Social Distancing AI

Using super-resolution to train an object detection model on low resolution images

Dennis Hein

Bachelor's thesis
2020:K23

Lund University

Faculty of Science
Centre for Mathematical Sciences
Mathematical Statistics

Centrum Scientiarum Mathematicarum

Social Distancing AI: Using super-resolution to train an object detection model on low resolution images

Dennis Oscar Hein

September 2020

Abstract

This paper addresses the following hypothetical situation: suppose that we want to train an object detection algorithm but that only low resolution data is available. As a tentative solution to this problem, this paper suggest to first super-resolve the low resolution data to obtain higher resolution data that is then fed to the object detection algorithm. Super-resolution is a technique that takes a lower resolution image and "paints" in extra details in a very convincing way to produce a higher resolution output. To evaluate this approach, this paper trains the object detection model on to separate datasets: 4× down-sampled and subsequently super-resolved images and the original images. For super-resolution we use ESRGAN which is based on RaGAN. Quantitative comparison suggests that there is little difference between the two networks and thus provides some support for the approach explored in this paper. Qualitative comparison indicates that this technique only works in certain cases–when super-resolution results in images that can properly be annotated. If the contrast between the objects of interest and the background is great enough, then super-resolution will result in much clearer images that are easily annotated. In this particular scenario, using super-resolution to train object detection algorithms represents a feasible solution to the hypothetical situation assessed in this paper. This approach is tested in the context of detecting social distancing violations in restaurants using areal footage. This is of high relevance in 2020 due to the ongoing Coronavirus pandemic.

Popular Abstract

This paper addresses the following situation: suppose that we want to train an object detection model but we only have access to low resolution data. This situation can emerge if we wish to use satellite data and detect "smaller," for instance table sized, objects. The particular application of interest is to use satellite data to monitor violations of social distancing guideline imposed during the COVID-19 pandemic. As a tentative solution to the above problem, we suggest to first super-resolve the low resolution data to obtain a higher resolution dataset. This dataset is subsequently fed to an object detection model. Super-resolving these low resolution images might allow for the human eye to detect the presence of the objects of interest. If this is the case then the images can be annotated and used as training data for an object detection model. To assess the suggested approach, this paper uses drone footage shot over Lund, Sweden. The drone footage is of very high resolution compared to satellite imagery. To simulate the hypothetical scenario stated above these images are down-sampled to produce low quality images. Down-sampling will reduce the amount of pixels in the images and thus make it more blurry. To assess the merit of the suggested approach we train an object detection model on two separate datasets: the original images and images that have been down-sampled and subsequently super-resolved. The relative performance is the assessed both qualitatively, by visually observing the out, and quantitatively, by using some commonly used performance metrics from computer vision. In this paper we also show that it is possible to use the down-sampled and then super-resolved images the train an object detection algorithm to detect tables in the outdoor seating area of restaurants. To further show the utility of this application we create a program that will detect the distance between the center of the detected tables. The purpose of this additional feature is to detect violations of social distancing regulations imposed as a measure to curb the spread of COVID-19.

The key mathematical component in this paper is the Generative Adversarial Network (GAN). The original GAN was introduced in 2014 and have since received a lot of attention. GANs are essentially trying to solve the problem: given that we have a dataset of objects with a degree of consistency, can we generate similar objects artificially? The GAN trains a generator which generates the artificial objects by pitting it against a discriminator in a minimax game. Perhaps the best way to build some intuition is with the following analogy. The generator can be thought of as a team of counterfeiters trying to produce some fake product, for instance currency. The discriminator can be thought of as the police, that is trying to catch the counterfeit currency but at the same time allowing genuine currency to be circulated. Competition between the counterfeiters and police will result in iterative improvement until the counterfeit currency is indistinguishable from the genuine currency. GANs have several interesting applications and in this paper we are concerned with super-resolution. Super-resolution is an image transformation task which involves estimating a higher resolution image from it's low resolution counterpart. This is a fundamentally ill-posed problem as there are myriad of high resolution images that could be associated with the low resolution counterpart. GANs are useful in super-resolution as the generator can produce a higher resolution image by "painting" details into the lower resolution image in a perceptually convincing way.

This project shows that it is possible to use the suggested approach to train a model that is able to detect violations of social distancing regulations using down-sampled drone footage. For this method to work it is crucial that the super-resolved images can be properly annotated. Annotating involves marking bounding boxes around the objects of interest and giving them a label. If these objects are not identifiable, then they cannot be annotated and the object detection model will be penalized during training for correctly predicting objects at these locations. For super-resolution to produce clearer images that are easily annotated it is important that the contrast between the object of interest and the background is sufficiently high.

Acknowledgements

I would like to thank Alexandros Sopasakis for excellent supervision throughout this project.

Contents

List of Figures

List of Tables

1 Introduction

Since its introduction in 2014 by [1] Generative Adversarial Networks (GANs) have received a lot of attention. GANs train the generator by pitting it against a discriminator in a minimax game. Perhaps the best way to build some intuition is with the following analogy. The generator can be thought of as a team of counterfeiters trying to produce some fake product, for instance currency. The discriminator can be thought of as the police, that is trying to catch the counterfeit currency but at the same time allowing genuine currency to be circulated. Competition between the counterfeiters and police will result in iterative improvement until the counterfeit currency is indistinguishable from the genuine currency. GANs have several interesting applications and in this paper we are concerned with the image transformation problem called super-resolution. Super-resolution is a technique that involves estimating a high resolution image from its lower resolution counter-part [2]. This is a fundamentally ill-posed problem since there are a myriad of possible high resolution images that associated with the low resolution input. Going from the lower resolution image to the high resolution one requires add extra detail into the image. GANs lend themselves well to this task as the generator can be used to "paint" in additional information in a visually very pleasing way. Using GANs for super-resolution was first introduced in Super-resolution Generative Adversarial Network (SRGAN) by [2] and then further developed in Enhanced Super-resolution Generative Adversarial Network (ESRGAN) by [3]. In this paper, we suggest using ESRGAN to super-resolve low quality images before being fed to an object detection model. The purpose of this exercise is to address the following hypothetical situation: suppose you need to train an object detection model but that you only have access to low resolution images. The practical situation of interest in this paper is when you need to use areal footage, mainly in terms of satellite imagery, and that these images come in too low resolution for them to be properly annotated and used as training data in an object detection model. Super-resolving these low resolution images might allow for the human eye to detect the presence of the objects of interest. If this is the case then the images can be annotated and used as training data for an object detection model. For further inference, new previously unseen data can be super-resolved before being given to the object detection model.

To evaluate the suggested approach, this paper uses drone footage shot over Lund. The original imagery is of very high resolution and will therefore be used as control. To simulate the hypothetical scenario stated above these images are $4\times$ down-sampled to produce low quality images. So, instead of the original images which are 1280×720 the low quality images are only 320×180. These images are subsequently super-resolved, resulting in images that are again 1280×720. We then train two separate object detection models on the original images and the images that have been down-sampled and then super-resolved. The merit of this approach is evaluated by comparing the performance of these two models. In this paper we also show that it is possible to use the down-sampled and then super-resolved images the train an object detection algorithm to detect tables in the outdoor seating area of restaurants. To further show the utility of this application we create a program that will detect the distance between the center of the detected tables. The purpose of this additional feature is to detect violations of social distancing regulations imposed as a measure to curb the spread of COVID-19. The longer term objective of this project is to use satellite imagery, instead of drone imagery, to help monitor adherence to social distancing guide lines. The results in this paper should be considered as a proof of concept towards that objective.

The rest of the paper is organized as follows. In chapter 2 we will cover briefly the most relevant theory for this paper. We will start by looking at the mechanics of MLPs and then more specifically convolutional neural networks (ConvNets). This is followed by a brief overview of some standard gradient based optimization techniques. This is relevant for training both super-resolution and object detection algorithms. We will subsequently go into the mechanics of the main models in this paper. The focus will be on the most important statistical mechanics behind ESRGAN. For the sake of brevity we will remain largely agnostic about other details. This is a paper that is fundamentally practical in nature and we do not aim for theoretical rigor and completeness. Both ESRGAN and YOLO are state of the art models from

their respective field and thus contain a lot of bells and whistles. To give a complete account on all details is simply beyond the scope of this paper. Hence, we only stress the parts that are of most importance from the perspective of mathematical statistics. In chapter 3 the actual implementation is carried out. Here we discuss data considerations, training of the model, and our final results. Finally in chapter 4 we conclude and discuss several important limitations and suggest improvements that can be made to the approach taken in this paper.

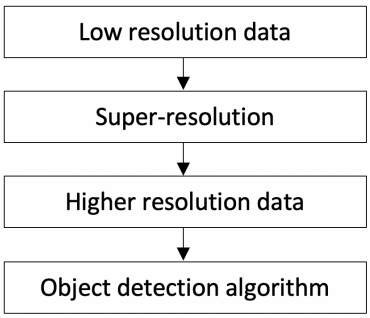


Figure 1: Overview of suggested approach. We start with low resolution input, in this case we have images that are of low resolution. The low resolution input is then super-resolved yielding higher resolution data. This is subsequently treated as the only data seen by the object detection algorithm—both for training and testing.

2 Theory

2.1 Deep learning

2.1.1 MLPs

Multilayered perceptrons (MLPs), also called feedforward neural networks, are one of the most common deep learning models. They are called feedforward as information only flows in one direction (i.e. unlike recurrent neural networks there are no feedback connections) [4]. The objective of MLPs are to approximate some function of interest. For instance, for a classification model, this function is the mapping of input into categories. When training the model, or using it for prediction, only the value of the output layer is shown. It is for this reason that the other layers are usually referred to as hidden layers. Deeper models tend to perform better [5]. The notion that deeper is better played a central part in the development of many famous architectures such as VGGnet [6] and ResNet [7] [8]. Nevertheless, a MLP with only one hidden layer is sufficient to approximate any function. This is a result called the universal approximation theorem [9] [10]. In particular, the universal approximation theorem states that any MLP with a linear output layer and at least one hidden layer with any "squashing" activation function (e.g., logistic sigmoid) can approximate[1] any Borel measurable[2] function from one finite-dimensional space to another given that the network has sufficient hidden units. Note that the universal approximation theorem only guarantees that there exists a MLP that approximates any function arbitrarily well. However, there is no guarantee that the training algorithm that we are using is able to learn that function [4].

[1] With any non-zero error.

[2] Following [4] we do not dive into the specifics of Borel measurability but only recognize that any continuous function on a compact (closed and bounded) subset of $\mathbb{R}^n$ is Borel measurable.

2.1.2 ConvNets

Convectional Neural Networks (ConvNets) are MLPs that applies the convolution operator in at least one of its layers [4]. When dealing with ConvNets one usually is required to work with higher order matrices (i.e. tensors). For instance, the input in a ConvNet is usually an colored image which is an order 3 tensor. This image will be stored as a $H \times W \times D$ tensor where H and W is the high and width of the image (e.g., 1280×720) and D represents the color channels (usually RGB format[3]). To make working with tensors more convenient, we occasionally want to store it as a long vector. For instance, we might like to store the colored imaged as a $HWD \times 1$ vector (which is an order 1 tensor). We will denote the vectorization as $\text{vec}(\cdot)$. Consider the following example for an order 2 tensor:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \ \text{vec}(A) = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}. \tag{1}$$

Following [11], we use equation 2 to give an abstract description of ConvNets:

$$\boldsymbol{x}^1 \to \boxed{\boldsymbol{w}^1} \to \boldsymbol{x}^2 \to ... \to \boldsymbol{x}^{L-1} \to \boxed{\boldsymbol{w}^{L-1}} \to \boldsymbol{x}^L \to \boxed{\boldsymbol{w}^L} \to z. \tag{2}$$

This represents the forward pass. The input, $\boldsymbol{x}^1$, an order 3 tensors goes through the L layers and results in an output $\boldsymbol{x}^L$. The parameters in layer i are denoted $\boldsymbol{w}^i$.[4] For a classification problem with C classes[5], $\boldsymbol{x}^L$ would be a C dimensional vector, where each component i represents the probability that the input is from class i. To make $\boldsymbol{x}^L$ a valid probability mass function, a transformation such as the softmax is commonly used. z in the final "layer" represents the loss. This is used to train the model. The loss will quantify the disagreement between the predicted output and the ground truth. For predictions we pass the input forward through to model. For training, we are interested in the both directions: a forward and backward pass [11]. The most common approach to training is using some version of gradient descent. The parameters in the i-th layer follow the update

$$\boldsymbol{w}^i = \boldsymbol{w}^i - \eta \frac{\partial z}{\partial \boldsymbol{w}^i} \tag{3}$$

As mentioned below, it is usually wasteful to compute the gradients for the entire sample and thus stochastic, also called stochastic mini-batch, gradient descent is most frequently used instead. The backward pass is used to simplify the task of calculating the gradients. In error back propagation, back propagation, or simply backprop, two sets of gradients are computed in each layer[11]:

- $\frac{\partial z}{\partial \boldsymbol{w}^i}$ as in (3) is used to update the current i-th layer and
- $\frac{\partial z}{\partial \boldsymbol{x}^i}$ which is used to update the parameters in the $(i-1)$-th layer (i.e. to update the parameters backwards).

