

GAN BASED OBJECT DETECTION OF NOISY IMAGES

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE AWARD OF THE DEGREE

OF

MASTER OF TECHNOLOGY

IN

INFORMATION SYSTEMS

Submitted by:

Jayanthi Adilakshmi Visali

2K17/ISY/07

Under the supervision of

Dr. ANIL SINGH PARIHAR



INFORMATION TECHNOLOGY

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi- 110042

JULY 2019

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi- 110042

CANDIDATE'S DECLARATION

I, Jayanthi Adilakshmi Visali, 2K17/ISY/07 student of M.Tech (Information System), hereby declare that the project Dissertation titled “GAN Based Object Detection of Noisy Images” which is submitted by me to the Department of Information Technology, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associate ship, Fellowship or other similar title or recognition.

Place: Delhi

JAYANTHI ADILAKSHMI VISALI

Date:

INFORMATION TECHNOLOGY
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi- 110042

CERTIFICATE

I hereby certify that the project Dissertation titled “GAN Based Object Detection of Noisy Images” which is submitted by Jayanthi Adilakshmi Visali, 2K17/ISY/07 to Department of Information Technology, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of degree of Master of Technology, is a record of the project work carried out by the student under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree of Diploma to this University or elsewhere.

Place: Delhi

Dr. ANIL SINGH PARIHAR

Date:

SUPERVISOR

Associate Professor

Department of Computer Science and Engineering

Delhi Technological University

(Formerly Delhi College of Engineering)

Bawana Road, Delhi- 110042

ACKNOWLEDGEMENT

First of all, I would like to express my deep sense of respect and gratitude to my project supervisor Dr. Anil Singh Parihar for providing the opportunity of carrying out this project and being the guiding force behind this work. I am deeply indebted to him for the support, advice and encouragement he provided without which the project could not have been a success.

Secondly, I am grateful to Dr. Kapil Sharma, HOD, Information Technology Department, DTU for his immense support. I would also like to acknowledge Delhi Technological University library and staff for providing the right academic resources and environment for this work to be carried out.

Last but not the least I would like to express sincere gratitude to my parents and friends for constantly encouraging me during the completion of work.

JAYANTHI ADILAKSHMI VISALI

ABSTRACT

Due to increased use of CGI imagery in many application across different fields, it is high time there is a Generative Adversarial Network which worked in synergy with an object detection algorithm like YOLO to overcome the difficulty in perception of various kinds of noises contributing to the decreased accuracy of object class prediction. The problem with CGI imagery in real time is that they are replete with objects which are of different proportions when compared to real life objects and also they anthropomorphize every type of object like cars, trees, houses, toys which makes it difficult for default anchor boxes to act as good priors in drawing the bounding boxes around them. So we have integrated a network which can generate denoised images with the help of generative and discriminator networks competing against each other and the generated denoised image will directly be pushed through another object detection network which here is YOLOv3 with improved IoU. So with the integration of these networks and tuning the parameters for our custom dataset the output will be an image which can be used for real-time rendering.

CONTENTS

Candidate's Declaration	i
Certificate	ii
Acknowledgement	iii
Abstract	iv
Contents	v
List of Tables	viii
List of Figures	ix
List of Abbreviations and Nomenclature	xii
CHAPTER 1 INRODUCTION	1
1.1 IMAGES, NOISE AND PHOTO-REALISTIC RENDERING	1
1.2 OBJECT DETECTION	3
1.3 DEEP LEARNING & CNNs	3
1.3.1 Convolutional Neural Networks (CNNs)	4
1.3.2 Spatial Arrangement	6
1.4 PROBLEM STATEMENT	8
CHAPTER 2 LITERATURE REVIEW	9
2.1 GENERATIVE ADVERSARIAL NETWORKS (GANS)	9
2.1.1 Algorithm	10
2.1.2 SR-GANs	11
2.1.3 Residual Blocks	12
2.1.4 Why SRGAN for Denoising?	14

2.2	OBJECT DETECTION USING YOU ONLY LOOK ONCE ALGORITHM	14
2.2.1	YOLO V 1	15
2.2.2	YOLO V 2	16
2.2.3	K-Means Clustering Algorithm	18
2.2.4	YOLO V 3	19
2.2.5	Need for Custom Anchor Boxes	20
2.2.6	Intersection Over Union(IoU)	21
2.3	RELU	21
2.3.1	Simple ReLU	22
2.3.2	Drawbacks of ReLU	22
2.3.3	Leaky ReLU	23
2.4	TRANSFER LEARNING	24
	CHAPTER 3 PROPOSED WORK	25
3.1	THE PIPELINE	25
3.2	IMAGE DENOISING USING GAN	26
3.2.1	Gaussian Distribution	26
3.2.2	Uniform Distribution	27
3.3	OBJECT DETECTION USING YOLO VERSION 3 WITH IMPROVED IOU	28
3.3.1	Tuning Leaky ReLU for the Improvement of Detection Accuracy	28
3.3.2	Generating custom anchor boxes for an improved IoU	30
3.4	EXPERIMENTAL SETUP	32
3.4.1	Data Set	32
3.4.2	Hardware Used	33

3.4.3	Software Used	34
CHAPTER 4 RESULTS AND DISCUSSION		35
4.1	RESULTS	35
4.1.1	Denoising of Photo-Realistic Rendered Images with Monochromatic Gaussian Noise Using GAN	35
4.1.2	Denoising of Photo-realistic Rendered Images with Monochromatic Uniform Noise Using GAN	36
4.1.3	Denoising of CT Scan and MRI Images with Monochromatic Gaussian Noise Using GAN	37
4.1.4	Denoising of CT Scan and MRI Images with Monochromatic Uniform Noise Using GAN	38
4.1.5	Denoising of Camera-taken Real Noisy Images using GAN	39
4.1.6	Object Detection in Photo-Realistic Rendered Images Using YOLO Version 3 with Improved IoU	39
4.1.7	Object Detection in Denoised Images Using YOLO Version 3 with Improved IoU	40
4.1.8	Object Detection in Photos Taken from Digital Camera Using YOLO Version 3 with Improved IoU	41
4.2	FALSE POSITIVES AND FALSE NEGATIVES	43
4.2.1	False Positives	43
4.2.2	False Negatives	43
CHAPTER 5 CONCLUSION		45
CHAPTER 6 FUTURE WORK		46
REFERENCES		47
LIST OF PUBLICATIONS		50

LIST OF TABLES

TABLE 2.1. DarkNet - 19	17
TABLE 2.2. DarkNet - 53	20

LIST OF FIGURES

FIGURE NO.	FIGURE	PAGE NO.
FIGURE. 1.1	Example of a Photo-realistic Rendered Image	2
FIGURE. 1.2	Basic network structure of Neural Networks	4
FIGURE. 1.3	Understanding the internal functioning of CNN	5
FIGURE. 1.4	Illustration of Average Pool layer	7
FIGURE. 1.5	Illustration of the Max Pool layer	7
FIGURE. 2.1	Generative Adversarial Network	10
FIGURE. 2.2	SR-GAN	12
FIGURE. 2.3	Residual Blocks	13
FIGURE. 2.4	YOLO Network	16
FIGURE. 2.5	K-Means Clustering: No. of Centroids vs. Mean IoU	19
FIGURE. 2.6	Illustration of IoU	21
FIGURE. 2.7	ReLU Activation Function	22
FIGURE. 2.8	Leaky ReLU/PReLU Activation Function	23
FIGURE. 3.1	The Block Diagram	25

FIGURE. 3.2	Gaussian Distribution	27
FIGURE. 3.3	Uniform Distribution	27
FIGURE. 3.4	Leaky_relu @ 0.090	28
FIGURE. 3.5	Leaky_relu @ 0.095	28
FIGURE. 3.6	Leaky_relu @ 0.100	29
FIGURE. 3.7	Leaky_relu @ 0.105	29
FIGURE. 3.8	Leaky_relu @ 0.110	29
FIGURE. 3.9	Leaky_relu @ 0.120	29
FIGURE. 3.10	Leaky_relu @ 0.111	30
FIGURE. 3.11	Mean IoU with VOC dataset(5)	31
FIGURE. 3.12	Mean IoU with custom dataset(5)	31
FIGURE. 3.13	Mean IoU with VOC dataset(9)	31
FIGURE. 3.14	Mean IoU with custom dataset(9)	31
FIGURE. 3.15	Default anchor box sizes visualized for representative purpose	32
FIGURE. 3.16	Custom anchor box sizes visualized for representative purpose	33
FIGURE. 4.1	GAN results on photo-realistic rendered images with monochromatic gaussian noise	35
FIGURE. 4.2	GAN results on photo-realistic rendered images with monochromatic uniform noise	36

FIGURE. 4.3	GAN results on CT Scan images with monochromatic gaussian noise	37
FIGURE. 4.4	GAN results on MRI images with monochromatic gaussian noise	37
FIGURE. 4.5	GAN results on CT Scan images with monochromatic uniform noise	38
FIGURE. 4.6	GAN results on MRI images with monochromatic uniform noise	38
FIGURE. 4.7	GAN results on real-noisy images	39
FIGURE. 4.8	Object Detection for Photo-Realistic Rendered Image 1	40
FIGURE. 4.9	Object Detection for Denoised Image 1	40
FIGURE. 4.10	Object Detection for Denoised Image 2	41
FIGURE. 4.11	Object Detection for Denoised Image 3	41
FIGURE. 4.12	Object Detection for Denoised Digital Image	42
FIGURE. 4.13	Example of False Positive	43
FIGURE. 4.14	Example of False Negative	44

LIST OF ABBREVIATIONS AND NOMENCLATURE

1. CNN: Convolutional Neural Network
2. GAN: Generative Adversarial Network
3. YOLO: You Only Look Once
4. CGI: Computer Generated Imagery
5. MRI: Magnetic Resonance Imaging
6. CT: Computed Tomography
7. SR-GAN: Super Resolution Generative Adversarial Network
8. GPU: Graphical Processing Unit
9. API: Application Programming Interface
10. ReLU: Rectified Linear Unit
12. PReLU: Parametric Rectified Linear Unit
13. R-CNN: Regions with CNN features
14. IoU: Intersection Over Union
15. HOG: Histogram of Oriented Gradients
16. SVM: Support Vector Machine
17. DL: Deep Learning
18. ML: Machine Learning
19. AI: Artificial Intelligence

20. LR: Low Resolution
21. HR: High Resolution
22. SR: Super Resolution
23. PDF: Probability Density Function
24. ISO: International Organization of Standardization
25. ANN: Artificial Neural Network
26. VOC: Visual Object Classes
27. COCO: Common Objects in Contexts
28. VGG: Visual Geometry Group
29. MSE: Mean Squared Error
30. NLP: Natural Language Processing
31. CV: Computer Vision
32. PSNR: Peak Signal-to-Noise Ratio
33. FC: Fully Connected
34. CONV: Convolution
35. v1: Version 1
36. v2: Version 2
37. v3: Version 3
38. CPU: Central Processing Unit
39. RMSE: Root Mean Square Error

- 40. FPN: Feature Pyramid Network
- 41. W: Input Volume Size
- 42. K: Kernel Field Size
- 43. S: Stride
- 44. P: Amount of Zero Padding applied

CHAPTER 1

INTRODUCTION

1.1 IMAGES, NOISE AND PHOTO-REALISTIC RENDERING

An image, derived from Latin word ‘Imago’ means an artifact that depicts , such as a 2D picture or a photograph. In , an image is nothing but a distributed amplitude of color(s). Digital Images are nothing but a collection of picture elements called pixels which are arranged in a rectangular array. These images may naturally contain noise in them due to external factors such as shooting at higher ISO settings, light available, etc. We can add whichever noise we deem necessary to the image externally. There are various noises like salt-and-pepper, gaussian, speckle, shot, poisson noises, etc. Image noise is nothing more than variations of brightness or information related to the specific color in the images, and is usually a part of the electronic noise. It could potentially be produced by the sensor of a digital camera or the circuit work in scanner of the same. Most common type of noise that is used in image processing are Gaussian Noise, which is named after the German mathematician Carl Friedrich Gauss and Uniform Noise. Statistically, Gaussian noise has the probability density function (PDF) same as that of normal distribution, also known as Gaussian Distribution, which means that the values of the noise have been distributed normally. Many natural sources for Gaussian noise are present like black-body radiation from the Earth and other warm objects, shot noise, the thermal vibrations of atoms in conductors also known as thermal or Johnson-Nyquist noise, and celestial sources such as Sun. It can also be added into the image externally by image processing techniques. The gaussian noise in images is mainly caused during acquisition e.g. sensor noise caused by high temperature and/or poor illumination.

