Content Aware Image Resizing using CNN

Project report submitted in partial fulfillment of the requirement for the degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

BY

ARSH KUMAR 181008

AND

TAVLEEN SINGH 181025

UNDER THE GUIDANCE OF

DR. HARSH SOHAL

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

MAY 2022

TABLE OF CONTENTS

DECLARATION

We hereby declare that the work reported in the B.Tech Project Report entitled ”CONTENT AWARE IMAGE RESIZING USING CNN” submitted at Jaypee University of Information Technology, Waknaghat, India is an authentic record of our work carried out under the supervision of Dr. Harsh Sohal. We have not submitted this work elsewhere for any other degree or diploma.

ARSH KUMAR
181008

TAVLEEN SINGH
181025

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Dr. Harsh Sohal
Date: 27-May-2022

Head of the Department/Project Coordinator

ACKNOWLEDGMENT

I would like to express my special thanks of gratitude to my mentor Dr. Harsh Sohal as well as our Head Of Department Dr. Rajiv Kumar who gave me the golden opportunity to do this wonderful project on the topic Content Aware Image Resizing Using CNN, which also helped me in doing a lot of research and I came to know about many new things, I am really thankful to them. Secondly, I would also like to thank my parents and friends who helped me a lot in finalizing this project within the limited time frame.

LIST OF FIGURES

Abstract

We use getting accustomed to consuming media via out phones, laptops, big screens etc. We know that there huge differences between screen sizes in them. When we browse various websites in our phones they automatically resize quite differently from our computer screens. However they do not rescale the important content they only rescale the nor important content and change the location of new lines. Obviously text has an advantage that we can break it wherever there's a space and start a new line. However common sense would say that photos or videos don't have same freedom they are restricted since they need the content they have to follow certain symmetry. Otherwise photos will look bad. Is it true though? Imagine a simple photo. You will find many details that are repeating and do not contribute to the photo content in detail. For example sky in the background. If we copy and paste the sky and just place it on top of the picture it's really not going to affect our content like faces. If we are able to blend that copy with the image. We would have essentially resized the image. Maybe changed the aspect ratio of the image. We have so many old movies, photos that need to be stretched in modern screen because they were captured in 4:3. Using Seam carving we can do that. Combining Seam carving with ML we can improve the accuracy of that.

We can also use Seam carving to assign more importance to certain element in picture. Assign least importance to certain elements. We can basically use it to remove elements in the picture.

Chapter 1

SEAM CARVING

1.1 Introduction

Today's display devices are diverse and versatile, putting new demands on digital media. Designers, for example, must produce numerous online content alternatives and design different layouts for different devices. Furthermore, HTML and other standards can accommodate dynamic changes in page layout and text. Nonetheless, photographs, despite being one of the most important elements of digital media, are typically stiff in size and cannot automatically morph to fit varied layouts. Other reasons for changing an image's size or aspect ratio include fitting it into different devices, such as cell phones or PDAs, or printing on a specific paper size or resolution. Standard image scaling is insufficient because it ignores the image's content and can only be performed uniformly. Cropping can only remove pixels from the image's periphery, therefore it's limited. Only by taking into account the image content, rather than just the geometric constraints, can more effective resizing be achieved. We present seam-carving, a simple image operator that may adjust the size of a picture by elegantly carving away or inserting pixels in various sections of the image. The relevance of pixels is defined by an energy function used in seam carving. A seam is a connected path of low-energy pixels that runs from top to bottom or left to right across the image. We can reduce and extend the size of an image in both directions by repeatedly eliminating or inserting seams. Seam selection ensures that, while preserving the image structure, we remove more low-energy pixels and fewer high-energy pixels when reducing images. The order in which seams are placed when expanding an image ensures a balance between the original image content and the artificially inserted pixels. These operators, in effect, resize photos based on their content. For aspect ratio adjustment, picture retargeting, image content augmentation, and object removal, we show how to use seam carving and insertion. Furthermore, we define multi-size images by recording the order of seam removal and insertion procedures and meticulously interleaving seams in both vertical and horizontal orientations. Such images can adjust their size in a content-aware manner on the fly. A designer may create a multi-size picture once, and the client application can scale it in real time to match the precise layout or display, depending on the size required. Several sorts of energy functions can be supported by seam carving, including gradient magnitude, entropy, visual saliency, eye-gaze movement, and more. The removal and insertion algorithms are parameter-free; nevertheless, we give a scribble-based user interface for applying weights to an image's energy and guiding the desired results to provide interactive management. This

programme may also be used to create photos in many sizes.