In order words, for each layer i we compute the partial derivatives of z with respect to the input $\boldsymbol{x}^i$ and parameters $\boldsymbol{w}^i$ of that layer. Since $\boldsymbol{x}^i$ is the output of the $(i-1)$-th layer, the second term can be thought of as the part of the "error" supervision information propagated from z backward to the current layer. Continuing in this layer by layer fashion one can back propagate all the way from z to the input layer. Hence, the term back propagation.

[3] RGB stands from red, green and blue. Each channel takes a value from 0 to 255. For instance, the color red is $(255, 0, 0)$.
[4] Here the biases (intercepts) $\boldsymbol{b}^i$ are ignored for simplicity. We will later use the notation $\boldsymbol{\theta}^i := (\boldsymbol{w}^i, \boldsymbol{b}^i)$ to also include biases.
[5] The classification problem of interest in this paper only has one class: tables.

2.1.2.1 ReLU The mapping of semantic information contained in an image into a class prediction is highly nonlinear [11]. The rectified linear unit (ReLU) is a commonly used layer to introduce nonlinearity into ConvNets without suffering from issues of vanishing and exploding gradients associated with, for instance, logistic sigmod layers [4]. Suppose that we are in the l-th layer with input $\boldsymbol{x}^l \in \mathbb{R}^{H^l \times W^l \times D^l}$. This layer will transform $\boldsymbol{x}^l$ into output $\boldsymbol{y}$ which is also the input of the $(l+1)$-th layer, $\boldsymbol{x}^{l+1}$. Any input in the order 3 tensor $\boldsymbol{x}^l$ is given by the triplet (i^l, j^l, d^l), where we follow [11] in using zero-based indexing convention such that $0 \leq i^l < H^l,\ 0 \leq j^l < W^l$ and $0 \leq d^l < D^l$. The ReLU can then be written as

$$y_{i,j,d} = \max\{0, x^l_{i,j,d}\} \tag{4}$$

with $0 \leq i^l < H^l = H^{l+1},\ 0 \leq j^l < W^l = W^{l+1}$ and $0 \leq d^l < D^l + D^{l+1}$ since the ReLU does not alter the size of the input. The intuition behind ReLU is as follows. Suppose that $x^l_{i,j,d} > 0$ if the region inside an image contains patterns that can be used to identify, for instance, a dog. If it does not have these patters then $x^l_{i,j,d} \leq 0$. ReLU will then only "activate" with from images that contain these useful patterns [11]. This helps with the vanishing gradient problem as the features that are set to 0 in the ReLU layer are not of interest to us, i.e. those are not activated features.

2.1.2.2 Convolution Let $\boldsymbol{f} \in \mathbb{R}^{H \times W \times D^l \times D}$ denote D kernels with spatial extent $H \times W$ and D^l channels [11]. Note that for a convolutional layer in layer l, $\boldsymbol{f}$ is equivalent to $\boldsymbol{w}^l$ in equation 2. In contrast to the ReLU layer, the convolutional layer will likely alter the output size. In particular, the spatial extent of the output will be smaller than for the input for any kernel larger than 1×1 [11]. This can quickly shrink the spatial extent as one goes through several layers in the network. To maintain the same spatial extent for the output as for the input one can apply so-called "same" padding [4]. Suppose the input is $H^l \times W^l \times D^l$ and the kernel is of size $H \times W \times D^l \times D$, then the output will have size $(H^l - H + 1) \times (W^l - W + 1) \times D$ (see Appendix B). To apply "same" padding we add, or pad, $\lfloor \frac{H-1}{2} \rfloor$ rows of zeros above the first row and $\lfloor \frac{H}{2} \rfloor$ rows of zeros below the last row of the input. In addition, one needs to pad $\lfloor \frac{W-1}{2} \rfloor$ columns of zeros to the left of the first column and $\lfloor \frac{H}{2} \rfloor$ columns of zeros to the right of the last column of the input. Padding, also called zero-padding is an important concept in ConvNets. Another important concept is the notion of stride, s. As the kernel moves over the pixels in the input the stride determines how many pixel locations the kernel skips. Most common is to set $s = 1$, but $s > 1$ is also possible. Following [11], for the remaining of this section we consider the case $s = 1$ and no zero padding. The convolution procedure can formally be written as

$$y_{i^{l+1}, j^{l+1}, d} = \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} \sum_{d^l=0}^{D^l-1} f_{i,j,d^l,d} \times x^l_{i^{l+1}+i, j^{l+1}+j, d^l} \tag{5}$$

$\forall\ 0 \leq d \leq D = D^{l+1}$ and spatial location (i^{l+1}, j^{l+1}) such that $0 \leq i^{l+1}, H^l - H + 1 = H^{l+1}, 0 \leq j^{l+1}, W^l - W + 1 = W^{l+1}$. Now, equation 9 looks very complicated. Fortunately, it is possible to write this convolution as a matrix product. To simplify the discussion, let us consider an example. Suppose that we have a 2×2 kernel and 3×3 input matrix. Let $*$ denote the convolution operation then

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 12 & 16 \\ 24 & 28 \end{bmatrix}. \tag{6}$$

To rewrite this as a matrix product, we consider the Matlab command im2col [11]. Let A be the 3×3 input matrix then[6]

$$B = \text{im2col}(A, [2, 2]) = \begin{bmatrix} 1 & 4 & 2 & 5 \\ 4 & 7 & 5 & 8 \\ 2 & 5 & 3 & 6 \\ 5 & 8 & 6 & 9 \end{bmatrix}. \tag{7}$$

Now, if we transpose B and vectorize the kernel we have that

$$B^\top \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 12 \\ 24 \\ 16 \\ 28 \end{bmatrix}. \tag{8}$$

With proper reshaping of the right hand side in equation 12 we will obtain the right hand side in equation 6. Hence, the convolution operations can be achieved by matrix multiplication and it is thus a linear one [11]. The transpose of im2col, im2row, converts $\boldsymbol{x}^l$ into a matrix $\phi(\boldsymbol{x}^{\boldsymbol{l}})$. Recall that $\boldsymbol{f}$ is an order 4 tensor in $\mathbb{R}^{H \times W \times D^l \times D}$. If $D = 1$ this can be vectorized into a $HWD^l \times 1$ vector. For $D > 1$ we can convert the order 4 tensor into a $HWD^l \times D$ matrix. We denote this matrix F. All of this heavy notation leads to a surprisingly simple equation for the convolutional layer

$$\text{vec}(\boldsymbol{y}) = \text{vec}(\boldsymbol{x}^{l+1}) = \text{vec}\left(\phi(\boldsymbol{x}^l)F\right). \tag{9}$$

A sanity check is in order. $\text{vec}(\boldsymbol{y}) \in \mathbb{R}^{H^{l+1}W^{l+1}D}$, $\phi(\boldsymbol{x}^l) \in \mathbb{R}^{(H^{l+1}W^{l+1}) \times (HWD^l)}$, and $F \in \mathbb{R}^{(HWD^l) \times D}$. Hence, the right hand side has the correct dimension $(H^{l+1}W^{l+1}D) \times 1$.

2.1.2.3 Pooling Pooling layers are also an important part of ConvNets. There are two types of pooling layers that are most widely used: max pooling and average pooling. Max pooling maps a subregion into its maximum value and average pooling maps it to its average value. Let the l-th layer be $\boldsymbol{x}^l \in \mathbb{R}^{H^l \times W^l \times D^l}$. More formally, for spatial extent of pooling $H \times W$ max pooling is

$$y_{i^{l+1},j^{l+1},d} = \max_{0 \le i < H, 0 \le j < W} x^l_{i^{i+1} \times H + i, j^{l+1} \times W + j, d} \tag{10}$$

and average pooling is

$$y_{i^{l+1},j^{l+1},d} = \frac{1}{HW} \sum_{0 \le i < H, 0 \le j < W} x^l_{i^{i+1} \times H + i, j^{l+1} \times W + j, d} \tag{11}$$

where $0 \le i^{l+1}, H^{l+1}, 0 \le j^{l+1}, W^{l+1}$ and $0 \le d < D^{l+1} = D^l$.

2.1.3 Residual Neural Networks

As mentioned above, deeper models tend to perform better. It then seems simple to improve model performance: just add more layers. However, as noted in [7] and [8] this is not the case. With deeper models, accuracy eventually gets saturated (and may even be degrading) due to the fact that they become very difficult to train. This is referred to as the degradation problem in [7]. As a remedy they suggest to learn residual mappings instead of the more complex underlying mappings in every few stacked layers. A bit more formally, suppose that the underlying mapping we wish to learn is denoted by $\mathcal{H}(\boldsymbol{x})$. Then the residual map

[6] For more information on these commands see https://arxiv.org/abs/1412.4564. We are not concerned with the specifics of this function but only think of it as an auxiliary function that will help with rewriting the convolution as a matrix product.

is defined as $\mathcal{F}(\boldsymbol{x}) := \mathcal{H}(\boldsymbol{x}) - \boldsymbol{x}$ and the original map can be reformulated as $\mathcal{F}(\boldsymbol{x}) + \boldsymbol{x}$. Deep residual networks (ResNets) are based on the idea that the residual map $\mathcal{F}$ is easier to optimize than the original mapping. In practice this can be achieved by using so-called shortcut connections, which are connections that skip one or more layers. In this case the shortcut connection used is the identity map. Figure 2 shows an example of a building block in a ResNet. These shortcut or skip connections are used extensively in recent state of the art object detection models.

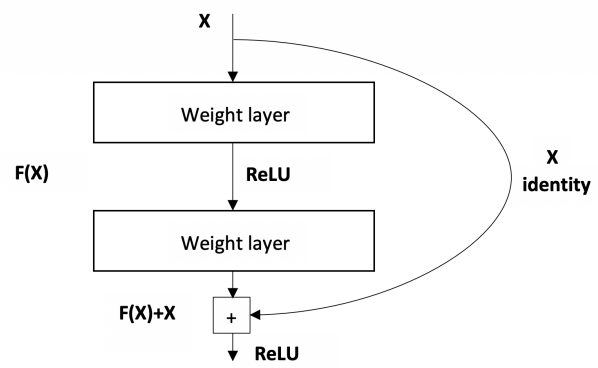


Figure 2: Illustration of a basic building block in ResNet. Reproduced from figure 2 in [7].