Computer Generated Imagery(CGI) has become a part of many daily life applications such as video games, movies and commercials. Many software like Pixar’s Renderman [1] software are available to generate photo-realistic imagery.

Nowadays, animation movie companies like Dreamworks and Pixar use a technique called pathtracing [2] for rendering their 3D scenes. Pathtracing is a technique in which hundreds of rays are directed into a single pixel randomly (achieved by “Monte Carlo” Method) to create high quality photo-realistic frames. The technique of pathtracing is well-described in Section 3.2. and the color of the pixel is obtained by averaging the colors generated by these rays, and this process is repeated for each and every pixel. Figure 1.1 is an example photo-realistic image taken from the Pixar movie Monsters University [3]. Frame by frame rendering of photo-realistic images is time consuming and very expensive as we need thousands of rays per pixel and thus increased computational complexity. With the development of large memories, efficient software APIs and CPUs, GPUs technology, there has been a great progress in rendering, but real-time rendering is less feasible as it takes around 1/3-2/3 of a whole day to render a single frame. We can apply image denoising methods to almost any scanned image, CGI or camera-captured image with noise.



Fig. 1.1 Example of a Photo-Realistic Rendered Image

1.2 OBJECT DETECTION

We, the humans have an amazing ability to identify objects that are in any image we look at, the location of the objects and relation of each other in the scene. Making a computer master this ability of human vision, which is to know what is where by looking as fast and as accurate as a human does is a complex problem which is widely studied as the field of Computer Vision. Even as humans, it is sometimes difficult to distinguish what we are seeing when the objects are too similar like differentiating between two dogs belonging to different breeds in retriever dogs like Nova Scotia Duck Tolling Retriever breed and Golden Retriever breed or when only some part of an object is only present in an image like head of an animal. Finding exactly where in a given image an object is present is called the problem of Object Localization. The problem of where an object is present and what it is in a given image is called Object Detection. Another challenging tasks of object detection is doing object detection when too many objects are present in the image.

With the advances in technology in the Computer Vision, Object Detection has become an integral part of many real-time applications. Object Detection has become a requisite in many problems of Computer Science and Artificial Intelligence(AI) like Self-driving cars, Responsive Robotics, medical applications, video surveillance, etc. Many algorithms have been discovered in order to solve the problem of object detection. Before Neural Networks, methods like Support Vector Machine and HOG are used for Object Detection. Nowadays, many efficient deep learning algorithms have been invented for detecting objects in an image.

1.3 DEEP LEARNING AND CNNs

With the advances in technology, computer software and hardware, Deep Learning(DL) has seen an incredible growth in the last two decades. There is an immense work going on this field. Artificial Neural Networks(ANN) have solved many complex problems till date. For unstructured data like images, video, text and audio, ANNs have been widely used. Especially for images, CNNs have showed a

better performance in solving many problems of image processing like image denoising, super resolution of images, deraining in images, coloring the images, etc. than traditional algorithms. Many promising results for the problems of Computer Vision like Image Localization and Classification, Semantic Segmentation, Object Detection Instance Segmentation, etc., have been given by CNNs. For the problem of object detection alone, many algorithms, both region-proposal and single shot detectors like R-CNN [4], Fast R-CNN [5], Faster R-CNN [6], YOLO Version 1 [7] [8], YOLO version 2 [7] [9], YOLO Version 3 [7] [10], SSD [11], etc., have been proposed. Convolutional Neural Networks are similar in architecture with conventional neural networks except that they assume image as an input. In visual recognition tasks, deeper networks tend to give better results.

1.3.1 Convolutional Neural Networks(CNNs)

CNNs [12] are like any other type of the Neural Networks that are made of neurons that have trainable biases and weights. Every neuron gets an input, performs an inner product and sometimes follows it with some non-linear function in between the network. Neural Networks transforms an input through a train of hidden layers which have multiple neurons in them, and each of the neuron is then connected to all of the neurons of the previous layer. The last FC layer is the output layer which throws out probabilities of classes or a binary classification. The basic network structure of neural networks is illustrated in Figure 1.2.

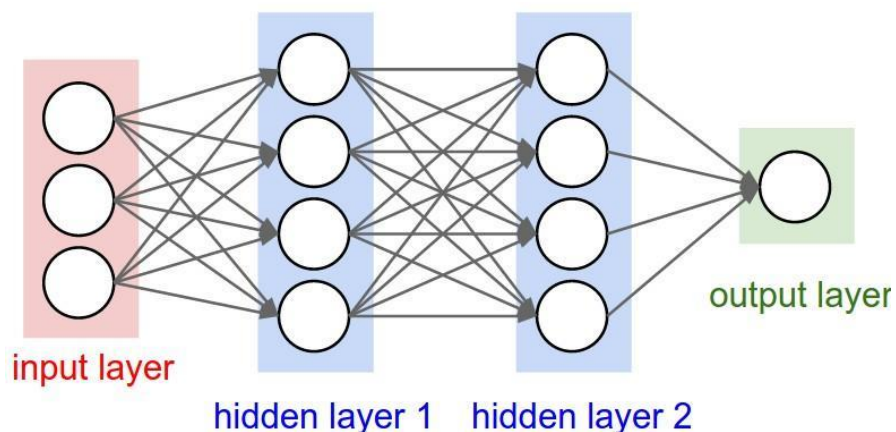


Fig. 1.2 Basic network structure of Neural Networks

ConvNet architectures are explicitly made for inputs that are images and these increases the efficiency of the forward function and try reducing the number of parameters the network usually depends on. CNNs have neurons arranged in three dimensions; width, height, depth. Figure 1.3 helps in understanding internal functioning of the CNNs.

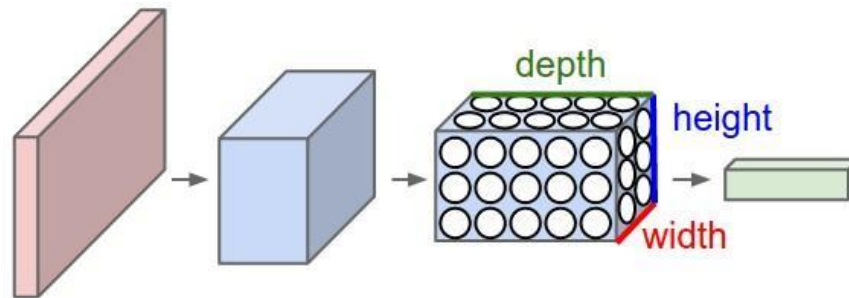


Fig. 1.3 Understanding the internal functioning of CNN

CNN is a sequence of layers, where each layer of a CNN converts one volume of activations into another with the help of a specific function called activation function. Most common type of activation functions in a CNN is one or the other variant of ReLU. Convolutional Layer, Fully-Connected Layer(FCL) and Pooling Layers are the 3 main types of layers used in CNN. Every CNN architecture is made by stacking these types of layers on top of each other in various configurations.

- INPUT $[W \times H \times C]$ will hold the raw image pixel values, where W is width, H is height and C represents number of channels (RGB).
- CONV layer is used to compute the neuron output which are inter-connected to sub-sections in the input, all computing a product between the weights resulting in $[W \times H \times F]$ where F represents filters.
- RELU layer is used to apply activation function to every element outputted by the layer. Others include Leaky-Relu, Sigmoid and Tanh activation functions.

- POOL layer is used for down-sampling (converting an image to low-resolution) operation along the dimensions. Types include average and max pooling. The pooling layer is used to reduce the representation size and thereby reducing the number of parameters which ultimately benefits in decreasing the amount of computations in the network. This ultimately controls overfitting. The figures, Figure 1.4 illustrates the average pool layer and figure 1.5 illustrates the max pool layer.
- FC Layer is used to compute the class scores at the output layer.

A CNN is able enough to inherently capture the ‘what’ and ‘where’ aspects of an image through the application of relevant and necessary filters. The performance of the architecture fits better to the dataset with images as the number of parameters are reduced and re-usability of weights is made possible. Unlike in ordinary neural networks, in a CONV layer, neurons get the input from something called a “receptive field”, which is a restricted area of previous layer.

1.3.2 Spatial Arrangement

Following hyper parameters decide the size of the output volume of the CONV layer:

- The depth of output volume is the number of filters we shall use; each filter tries to learn something unique in the input.
- The stride we use to slide the chosen filter. For instance, if the stride is 1, then the filter is moved one pixel at a time which leads to heavy receptive field overlap. Practically using smaller strides are advantageous.
- The padding of the input with zeroes all along the image border is another hyper parameter. With Zero padding we can control the spatial size of the output image. Usage of zero padding in CONV layers avoids the reduction of

the size of the volumes of the images by a certain amount after every CONV, and stops the fast escape of the information at the image borders.

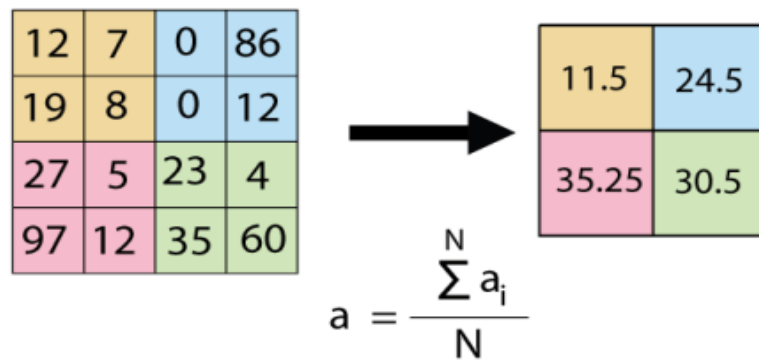


Fig. 1.4 Illustration of the Average Pool layer

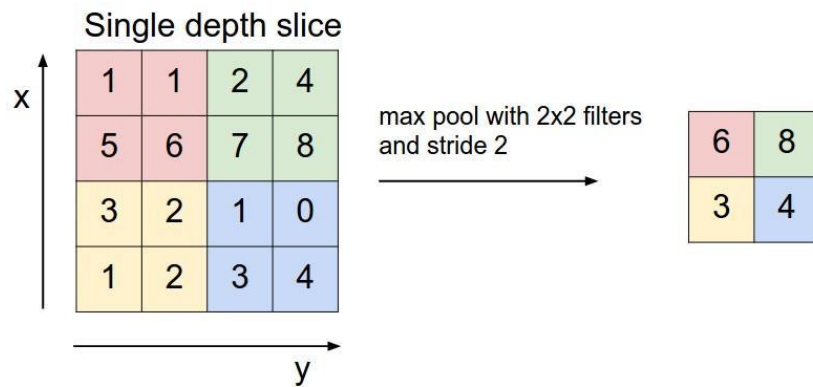


Fig. 1.5 Illustration of the Max Pool layer

The computation of the spatial size for the output is defined by the Eq. (1).

$$M = ((W - K + 2 P)/S) + 1 \tag{1}$$

where, M denotes Spatial size of the output,

W denotes size of input,

P denotes amount of applied zero padding,

K denotes kernel field size and

S denotes stride.