1.2 Background

In many image processing software, picture scaling is a regular feature. It operates by scaling the picture to a specific size in a consistent manner. Recently, there has been a surge in interest in image retargeting, which aims to reduce the size of a picture while preserving significant elements, which can be recognised top-down or bottom-up. Bottom-up approaches depend on visual saliency methods to generate a visual saliency map of the picture, whereas top-down methods employ tools like face detectors to find key parts in the image. Cropping can be used to display the most essential portion of the image once the saliency map has been created. on either a saliency map or the output of a face detector. The huge image is then cropped to capture the image's most important portion. Similarly, looked at the issue of picture adaptation for mobile devices. The most essential section of the image is automatically recognised and supplied to the mobile device in their approach. addressed picture re targeting to mobile devices as well, recommending that time be exchanged for space. They design an ideal path across a set of places of interest and present them serially, one after the other, to the viewer. crop photos intelligently using eye tracking and composition principles. All of these approaches provide remarkable effects, but they rely on standard picture resizing and cropping processes to modify the image's size. the first solution to the basic challenge of warping a picture into an arbitrary form while keeping user-specified attributes. A special version of the Laplacian editing approach, tailored to handle similarity restrictions on sections of the domain, is used to produce feature-aware warping. Because the global optimization process propagates local restrictions, not all of them can be met at the same time. Because our approach is discrete, cutting one seam has no bearing on the remainder of the picture.

1.3 Method

3 steps are used to carve seams:

1. Working out how much energy each pixel has.

This is accomplished by smoothing the picture and then applying a 3x3 Sobel filter to compute the initial x- and y-derivatives at each location.

2. Using dynamic programming, determine the least total energy seam from top to bottom(or left to right).

Seam as the name would suggest is continuous pixels in each direction(can either be vertical or horizontal). A vertical seam, if taken the example of would be made of one pixel per row with pixels in

consecutive rows.

3. Creating a smaller/larger(limited) picture by removing or adding pixels along the traced seam. Let's say there were an area where there is no major energy. That would usually mean that the area itself is not really that useful usually which means we can add or remove pixels without affecting the main content of the picture. If we are to add the pixels it will enlarge the pixels. The pixels that will be added are added next to the traced seam and pixel values will be the average of the surrounding area. The same goes for deleting the pixels.

The steps are repeated until desired result is achieved. Obviously the image expansion is limited since the algorithm is not omniscient. You can't keep on adding pixels forever sooner or later you will start to add pixels using previously average value which will result in compounding affect. Since you are just adding up averages. However still in limited case that is, just changing or adjusting image for dynamic content delivery the algorithm is suffice. Before when we were calculating the seam carve after the initial deletion or addition we were then recalculating the whole energy level again but that is useless since the change of energy levels would usually be around the area where the seam was added or removed which then means that the only place that we really need to calculate seam is at the are near new deleted or added seam.

This does not affect the calculation times much to be of great significance in smaller picture but again due to compounding affect the bigger picture calculation time if 10-20 seconds faster which is very useful since for real time application the need to reduce the time would be very important.

1.4 Unsupervised Learning

Now unsupervised learning is a bit different from supervised learning. It's almost similar in its structure but quite different in functioning. Here the dataset we provide does not have any labels. As we know most of the data around us would be unlabeled so if we can use ML to analyze this data it only makes the prediction better. It is dependent on whether the model can establish any link between data on its own. There's no supervisor here because the data does not have any outcomes. The model need to find regularities some kind of pattern in the data. One of the methods of unsupervised learning is Clustering. In Clustering "the aim is to find clusters or groupings of input" - Alpaydin [1, p.-112] If the model is implemented properly it is able to detect outliers in a set of data. It can be used by companies or groups to filter their data to a more niche market. It can be used to remove noise in a media because noise would usually stand out from the rest of the bits. There are various implementations of clustering or unsupervised learning in general. Wherever there's a requirement for classification of data be it search engine, data grouping we can use unsupervised learning to analyze datasets.

1.4.1 KNN

KNN or K-nearest neighbor is an unsupervised machine learning algorithm used to predict what category a new point might belong to. We will be able to use KNN to predict the value of pixel. Now, remember this is only required in case of adding a new seam.

1.4.2 Live Media editing

Finally we consume a lot of media everyday. Now many times we might have tried cropping an image or resizing it only to find out that we have cut or changed dimensions of important picture. If you want to watch an old movie that was shot in 4:3 aspect ratio on modern 16:9 or 16:10 TV you will need to stretch the video. However with techniques like seam carving we can at least change the dimensions of an image. Now there have been fast implementations of the same algorithm that can be used. However seam carving has one problem that it cannot detect faces or important feature by itself. It's working assuming that faces will be brightly lit and can be detected with edges. Which is not always true as sometimes photos might be underexposed which can be a problem and will result in badly carved image. However this can be improved if we take help of Machine Learning Models.

1.5 Chapter Conclusion

This introduction of basic concepts was about Seam Carving and ML. How we can unite them and make a useful product which can be implemented in various fields. Most people don't realize this but they need seam carving everywhere today. When you rotate your phone the photo get squished and smaller which might be a minor inconvenience. Not worrying about aspect ratios and focusing on content is really helpful in everyday life. This will be described in more detail in upcoming chapters.