2.2 Optimization

2.2.1 Problem statement

In this section we focus on how to solve the following problem

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}). \tag{12}$$

That is, we wish to find the parameters $\boldsymbol{\theta}$ that minimizes the loss function J. Note that since most practical problems will be non-convex it may not be feasible to get the global minimum. A non-convex problem does not have any theoretical guarantees of convergence. Instead, we look for a $\boldsymbol{\theta}$ that makes the loss function as small as possible. Despite the lack of theoretical guarantees the empirical success enjoyed by deep learning models indicate that it is reasonable to use learning algorithms based on some form of gradient descent [4]. In a pure optimization problem, we would like to minimize the loss function on average across the data generating distribution, i.e.

$$J^*(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x},y\sim p_{data}} L(f(\boldsymbol{x},\boldsymbol{\theta}),y) \tag{13}$$

where $L(\cdot)$ is the per-example loss function. The expected loss is known as called risk. However, as we do not observe the data generating process we cannot optimize J^*. Instead, we minimize the loss on average over the data. Estimating p_{data} using the empirical distribution function leads to the empirical risk which for m training examples is

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x},y\sim \hat{p}_{data}} L(f(\boldsymbol{x},\boldsymbol{\theta}),y) = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)},\boldsymbol{\theta}),y^{(i)}). \tag{14}$$

Solving (10) using the empirical risk is known as empirical risk minimization [4]. The objective of machine learning algorithm is usually to provide a model that has a good out of sample fit. Empirical risk minimization

is very prone to overfitting and, using gradient based methods, excludes many useful loss functions. Therefore the actual function we set out to minimize is further away from (11) as surrogate loss functions and different forms of regularization is commonly used [4].

2.2.2 Stochastic gradient descent

2.2.2.1 Vanilla The idea behind gradient descent is perhaps best explained by the hill climbing analogy. Suppose that you are on top of a hill blindfolded and you want to find your way down. Iteratively taking a small step in the opposite direction of the gradient, the direction of steepest ascent, will eventually take you down to the bottom provided that the hill is convex and that you use a reasonable step size. In machine learning the step size is referred to as the learning rate. For a given learning rate ϵ, batch gradient descent for a dataset with M examples and initial parameter $\boldsymbol{\theta}$ would apply the following update:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \frac{1}{M} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{M} L(f(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}), y^{(i)}) \tag{15}$$

where we use "$\leftarrow$" instead of "$=$" to stress that this update is iterative. Nevertheless, this update is computationally very costly as it requires evaluating the model at all examples in the dataset. In addition, there is likely a large amount of redundancy. Consider the worst case scenario where all M examples are identical. The correct gradient could then be found by sampling a single example. This worst case scenario is unlikely to occur in any practical application, however there are nonetheless important benefits from randomly sampling a mini-batch $m < M$ and compute the gradient based on those m examples [4]. Another argument for using mini-batches instead of the entire sample is based on the standard error of the mean, which is given by $\sigma/\sqrt{m}$ for a sample of size m. Suppose that we draw samples of sizes 10^4 and 10^8. The second approach requires 10000 times as many computations but it decrease the standard error of the mean only by a factor of 100 [4]. By sampling a mini-batch of size m and making the update

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}), y^{(i)}). \tag{16}$$

we arrive at the "vanilla" stochastic gradient descent, or mini-batch gradient descent[7]. The learning rate is a very important hyperparameter and must be chosen with care. So far we have treated it as fixed; however, in practice it must be gradually decreased. We restate the following sufficient condition on the ϵ_k's from [4] without proof[8]. Let ϵ_k be the learning rate for iteration k, then a sufficient condition on the learning rate for stochastic gradient descent to converge is

$$\sum_{k=1}^{\infty} \epsilon_k = \infty \tag{17}$$

and

$$\sum_{k=1}^{\infty} \epsilon_k^2 < \infty. \tag{18}$$

Intuitively, this conditions states that the learning rate may not be too large nor too small.

[7] Note that this terminology is by no means universal. For some, stochastic gradient descent means exclusively when only one example is sampled. Nevertheless, we use this terminology where the case of $m = 1$ is a subset of the more general mini-batch sampling.

[8] This is a famous and well known statement. We omit the proof here as it is not the main focus of this paper.

2.2.2.2 Momentum Adding momentum to stochastic gradient descent may accelerate learning [12]. This algorithm will consider past gradients by a exponentially decaying moving average and let its direction guide it. We add a variable $\boldsymbol{v}$ and do the update

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} \epsilon \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}), y^{(i)}) \tag{19}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v} \tag{20}$$

where $\alpha \in [0, 1)$ is a hyperparameter that determines the strength of this exponentially decaying moving average of previous gradients. The name momentum derives from its physical analogy where $\boldsymbol{v}$ represents the velocity of a particle and we arrive at the momentum by multiplying this with its mass [4]. Continuing this physical analogy, if the algorithm repeatedly observes gradient $\boldsymbol{g}$ it will accelerate in the direction of $-\boldsymbol{g}$ until terminal velocity is reached. When reaching terminal velocity the step size will be

$$\frac{\epsilon ||\boldsymbol{g}||}{1 - \alpha} \tag{21}$$

and thus it might be useful to think of the hyperparameter α on the form $1/(1-\alpha)$[4]. For instance, $\alpha = 1/2$ implies that we multiply the maximum speed by $1/(1 - 1/2) = 2$ relative to the vanilla algorithm.

2.2.2.3 Adaptive learning parameters AdaGrad updates a global learning rate adaptively [13]. In particular, the updates are based on scaling the learning rate by the inverse square root of the sum of its previous values. Formally, for a global learning rate ϵ, a small constant to avoid numerical issues δ, and a initial gradient accumulation vector $\boldsymbol{r} = \boldsymbol{0}$, the update is found as follows

$$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}), y^{(i)}) \tag{22}$$

$$\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g} \tag{23}$$

$$\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g} \tag{24}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta} \tag{25}$$

where $\odot$ denotes the Hadamard product. The RMSProp algorithm [14] improves the performance of AdaGrad in non-convex settings by allowing gradient accumulation to follow an exponentially decaying moving average. Formally, let ρ be the decay rate, the update is computed as

$$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}), y^{(i)}) \tag{26}$$

$$\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g} \tag{27}$$

$$\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g} \tag{28}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}. \tag{29}$$

Note that the only difference from AdaGrad is how we update $\boldsymbol{r}$. Nevertheless, this increases performance and RMSProp is one of the standard go-to optimization algorithms used in practical applications [4].

2.2.2.4 ADAM ADAM [15] can be thought of as a combination of momentum and RMSProp. It derives it name from "adaptive moments." It estimates the first-order moment (momentum) and the (uncentered) second-order moment whose biased version are updated by exponentially weighted moving averages with decay parameters ρ_1, ρ_2 [4]. These are biased as they are initialized at the origin. To correct for this bias, ADAM applies a bias correction to each term. For initial 1st and 2nd order moment variables $\boldsymbol{s} = \mathbf{0}, \boldsymbol{s} = \mathbf{0}$ and time step $t = 1$ the update is computed as

$$\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}), y^{(i)}) \tag{30}$$

$$t \leftarrow t + 1 \tag{31}$$

$$\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g} \tag{32}$$

$$\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g} \tag{33}$$

$$\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t} \tag{34}$$

$$\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t} \tag{35}$$

$$\Delta\boldsymbol{\theta} \leftarrow -\epsilon \frac{\hat{\boldsymbol{s}}}{\delta + \sqrt{\hat{\boldsymbol{r}}}} \tag{36}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}. \tag{37}$$

ADAM is another commonly used optimization algorithm that we will use below when training our YOLO model.

2.3 Generative Adversarial Networks

2.3.1 Introduction

Since its introduction in 2014 by the seminal paper [1], Generative Adversarial Networks (GANs) has been a topic that has received a lot of attention. This popularity has lead to an abundance of new versions and applications. One application that has worked particularly well is super-resolution. In this section we will cover the most basic building blocks of the original GAN in [1]. As correctly noted in [16], this presentation is not to be considered formal and instead we stress the intuition behind the most important mechanics. The interested reading is referred to [16] and references therein for a more formal mathematical treatment of GANs.

2.3.2 Intuition

The term generative model is used in the sense that the model uses the training data, which consists of samples drawn from p_{data}, to estimate p_{data} [17]. There are two main ways to go about this. First, one can explicitly model p_{data} by either using some tractable density or some form of approximation. The approximation approach is taken in variational autoencoders (VAE) [18] [19] and Boltzmann machines [20] [21]. The second approach is to model p_{data} implicitly in the sense that the model only generates samples drawn from p_{data}. GANs usually take this latter approach. The "adversarial" part comes from the main idea in GANs which is to train two separate networks that compete with each other. There is a generative model G and a discriminative model D whose objective is to estimate the probability that an observed sample came from the training data rather than the generative model G [1]. The training of G corresponds to maximizing the probability that D is incorrect. Hence, the problem is framed as a minimax game between two players.

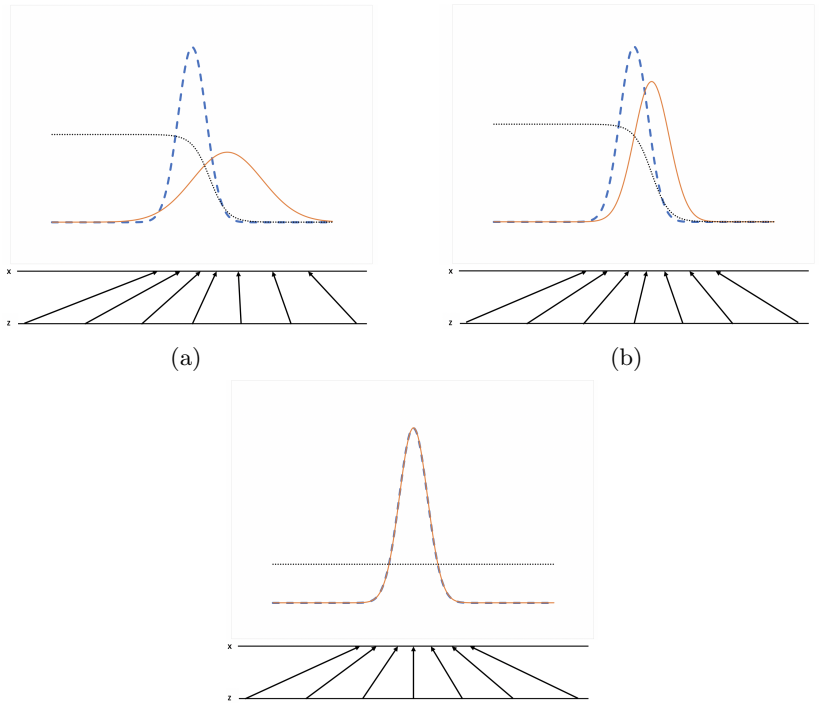


Figure 3: The blue dashed curve is p_{data}, the orange solid curve is p_g (G), and the dotted black one is the discriminator D. The upper horizontal line is a part of the domain $\boldsymbol{x}$ and the lower horizontal line is the domain from which we sample $\boldsymbol{z}$. The upward arrows represent the mapping $\boldsymbol{x} = G(\boldsymbol{z})$. We can see how G imposes the non-uniform distribution p_g on the generated samples. (a) Close to convergence. p_{data} is similar to p_g and D is trained to estimate the probability that a given sample is from p_{data}, eventually converging to $D(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x})+p_g(\boldsymbol{x})}$. (b) G flows to regions that are more likely to be classified as from p_{data}. This is to "trick" D into assigning high values to samples from G. (c) After several steps, $p_{data} = p_g$ if G and D have enough capacity. At this stage the two distributions are indistinguishable and D is unable to discriminate. Thus, $D(\boldsymbol{x}) = \frac{1}{2}$. This is a partial reproduction of figure 1 in [1].

To improve our intuition we can think of this as a game between a team of counterfeiters and the police. The counterfeiters objective is to make their fake product as similar to the real one as possible. Simultaneously, the police wish to allow for circulation of the real product and catch the counterfeiters. Competition in this game will drive both teams to improve their performance until the counterfeit and real products are indistinguishable [1]. An illustration is given in figure 3.