1.4 PROBLEM STATEMENT

Even with the availability of computationally high GPU machines, the real-time rendering and detection of the photo-realistic images is still an expensive problem. Nowadays there is an increased requirement of CGI imagery in many fields like entertainment industry, video gaming, and medicine. Even with increase in powerful GPUs, the real-time rendering of a single image takes around 8 - 16 hours. So, real-time object detection in photo-realistic image while they are rendered real time is a challenging task. Also, when the problem of object detection is concerned, when there is noise in the images, it is very difficult for the object detection algorithms to accurately detect the objects. Especially in the real-time, it is very difficult to detect objects in applications like video surveillance where the images are usually very noisy and in the wild where noise causing factors like rain, fog, smoke, etc., are present.

CHAPTER 2

LITERATURE REVIEW

2.1 GENERATIVE ADVERSARIAL NETWORKS (GANS)

GAN [13] [14] was first introduced in 2014 by Ian Good Fellow, et al. GANs are basically a scenario of game theory where “the generator network will compete against an adversary (discriminator network)”. The Generator is forced to generate new data similar to the expected values, and constantly convince the discriminator to believing that the generated samples are real. The Discriminator attempts to learn to classify samples as real (the data the network will be trained on) and fake (the samples generated by the Generator network) correctly. This process is illustrated in the figure 2.1.

Generator network produces sample $x = g(z, \theta_g)$, which is a mapping of input (usually drawn from a random probability distribution) noise variables Z to a desired network space x (images). Discriminator gives out a probability value solved by $d(x, \theta_d)$, depicting the probability of x being a real training sample instead of a fake one. θ_i represents weights of parameters that define each neural network.

Learning in GANs is like a zero-sum game, where the function $v(\theta_g, \theta_d)$ calculates the loss of the discriminator, as the function is received by the generator as its own loss. During learning, each network attempts to minimize its own loss, so at the convergence $g^* = \arg \min \max v(g, d)$. Discriminator’s weights are updated in order to “maximize the probability that any real data input x is classified correctly as being real, while minimizing the probability that any fake image is classified as being real”. Maximize $d(x, \theta_d)$, minimize $d(g(z, \theta_g))$.

Generator, meanwhile, updates weights so as to maximize the probability of the fake sample is classified by the discriminator as being from a real dataset, compared to being coming to decrease the log-probability that the prediction made by the discriminator is correct.

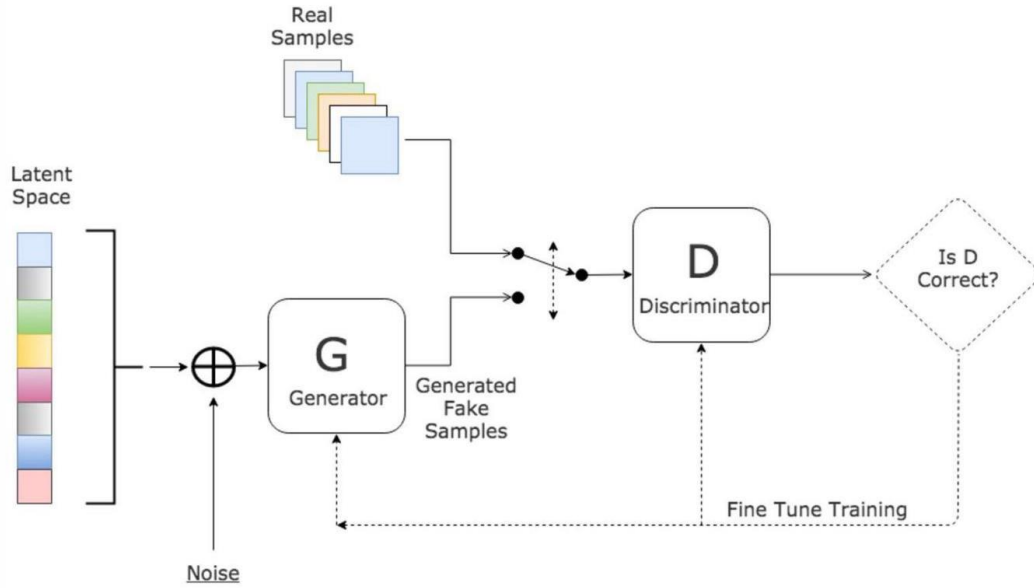


Fig. 2.1 Generative Adversarial Network

This rework is different from both zero-sum and maximum likelihood. And was introduced with the heuristic motivation in which it will keep the difference between the derivative of the generator's cost function and the discriminator's log huge even when discriminator keeps confidently rejecting all the generated samples.

The loss functions of the generative adversarial network are given by the following Eq.2.

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

$$\max_D V(D) = \underbrace{\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})]}_{\text{recognize real images better}} + \underbrace{\mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]}_{\text{recognize generated images better}}$$

$$\min_G V(G) = \underbrace{\mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]}_{\text{Optimize G that can fool the discriminator the most.}}$$

(2)

2.1.1 Algorithm

Following is a pseudo code that comprehensively describes the functioning of a Generative Adversarial Network:

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k , is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $p_g(\mathbf{z})$.
- Sample minibatch of m examples $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\mathbf{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[\log D(\mathbf{x}^{(i)}) + \log \left(1 - D(G(\mathbf{z}^{(i)})) \right) \right].$$

end for

- Sample minibatch of m noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $p_g(\mathbf{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log \left(1 - D(G(\mathbf{z}^{(i)})) \right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

There are various types of GANs like SR-GANs, Conditional GANs, Deep Convolutional Gans, infoGAN, Obj-GANs, etc.

2.1.2 SR-GANs

Super Resolution Generative Adversarial Networks (SR-GANs) [15] are a type of GANs used to convert low-res images to high-res images. Super-resolution GAN employs a deep neural network which works in synergy with a discriminator network to produce images with higher resolution. During the training, an image of high-res (HR) is down-sampled to a low-resolution image (LR). Then, a GAN generator up-samples images to super-resolution images (SR) from low-res (LR). A discriminator is used for distinguishing the HR images and back-propagate the GAN loss for further training both the discriminator and the generator. Figure 2.2 describes an SR-GAN.

The network below composes of convolution layers which go through batch normalization and parameterized ReLU (PReLU). Similar to ResNet, the generator implements skip connections. The convolution layer with 3x3 kernel filters outputting 64 channels with stride 1 is represented as “k3n64s1”.

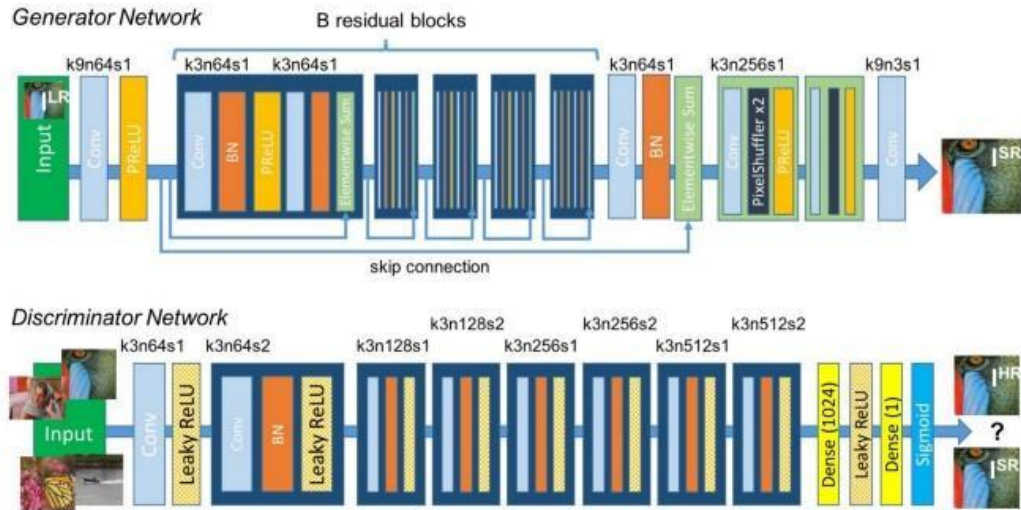


Fig. 2.2 SR-GAN

2.1.3 Residual Blocks

As it is comparatively more difficult to train deeper networks, the residual learning framework facilitates the training of these networks, and contributes to the improvement of the performance with deeper network. 16 “Residual Blocks” are used in Generator. Skip architecture or residual connection is commonly used as a “Skip Connection” between the output and the input, or between convolution and transposed convolution. To use skip connection, the dimension equality of both input and output should be maintained. By using skip connection, it provides an alternative for gradient to back propagation bettering the model’s convergence as depicted in Figure 2.3.

The perceptual loss function l^{SR} is vital for the generator network’s performance. While l^{SR} is inspired from the Mean Square Error, a new loss function is created that inspects a solution to those characteristics that are perceptually relevant. The perceptual loss is calculated as the weighted sum of a content loss and an adversarial loss component as given in the following Eq.3.

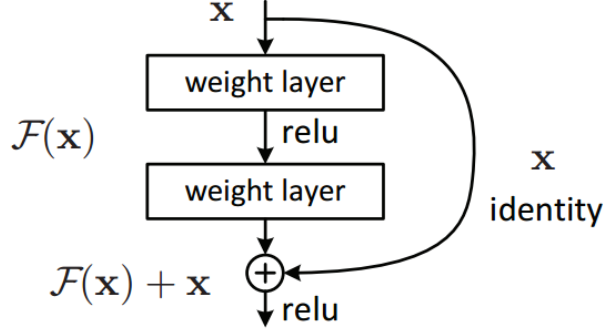


Fig. 2.3 Residual Blocks

The pixel-wise Mean Squared Error (MSE) loss is calculated as per the following Eq.4.

$$l^{SR} = \underbrace{l_X^{SR} + 10^{-3}l_{Gen}^{SR}}_{\text{perceptual loss (for VGG based content losses)}} \quad (3)$$

$$l_{MSE}^{SR} = \frac{1}{r^2WH} \sum_{x=1}^{rW} \sum_{y=1}^{rH} (I_{x,y}^{HR} - G_{\theta_G}(I^{LR})_{x,y})^2 \quad (4)$$

The VGG loss is nothing but Euclidean distance between the reconstructed image and the reference image's feature representations because although achieving particularly high PSNR, MSE optimization problem solutions couldn't produce high frequency content resulting in unsatisfying solutions with overly smooth textures. The following Eq.5 gives this VGG Loss that is based upon the ReLU activation layers of pretrained 19-layer VGG network.

$$l_{VGG/i,j}^{SR} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2 \quad (5)$$

Here $W_{i,j}$ and $H_{i,j}$ define within the VGG network, dimensions of their respective feature maps. The generative loss is defined depending upon the probabilities of discriminator over all training samples as given by the following Eq.6.

$$l_{Gen}^{SR} = \sum_{n=1}^N -\log D_{\theta_D}(G_{\theta_G}(I^{LR})) \quad (6)$$

Here, $D_{\theta_D}(G_{\theta_G}(I^{LR}))$ is the probability that the reconstructed image $G_{\theta_G}(I^{LR})$ is a natural High Resolution image. For better gradient behavior we minimize $-\log D_{\theta_D}(G_{\theta_G}(I^{LR}))$ instead of $\log [1 - D_{\theta_D}(G_{\theta_G}(I^{LR}))]$.

2.1.4 Why SRGAN for Denoising?

The most related GAN-based work to denoise is the SRGAN, a GAN for Single Image Super-Resolution. Unlike the networks previously proposed for image super resolution that used “Mean Squared Error(MSE) as the optimization function, SRGAN proposes a loss function that resolves perceptually satisfying high-resolution image. Architecture used is a very deep residual net architecture based on GAN. By minimizing the perceptual loss function (weighted sum of content loss and adversarial loss) the training of the network is achieved”. When we do not rely upon the pixel wise error measures like MSE based optimization, a better perceptual loss function that has both adversarial loss and content loss is implemented.

2.2 OBJECT DETECTION USING YOU ONLY LOOK ONCE (YOLO) ALGORITHM

YOLO [7] [10] [8] [9] is a popular state-of-the-art real-time object detection algorithm. YOLO was first developed by J. Redmon et al [8]. There are three versions of the algorithm which is refined through years. The advantage of YOLO over other object detection algorithms is that it can be generalized to any other type of images like artwork [8] [16]. In this project, we improved accuracy of YOLO Version 3 for object detection in photo-realistic rendered images.