Chapter 2

FUNCTIONING

First we need to implement and understand basic Seam Carving algorithm only then can we actually implement changes. Avidan and Shamir [2] However in normal seam carving the new seam is predicted using arithmetic average here we will use KNN, a Machine Learning algorithm.

2.1 Overview of the Seam Algorithm

First of all Seam Algorithm has a pretty basic step. That is carving the seam. Now obviously the question is what are 'Seams'? In a photo we have many regions made of pixels. Some pixels or some regions might not be that important to the image. So we pick those pixel and create a path along them. Obviously the path need to be interconnected so we need to find pixel in the new row that are either at the bottom, bottom-left, bottom-right. Something similar to this.

Figure 2.1: *Image matrix representation*

As you can see if we only stick to bottom three pixel we can create a path. Proper method will be covered in next few sections. Now this path shows the path of least energy. Meaning of least energy is that it's less important. The way we find importance is by first finding edges and then finding pixel that would be of low intensity.

Therefore steps required for Seam carving Removal are:-

- Making an energy map. Energy map is created by finding gradient of the Image. The gradient for x direction and y direction are found separately. Gradient is calculated using a kernel which is passed over the image. They are then combined to form a image that highlights the edges. This is called sobel edge detection.[3]
- Now we pick a top point in that energy map and start to find min path towards the bottom edge.
- Remove the Seam.

- Rep Steps 1–3 until we hit the desired size(Only in case of removal). Every time a seam is removed, the energy map must be recalculated.

Now the steps to add seams.

- Making an energy map. Energy map is created by finding gradient of the Image. The gradient for x direction and y direction are found separately. Gradient is calculated using a kernel which is passed over the image. They are then combined to form a image that highlights the edges. This is called sobel edge detection.[3]
- Now we pick a top point in that energy map and start to find min path towards the bottom edge.
- Calculate the Seam by finding the average along the path.
- Add the Seam.
- Rep Steps 1–4 until we hit the desired size(Only in case of addition).

2.2 Insertion of the Seam

Seam insertion is basically similar to removal however we are just reversing the steps(technically) to achieve a enlarged picture. Insertion is very useful to change the aspect ratio of image by adding unnecessary information that can enlarge the picture but not change the dimension of useful content for example a face.

2.3 Object Removal

Objects can also be removed from a picture. The basic concept stays the same we still find path with least energy. However now we control which pixel are least energy. If we implement some kind of interface that can allow the user to make selection for areas that should be high energy and areas that should be low energy seam carving can be use to remove certain elements from the image. Since the algorithm itself doesn't know what it's removing we can easily manipulate it.

Chapter 3

SEAM CARVING STEPS THEORY

3.1 Image Pre-Processing

Figure 3.1: *Original Image*

So when we first take image as a input we obviously can't draw seam directly on it. We first open an RGB image. RGB images have three channels Red, Green and blue. Each of them contains pixel that can range from 0 to 255 total of 256 values. First we convert these channels into gray scale. Converting to gray scale in better because edge detection works better in gray scale. Afterwards we find gradient of the image using kernel usually This kernel find the gradient for y axis after we

-1	0	1
-1	0	1
-1	0	1

Figure 3.2: *Kernel for dy gradient*

convolve it with the image. Similarly we invert it to find gradient for x axis. Then the resultant

Figure 3.3: *Image Edges*

gradients are combined and merged. We then get a image that contains the edges. As you can see the edges have been detected and the part that are bright are important pixel whereas dark parts like the sky are unimportant pixels.

3.2 Processing of the image

Now that we have our edges we can simply start finding the minimum path and then delete it. However that would increase the time complexity to 3^n which is not desirable at all. So we use dynamic programming to solve the problem. Let's say that we have a matrix as below: Now we want to find

5	2	1
1	6	10
10	10	1

Figure 3.4: *An example matrix*

the minimum sum towards the bottom from the mid point. So first we try finding minimum number in the next row. However that is wrong since in this example you can see the minimum path is actually $2- > 10- > 11$ but if we just look at the row below we would choose 1 after 2 and then 10 which

we know is the wrong path. After that we can try naive method. Basically trying every possible path and then saving the path with smallest sum but as already discussed it is too slow. So we use dynamic programming to create energy function. We start from the bottom row of the matrix. We can assume that there's a row with zeroes below the last row in this matrix. We create a new matrix for energy function and then we copy the last row in it. In this case it would be the imaginary row of zeroes. Now we will start filling the sums from bottom. So we take a look at the original matrix and first element is 10 now we look at the energy matrix but in a row below and find the smallest element out of three(in this case out of 2). In the case of first original row it will be the same elements but the second row will be different. We take number from the original matrix but take the minimum from the energy matrix from a row below. Then add the min and original. Doing this for every pixel we create an energy matrix.