2.3.3 More formal results

As mentioned above, MLPs are extremely good at approximating essentially any mapping from the data to the desired output. It therefore comes as no surprise that we will use MLPs for the discriminator and generator. As mentioned above, GANs usually model the data distribution only implicitly in the sense that we can only observe samples from this estimated distribution. Let $\boldsymbol{z} \sim p_{\boldsymbol{z}}$ be a prior input noise variable. Then applying the map $\boldsymbol{x} = G(\boldsymbol{z}, \boldsymbol{\theta}_g)$ yields our sample from p_g, where $G(\cdot, \boldsymbol{\theta}_g)$ is a differentiable function

represented by a MLP with parameters $\boldsymbol{\theta}_g$ mapping the input into the data space, i.e. if $\boldsymbol{z} \in \mathbb{R}^d$ and our training data is $\mathcal{X} \subset \mathbb{R}^n$ then $G : \mathbb{R}^d \to \mathbb{R}^n$. In other words, G transforms samples $\boldsymbol{z}$ from $p_{\boldsymbol{z}}$ into generated samples $G(\boldsymbol{z})$ from p_g. Our objective is to learn this mapping G by learning the parameters $\boldsymbol{\theta}_g$. In addition, we define another MLP $D(\boldsymbol{x}, \boldsymbol{\theta}_d)$ which maps the input into a single scalar. This discriminator represents the probability that a given sample came from p_{data} rather than p_g. To train these two MLPs, we will pit then against each other in a minimax game. In particular,

$$\min_G \max_D V(D, G) = \min_G \max_D \left(\mathbb{E}_{\boldsymbol{x} \sim p_{data}}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}}[\log(1 - D(G(\boldsymbol{z})))]\right). \tag{38}$$

Note that training G to minimize $\log(1-D(G(\boldsymbol{z})))$ in practice may result in problems with vanishing gradient during early training. To sidestep this issue [1] train G instead to maximize $\log(D(G(\boldsymbol{z})))$. This is purely a heuristic solution to avoid issues during early training that does not change the dynamics of G and D [1]. Since we do not observe p_{data} we will approximate the expectations by sample averages. An overview of how to train this GAN is presented in algorithm 1. Note that we are alternating between optimizing D for k steps and optimizing G one step. This is done, instead of optimizing D to completion, to avoid over-fitting [1].

Algorithm 1: Stochastic gradient descent for training GAN. k, the number of steps to sequentially train D, is a hyperparameter. N is the number of training iterations and m the size of the minibatch. The stochastic gradient updates can use any standard gradient based approach. [1] use momentum. This is reproduced from algorithm 1 in [1].

for $1, 2, ..., N$ **do**
  for $1, 2, ..., k$ **do**
    Sample $\{\boldsymbol{z}^1, ..., \boldsymbol{z}^m\}$ from p_g
    Sample $\{\boldsymbol{x}^1, ..., \boldsymbol{x}^m\}$ from p_{data}
    Update the discriminator using (ascending)

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

  end
  Sample $\{\boldsymbol{z}^1, ..., \boldsymbol{z}^m\}$ from p_g
  Update the generator using (descending)

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

end

The main results from [1] are restated here for the convenience of the reader. Proposition 2.1. gives the optimal discriminator for any given generator. Theorem 2.2. shows that the global optimum of virtual training criterion is achieved for $p_g = p_{data}$. Finally, proposition 2.2. proves convergence of algorithm 1.

Proposition 2.1 ([1]). *The optimal discriminator D for any given G is*

$$D_G^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \tag{39}$$

Proof. See [1]. □

We can interpret the training objective within the framework of maximum likelihood estimation. In particular, the training objective for D can be viewed as trying to maximize the log-likelihood of the conditional probability $P(Y = y|\boldsymbol{x})$. Where $Y := \mathbb{I}(\boldsymbol{x}$ sampled from $p_{data})$ and $\mathbb{I}(\cdot)$ denote the indicator function. The minimax game in (40) can now be written as [1]:

$$\begin{aligned}
C(G) &= \max_D V(G, D) && (40)\\
&= \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[\log(1 - D_G^*(G(\boldsymbol{z})))] && (41)\\
&= \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D_G^*(\boldsymbol{x}))] && (42)\\
&= \mathbb{E}_{\boldsymbol{x}\sim p_{data}}\left[\log \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log\left(1 - \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right)\right] && (43)\\
&= \mathbb{E}_{\boldsymbol{x}\sim p_{data}}\left[\log \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log \frac{p_g(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] && (44)
\end{aligned}$$

where $C(G)$ is the virtual training criterion [1]. Before we continue, we define the Kullback-Leibler (KL) and Jensen-Shannon divergences. Let $p(\boldsymbol{x})$ and $q(\boldsymbol{x})$ be two probability density functions defined on $\mathbb{R}^n$, then the KL-divergence is

$$\mathrm{KL}(p \parallel q) := \int_{\mathbb{R}^n} p(\boldsymbol{x}) \log\left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right) d\boldsymbol{x}$$

and the Jensen-Shannon divergence is

$$\mathrm{JSD}(p \parallel q) := \frac{1}{2}\mathrm{KL}(p \parallel M) + \frac{1}{2}\mathrm{KL}(q \parallel M),$$

where $M := \frac{p+q}{2}$ [16]. Note that, unlike the KL divergence, the Jensen-Shannon divergence is symmetric.
We will need Jensen's inequality for the next lemma. Jensen's inequality states that for a convex function f

$$\mathbb{E}(f(x)) \geq f(\mathbb{E}(x)).$$

Moreover, if f is strictly convex then $\mathbb{E}(f(x)) = f(\mathbb{E}(x))$ implies $x = \mathbb{E}(x)$ with probability 1 (i.e. x is constant) [22].

Lemma 2.1.

$$\mathrm{KL}(p \parallel q) \geq 0 \text{ with equality if and only if } p = q.$$

Proof.

$$\begin{aligned}
\mathrm{KL}(p \parallel q) &= \int_{\mathbb{R}^n} \log\left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right) p(\boldsymbol{x}) d\boldsymbol{x} \\
&= \mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}\left(\log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right) \\
&= \mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}\left(-\log \frac{q(\boldsymbol{x})}{p(\boldsymbol{x})}\right) \\
&\geq_{(1)} -\log\left(\mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})} \frac{q(\boldsymbol{x})}{p(\boldsymbol{x})}\right) \\
&= -\log\left(\int_{\mathbb{R}^n} p(\boldsymbol{x}) \frac{q(\boldsymbol{x})}{p(\boldsymbol{x})} d\boldsymbol{x}\right) \\
&= -\log\left(\int_{\mathbb{R}^n} q(\boldsymbol{x}) d\boldsymbol{x}\right) \\
&= -\log 1 \\
&= 0.
\end{aligned}$$

where $\geq_{(1)}$ is due to Jensen's inequality and that $-\log(t)$ is a convex function. Since $-\log(t)$ is even strictly convex $\geq_{(1)}$ holds with equality if and only if $p(\boldsymbol{x})/q(\boldsymbol{x})$ is constant, i.e. $q(\boldsymbol{x}) = cp(\boldsymbol{x})\ \forall \boldsymbol{x}$. But

$$1 = \int_{\mathbb{R}^n} q(\boldsymbol{x}) d\boldsymbol{x} = c \int_{\mathbb{R}^n} p(\boldsymbol{x}) d\boldsymbol{x} = c \tag{45}$$

so $\geq_{(1)}$ holds with equality if and only if $p(\boldsymbol{x}) = q(\boldsymbol{x})\ \forall \boldsymbol{x}$, i.e. $p = q$. □

Corollary 2.1. $\mathrm{JSD}(p \parallel q) \geq 0$ *with equality if and only if* $p = q$.

Proof. Follows immediately from the definition of Jensen-Shannon divergence and lemma 2.1. □

Theorem 2.2 ([1])**.** *The global minimum of C(G) is obtained* $\iff$ $p_g = p_{data}$ *where it takes the value* $-\log(4)$.

Proof. Suppose $p_g = p_{data}$. Then by proposition 2.1. we have that $D^*_G(\boldsymbol{x}) = 1/2$. By equation (44) it follows that

$$\begin{aligned}
C(G) &= \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D^*_G(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D^*_G(\boldsymbol{x}))] \\
&= \mathbb{E}_{\boldsymbol{x}\sim p_{data}} \log \frac{1}{2} + \mathbb{E}_{\boldsymbol{x}\sim p_g} \log\left(1 - \frac{1}{2}\right) \\
&= \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[-\log 2] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[-\log 2] \\
&= -\log 2 - \log 2 = -\log 4.
\end{aligned}$$

We now wish to show that this is the minimum of $C(G)$. Adding $0 = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})} \log 2 + \mathbb{E}_{\boldsymbol{x}\sim p_g(\boldsymbol{x})} \log 2 - \log 4$

on the right-hand side of (44) yields

$$C(G) = -\log 4 + \mathbb{E}_{\boldsymbol{x}\sim p_{data}}\left[\log \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} + \log 2\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log \frac{p_g(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} + \log 2\right] \tag{46}$$

$$= -\log 4 + \mathbb{E}_{\boldsymbol{x}\sim p_{data}}\left[\log \frac{2p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\log\left[\frac{2p_g(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] \tag{47}$$

$$= -\log 4 + \int_{\mathbb{R}^n} p_{data}(\boldsymbol{x}) \log \frac{2p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} d\boldsymbol{x} + \int_{\mathbb{R}^n} p_g(\boldsymbol{x}) \log \frac{2p_g(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})} d\boldsymbol{x} \tag{48}$$

$$= -\log 4 + \int_{\mathbb{R}^n} p_{data}(\boldsymbol{x}) \log \frac{p_{data}(\boldsymbol{x})}{\frac{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}} d\boldsymbol{x} + \int_{\mathbb{R}^n} p_g(\boldsymbol{x}) \log \frac{p_g(\boldsymbol{x})}{\frac{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}{2}} d\boldsymbol{x} \tag{49}$$

$$= -\log 4 + \mathrm{KL}\left(p_{data} \,\middle\|\, \frac{p_{data} + p_g}{2}\right) + \mathrm{KL}\left(p_g \,\middle\|\, \frac{p_{data} + p_g}{2}\right) \tag{50}$$

$$= -\log 4 + \mathrm{JSD}\left(p_{data} \,\|\, p_g\right) + \mathrm{JSD}\left(p_g \,\|\, p_{data}\right) \tag{51}$$

$$=_{(1)} -\log 4 + 2\,\mathrm{JSD}\left(p_{data} \,\|\, p_g\right) \tag{52}$$

where $=_{(1)}$ follows since the Jensen-Shannon divergence is symmetric. That $-\log 4$ is the unique global minimum for $p_{data} = p_g$ follows from corollary 2.1. □

Proposition 2.2 ([1]). *Suppose that for each iteration in Algorithm 1 the discriminator achieves an optimum given G, that G and D have enough capacity, and p_g is updated s.t. the following criterion is improved*

$$\mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D^*_G(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D^*_G(\boldsymbol{x}))] \tag{53}$$

then p_g converges to p_{data}.

Proof. See [1]. □

There are some major differences between these theoretical results and how GANs work in practice. First of all, optimization is based on $\boldsymbol{\theta}_g$ instead of p_g since the D and G are MLPs. As these updates are made in the parameter space, the above results which all depend on convexity, are no longer valid [17]. Nevertheless, despite the lack of theoretical guarantees, GANs based on MLPs work very well in practice [1].