2.2.1 YOLO V 1

Usually to perform detection object detection employs classifiers. Instead, in YOLO [8] object detection is seen as a regression-problem intended to separate bounding boxes spatially and the probabilities associated thereof the classes. The network “can be optimized end-to-end on detection performance directly as a single neural network is enough to predict bounding boxes and class probabilities directly in one evaluation of images. Although YOLO makes more errors related to localization when compared to classical methods, it is less likely to predict false positives on background. It also learns very general representation of objects in an image. It is better than other networks like DPM, R-CNN when it comes to generalization from an image to artwork.

While deformable parts models (DPM) uses an approach called sliding window in which the classifier is run at evenly spaced locations over the entirety of the image and systems like R-CNN use region proposal methods to generate potential bounding boxes in an image first and then run a classifier on these proposed boxes which are refined in post-processing. The duplicate detections are eliminated and scoring of the boxes is done based on the other objects in the scene. But in YOLO, you only look at the image once to predict and locate. Unlike sliding window and region proposal-based techniques, YOLO implicitly understands contextual information about the classes as well as their appearance because during training and test time it looks at the entire image.

An input image taken and is marked into an $S \times S$ grid. And if the center of a particular object falls into that grid cell, it is that cell’s responsibility for detecting an object. Every grid cell predicts B bounding boxes and confidence scores (tells how sure the model is that the box has an object) respectively. They also tell how accurate it thinks the box is that it predicts. Confidence is defined as $\text{Pr}(\text{Object}) * \text{IOU}(\text{prediction}, \text{truth})$. Absence of the object means the confidence scores will be zero. The confidence score is the intersection over union (IOU) between the predicted box and the ground truth. The bounding box prediction is made of 5 components: $(x, y, w, h, \text{confidence})$. The (x, y) are coordinates normalized to $[0,1]$ to represent the box

centers, relative to location of the grid cell. The (w, h) box dimensions are normalized to $[0, 1]$ too, relative to the size of the image.

The detection network of YOLO has 24 CONV layers that have been followed by another two FC layers. The feature space from the preceding layers is reduced by the alternating 1×1 CONV layers. The network architecture of YOLO is as depicted in Figure 2.4.

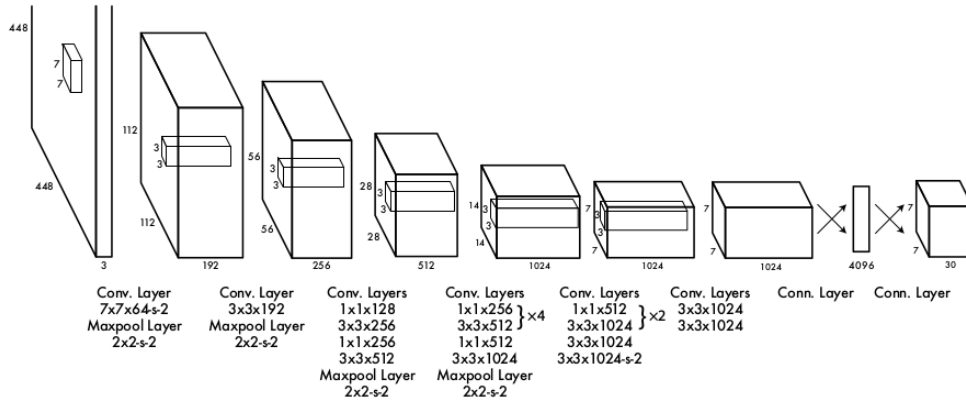


Fig. 2.4 YOLO Network

2.2.2 YOLO V 2

The second version of YOLO is called YOLO9000 [9] as it can detect over 9000 different object categories. They improved it over the first version of YOLO for better, accurate and faster detections. Improvements are made in YOLOv2 over the network. Darknet-19 [7] has been introduced as a new classification model that is used as a base for YOLOv2. It mostly uses 3×3 filters and the doubling of number of channels is done after each pooling step. Table 2.1 describes the Darknet-19 model. It has got 19 convolutional layers along with 5 max pooling layers. It requires around 5.6 billion operations for a single image to be processed.

Loss function is given in Eq.7. Of the loss function equation, the first one is implemented to penalize the objectness score prediction of the bounding boxes that are responsible for predicting objects, and the second one for bounding boxes having no objects, the last one acts as a punishing parameter for the class prediction for the bounding box which predicts the objects. In the YOLO V2, instead of directly

predicting the bounding boxes, YOLO predicts off-sets from a predetermined set of boxes with particular height-width ratios called anchor boxes. They are defined to capture the scale and aspect ratio of an object class you want to detect and are typically chosen based on the object sizes in the dataset used for training. During detection, the anchor boxes are spread across the test image and the network predicts the probability and other metrics like intersection over union (IoU) and offsets for all tiled anchor boxes. Post prediction, each individual anchor box is refined.

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

TABLE. 2.1 Darknet – 19

$$\begin{aligned}
& \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \\
& + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\
& + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\
& + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned} \tag{7}$$

2.2.3 K-Means Clustering Algorithm

K-Means [17] [9] is a very often used clustering algorithm known for its efficiency. The algorithm works by storing k centroids that it uses to define the clusters. When a point is closer to a cluster's centroid than rest of the centroids, then that point is said to belong to that particular cluster. The best centroids are found by switching constantly between assigning data points to clusters depending on the current centroids and choosing centroids based on the current allocation of data points to clusters. The Algorithm is given below. Euler distance metric for K-means minimizes error for larger bounding boxes, but not for smaller ones. Hence, in YOLOv2, intersection over union (IOU) is used as the distance metric.

1. Initialize **cluster centroids** $\mu_1, \mu_2, \dots, \mu_k \in \mathbb{R}^n$ randomly.

2. Repeat until convergence: {

For every i , set

$$c^{(i)} := \arg \min_j \|x^{(i)} - \mu_j\|^2.$$

For each j , set

$$\mu_j := \frac{\sum_{i=1}^m 1\{c^{(i)} = j\}x^{(i)}}{\sum_{i=1}^m 1\{c^{(i)} = j\}}.$$

}

Figure 2.5 depicts the analysis of no. of centroids vs. Mean IoU. It is clear from the plot in the following figure 2.5 that as the number of centroids increases, the mean IoU between anchor boxes and bounding boxes plateaus after a steady increase after about 15 centroids. By Elbow method, in the plot the location of a bend (elbow) is usually considered an indicator for deciding the appropriate number of clusters. So from the plot it is apparent that 5 clusters will yield good results. But in the thesis, use of different number of cluster centroids has been experimented with to see how YOLO algorithm performs.

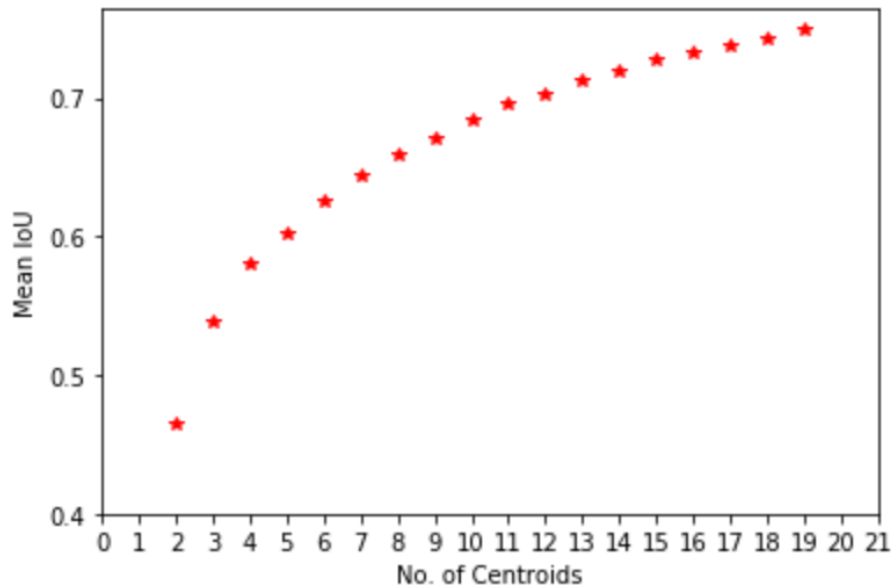


Fig. 2.5 K-Means Clustering: No. of Centroids vs. Mean IoU

Every few iterations, the network in YOLO9000 is changed, rather than fixing the input image size. For every 10 batches, a new image dimension size is chosen by the network randomly. As the model is down-sampled by a factor of 32, the smallest dimension is 320×320 and the largest is 608×608 for the detection. Resizing of the network happens during training which makes it possible for the network to predict on variety of the input dimensions. This means that the same network can be used to predict detections at various resolutions. Given smaller dimensions, the network runs faster. Thus, YOLOv2 offers a tradeoff between accuracy and speed.

2.2.4 YOLO V 3

YOLOv3 [10] incorporates residual blocks, up-sampling and skip connections. Although YOLOv3 is better, it is not necessarily faster. YOLOv3 has a 53-layer network that has been trained on Imagenet [18]. But for these detection tasks, another 53 more layers are stacked on it to get a 106 layered fully convolutional architecture making it slower compared to YOLOv2. Whereas in YOLOv2 we are using only a 19 layered network. In YOLOv3, for each and every bounding box an objectness score is predicted using logistic regression method. Feature extraction is done by the robust Darknet – 53 which has been depicted in Table 2.2. Residual connections are added to the new Darknet – 53.

YOLOv3 predicts the bounding boxes at three different scales. The system extract features from these scales to feature pyramid networks (FPNs). From the base feature extractor, several convolutional layers were added. The last one predicts a three-dimensional tensor encoding bounding box, objectness, and class predictions. The confidence and class predictions of the objects are done through logistic-regression.

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

TABLE.2.2 Darknet-53

Detections at various layers are used to solve the issue of detecting smaller objects which cannot be done properly in YOLOv2. We up sample the layers that are conjoined with previous layers so as to preserve the fine features of the smaller objects for better detections. Different layers – a 13×13 layer is used to detect large objects, a 52×52 layer is used to detect smaller objects and a 26×26 layer is used to detect medium size objects. In YOLOv3, the last three terms are cross entropy error terms unlike the squared errors in YOLOv2.

2.2.5 Need for Custom Anchor Boxes

Given that our dataset comprises of animated images where the sizes of the

objects are exaggerated for visual effect, it is necessary to understand that default anchor boxes provided by the YOLOv3 algorithm may work, but use of custom anchor boxes should improve the accuracy of the bounding box around the object classes. With increase in cluster centroids in the generation of anchor boxes increases the mean IoU of the generated priors to the actual ground truth. But after a certain number of centroids, the mean IoU plateaus and it will no longer be a relevant parameter to decide whether we can decide the trade-off of IoU on it. YOLOv3 uses 9 anchor boxes generated based off on the VOC dataset [19].

2.2.6 Intersection-Over-Union(IoU)

Intersection over Union, also called Jaccard Index, is a popular, and efficient metric of evaluation used to “measure the accuracy of an object detector on any given dataset. It is often used in object detection challenges like PASCAL VOC challenge [19] or any other classification tasks which require the understanding of correctness. The ground-truth bounding boxes are needed in order to apply IoU to evaluate a given object detector and the predicted bounding boxes from our model. Intersection over Union (IoU) is depicted in the figure 2.6.

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Fig. 2.6 IoU Illustration

2.3 RELU

There are many activation functions that are implemented in the CNNs like sigmoid, tanh, ReLU, PReLU, etc. Among all the activation functions used, variants of Rectified Linear Unit, also called as ReLU are the most frequently used.

2.3.1 Simple ReLU

ReLU is an activation function defined as $y = \max(0, x)$. ReLU is normally applied element-wise to the output of any other function like a product of matrices etc. The simple ReLU activation function is illustrated in Figure 2.7. ReLU is one of the most common activation functions in NNs, especially in CNNs. From the function definition one can see that “ReLU is linear (identity) for all positive values, and zero(0) for all the negative values”.