12	9	8
11	7	11
10	10	1
0	0	0

Figure 3.5: *An example of energy matrix*

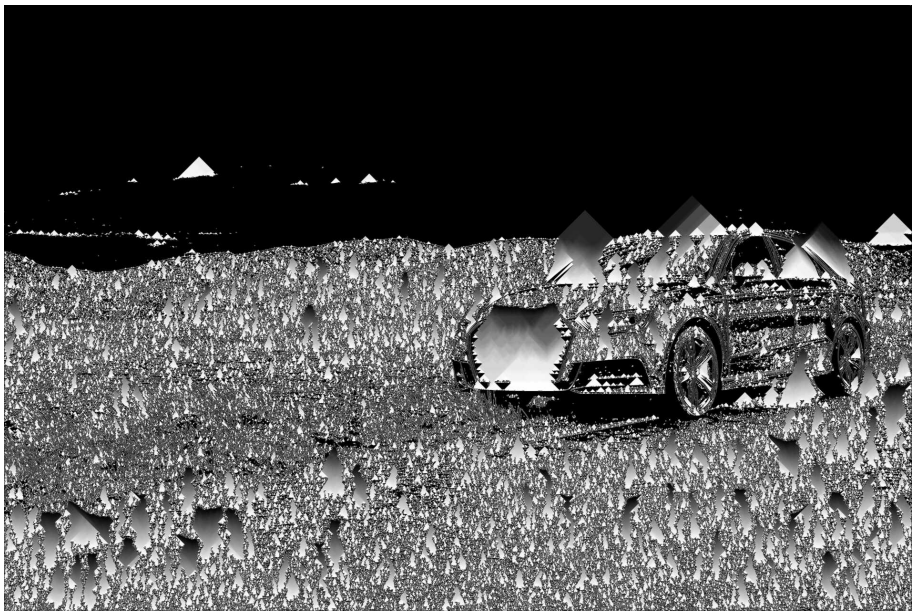

Figure 3.6: *Energy function photo*

The triangle are basically the most important part of picture. They have the highest energy and seam will always try to avoid them.

As we were filling them we also should create an array of the direction that can speed up our navigation when creating a seam. We create an array with similar dimension to our image. Then as we were filling the energy function alongside we also fill this array. The method to do so is to represent the three direction with values. $bottom - left, bottom, bottom - right = -1, 0, 1$ So let's say our minimum value for point (3,4) was in the bottom right. We will put 1 in the array at position (3,4) representing that we needed to move to the bottom-right pixel from that position.

3.3 Drawing the Seam

Drawing or finding the seam is now very simple we just pick a pixel on the top most array. Picking pixel from left side in this case. Then we just follow the direction matrix we created and find the seam. We can continue this along the width if we want to increase the width and along the height if we want to increase the height.

Figure 3.7: *Single seam in the photo*

3.4 Removing and adding the seam

Removing the seam is very simple we just recreate the image but just skip the pixel that are in seam. However adding seam can be simple or can be made complex. In this case we are using machine learning KNN to predict the values for the seam. So let's say we want to add a new seam first we need to understand basics of KNN.

3.4.1 KNN

KNN is a machine learning algorithm that can predict value for a new point. It basically select the nearest neighbors based on euclidean distance and then calculate the probability based on frequency of values at the points. The reason we use this algorithm rather than hardcoding to find values around it is because it can help us to scale up the program later easily.

Now that we understand KNN we can easily implement this for seam carving. We first need to control the distance of data from the point. We can limit that to 30 pixels and then limit the k to 20 so that we can find the approximate value.

Chapter 4

PROGRAM EXPLANATION

4.1 Programming Language

We are using Rust for programming seam carving. Rust is a fast compile language which would help us to expand the project later and add other deep learning models in it. We use basic image crate(library) in rust to process images and control them.

4.2 Basic Input

First we need to input the file name and required dimension for picture. For that we have a file called input.rs that takes in input paramter from command line. We just check if the file exists and then check if new width and height are proper. They are then stored in a Struct so that we can combine them together.

4.3 Seam

We have a file called seam.rs that contains the seam functions. Basically the file name that was taken through the input.rs file is now opened and checked if it exists. If it does then we open it and create its clone. Then we create energy function and after that we start from the top and carve the image. Now to remove or add new seam we can call the functions accordingly.

4.3.1 Removal

For removal we are just skipping pixels of photo that are detected by the current seam.

4.3.2 Addition

We call and external binary to handle the KNN algorithm. Since local processing might be slow because of limited power.