2.4 Relativistic Generative Adversarial Networks

2.4.1 Intuition

Since the introduction of the original GAN in 2014 by [1] an plethora of GANs have been developed. Most can be expressed in the following, fairly general, form [23]

$$L_D = \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\tilde{f}_1(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[\tilde{f}_2(D(G(\boldsymbol{z})))] \tag{54}$$

$$L_G = \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\tilde{g}_1(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[\tilde{g}_2(D(G(\boldsymbol{z})))] \tag{55}$$

where $\tilde{f}_1, \tilde{f}_2, \tilde{g}_1, \tilde{g}_2$ are all scalar to scalar functions. We can categorize GANs based on whether they use saturating or non-saturating loss functions [23]. For saturating we have that $\tilde{g}_1 = -\tilde{f}_1, \tilde{g}_2 = -\tilde{f}_2$ and for non-saturating $\tilde{g}_1 = \tilde{f}_2, \tilde{g}_2 = \tilde{f}_1$. The loss function in [1] is a saturating cross-entropy loss. In other words, $\tilde{f}_1 = -\log(D(\boldsymbol{x}))$ and $\tilde{f}_2 = -\log(1 - D(\boldsymbol{x}))$, where $D(\boldsymbol{x}) := \sigma(C(\boldsymbol{x}))$ and $\sigma(\cdot)$ is the logistic sigmoid function. Here $C(\boldsymbol{x})$ is the untransformed discriminator output, called the critic [23]. Writing this all out we get,

$$L_D = \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[-\log(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[-\log(1 - D(G(\boldsymbol{z})))] \tag{56}$$

$$L_G = \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[\log(1 - D(G(\boldsymbol{z})))] \tag{57}$$

which is the objectives we get in [1] but written as two loss functions to be minimized. As we showed above, this objective is approximately the same as minimizing the Jensen-Shannon divergence between p_{data} and p_g. This objective is very intuitive as minimizing this divergence to optimality would imply that the distribution of our generator p_g is a very good approximation of the data generating distribution p_{data}. However, [23] notes that the dynamics of training a standard GAN [1] is very different from that of minimizing the Jensen-Shannon divergence. The Jensen-Shannon divergence is minimized when p_{data} is indistinguishable from p_g and the discriminator cannot do better than a random guess guessing, i.e. $D(\boldsymbol{x}) = D(G(\boldsymbol{z})) = 1/2$. The Jensen-Shannon divergence is as large as it can be when the p_{data} and p_g are perfectly distinguishable and $D(\boldsymbol{x}) = 1, D(G(\boldsymbol{z})) = 0$. In other words, the discriminator is never mistaken. Hence, we expect the dynamics of minimizing Jensen-Shannon divergence should similar to the following. $D(\boldsymbol{x})$ decreases smoothly from 1 to 1/2 and that $D(G(\boldsymbol{z}))$ increases smoothly from 0 to 1/2. Instead when training the standard GAN, $D(\boldsymbol{x})$ is kept constant while $D(G(\boldsymbol{z}))$ is increasing. The main argument in [23] is that $D(\boldsymbol{x})$ should also decrease while $D(G(\boldsymbol{z}))$ increases. This can be achieved by making the discriminator relativistic, i.e. depending on both on p_{data} and p_g.

2.4.2 Relativistic GAN

A simple way of making the discriminator relativistic is simply to change D such that it is a function of $\tilde{\boldsymbol{x}} = (\boldsymbol{x}, G(\boldsymbol{z}))$, for instance $\tilde{D}(\tilde{\boldsymbol{x}}) = \sigma(C(\boldsymbol{x}) - C(G(\boldsymbol{z})))$. We can rewrite the loss function (56) and (57) with relativistic discriminators to get the loss functions of Relativistic GAN (RGAN) [23]

$$L_D^{RGAN} = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{z})\sim(p_{data},p_{\boldsymbol{z}})}[\tilde{f}_1(C(\boldsymbol{x}) - C(G(\boldsymbol{z})))] + \mathbb{E}_{(\boldsymbol{x},\boldsymbol{z})\sim(p_{data},p_{\boldsymbol{z}})}[\tilde{f}_2(C(G(\boldsymbol{z})) - C(\boldsymbol{x}))] \tag{58}$$

$$L_G^{RGAN} = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{z})\sim(p_{data},p_{\boldsymbol{z}})}[\tilde{g}_1(C(\boldsymbol{x}) - C(G(\boldsymbol{z})))] + \mathbb{E}_{(\boldsymbol{x},\boldsymbol{z})\sim(p_{data},p_{\boldsymbol{z}})}[\tilde{g}_2(C(G(\boldsymbol{z})) - C(\boldsymbol{x}))] \tag{59}$$

where $\tilde{f}_1, \tilde{f}_2, \tilde{g}_1, \tilde{g}_2$ are as above. The interpretation of the discriminator is very different in RGAN compared to GAN. In RGAN, the discriminator is predicting the probability that the given data is more "realistic" than randomly sampled data from p_g.

2.4.3 Relativistic average GAN

Due to the large difference in interpretation of the discriminator in GAN and RGAN, [23] also develops a middle-ground called the Relativistic average GAN (RaGAN). The discriminator in RaGAN is predicting the probability that some given data is more "realistic" on average than data from p_g. More formally, the loss functions for RaGAN are formulated as

$$L_D^{RaGAN} = \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\tilde{f}_1\left(C(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}C(G(\boldsymbol{z}))\right)] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[\tilde{f}_2\left(C(G(\boldsymbol{z})) - \mathbb{E}_{\boldsymbol{x}\sim p_{data}}C(\boldsymbol{x})\right)] \tag{60}$$

$$L_G^{RaGAN} = \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\tilde{g}_1\left(C(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}C(G(\boldsymbol{z}))\right)] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[\tilde{g}_2\left(C(G(\boldsymbol{z})) - \mathbb{E}_{\boldsymbol{x}\sim p_{data}}C(\boldsymbol{x})\right)] \tag{61}$$

where $\tilde{f}_1, \tilde{f}_2, \tilde{g}_1, \tilde{g}_2$ are as above. The RGAN and RaGAN can be trained using stochastic gradient descent in a similar manner as in algorithm 1. For more details see the appendix of [23]. Moreover, [23] recommends using non-saturating loss functions for RGAN and RaGANs.

2.5 Super-resolution

2.5.1 Introduction

Many classical image processing tasks are framed as image transformation task where the system takes an input image and transform it into an output image [24]. One common example of an image transformation task is super-resolution. Super-resolution involves estimating a high resolution image from it's low resolution

counterpart [2]. This is a fundamentally ill-posed problem since there are a myriad of possible high resolution images that could be associated with the low resolution input. GANs are useful in this respect as the super-resolved images require more information to be filled in. A generator can be used to "paint" more details into the image in a convincing way. The seminal paper [2] introduced the Super-resolution Generative Adversarial Network (SRGAN), a method for single image super-resolution based on GANs. The main objective is to train a generative function G that can produce high quality up-scaled images from the low resolution input images. [3] argues that despite of the overall success of SRGAN, the hallucinated details occasionally add undesirable artifacts. By revisiting and changing some key components of SRGAN, [3] introduces Enhanced Super-resolution Generative Adversarial Network (ESRGAN). The change that is of main interest for the discussion in this paper is that, in contrast to SRGAN which uses the original discriminator from [1], ESRGAN makes the discriminator relativistic as in [23].

2.5.2 ESRGAN

ESRGAN uses RaGAN [23] and the discriminator and adversarial loss are then symmetrically defined as

$$L_D^{RaGAN} = -\mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log\sigma\left(C(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}C(G(\boldsymbol{z}))\right)] - \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[\log(1-\sigma\left(C(G(\boldsymbol{z})) - \mathbb{E}_{\boldsymbol{x}\sim p_{data}}C(\boldsymbol{x})\right))] \quad (62)$$

$$L_G^{RaGAN} = -\mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log(1-\sigma\left(C(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}C(G(\boldsymbol{z}))\right))] - \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[\log\sigma\left(C(G(\boldsymbol{z})) - \mathbb{E}_{\boldsymbol{x}\sim p_{data}}C(\boldsymbol{x})\right)] \quad (63)$$

where $\sigma(\cdot)$ is the logistic sigmoid function. The data will consist of high and low resolution versions of the same images. In (62) and (63) $\boldsymbol{z}$ would represent the low resolution images and $\boldsymbol{x}$ the high resolution counterparts. In ESRGAN L_G^{RaGAN} is only a part of the total loss for the generator. Previous work in super-resolution is usually based on minimizing the mean squared error (MSE) of pixel per pixel differences between low and high resolution images. However, as argued in [24], this fails to capture high level perceptual differences. These higher level perceptual difference is what is going to matter to the human eye. To rectify this, SRGAN [2] follows [24] and [25] in a loss function that considers perceptually relevant characteristics. Following the work in SRGAN, ESRGAN also uses a perceptual loss function, L_{percep}. However, in contrast to convention, [3] suggest to use features before, rather than after, the activation layers. First, activated features are very sparse and thus provides weak supervision. Second, [3] empirically observe that using features after activation leads to inconsistent reconstructed brightness. The total generator loss is in ESRGAN defined as

$$L_G = L_{percep} + \lambda L_G^{RaGAN} + \eta L_1 \quad (64)$$

where $L_1 = \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}\|G(\boldsymbol{z}) - \boldsymbol{x}\|_1$ is the ℓ_1-norm distance between reconstructed $G(\boldsymbol{z})$ and the ground-truth image $\boldsymbol{x}$. λ and η are coefficients used to balance between the different terms. In addition to using a relativistic discriminator, ESRGAN also make some changes to the network architecture compared to SRGAN. We omit details on this for the sake of brevity. An example of what ESRGAN can do is given in figure 4.

2.6 You Only Look Once

2.6.1 Introduction

You only look once (YOLO) refers to a series of state of the art object detection models originally developed by [28]. Subsequent improvements to the models where made in YOLO9000 [29] and YOLOv3 [30]. YOLOv3 was the final model created by the original authors. However, other researchers continued the development of YOLO. A widely recognized update to the v3 model is YOLOv4 [31]. Consistent with previous versions of YOLO, the v4 still uses Joseph Redmon's open source neural network framework Darknet[9]. This paper uses the latest iteration of the YOLO models, YOLOv5 [32] which, instead of Darknet, is implemented using PyTorch.[10] This version has not been widely accepted within the computer vision community and

[9] See https://pjreddie.com/darknet/.
[10] See https://pytorch.org/.

(a) Original

(b) Super-resolved

Figure 4: This figure illustrates an example output from ESRGAN [3] using an image from the div2K dataset [26] [27]. (a) shows the original image and (b) shows the super-resolved image.

is surrounded with a lot of controversy (e.g., should it really be called YOLO?). Another issue with the YOLOv5 model is that it does not have any corresponding paper—all information is given from its Github page and soure code within. Nevertheless, after trying out v3, v4, and v5, the author found that the v5 greatly outperforms the other versions in terms of training speed. Since the actual version of YOLO is of negligible importance in the project, the practical benefits of using YOLOv5 exceeds the cost. Moreover, since the particular mechanics of the object detection model is of tertiary importance to this project we will only give very brief overview of the first 3 iterations of YOLO just to give an introduction to the YOLO class of models. The interested reader is referred [28] [29] [30] [31] [32].

2.6.2 YOLO

You Only Look Once (YOLO) was introduced by [28]. Unlike previous detection models, which re-purposed classifiers to perform detection, YOLO treats object detection as one single regression problem using a single convolutional neural network which simultaneously predicts bounding boxes and class probabilities. In contrast to previous object detection models, this allows the model to predict where and what is present in the images while only looking at the images once—hence the name. Due to its simplicity, YOLO is also much faster than many of its predecessors taking a significant step towards real time inference. The following brief overview of the most important aspects of the YOLO models is partially based on [33]. Imagine that we split the images in to $S \times S$ grids. YOLO predicts B boundary boxes and C classes. The loss function consists of three different parts. The confidence loss, if an object is detected in the box, is

$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{i,j}^{obj} \left(C_i - \hat{C}_i \right)^2 \tag{65}$$

where $\hat{C}_i$ is the box confidence score in cell i and $\mathbb{I}_{i,j}^{obj} = 1$ if the j-th box in cell i is responsible for detecting the object. Otherwise $\mathbb{I}_{i,j}^{obj} = 0$. If no object is detected within the box, then the confidence loss is

$$\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{i,j}^{noobj} \left(C_i - \hat{C}_i \right)^2 \tag{66}$$

where $\mathbb{I}_{i,j}^{noobj}$ is the complement of $\mathbb{I}_{i,j}^{obj}$ and λ_{noobj} is a weight applied to balance out the fact that most boxes do not contain an object. It's default value in [28] is $\lambda_{noobj} = 0.5$. Then we have the localization loss

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{i,j}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{i,j}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \tag{67}$$

where λ_{coord} fills a similar function as λ_{noobj}, with default value $\lambda_{coord} = 5$. x, y, w, h are the x, y cooridantes of the center of the bounding box and its width and height. $\hat{x}, \hat{y}, \hat{w}, \hat{h}$ are the estimated counterparts. Finally we have the classification loss

$$\sum_{i=0}^{S^2} \mathbb{I}_{i}^{obj} \sum_{c \in classes} \left(p_i(c) - \hat{p}_i(c) \right)^2 \tag{68}$$

where $\mathbb{I}_{i}^{obj} = 1$ if there is an object in cell i, other wise its equal to 0 and $\hat{p}_i(c)$ denotes the conditional class probability for class c in cell i. Note that the classification loss is always zero for a single class model.

