters; bx, by , bw, bh. If however, the model passes both of these checks, the predicted object is then compared to the actual labelled object, where an error value is calculated based upon the prediction. This process is iterated

over and over again until there isn't anymore training data or the model converges. In this manner, the network learns to detect objects, accurately locate the centroid, fit an appropriately sized bounding box and then classify the type of object.

3.7.2 Inference in the Region Layer

In inference of the region layer the object probability is multiplied by each class probability for each feature box. If this product is higher than a set threshold and all other class probability products for that feature box, the prediction is added to a vector of other predictions. Essentially, the best prediction for each feature box is propagated further if it above a certain threshold. All of these predictions are then filtered through a non-maximum suppression algorithm. This algorithm discards multiple bounding box predictions of the same class in the same area. This is shown in figure 18.

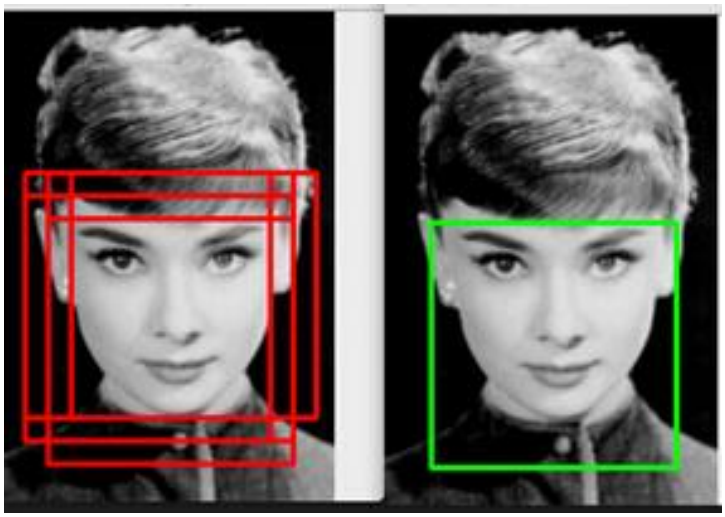


Figure 18: Example of non-maximum suppression of bounding boxes predicting a face

The bounding boxes are then displayed over the input image and the next frame or image is taken.

4.0 Data Augmentation

Data augmentation is the manipulation and distortion of in the input images. This vastly improves the robustness of the network and its ability to detect objects [16]. Additionally it provides more training data for the network to learn from. There are a few common data augmentation methods used in convolutional neural networks.

4.1 Image Flips

Flipping or mirroring images 50 percent of the time allows objects to appear on both the sides of the image and provides a different perspective for the ConvNet to model.

4.2 Random Crops

Randomly cropping a portion of the image allows the network to learn sections of an object, reducing over-fitting and improving the detection rate.

4.3 Colour Jittering

Colour jittering alters the original input colour by a specified hue, saturation and gain. This distorts the input colour channels to account for a change in brightness and contract in testing.

5.0 PRSNet Dependencies

PRSNet uses multiple libraries to function. These libraries include:

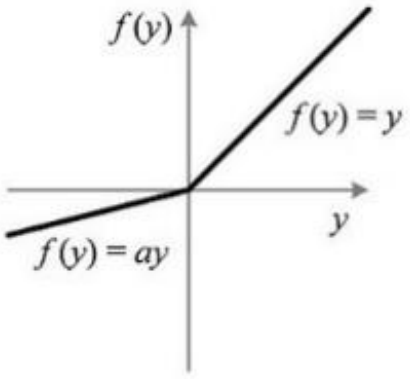
1. CUDA Toolkit 8 (Feb Release)
2. CUDA Deep Neural Network Library (cuDNN) version 6
3. OpenCV (core, highgui, imgproc)
4. libxml2
5. libconfig.h++
6. FFMPEG Library (avutil, avcodec, avformat, swscale)

6.0 Next Steps

Currently PRSNet is based off the YOLO architecture but there are a few implementations that can be done to improve the detection rate of the network.

6.1 PReLU Activation

Parametric Rectified Linear Units (PReLU) are generalized versions of Leaky ReLUs. This concept is demonstrated in figure 19.



$$PReLU(x_i) = \begin{cases} x_i & if\, x_i > 0 \\ a_i x_i & if\, x_i \leq 0 \end{cases}$$

Figure 19: PReLU function and plot

In this activation function the alpha parameter is learnt during the training phase of the convolutional neural network. They have been shown to improve the mean average precision of a network by 1 percent [14]. However, this improvement comes at a marginal computation cost during training.

6.2 Recurrent Neural Networks and LTSM

One of the problems with ConvNets is that they don't have any prior knowledge of the previous detection. This is a problem in video analysis or real time detection as the network can miss a detection every couple frames. Recurrent Neural Networks (RNN), or more precisely Long Short-Term Machines (LTSM) solve this problem. They work in a similar manner to ConvNets but instead of just propagating the output to the next layer in the network, they also store this output to be analyzed again. A simple RNN is shown in figure 20.