Chapter 5

NEURAL NETWORKS

5.1 Introduction to Neural Networks

Neural Network are used to simulate the activity of human brain and function a bit similar to the human brain so that to solve AI or ML problems.*IBM* [4] Now Neural Networks are a subset of Machine Learning used to solve problem in ML by Simulating the activity of human brain.Hurbans [5] They are inspired by the way human brain functions in that they function how biological neurons send signal to each other. In general a Neural network would have Layers where the information is processed one by one. One layer would be the input layer and the other would be the output layer. There might be one or more hidden layer that process the data in some way. Each layer is built of multiple artificial neurons or nodes that are interconnected to nodes in the other next layer. In simple terms we can say there's a threshold value of each node that activates it until the signal crosses that threshold value the node remains inactivated but as soon as the signal crossed the threshold value the node activates and the signal is activated. We also need to train the neural network for some specific application first that can adjust these nodes to understand the data that will be given to the neural net. We can also say that each individual node is some sort of mathematical entity that has a lot of operations going on inside it. A node will have certain weight and a bias assigned to it what they are will be explained later in details.. Each node changes when it's learning new information, i.e it's weight and bias change so that the new information can be correctly processed.

5.2 Working of Neural Networks

A basic Neural Network would be comprised of an Input Layer, Output Layer there will be Hidden Layers sandwiched between them. The Layer have nodes that would be connected with other layer's node. A node represents a neuron in human brain mathematically. Just like when neuron would be activated if some activity provides some stimulus in brain the nodes are only activated when certain threshold is met. Signal start from input layer and goes layer by layer in order to be processed in some way and we get our results in output layer. In layman's term we can explain the processing of data to be like this. The input layer get's some signal, for sake of simplicity let's assume the first input layer consist of a single node and hidden layers have more nodes. we would represent the figure in this way.

As we can see the basic input layer is connected with two hidden layer(1, 2) and an output layer.

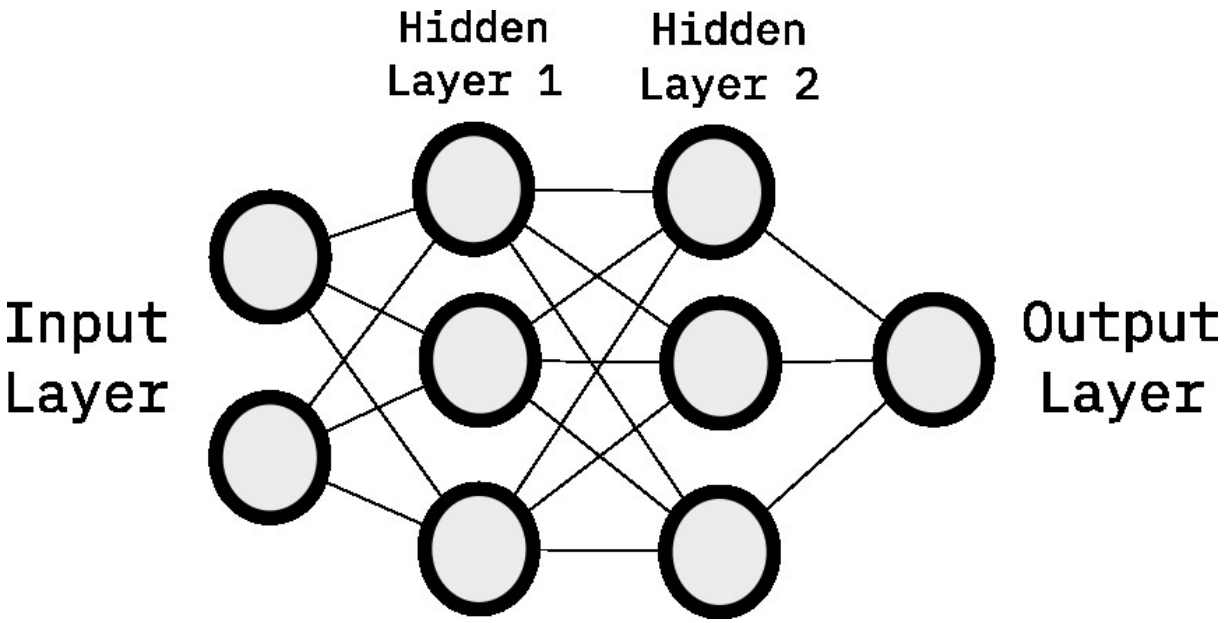

Figure 5.1: *Basic representation of neural network*

For sake of understanding, let's assume that the input layer takes in the price and the dimensions of the house, the hidden layer one then process the information about price whether the price is high or low and then the next hidden layer compares the price to the house dimensions and the final output layer gives us a simple YES or NO answer. The model will be trained by taking data of various prices and dimensions of house and then telling the model which is passable and which is not. This will be done by adjusting the weight and bias of each node.

5.2.1 Nodes

$$\sum_{i=1}^{n} w_i x_i + bias$$

$$w_1 x_1 + w_2 x_2 + w_3 x_3 + bias$$

- The formula above represent how we would calculate the threshold of a node. First variable we talk about is w_i .

  As the name suggest the weight is the actual weight of the 'thing'. Weight represent the importance of any feature the node is supposed to represent the more the weight the higher the importance of the feature which it is representing.