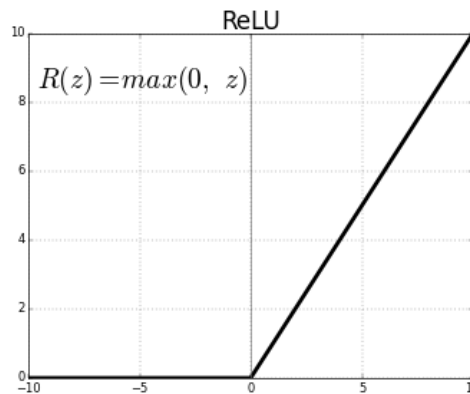


Fig. 2.7 ReLU Activation Function

Simple mathematics make it easily understandable for someone starting out with NN. Due to simplicity of mathematics the model will now take less time to train too. It converges faster. Linearity as in “the slope doesn’t plateau when x gets very large. Unlike sigmoid function, it won’t be suffering from the vanishing gradient problem. Since ReLU is zero(0) for all negative inputs, indicating it being inactivate for any given value” which is desirable most of the times.

2.3.2 Drawbacks

The drawback for having zero for every negative value is a problem called “dying ReLU. A ReLU neuron is considered dead if it cannot come out of the negative side and always outputs 0 irrespective of the input. Because the slope of ReLU in the negative range is also zero(0), once a neuron gets negative, the neuron can’t come out of it. Such neurons don’t play any role in discriminating the input and is basically useless”. Over the time it can be observed that a large part of the neural network is useless or inactive. So, we use another variant of ReLU like parametric ReLU which

do not suffer from these drawbacks.

2.3.3 Leaky RELU

“Leaky ReLU” has a small slope for negative values, instead of altogether zero(0) which is developed on top of ReLU. The function looks like its leaking with a slope in the negative values, hence the name. For instance, leaky ReLU might have $y = 0.01x$ when $x < 0$.

“Parametric ReLU (PReLU)” is another kind of leaky ReLU, which instead of having a predetermined slope like 0.01, makes it a parameter for the NN that has to be trained and learned: $y = ax$ when $x < 0$. Figure 2.8 illustrates Leaky ReLU/ PReLU activation function.

Leaky ReLU has two benefits:

- Due to the lack of zero-slope parts, it doesn't suffer from the problems ReLU suffers from.
- Training with a leaky ReLU is faster. It can be observed empirically that with “mean activation” being closer to 0 makes training faster. Leaky ReLU is much “balanced,” and may learn faster than ReLU.

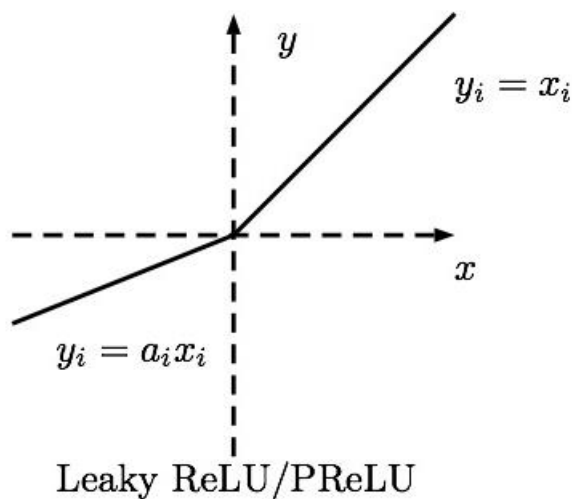


Fig. 2.8 Leaky ReLU/PReLU Activation Function

Although the result is not always consistent, leaky ReLU is usually better than plain ReLU, but not always is, and should be considered only as an alternative.

2.4 TRANSFER LEARNING

Transfer learning [20] is an ML method in which a model developed to solve a specific task, is reused for learning another model on another task. It is a very usual thing in DL where pre-trained models are used as the starting points of various tasks of CV and NLP. Due to large computing time and resources needed for the development of the neural networks on these problems, transfer learning is vital here. The knowledge acquired in solving a task ‘X’ is used to solve another task ‘Y’.

There are two types of transfer learning -

1. Transductive Transfer Learning
2. Inductive Transfer Learning

In Inductive Transfer Learning is a traditional supervised learning approach in which we learn a model from the given labeled examples and then we predict the labels of those examples which we have not known about or seen. Where as in Transductive Transfer Learning, we learn from a lot of examples, and then we try only to predict a known (test) set of unlabeled examples. As we can see, transductive transfer learning is a less ambitious approach compared to the inductive transfer learning approach. In Deep Learning, we use transfer learning for speeding up the training and for improving the performance of the DL models. We use the pre-trained models as starting point on the Natural Language Processing(NLP) and Computer Vision(CV) tasks as it saves lots of computing resources and time.

It is a very normal practice to perform transfer learning with the predictive modeling problems that use image or video data as input. For the problems in Computer Vision, it is a very common approach to use the DL models that are pre-trained for challenging image classification tasks like ImageNet [19] 1000-class photograph classification competition. Many organizations like Google, Microsoft, etc., develop models for these types of competitions and often release their final models for reuse under permissive licenses. These complex models take days or weeks to train using these modern hardware and infrastructure which may be not be available in normal schools and colleges and even in mid-sized research laboratories.

Some of these models include Google Inception Model [21], Oxford VGG Model [22], Microsoft ResNet Model [23].

CHAPTER 3

PROPOSED WORK

3.1 THE PIPELINE

The block diagram of our implementation is as depicted in the figure 3.1. We take an image with noise and pass it through a Denoising Generative Adversarial Network to get the denoised image. The image which is inputted into the GAN could contain any type of noise like real-time noise, or gaussian or uniform with varied amounts of noise in them. For the sake of experiments, we explicitly added gaussian and uniform noise of varied amounts to the images and made them noisy. The denoised image, which is an output of the GAN, is again given as input to the improved YOLO Version 3 algorithm for object detection of the images so that bounding boxes with much improved IoU are predicted.

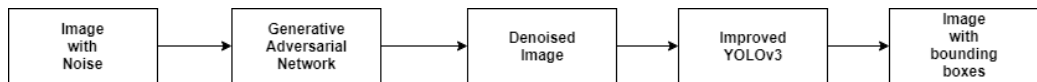


Fig. 3.1 The Block Diagram

The work-flow of our implementation is well understood by the figure 3.1. The entire implementation can be broadly divided into two parts. One, denoising of the image using a generative adversarial network and two, object detection using YOLOv3 with improved IoU for better detection. Denoising of an image is required before object detection as it may be difficult for the object detection algorithms to predict the bounding boxes when noise is present. A lot of noise will be present when photo-realistic rendering is done with four or eight rays instead of thousands of rays. We use a denoising generative adversarial network for the purpose of denoising. Then, we input the denoised image to the object detection algorithm YOLOv3 with improved Intersection over Union (IoU) for predicting the bounding boxes. which can be generalized to any type of images like artwork, etc. Finally, an image with bounding boxes is obtained as the output.

3.2 IMAGE DENOISING USING GAN

Path-tracing is the technique that is used in rendering high quality photo-realistic frames of 3D scenes. This technique involves shooting of thousands of rays into a single pixel using the Monte Carlo simulation. When a ray hits the objects in the image scene which will either refract or reflect or become absorbed. Average of the color generated will be the color of that pixel. As this method will be applied for all the pixels rendering photo-realistic scenes frame by frame is very time consuming and also expensive. The implemented method is to render using very small number of samples per pixel and pass the noisy image to the network, which will generate a photo-realistic image with high quality. The network is based on the ResNet, which specializes in residual blocks which will carry important features forward into the network without fail. The key is nothing but the defined loss function and the deep GAN. Also a refined perceptual loss has been defined to preserve color, texture, properties of the scene.

The check points of the pre-trained network are used for implementing the GAN using the Tensorflow [24] API. We introduced monochromatic Gaussian and Uniform noises of varying amounts to our ground truth images.

3.2.1 Gaussian Distribution

Gaussian functions are very much often used to represent the PDF of a normally distributed random variable with expected value(μ) and variance(σ^2). Figure 3.2 depicts the Gaussian distribution. And the Gaussian distribution shown is normalized so that the sum over all values of x gives a probability of 1. In this case, the Gaussian is as depicted in figure 3.2 and is given by the Eq.8.

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \quad (8)$$

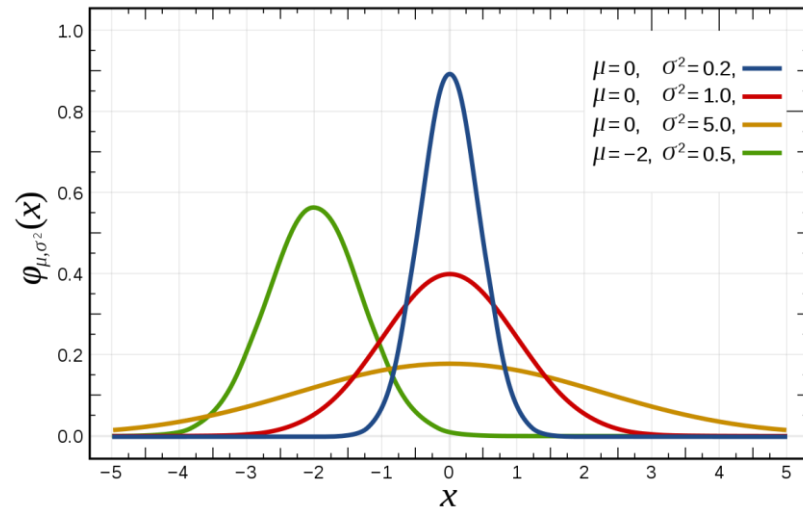


Fig. 3.2 Gaussian Distribution

3.2.2 Uniform Distribution

The continuous uniform distribution is a probability distribution in which all outcomes are equally likely; each variable has the same probability that it will be the outcome. The distribution is defined by the two parameters, a (minimum) and b (maximum). And the Uniform distribution is abbreviated U (a,b) and is illustrated in figure 3.3.

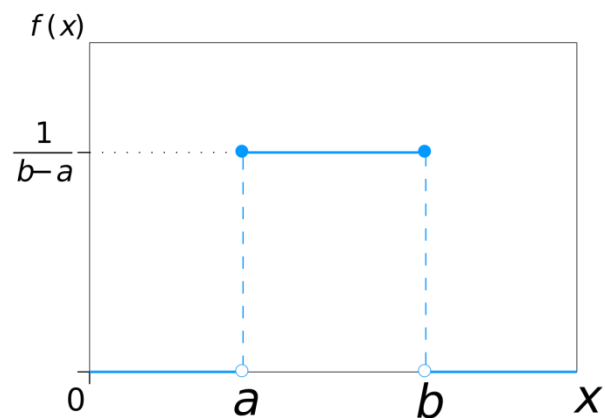


Fig. 3.3 Uniform Distribution

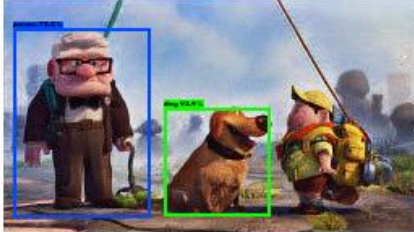
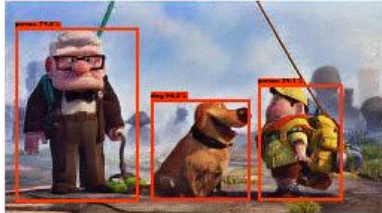
The probability density function of continuous uniform distribution as in the given Eq.9.