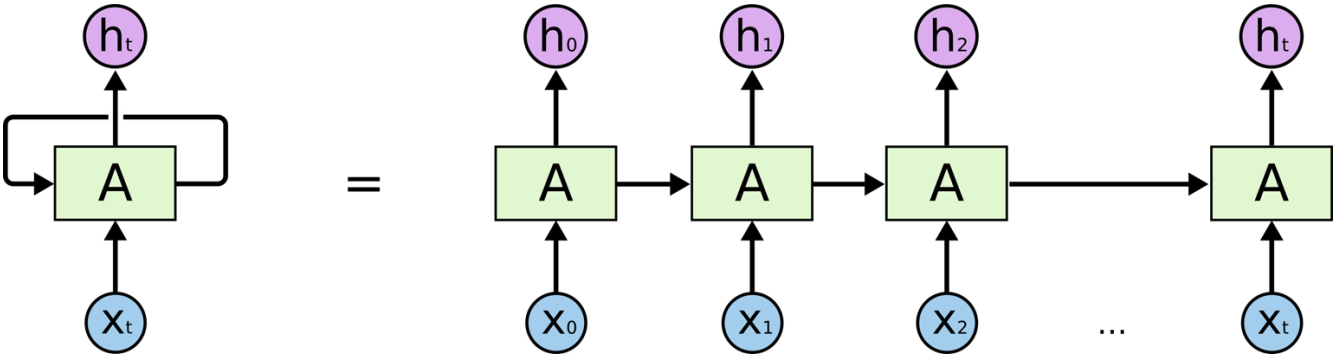


Figure 20: Example of a one layer Recurrent Neural Network (right) and the same layer expanded out (left)

ROLO is an adapted version of the YOLO neural network with an LSTM network running at the end to track previous detections [17].

6.3 Generative Adversarial Network (GAN)

Generative Adversarial Networks are the ultimate form of data augmentation [16]. GANs are taught to create similar images of objects trained on a network [18]. They work against the original network to attempt to trick the network into not detecting an object that is actually there. They are an incredibly interesting concept that is still being researched but trials have shown to improve a networks ability to detect objects without labelling images, thus becoming a form of unsupervised learning.

7.0 Conclusion

Convolutional neural networks are at the forefront of machine learning and have revolutionized the field of image classification and object detection. Although the use of each of the layers is understood, the overall architecture of each network is still explored in order to obtain a sufficient model of the data. Further improvements in activation functions and weight optimization will improve the efficiency and accuracy of networks in the future. Moreover, advances in GPUs will allow for deeper and more complex networks that will benefit the entire artificial intelligence field. PRSNet will therefore continue to improve and is why convolutional neural networks were ultimately chosen as the best object detector to use.

8.0 References

1: Joseph Redmon, Ali Farhadi, YOLO9000: Better, Faster, Stronger , 2016

2: Karpathy, CS231n Convolutional Neural Networks for Visual Recognition, 2016, http://cs231n.github.io/convolutional-networks/

3: N/A, Face Detection using Haar Cascades, 2017,

4: Hilton Bristow, Simon Lucey, Why do linear SVMs trained on HOG features perform so well?, 2014

5: Alex Krizhevsky , Ilya Sutskever, Geoffrey E Hinton, ImageNet Classification with Deep Convolutional Neural Networks, 2012

6: Shaoqing Ren, Kaiming He, Ross B. Girshick, Jian Sun, Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, 2015

7: Isaac Changhau , Weight Initialization in Artificial Neural Networks , 2017, https://isaacchanghau.github.io/2017/05/24/Weight-Initialization-in-Artificial-Neural-Networks/

8: Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition, 2015

9: Sebastian Ruder, An overview of gradient descent optimization algorithms, 2016

10: Diederik P. Kingma, Jimmy Ba, Adam: A Method for Stochastic Optimization, 2014

11: Karl N, Batch Normalization—What the hey?, 2016, https://gab41.lab41.org/batch-normalization-what-the-hey-d480039a9e3b

12: Sergey Ioffe, Christian Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015

13: Aditya Sharma, Understanding Activation Functions in Deep Learning, 2017, https://www.learnopencv.com/understanding-activation-functions-in-deep-learning/

14: Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, 2015

15: Azad Naik, K-Means Clustering Algorithm, 2011,

16: Luis Perez, Jason Wang, The Effectiveness of Data Augmentation in Image Classification using Deep Learning, 2017

17: Guanghan Ning, Zhi Zhang, Chen Huang, Zhihai He, Xiaobo Ren, Haohong Wang, Spatially Supervised Recurrent Convolutional Neural Networks for Visual Object Tracking, 2016

18: Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, Generative Adversarial Networks, 2014