- Next is x_i.

  This is the input. It is a group of value of the input that we need to calculate against. As we can understand mathematically that the weight get multiplied with the value thus increasing the importance of value basically higher would be better. The greater ones will be contributing more to the output.

- Finally we have **bias**

  Explaining the bias is a bit difficult in simple terms but the bias will help us shift the value around for which the node will be activated. Take a normal sng. function as an example which is show in the figure below.

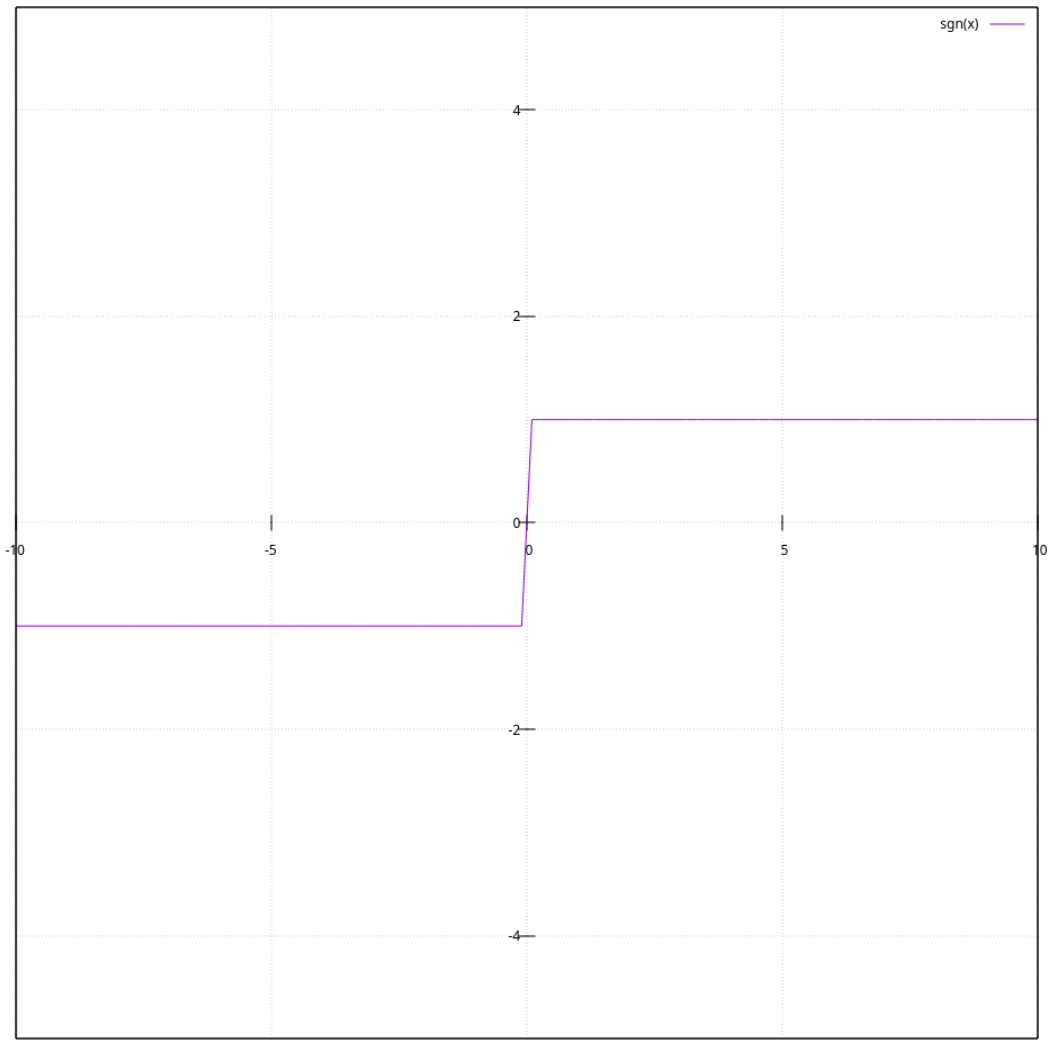

Figure 5.2: *A sig. function without bias*

Now let's say we apply bias of 2 to this function it just moves the function around. That's all bias does it moves around the activation function.

- Activation Function The sum of weights and bias themselves is called the activation function.