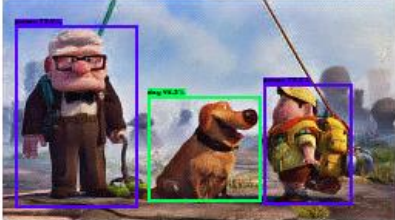
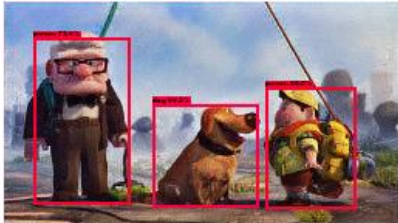
$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b, \\ 0 & \text{for } x < a \text{ or } x > b \end{cases} \quad (9)$$

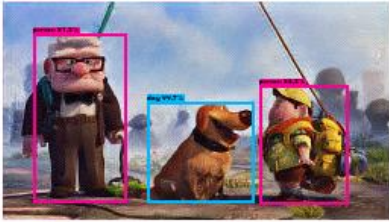
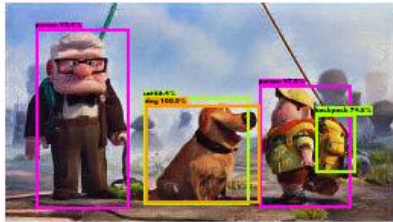
3.3 OBJECT DETECTION USING YOLO VERSION 3 WITH IMPROVED IOU

3.3.1 Tuning Leaky ReLU for the Improvement of Detection Accuracy

Leaky ReLU converges faster if both positive and negative values are expected in the output unlike traditional ReLU. It is one of the parameter which can be customized based on the working dataset and an increase in the object prediction can be achieved by tuning it. Following is an experiment which illustrates the increase in accuracy of the prediction of different classes in an image and improper increase in the slope of the leaky ReLU will result in unnecessary detections or false positives.

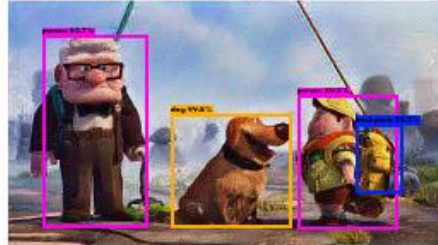
<p>Class name: person Confidence: 70.20437121391296 Class name: dog Confidence: 93.35347414016724</p>  <p>Total time: 25.487714290618896 secs</p>	<p>Class name: person Confidence: 74.76561665534973 Class name: person Confidence: 59.104371070861816 Class name: dog Confidence: 96.49674892425537</p>  <p>Total time: 23.343185663223267 secs</p>
<p>Fig. 3.4 leaky_relu @ 0.090</p>	<p>Fig. 3.5 leaky_relu @ 0.095</p>

<p>Class name: person Confidence: 75.906240940094 Class name: person Confidence: 70.18781304359436 Class name: dog Confidence: 98.32143783569336</p>  <p>Total time: 23.721565008163452 secs</p>	<p>Class name: person Confidence: 79.98812794685364 Class name: person Confidence: 72.93789982795715 Class name: dog Confidence: 99.28789734840393</p>  <p>Total time: 20.99589204788208 secs</p>
<p>Fig. 3.6: leaky_relu @ 0.100</p>	<p>Fig.3.7 leaky_relu @ 0.105</p>

<p>Class name: person Confidence: 88.22876214981079 Class name: person Confidence: 81.33618235588074 Class name: dog Confidence: 99.70197081565857</p>  <p>Total time: 21.618655681610107 secs</p>	<p>Class name: person Confidence: 96.96634411811829 Class name: person Confidence: 92.35162138938904 Class name: cat Confidence: 86.44795417785645 Class name: dog Confidence: 99.96446371078491 Class name: backpack Confidence: 74.84027743339539</p>  <p>Total time: 21.34204888343811 secs</p>
<p>Fig. 3.8 leaky_relu @ 0.110</p>	<p>Fig. 3.9 leaky_relu @ 0.120</p>

The above figures, figure 3.4, figure 3.5, figure 3.6, figure 3.7, figure 3.8, figure 3.9 and figure 3.10 depict the leaky_relu for different slopes. In the above results, we can observe that for a slope of 0.090, the detection of the person could not be made, but for a slope of 0.120, there is a false positive of cat object. And fine tuning of the parameter resulted in the result below in figure 3.10 which illustrates the perfect detection system for the used dataset at slope 0.111 in the duration of detection time.

Class name: person
Confidence: 89.802485704422
Class name: person
Confidence: 82.71081447601318
Class name: dog
Confidence: 99.75413084030151
Class name: backpack
Confidence: 53.162264823913574



Total time: 21.792546033859253 secs

Fig. 3.10 leaky_relu @ 0.111

3.3.2 Generating custom anchor boxes for an improved IoU

A few ground truth images are taken from the training data of the GANs and annotated to get the values of the bounding boxes in .xml format. The annotated files will be the input for the custom anchor box generation script, which takes in the x_{min} , x_{max} , y_{min} , y_{max} values for each object in an image to understand the aspect ratio and the scale of the object with respect to the whole image. Then after the calculation with the help of clustering algorithm, which here is 'k means' algorithm, it will give out the possible cluster centroids with priors bound around them in various aspect ratios and scales. Here we can see from the results that when compared to a traditional IoU which is obtained by training on a large common dataset, the mean IoU of the custom made anchor boxes fit better with the predictions and are higher considerably. Further increase in diversity of the dataset with various exaggerated shapes of the animated characters could provide more insights into efficiency of the custom made anchor boxes on the detection accuracy and necessity to decide the priors based on the cluster centroids which converge better with the subject dataset. Figure 3.11 shows mean IoU with VOC dataset with $k=5$. Figure 3.12 shows mean IoU with custom dataset with $k=5$. Figure 3.13 shows mean IoU with VOC dataset with $k=9$. Figure 3.14 shows mean IoU with custom dataset with $k=9$.

<pre> Accuracy: 60.68% Boxes: [[0.792 0.77066667] [0.042 0.07733333] [0.368 0.52533333] [0.096 0.152] [0.176 0.324]] </pre>	<pre> Accuracy: 86.01% Boxes: [[0.328125 0.47265625] [0.33398438 0.5390625] [0.20117188 0.35546875] [0.171875 0.46875] [0.47265625 0.45507812]] </pre>
<p>Fig. 3.11 Mean IoU with VOC dataset(5)</p>	<p>Fig. 3.12 Mean IoU with custom dataset(5)</p>
<pre> Accuracy: 67.54% Boxes: [[0.038 0.07185629] [0.374 0.70133333] [0.102 0.288] [0.634 0.45066667] [0.176 0.16533333] [0.308 0.31466667] [0.834 0.836] [0.192 0.49866667] [0.078 0.128]] </pre>	<pre> Accuracy: 93.49% Boxes: [[0.23046875 0.3515625] [0.29882812 0.42578125] [0.46875 0.48828125] [0.49023438 0.41015625] [0.171875 0.36132812] [0.171875 0.46875] [0.3046875 0.28125] [0.33398438 0.5390625] [0.33398438 0.49609375]] </pre>
<p>Fig. 3.13 Mean IoU with VOC dataset(9)</p>	<p>Fig. 3.14 Mean IoU with custom dataset(9)</p>

The above values titled boxes are the different sizes of prior boxes created by the k means clustering algorithm which play a major role in deciding the IoU of the predictions on the images. Diversity in the ratios will be decided by the annotations of the custom dataset and the requirement of the proportions at the time of detection. Ideal algorithm creates thousands of anchor boxes for every prediction that depicts the ideal location, size and shape of the object it particularly specializes in predicting.

Figure 3.15 and figure 3.16 are visualization of the default anchor boxes in RetinaNet and beside is the amount of reduction in size of the boxes that can be achieved in order to maintain higher IoU and to detect much smaller objects of rare proportions.

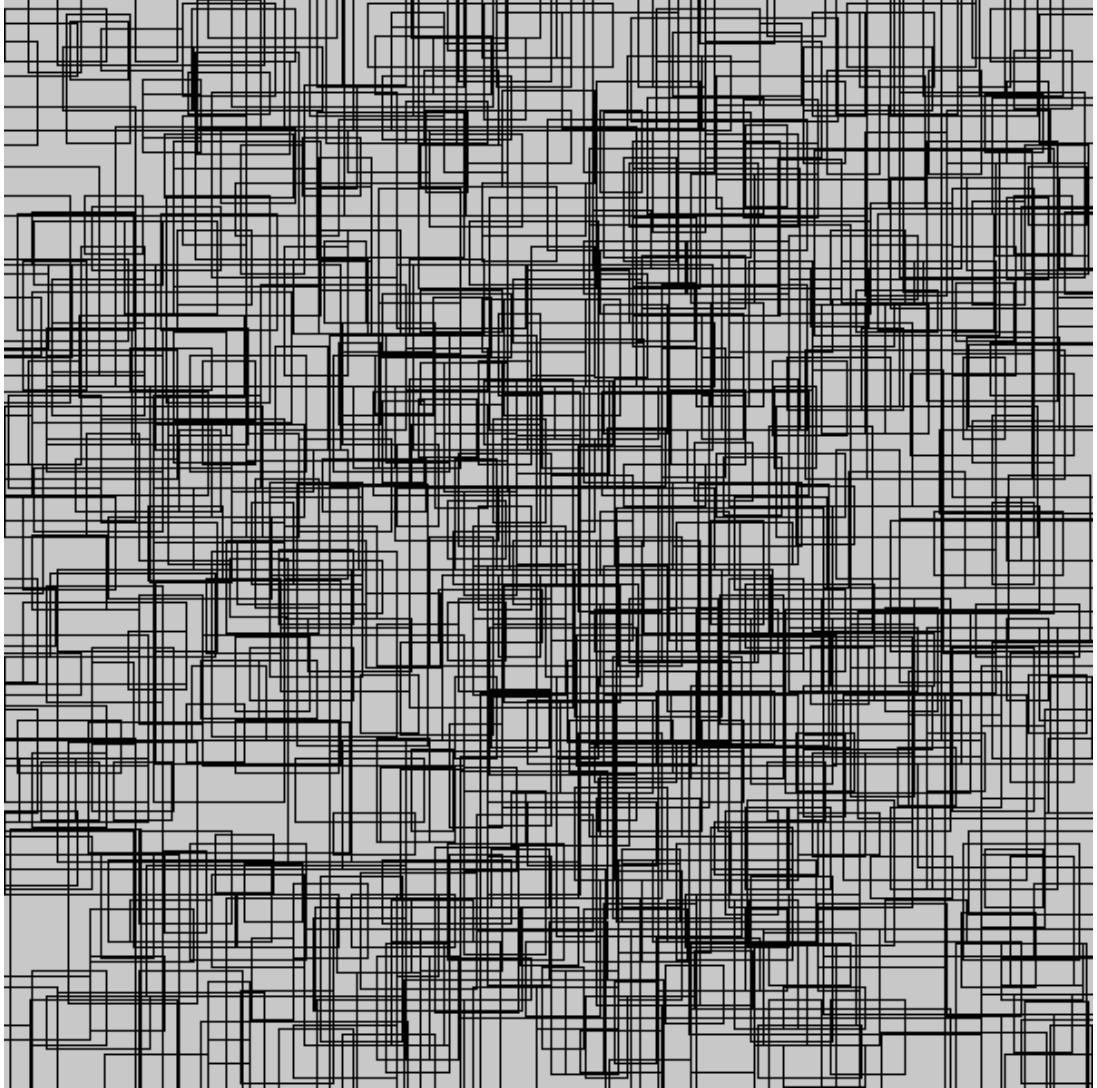


Fig. 3.15 Default anchor box sizes visualized for representative purpose

3.4 EXPERIMENTAL SETUP

3.4.1 Data Set

All the image samples are image frames of the Pixar movie [3] and Gaussian noise of varied standard deviation has been added to diversify the DataSet [15]. All the images are of the size 256×144 . The observed images set consists of the noisy images and the ground truth image set consists of the corresponding clean images. The test set for YOLO Version 3 with improved IoU are nothing but the denoised images which are output of the denoising GAN. The camera-taken images of the test set for improved YOLO Version 3 are taken from the drive.ai sample dataset [25]. CT scan [26] and MRI

Images [26] are also included in the test set. We have also created custom test set of about 100 photo-realistic rendered images from Pixar movies [3].