2.6.3 YOLO9000

The 9000 suffix refers to the fact that this version extends YOLO to more than 9000 classes [29]. In addition, they add batch normalization [34]. The big new thing with YOLO9000, however, is the use of anchor boxes. This somewhat difficult to grasp idea, introduced in [35], has played an important part in object detection models as it helps stabilize early training. The main idea is that most objects that we are interested in detecting have similar shapes within that class. In other words, most humans will have a certain aspect ratio and thus fit into a similarly shaped bounding box. One can then introduce, for instance, 5 different bounding boxes—called anchors or priors in [29]. Instead of directly predicting coordinates, [35] predicts the bounding boxes using these user-defined anchors. The particular priors used are important for the performance of the model. To find good priors, [29] uses k-mean clustering on the training dataset bounding boxes. $k = 5$ has been found good in practice [29]. Instead of offsets to prior bounding boxes as in [35], [29] uses an approach they call direct location prediction. The network predicts the following 4 coordinates for each bounding box:

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \end{aligned}$$

where t_x, t_y, t_w, t_h are the untransformed output from YOLO, $\sigma(\cdot)$ the logistic sigmoid function, (c_x, c_y) the top left corner of prior bounding box, p_w, p_h the width and height of prior bounding box, and b_x, b_y, b_w, b_h is the predicted boundary box. Another important addition in YOLO9000 is the use of hierarchical classifications.

2.6.4 YOLOv3

The paper introducing YOLOv3 is called "YOLOv3: An Incremental Improvement", indicating that a lot remains the same as in YOLO9000. Nevertheless, there are a couple of changes. Perhaps the most significant one is the use of a much deeper feature extractor. YOLO9000 uses Darknet-19 which has 19 convolutional layers. YOLOv3 uses a significantly deeper feature extractor called Darknet-53. As the name suggests, Darknet-53 contains 53 convolutional layers [30]. As mentioned above, deeper models tend to perform better. Darknet-53 is using skip-connection from [7] and [8] to get all the benefits of extra depth.

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

Table 1: Darknet-53. This is the feature extractor utilized in YOLOv3. The numbers next to be block on the left hand side indcates that that particular stack of layers in repeated that many times. The size is the spatial extent of the convolution and /2 indicates that a stride of 2 instead of 1 is used. The output part shows how the size of the image changes throughout the network. From 256×256 to 8×8. This table is reproduced from [30].

2.7 Performance measures for computer vision

The discussion in this section is largely based on [36].

2.7.1 Precision and Recall

The two most straightforward metrics are precision and recall. Precision P is measuring the fraction of correct guesses and recall R is measuring the fraction of guesses where there is something to guess. More formally

$$P = \frac{TP}{TP + FP} \tag{69}$$

$$R = \frac{TP}{TP + FN} \tag{70}$$

where TP, FN, FP are true positive, false negative, and false positive, respectively. If we plot a model's precision and recall as a function of the model's confidence threshold we get the precision recall curve. A

sketch of a typical precision-recall curve can be seen in figure 5. It is downward sloping as when the confidence threshold is reduce there will be more incorrect guesses.

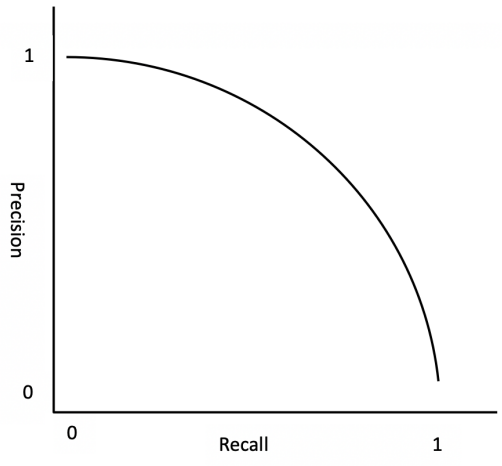


Figure 5: Precision-recall curve. Reproduced from [36].

2.7.2 Average Precision and Mean Average Precision

The next metrics is also using the precision-recall curve. Imagine that we compute the precision for the different confidence thresholds marked by the orange bars in figure 6. Taking the average of these yields the average precision (AP) metrics. The model will make an prediction in terms of a bounding box enclosing the object. To what extent must this predicted bounding box overlap with the ground-truth (user provided) bounding box?

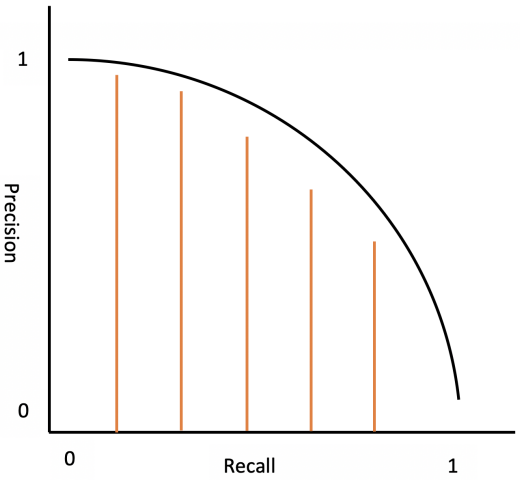


Figure 6: Average precision (aP). Reproduced from [36].

There are many possible answers to this question. We formalize the notion of this overlap with the intersection over union (IoU) metrics. Figure 7 gives an visualization of this. It is common to average the average

precision over several different IoU thresholds. This yields the most commonly used performance metrics in object-detection models: the mean average precision (mAP). We follow the COCO challenge [37] by using the IoU thresholds $0.5, 0.55, ..., 0.95$. We denote this metrics mAP0.5:0.95. We will also denote the mAP at IoU threhold 0.5 as mAP0.5.

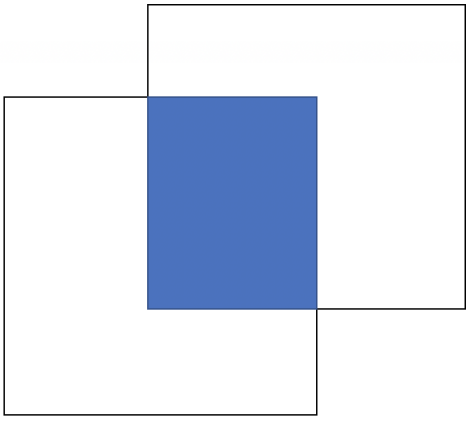


Figure 7: Intersection over union (IoU). Reproduce from [36].

The final metrics that will be used in this paper is called fitness. To the best of the author's knowledge this is defined by the Ultralytics team that is behind the YOLOv5 model. This metric is actually what defines the "best" model when training.[11] It is simply defined as a weighted average between mAP0.5 and mAP0.5:0.95,

$$\text{fitness} = 0.1 \cdot \text{mAP0.5} + 0.9 \cdot \text{mAP0.5:0.95}.$$

This particular weighting is the same as used in the original implementation of the YOLOv5 model[12].

3 Implementation

3.1 Overview

A visual overview of the suggested approach in the paper can be found in figure 1. To evaluate the approach we simulate the situation of only having low quality input by down-sampling the original images. This enables us to evaluate the merits of this approach as it is possible to compare the performance of the model trained on down-sampled and subsequently super-resolved images with the performance of the model trained on the original images. Hence, this will require to train the object detection model on two separate datasets. This is done by the following steps:

(i) Down-sample original images using Matlab's imresize (4× downsampling).

(ii) Super-resolve the down-sampled images using ESRGAN. This was achieved by using the ISR package [38]. This yields 4× up-sampling and thus the dimension of the super-resolved images is the same as the original images.

[11] An interesting observation during training is that the model with the best fitness is not necessarily the one with the lowest loss.

[12] For more information, see `https://github.com/ultralytics/yolov5`.

(iii) Annotate the images. This was done using LabelImg [39].

(iv) Train YOLOv5. To speed up this process Google Colab was used.

Steps (iii)-(iv) were repeated for the original images as well. Once this is completed we have two sets of trained weights. We can then compare the performance of the two models. The performance was evaluated both qualitatively (by visually observing the test output) and quantitatively (by comparing loss, precision, recall, mAP, and fitness). All the code used in the project has been added to a GitHub repository.[13]

3.2 Ethical considerations

Using computer vision to detect violations of social distancing regulations is great in the sense that it might help us curb the spread of viruses such as COVID-19. Nevertheless, this sort of surveillance presents important ethical consideration due to privacy concerns. This project is doing several things to allow for detection of social distancing violations in an ethical way. First of all, the focus is on tables. We specifically chose to track tables (where persons later may sit) than individuals. This allows us to see if proper precautions has been made at restaurants so that when they later fill up this is done in a way that do not violate social distancing measures. Hence, we do not specifically track individuals. Secondly, the imagery presented in this paper either do not contain people in a way that they can be identified or has been down-sampled to the extent that individuals are completely unidentifiable. We believe that these two precautions allow us to carry out this project is an ethical manner with individuals' privacy in mind.

3.3 Data

The data used in this paper is drone footage of the outdoor seating area of several restaurants in Lund. Due to privacy concerns, these data cannot be made public. As mentioned in previous section, the author has taken actions to ensure that no individual can be identified in the images shown in this project. An overview of the data can be found in table 2. As can be seen in this table, the videos were split into training and test sets. In particular, the first x seconds (indicated in the training column) were used for the training dataset and the remaining video was used for test data. The variation in where the video was cut was based subjectively on what "information" was contained in the video. For instance, if there is nothing changing in the video for a longer amount of time then this represents essentially the same image repeated over and over and thus does not add any variation to the dataset. The video was shot in about 30 frames per second. However, following the same train of thought as above, using each of these frames would prove extremely redundant. Using all 30 frames per second adds little to no extra value but at the same time comes at the cost of additional time when annotating the images. As a trade-off, 3 frames per second were used.[14] This resulted in a training dataset of 104 images. We should comment directly on the size of this dataset. First of all, we do by no means aim to train a models that is very general and is able to detect tables of any variation of size, shape and color. This project is mainly intended as a proof of concept and thus a very small dataset is used due to resource limitations.

[13] For now this has been kept private due to public and private interest in this project.

[14] For all video and image processing we used ffmpeg. See `https://ffmpeg.org/`.

Overview of data			
Video	Resolution	Total length	Training length
3308.mp4	(1280, 720)	00:00:11	00:00:06
3628.mp4	(1280, 720)	00:00:17	00:00:06
3715.mp4	(1280, 720)	00:00:23	00:00:14
0708.mp4	(1280, 720)	00:00:15	00:00:09

Table 2: This table gives a concise overview of the data used in this paper. The first column just serves as an identifier. The second columns gives the resolution. The third column states the total length of the video in the format hours:minutes:seconds. The final columns describes how the video has been split to create training and testing datasets.