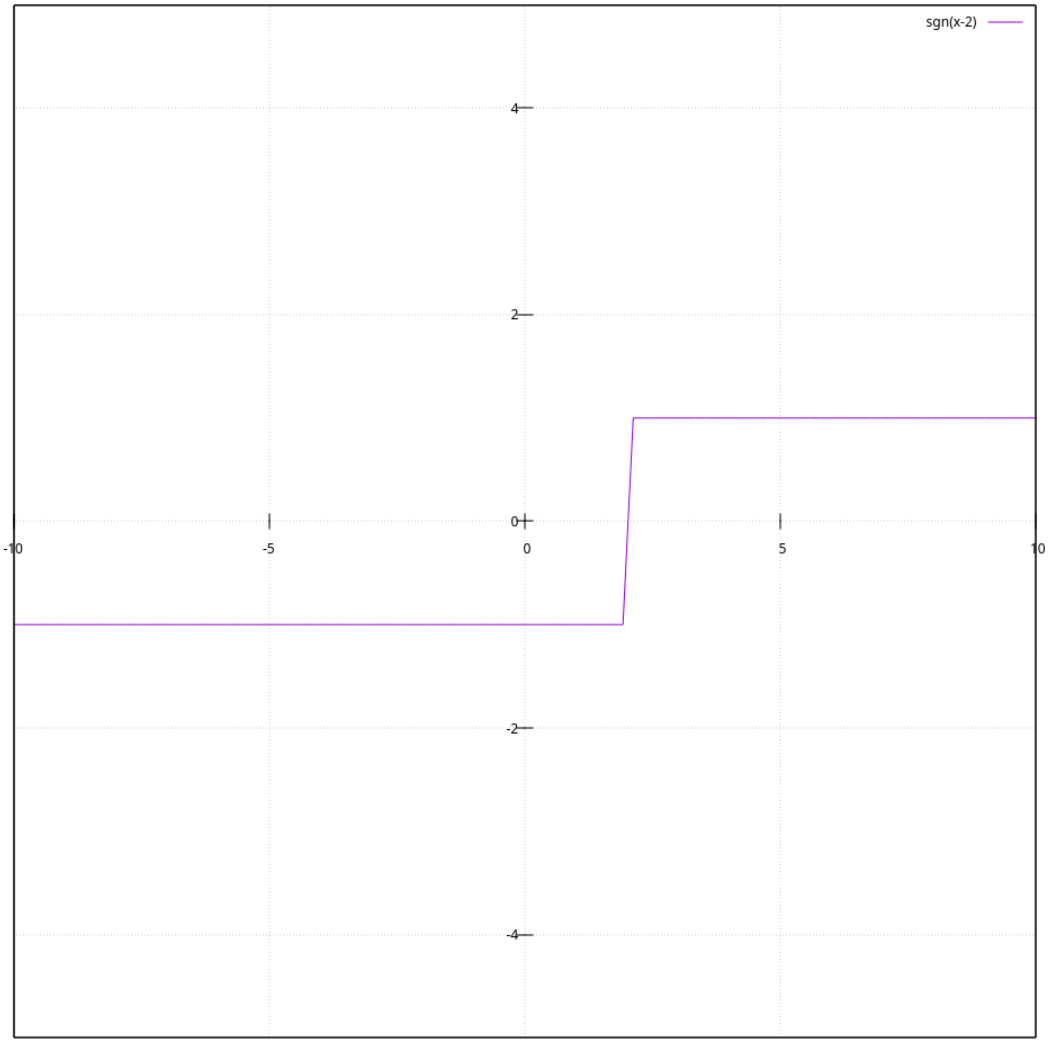

Figure 5.3: *A sig. function without bias*

The formula above represents the activation function. So we can easily say that the activation function refers to the mathematical function that defines whether the neuron or the node will be activated or not.

5.3 Training

We need to train the Neural Net. to make it functional. To train the Neural Network it means to adjust the weights so that the input that we have in the database best match the output. So that the neural net can predict new outputs. This training process could be considered iterative because the weights are updated each time in small changes for every data entry in the data set. Now this learning process need to be optimized. In this case the optimized Neural Network would mean Neural network with less error or Mean square error.

$$CostFunction/MSE = 1/2 \sum_{i=1}^{n} (y' - y)^2$$

Figure 5.4: *Formula for mean square error*

With every iteration the main purpose of the neural network is to reduce the Mean Square Error. So the Neural Network Works on the basis of reducing this error and finding a minima to converge to. Most Neural Network can only send their information in one direction which is called feed forward neural network. Neural Networks can also be trained for back propagation. Meaning sending information from output side to input side.