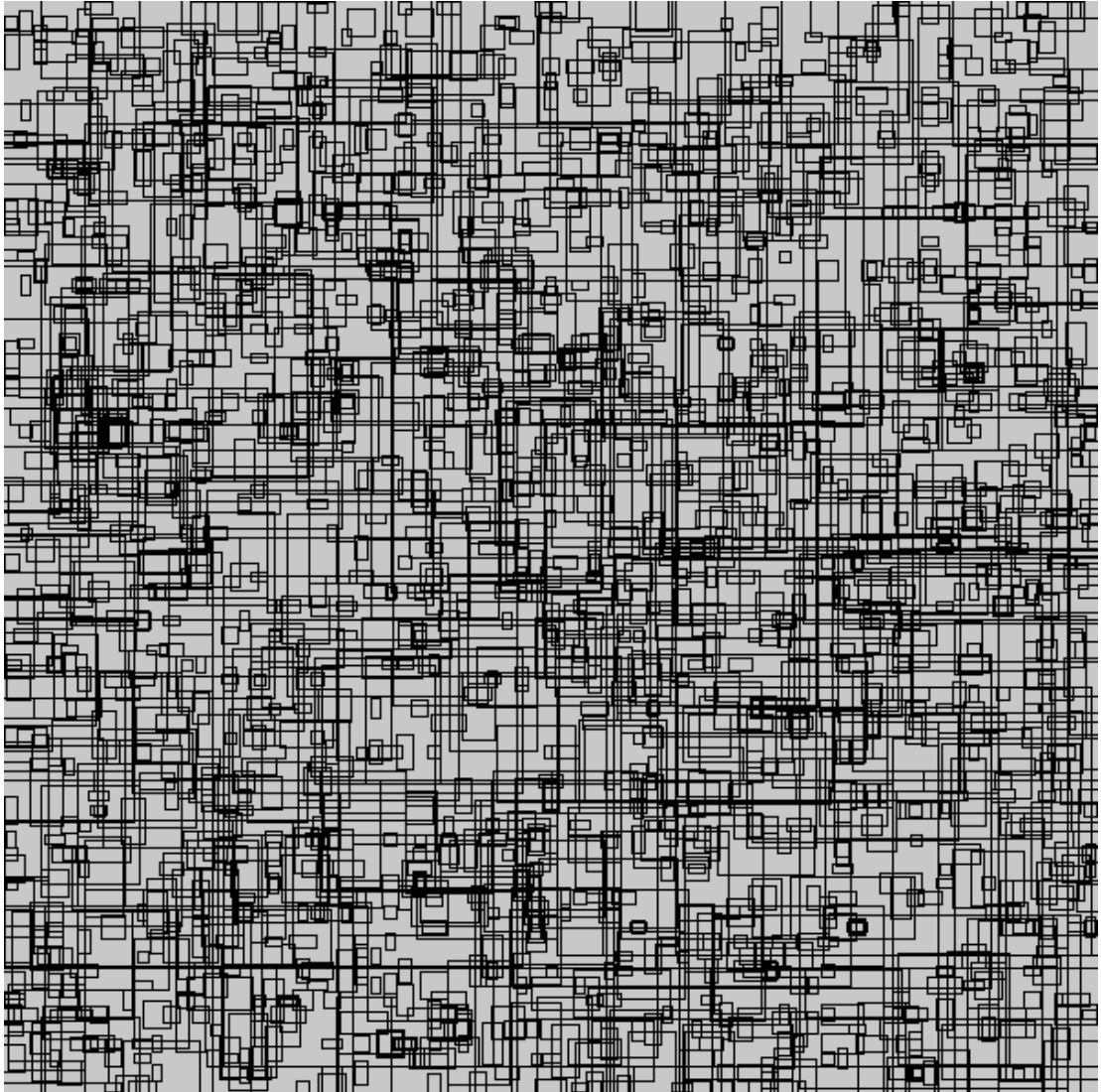


Fig. 3.16 Custom anchor box sizes visualized for representative purpose

3.4.2 Hardware Used

- Lenovo Ideapad 320 Notebook
- Nvidia GeForce 920MX/PCIe/SSE2 GPU
- Intel Core i5-7200U CPU @ 2.50GHz × 4
- 1 TB SATA Hard Disk
- 8.00 GB RAM

3.4.3 Software Used

- Ubuntu 16.04 LTS 64-bit Operating System
- Python 3
- Tensorflow 1.12 API
- LabelImg Tool
- PyCharm 2018.2.4 (Community Edition)
- Jupyter Notebook
- Open CV Version 3

CHAPTER 4

RESULTS AND DISCUSSION

4.1 RESULTS

The results of GAN based object detection of noisy images are illustrated and discussed here. Using GAN, we did denoising of photo-realistic rendered images which have varying amounts of monochromatic Gaussian and monochromatic Uniform Noises in them.

4.1.1 Denoising of Photo-realistic Rendered Images with Monochromatic Gaussian Noise Using GAN

Using GAN, denoising is done for photo-realistic rendered images with monochromatic gaussian noise and the results are shown in figure 4.1.



Fig. 4.1 GAN results on photo-realistic rendered images with monochromatic gaussian noise

4.1.2 Denoising of Photo-realistic Rendered Images with Monochromatic Uniform Noise Using GAN

Using GAN, denoising is done for photo-realistic rendered images with monochromatic Uniform Noise and the corresponding results are depicted in the figure 4.2.



Fig. 4.2 GAN results on photo-realistic rendered images with monochromatic uniform noise

4.1.3 Denoising of CT Scan and MRI Images with Monochromatic Gaussian Noise Using GAN

Using GAN, denoising is done for CT Scan images with monochromatic gaussian noise and MRI images with monochromatic gaussian noise and the results are depicted in figure 4.3 and figure 4.4.

CT Scan

Noisy Image



Denoised Image

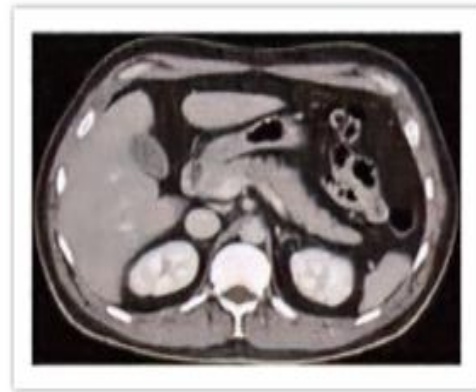


Fig. 4.3 Results on CT Scan

MRI Scan

Noisy Image



Denoised Image



Fig. 4.4 Results on MRI

4.1.4 Denoising of CT Scan and MRI Images with Monochromatic Uniform Noise Using GAN

Using GAN, denoising is done for CT scans image with monochromatic uniform noise and MRI images which have monochromatic uniform noise and the results are shown in the following figures, Figure 4.5 and Figure 4.6.

Uniform Monochromatic Noise

Noisy Image



Denoised Image



Fig. 4.5 Results on CT Scan

Uniform Monochromatic Noise

Noisy Image



Denoised Image

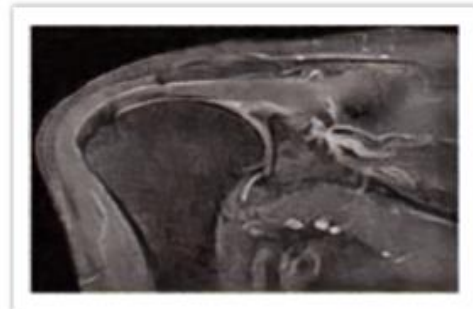


Fig. 4.6 Results on MRI

4.1.5 Denoised Images of Camera-taken Real Noisy Images using GAN

Using GAN, denoising of images that are taken by camera with real noisy images is performed and the results are depicted in the following figure 4.7.

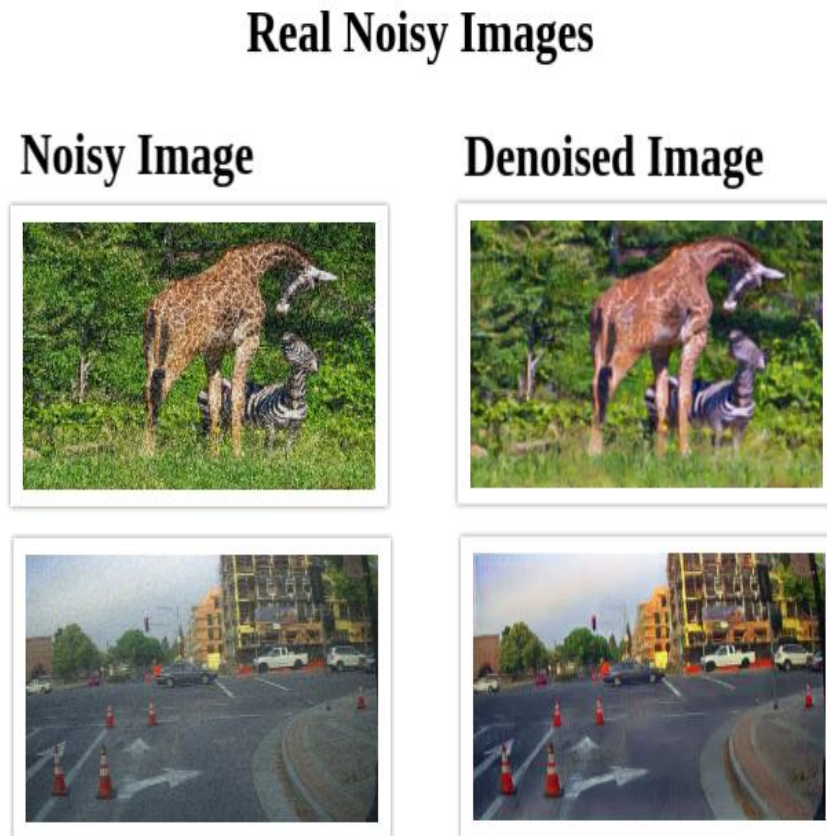


Fig. 4.7 GAN Results on real-noisy images

4.1.6 Object Detection in Photo-Realistic Rendered Images Using YOLO Version 3 with Improved IOU

Object detection has been done on photo-realistic rendered images using YOLO Version 3 with Improved IoU and the results of cross-depiction problem is depicted in the following figure 4.8.