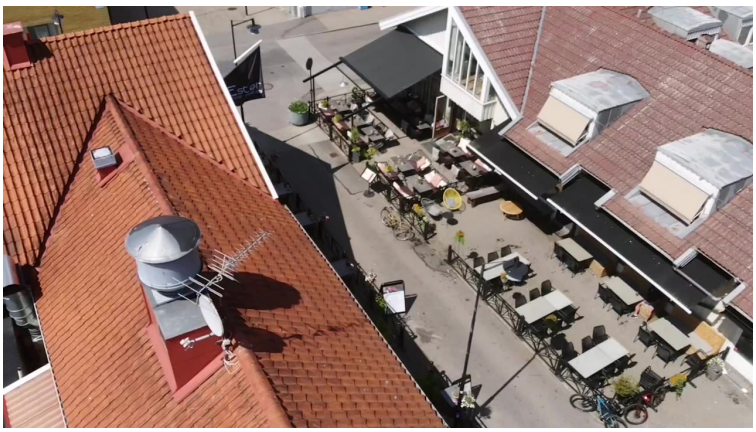
3.4 Preprocessing

The first step simply involves taking each image and $4\times$ down-sample it using Matlab's imresize command.[15] This converts $(1280, 720)$ images to $(320, 180)$. These images correspond to the "low quality dataset." We then proceed to the second step: super-resolution. As GANs may be unstable to train [17] [1], we opted in this project to used pre-trained weights only. In particular, we used the pre-trained weights from the ESRGAN [3] implemented in the ISR package [38]. The super-resolution performs 4x up-sampling and thus the down-sampled and subsequently super-resolved images are again $(1280, 720)$. An example of a training data images is shown in figure 8.[16] There is an element of subjectivity in whether or not it is easier to identify, and thus annotate, objects in the super-resolved images compared to the down-sampled ones. In the author's experience, the dataset contains examples where super-resolution helps and where it does not. That images can be properly annotated is crucial for the success of the approach suggested in this paper. The third step is the most time consuming and prone to error as it requires human interaction. The annotation was done using LabelImg [39]. This Python based API[17] allows the user to go through the dataset image by image and mark bounding boxes by using the cursor. Annotating the images will result in two files per image: a .jpg and a .txt file. An example .txt file is shown in figure 9. The first column is the class, since this is a single clas model there are all equal to zero. The following four columns give the location of the bounding box. The YOLO format is x, y, w, h where x, y is the center of the bounding box and w and h the width and height respectively. Note that these are normalized values, relative to the width and height of the input image. One significant concern is human error when annotating the images. For the super-resolved images, and in some cases even the original images, it is difficult even for an human eye to detect an object. This could be due to partial occlusion, shade, or simply that the object is too small to be correctly identified. Working with down-sampled and subsequently super-resolved images greatly amplified this issue. During the annotating process there were several examples when the author knew where all the tables are located (due to having worked with original images) but it was not possible to see them in the super-resolved image. To keep this proof of concept as valid as possible, these tables were not annotated. Hence, during training the model will be penalized for, correctly, predicting the existence of tables at these locations.

3.5 Training

The fourth step mentioned above is training the model. It is uncommon and inefficient to train a model from scratch in computer vision. Instead, one commonly initialize the model using pre-trained weights. This is referred to as transfer learning. In this project we use the pre-trained YOLOv5 weights from [32] which

[15] See `https://se.mathworks.com/help/images/ref/imresize.html` for more information.
[16] To reduce subjectivity a random image was chosen as a showcase in this example.
[17] Application Programming Interface.



(a)

(b)

(c)

Figure 8: This figure shows an example images from the training data. (a) Original image. (b) 4× downsampled image. (c) Image in (b) super-resolved. Large scale versions of these images can be found in appendix A.

0	0.448437	0.676389	0.042188	0.050000
0	0.485938	0.628472	0.037500	0.043056
0	0.510156	0.592361	0.035937	0.037500

Figure 9: Example of content in .txt file associated with each annotated image. The first column is the class identifier. The second two columns are the x and y coordinates indicating the center of the bounding box. The last two columns are the width and height of the bounding box. Note that all of these values have been been normalized.

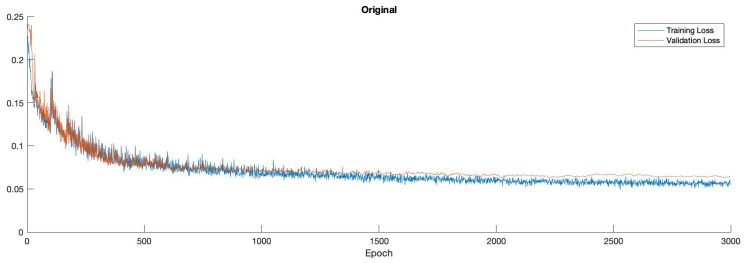
are trained on the MS COCO dataset [37]. For optimization we used ADAM [15]. As fine tuning the object detection algorithm is tertiary in this paper, we did not bother tuning the hyperparameters. Instead, the hyperparameters that were used are the same as in [32]. Training ConvNets is a very computation intensive. To make training feasible Google Colab was used. An overview of the training results is given in figure 10. "Super-resolved" refers to the images that have first been down-sampled and subsequently super-resolved.

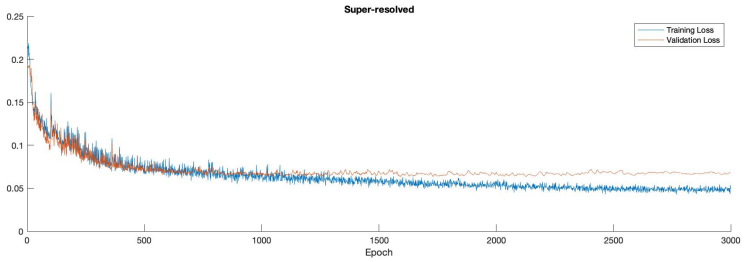
3.6 Social distancing

To extend the utility of the work in this project we want more than just predictions of the location of tables. Due to COVID-19 a lot of social distancing regulations and guidelines has been set in place along with other measures aimed to curb the spread of the virus. Social distancing involved maintaining a certain distance to individuals from outside your immediate social circle. Hence, we would like to detect the presence of tables as well as having a warning flag if they are placed too close to each other. To make a prediction using the Python program accompanied with this paper the user is asked to provide the image that inference should be run on as well as which weights should be used (i.e. for super-resolved or original images). The user will then be asked to define 6 points in the image by simply clicking on these locations. The first four specify the region of interest. This region will then be mapped into a bird's eye view. The last two points will specify the desired distance threshold. An example of this is shown in figure 11. If the center of the tables, from a bird's eye view, are closer than the distance threshold then they will be marked by a red dot at their center and be labelled "high risk."

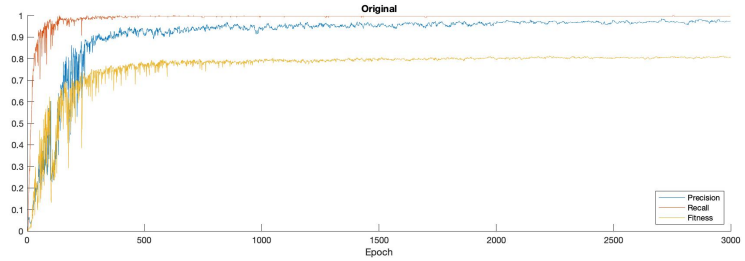
3.7 Results

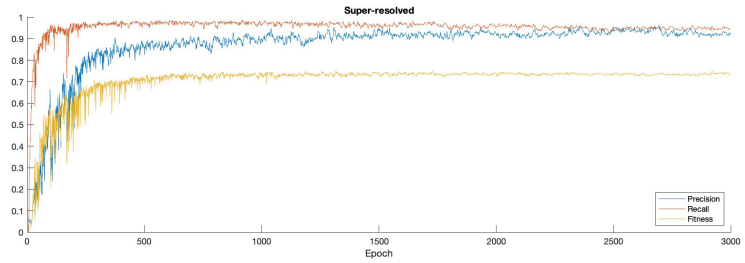
The results are evaluated both quantitatively and qualitatively. It is nice working with images as the result is usually very clear by just observing the output. Nevertheless, this type of qualitative assessment is highly subjective and conclusions can thus differ across individuals. Figures 12 and 13 shows the result from some test images. It is quite clear from these two examples that the approach taken in this project works in only certain scenarios. In figure 12 the super-resolved model is under performing and in figure 13 there is essentially no difference. As mentioned above, in certain super-resolved images it was extremely challenging to annotate. In particular cases this lead to tables not being annotated as tables and subsequently the algorithm is penalized during training for correctly predicting tables at these locations. On the other hand, when objects in the super-resolved images are properly identifiable then then performance is at least as good as that of the model trained on the original images. This gives the key conclusion of this paper. The suggested approach will only work when the objects of interest are properly identifiable to the human eye in the super-resolved images. Our quantitative assessment is based on the performance metrics defined above. These metrics are created so that one can objectively compare the performance of two models on the same dataset. Since the two datasets are not the same we should take the P, R, mAP and fitness metrics with a grain of salt. We can see in table 3 that these measures are very similar for the original and super-resolved models. Fitness is higher for the original model, however the difference is very small. The loss functions





(a)





(b)

Figure 10: (a) Shows training loss and validation loss for the model trained on the original data and the model trained on the down-sampled then super-resolved data. (b) Performance metrics for the two different networks. Note that fitness in a weighted average of mAP0.5 and mAP0.5:0.95.

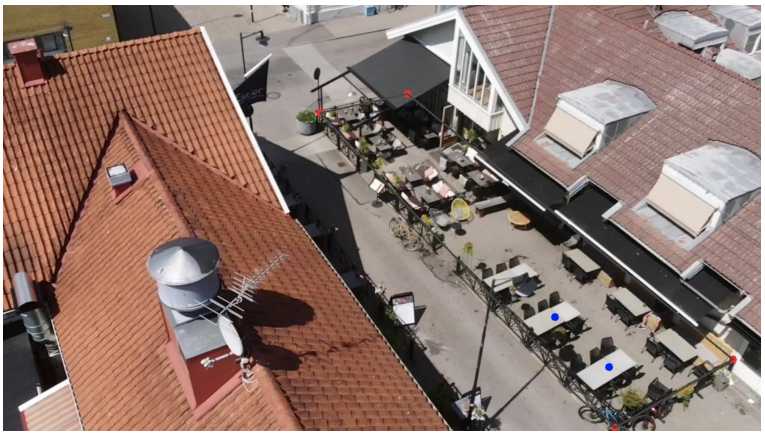


Figure 11: The red dots define the region of interest. This region is also drawn by connecting these red dots with a grey line. The distance between the two blue dots define the distance threshold. Large scale version of this images is available in the appendix A.

are comparable (since the model is exactly the same) but we should again acknowledge that the datasets are different. Interestingly, the model trained on the super-resolved images achieves both lower training loss and validation loss. Nevertheless, the difference is again very small and might only be a statistical fluke. It is also worth noting that the two datasets have been annotated differently. Hence, from the quantitative assessment we conclude that it does not seem to be a significant difference in performance across the models.

Data	Training Loss	Validation Loss	P	R	mAP0.5	mAP0.5:0.95	Fitness
Original	0.0597	0.0632	0.9737	0.9960	0.9924	0.7923	0.8123
Super-resolved	0.0547	0.0483	0.9202	0.9761	0.9819	0.7255	0.7511

Table 3: This table provide a summary of the quantitative assessment of the two models. The first two columns show the training loss and validation loss of the model trained on the two different datasets. Since the models are exactly the same these numbers are comparable. We note that both the training and validation loss of the model trained on down-sampled and the super-resolved images is actually lower than that of the network trained on the original images. Nevertheless, this difference is very small. The remaining five columns give various performance measures that are commonly used to compare the performance of object detection models: Precision, Recall, mean average precision at Iou 0.5, mean average precision over IoU 0.5,...,0.95 and fitness which is a weighted average of the later two. It is this final measure, fitness, that defines the "best" model. These particular results are those associated with the epoch that achieved the best fitness for each dataset. According to these measures the model trained on original images is performing a little bit better, but again the difference is very small.