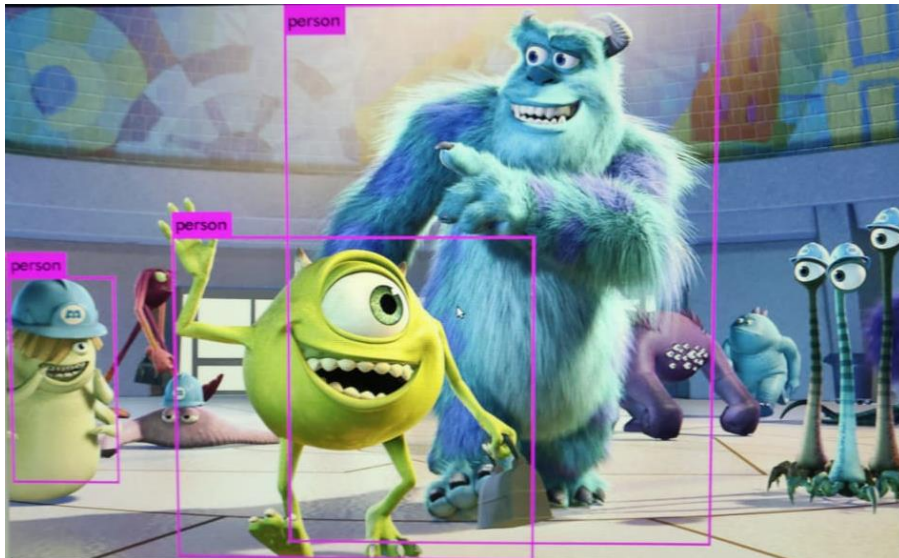


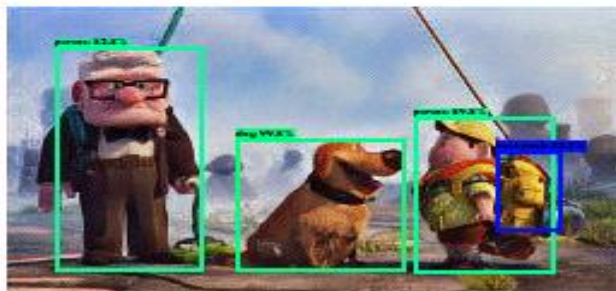
Fig. 4.8 Object Detection in Photo-Realistic Image 1

4.1.7 Object Detection in Denoised Images Using YOLO Version 3 with Improved IoU

Object detection has been done on denoised photo-realistic rendered images using YOLO Version 3 with Improved IoU and the results of various class detections are depicted in the following figures, figure 4.9, figure 4.10 and figure 4.11.

```

Class name: person
Confidence: 89.77555632591248
Class name: person
Confidence: 82.77071118354797
Class name: dog
Confidence: 99.75026249885559
Class name: backpack
Confidence: 53.20969223976135
  
```



Total time: 20.80360507965088 secs

Fig. 4.9 Object Detection on Denoised Photo-Realistic Rendered Image 1

Class name: person
Confidence: 89.17433023452759
Class name: person
Confidence: 83.43595862388611

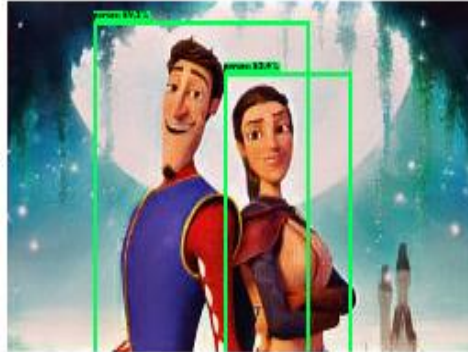


Fig. 4.10 Object Detection on Denoised Photo-Realistic Rendered Image 2

Class name: person
Confidence: 99.95991587638855
Class name: laptop
Confidence: 98.8497257232666

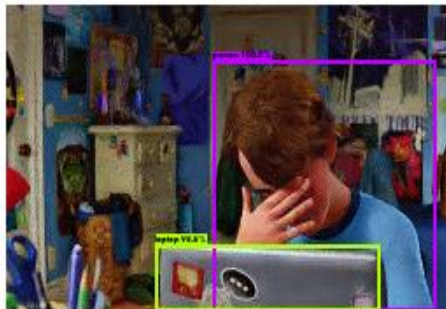
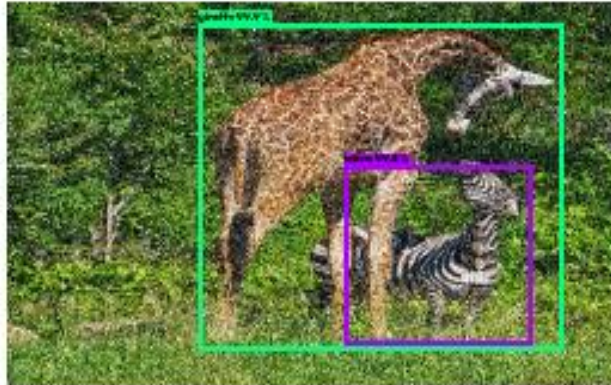


Fig. 4.11 Object Detection on Denoised Photo-Realistic Rendered Image 3

4.1.8 Object Detection in Photos Taken from Digital Camera Using YOLO Version 3 with improved IoU

Object detection has been done on denoised camera-captured images using YOLO Version 3 with Improved IoU and the results of various class detections are depicted in figure 4.12.

Class name: zebra
Confidence: 99.80953335762024
Class name: giraffe
Confidence: 99.93719458580017



Total time: 29.125305891036987 secs

Fig. 4.12 Object Detection in Denoised Digital Image

4.2 FALSE POSITIVES AND FALSE NEGATIVES

4.2.1 False Positives

False positive is a test result that wrongly shows that a particular attribute is present.

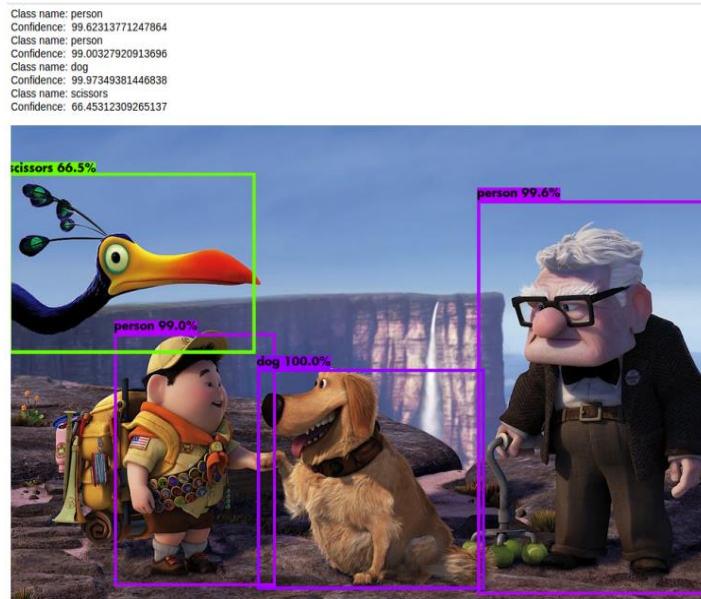


Fig. 4.13 Example of False Positive

In the Figure 4.13, even though an object scissors is not present, it still shows that it is present, thus a false positive.

4.2.2 False Negatives

False negative is a test result that wrongly shows that a particular attribute is absent. In the Figure 4.14, even though an object person is present thrice, it detected it only one instance and left the other two instances of the person class undetected, thus a false negative.

Class name: person
Confidence: 64.1047716140747
Class name: sports ball
Confidence: 55.05834221839905

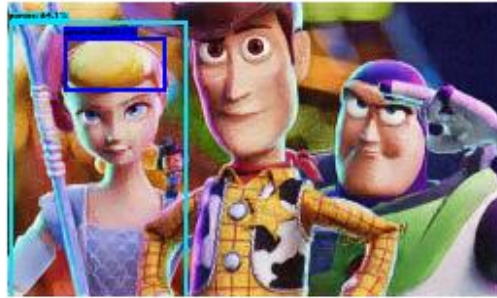


Fig. 4.14 Example of False Negative

CHAPTER 5

CONCLUSION

In this project, we implemented the proposed pipeline for denoising photo-realistic images using GANs for detecting objects using YOLOv3 with improved IoU. We discovered the slope at which Leaky ReLU works best for our dataset, also, we increased the mean IoU of the bounding boxes by generating custom anchor boxes using k means clustering for our dataset. Different noisy MRI and CT scan images, CGI and camera captured images with different types of noises and different amounts in them have been denoised using denoising GAN. We generalized YOLO for photo-realistic images. So real-time object detection in real-time rendering helps identify the objects and localize them in a rendered image. Denoising photo-realistic images helps reduce the production time from many hours to a few seconds. Thereby, improving the speed of real-time rendering.

CHAPTER 6

FUTURE WORK

In future, noises that are produced by Monte Carlo Rendering can be included and we can extend this to other types of noises other than just monochromatic noise. This can further be extended to work in real-time rendering and for real-time object detection and for object detection in the wild for counting the wild-life, to detect forest-fires, etc and during rain to detect the objects by deraining the images. Also this can be extended for other types of images like Artwork. We can also try implementing K-Means++ for initialization in K-Means clustering in YOLO algorithm.

REFERENCES

- [1] "<https://renderman.pixar.com/>," [Online].
- [2] "<https://sciencebehindpixar.org/pipeline/rendering/>," [Online].
- [3] "<https://www.pixar.com/feature-films-launch/>," [Online].
- [4] "Girshick, R.B., Donahue, J., Darrell, T., & Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. 2014 IEEE Conference on Computer Vision and Pattern Recognition, 580-587".
- [5] "R. B. Girshick. Fast R-CNN. CoRR, abs/1504.08083, 2015R. B. Girshick. Fast R-CNN. CoRR, abs/1504.08083, 2015".
- [6] "Ren, S., He, K., Girshick, R., et al.: 'Faster R-Cnn: Towards Real-Time Object Detection with Region Proposal Networks', IEEE transactions on pattern analysis and machine intelligence, 2017, 39, (6), pp. 1137–1149".
- [7] "J. Redmon. Darknet: Open source neural networks in c. <http://pjreddie.com/darknet/>," [Online].
- [8] "J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 779-788".
- [9] "Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. In Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on, pages 6517–6525. IEEE, 2017".
- [10] "J. Redmon and A. Farhadi. Yolov3: An incremental improvement. ArXiv, 2018".
- [11] "W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In European conference on computer vision, pages 21–37. Springer, 2016".
- [12] "Zhang, Qianru & Zhang, Meng & Chen, Tinghuan & Sun, Zhifei & Ma, Yuzhe & Yu, Bei. (2018). Recent Advances in Convolutional Neural Network Acceleration".
- [13] "Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, and Aaron Courville Yoshua Bengio. Generative adversarial nets. In NIPS, 2014".
- [14] "William Fedus, Mihaela Rosca, Balaji Lakshminarayanan, Andrew M. Dai, Shakir Mohamed, and Ian Goodfellow. Many paths to equilibrium: GANs do not need to

decrease a divergence at every step. In ICLR, 2018”.

- [15] “A. Alsaiari, R. Rustagi, A. Alhakamy, M. M. Thomas and A. G. Forbes, "Image Denoising Using A Generative Adversarial Network," 2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT), Kahului, HI, USA, 2019, pp. 126-132”.
- [16] “H. Cai, Q. Wu, T. Corradi, and P. Hall. The cross-depiction problem: Computer vision algorithms for recognising objects in artwork and in photographs. arXiv preprint arXiv:1505.00110, 2015”.
- [17] “https://en.wikipedia.org/wiki/K-means_clustering,” [Online].
- [18] “<http://www.image-net.org/>,” [Online].
- [19] “<http://host.robots.ox.ac.uk/pascal/VOC/>,” [Online].
- [20] “Tan, Chuanqi & Sun, Fuchun & Kong, Tao & Zhang, Wenchang & Yang, Chao & Liu, Chunfang. (2018). A Survey on Deep Transfer Learning: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4–7, 2018, Proceedings, Part III. 10.10”.
- [21] “C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. CoRR, abs/1602.07261, 2016”.
- [22] “http://www.robots.ox.ac.uk/~vgg/research/very_deep/,” [Online].
- [23] “He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 770-778)”.
- [24] M. i. A. P. B. J. C. Z. C. A. D. J. D. M. D. S. G. G. I. M. I. M. K. J. L. R. M. S. M. D. M. B. S. P. T. V. V. P. W. M. W. Y. Y. and X. Z. , “Tensorflow: A system for large-scale machine learning,” in *12th USENIX Conference on Operating Systems Design and Implementation, OSDI'16*, Berkeley, CA, USA, 2016.
- [25] “<https://www.drive.ai/>,” [Online].
- [26] “<https://imaging.sansumclinic.org/medical-services/medical-service/details/radiology/>,” [Online].
- [27] “J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama et al., "Speed/accuracy trade-offs for modern convolutional object detectors", IEEE CVPR, 2017”.

- [28] "T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In European conference on computer vision, pages 740–755. Springer, 2014".
- [29] "Sang, Jun et al. "An Improved YOLOv2 for Vehicle Detection." Sensors (Basel, Switzerland) vol. 18,12 4272. 4 Dec. 2018, doi:10.3390/s18124272".
- [30] "<http://cocodataset.org/#home>," [Online].
- [31] "Lin, Tsung-Yi & Goyal, Priya & Girshick, Ross & He, Kaiming & Dollar, Piotr. (2017). Focal Loss for Dense Object Detection. 2999-3007. 10.1109/ICCV.2017.324".

LIST OF PUBLICATIONS

- [1] A research paper titled “A Novel Cipher using Cipher Squares” has been submitted.