4 Conclusion and limitations

There are some important limitations regarding the results in this paper. Here we discuss briefly the most important ones and how they give natural extensions to the current paper. First, the simulated situation in this project is likely to be easier than a real situation when one would be working with low resolution images to start with. This is due to the fact that our simulated data is directly generated from the higher resolution counterparts using down-sampling. Second, the quantitative comparison is not very rigours. Since

(a) Original (b) Super-resolved

Figure 12: This figure illustrates output of the two models for the same frame. (a) Prediction of the model trained on the original images. (b) Prediction of model train on down-sampled then super-resolved images. The green box indicates the prediction of a table. Tables with a red dot in their center and the text "high risk" are at a distance closer than the user defined threshold. This particular image was selected as it nicely shows the limitations of the suggested approached tested in this paper. Tables in the center of the image are simply too blurry to be identified by the human eye. Consequently, they have not be annotated as tables and thus the model get penalized during training for predicting tables at these locations. These images have been cropped due to privacy concerns. Large scale versions of these images are available in appendix A.

we do not know the distribution of the difference in loss functions, we cannot claim that there is statistically insignificant difference. Instead, we just have to consider whether or not the difference is large enough that it is reasonably not due to chance—hence introducing a degree of subjectivity. In addition, due to variation in the datasets our quantitative comparison must be taken with a grain of salt. Third, there is a degree of subjectivity and inevitable human error evolved in the annotation part. It is possible that two different individuals have a different opinion on whether or not there is a table in a particular location. In addition, since the author annotated both the original and super-resolved images extra bias might have been introduced. Fourth, the models trained in this paper are trained on extremely small datasets. A particular issue here is that the training data and validation data comes from the same videos. Thus, monitoring the validation error as a way to avoid over-fitting is not likely to be reliable. That being said, we never aimed to train a very general model and this paper was only meant as a proof of concept. It would be an interesting extension of this paper to launch a larger scale test. Fifth, it is difficult to be exact with the user defined distance in the social distancing program. A good next step would be to improve how this distance is being defined.

The main application of interest in this project is to be able to use satellite images as data for object detection algorithms where the objects of interest are too small to be identified. This is likely the case with tables, even if you have access to satellite images with resolution of 10-20 cm per pixel[18]. An obvious extension of this project would be to try out this approach on satellite imagery. The results in this paper works as a proof of concept indicating that this would be worth while. Nevertheless, in the end we cannot know until we try. Implementing this approach on actual satellite data would address the first caveat

[18]It should be noted that getting satellite data as a private individual with resolution higher than 30 cm per pixel is difficult due to of legal regulations.

(a)

(b)

Figure 13: This figure illustrates output of the two models for the same frame. (a) Prediction of the model trained on the original images. (b) Prediction of model train on down-sampled then super-resolved images. This is one of many examples where the two models perform qualitatively the same. Note that almost all tables are clearly identifiable for the human eye even in (b). Large scale versions of these images are available in appendix A.

mentioned above.

In this paper we have assessed a tentative solution to the problem of needing an object detection model while only having low resolution data available. To simulate the above scenario we created a low resolution dataset by down-sampling the original images. These lower resolution images were subsequently super-resolved. We trained an object detection model on each of these two datasets: the original images and the down-sampled and then super-resolved images. To assess the merits of this tentative solution we compared these two models qualitatively and quantitatively. The quantitative comparison consist of comparing the loss functions of the two models as well as some other commonly used performance metrics in computer vision. The main conclusion from this comparison was that there seems to be no significant difference in terms of these metrics. However, as noted above, due to variations in the two datasets the loss functions, precision, recall, and mAP are not strictly comparable. Moreover, the qualitative comparison consisted of visually comparing output from the two models. Within the test data there are examples where the two models perform essentially the same but there are also cases where the model based on the original images clearly performs better. These particular examples gave some guidelines on important limitation of the approach tested in the paper. In order to train an object detection model one needs to feed the model annotated images. This annotation step involves a human marking object in the training data with bounding boxes. There were several examples in the training dataset when the contrast between the objects of interest, tables, and the background was simply not clear enough for super-resolution to yield images where these object are identifiable for the human eye. Hence, they were subsequently not annotated as objects and during training the model would be penalizing for, correctly, predicting the existence of tables at these locations. This issue is one of the main limitations of tentative solution to the problem presented in this paper. Using super-resolution to generate higher resolution data that can be used for an object detection model is perfectly viable as long as the super-resolved images are such that they can be properly annotated.

5 References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[2] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew P. Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802, 2016.

[3] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Chen Change Loy, Yu Qiao, and Xiaoou Tang. Esrgan: Enhanced super-resolution generative adversarial networks, 2018. https://arxiv.org/abs/1809.00219.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[5] Ian Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. In *ICLR2014*, 2014.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 630–645, Cham, 2016. Springer International Publishing.

[9] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[10] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[11] Jianxin Wu. Introduction to convolutional neural networks, 2017. https://cs.nju.edu.cn/wujx/teaching/15_CNN.pdf.

[12] B.T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[13] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12(null):2121–2159, July 2011.

[14] G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. https://arxiv.org/abs/1412.6980.

[16] Yang Wang. A mathematical introduction to generative adversarial nets (gan), 2020. https://arxiv.org/abs/2009.00169.

[17] Ian J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017.

[18] Diederik P. Kingma. Fast gradient-based inference with continuous latent variable models in auxiliary form. *CoRR*, abs/1306.0733, 2013.

[19] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1278–1286, Bejing, China, 22–24 Jun 2014. PMLR.

[20] G. E. Hinton and T. Sejnowski. Learning and relearning in boltzmann machines. In *Parallel distributed processing: Explorations in the microstructure of cognition*, pages 282–317–. MIT Press, Cambridge, MA, 1986.

[21] Geoffrey E. Hinton, Terrence J. Sejnowski, and David H. Ackley. Boltzmann machines: Constraint satisfaction networks that learn. Technical Report CMU-CS-84-119, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1984.

[22] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, USA, 2006.

[23] Alexia Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard gan. 2018. https://arxiv.org/abs/1807.00734.

[24] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. *Computer Vision – ECCV 2016*, pages 694–711, 2016.

[25] Joan Bruna, Pablo Sprechmann, and Yann LeCun. Super-resolution with deep convolutional sufficient statistics. January 2016. 4th International Conference on Learning Representations, ICLR 2016 ; Conference date: 02-05-2016 Through 04-05-2016.

[26] Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[27] Radu Timofte, Eirikur Agustsson, Luc Van Gool, Ming-Hsuan Yang, Lei Zhang, Bee Lim, et al. Ntire 2017 challenge on single image super-resolution: Methods and results. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[28] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[29] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, 2017.

[30] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018. https://arxiv.org/abs/1804.02767.

[31] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020. https://arxiv.org/abs/2004.10934.

[32] Glenn Jocher et al. Yolov5. https://github.com/ultralytics/yolov5, 2020.

[33] Jonathan Hui. Real-time object detection with yolo, yolov2 and now yolov3. https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088, 2018.

[34] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[35] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015.

[36] Jacob Solawetz. What is mean average precision (map) in object detection? https://blog.roboflow.com/mean-average-precision/, 2020.

[37] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014.

[38] Francesco Cardinale et al. Isr. https://github.com/idealo/image-super-resolution, 2018.

[39] Tzutalin. Labelimg. https://github.com/tzutalin/labelImg, 2015.

[40] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2016. https://arxiv.org/abs/1603.07285.

6 Appendix A

Figure 14: Large scale version of figure 8 (a).

Figure 15: Large scale version of figure 8 (b).

Figure 16: Large scale version of figure 8 (c).

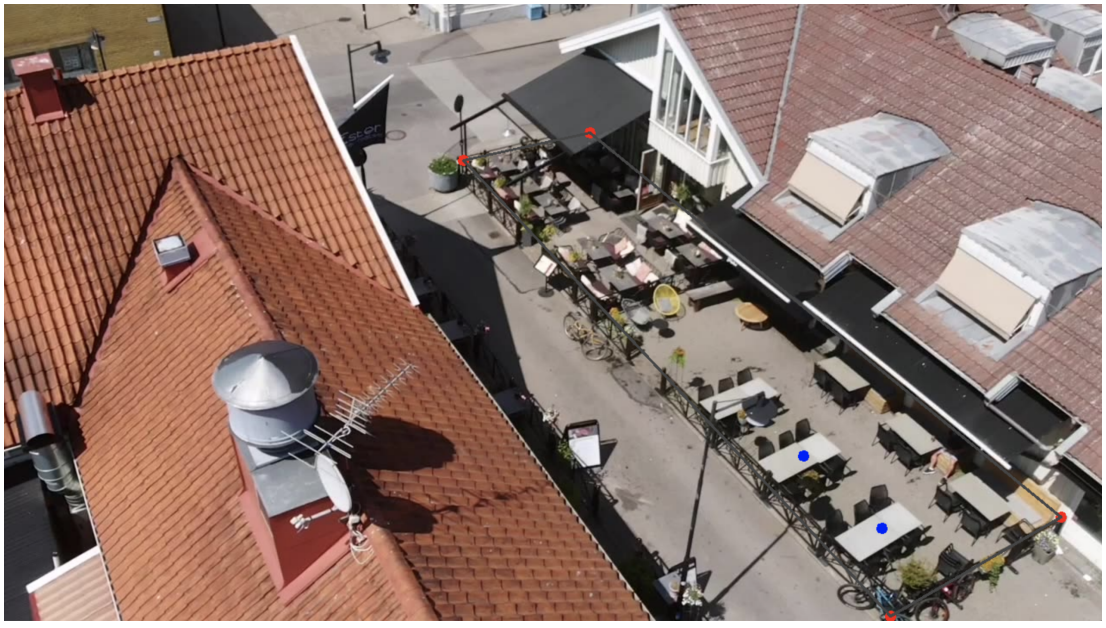


Figure 17: Large scale version of figure 11.

Figure 18: Large scale version of figure 12 (a).

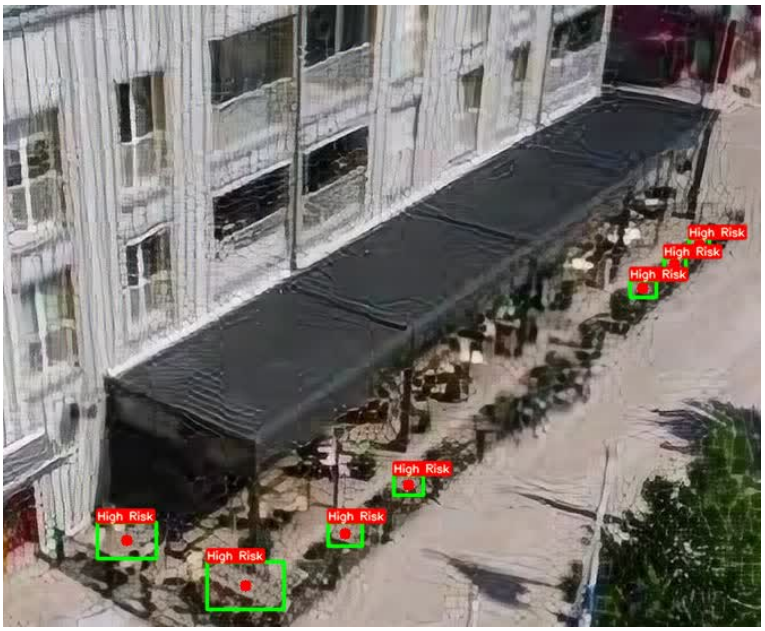


Figure 19: Large scale version of figure 12 (b).

Figure 20: Large scale version of figure 13 (a).

Figure 21: Large scale version of figure 13 (b).

7 Appendix B

7.1 Deep learning arithmetic

The following are two useful proposition for deep learning arithmetic [40]. Let o, i, k, p, and s denote the output size, input size, kernel size, zero padding, and stride, respectively. The proposition are stated under the following simplifications:

- 2-D discrete convolutions.
- square inputs ($i_1 = i_2 = i$).
- square kernel ($k_1 = k_2 = k$).
- same stride applied along both axes ($s_1 = s_2 = s$).
- same zero padding applied along both axes ($p_1 = p_2 = p$).

These simplifications are justifiable based on the observation that relationship between layers do not interact across axes [40].

Proposition 7.1 (Convolution arithmetic)**.** $\forall i, k, p, s$

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

where $\lfloor \cdot \rfloor$ is the floor function.

Proposition 7.2 (Pooling arithmetic)**.** $\forall i, k, s$

$$o = \left\lfloor \frac{i - k}{s} \right\rfloor + 1.$$

Bachelor's Theses in Mathematical Sciences 2020:K23
ISSN 1654-6229

LUNFMS-4047-2020

Mathematical Statistics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/