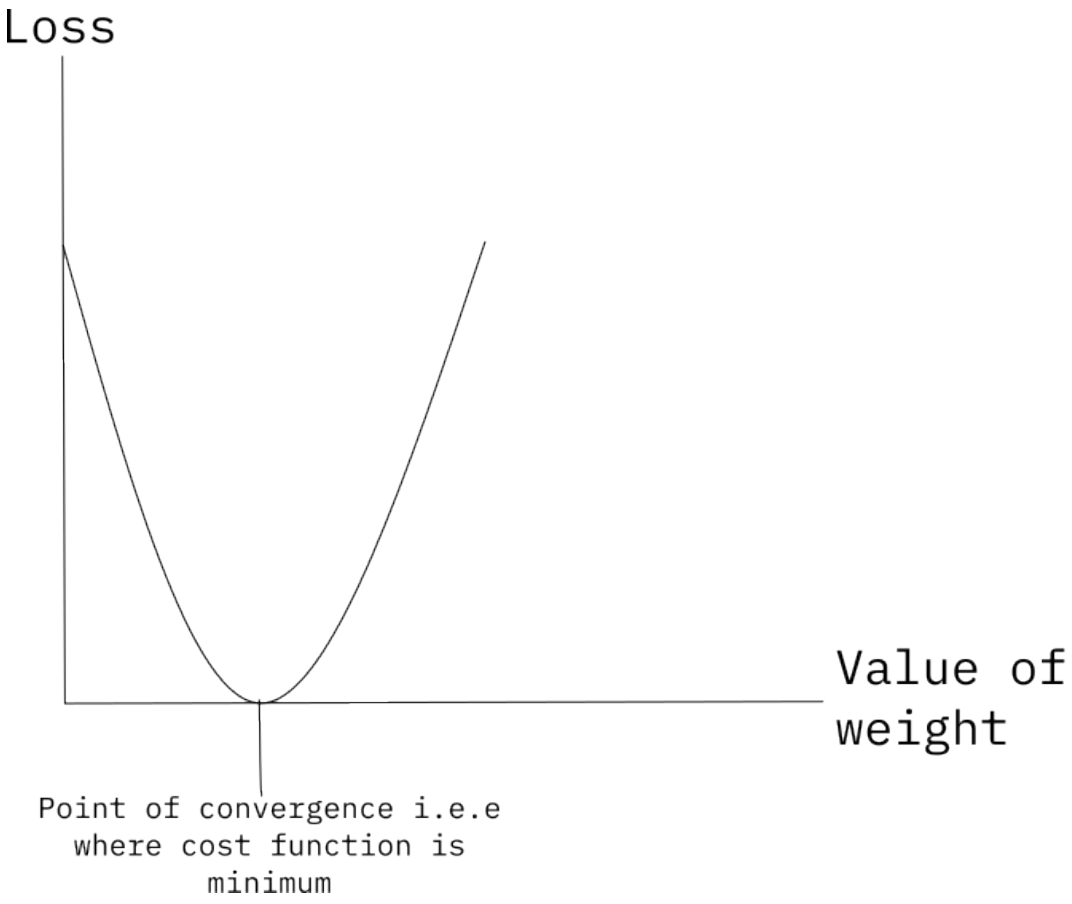

Figure 5.5: *Point of convergence*

5.4 Type of Neural Network

- **The Perceptron :-**

  The oldest and most simple type of Neural Network. The perceptron unlike node is really simple in its architecture. The key difference that separates perceptron from the node is the fact that perceptron does not work on linearly inseparable data. That is because perceptron cannot Build XOR gate.

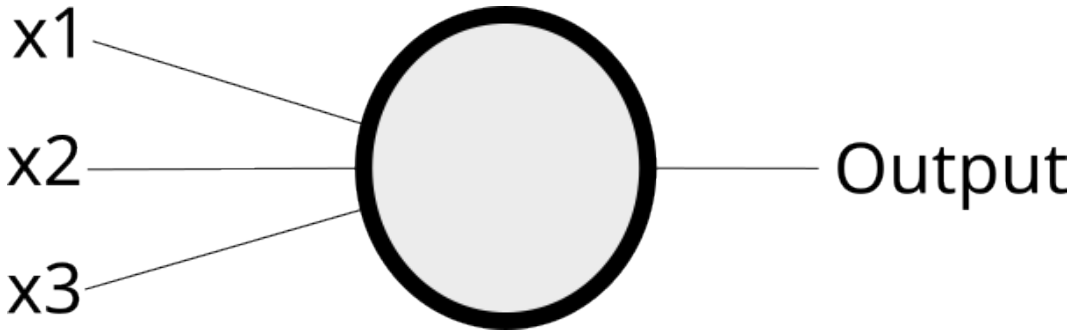

Figure 5.6: *The Perceptron*

- **Feedforward Neural Network :-**

  This is the modern Neural Network. The nodes contain the sigmoid function we talked about earlier. The Neural Network is basically the base for all other Neural Networks.

- **Recurrent Neural Network :-**

  These Neural Network are used to have Neural Network with Memory. Basically these Neural Networks have feedback loop that move the data back to improve the Neural Net. These are used for prediction work usually. For eg. predicting stock and such. Graves [6]

- **Convolutional Neural Network :-**

  As the name suggests that these Neural Network use convolution. Convolution is really great for processing images because that is how various operation like edge detection, blurring are done.Burrus and Parks [7] By convolution the image is processed and data is extracted inside the hidden layers.

Since Convolutional Neural Network are really good at image processing we are going to use them for extra help in seam carving.

Chapter 6

CNN

Seam Carving without any CNN's help has one problem. Let's say that the image has certain region that have under exposed part or over exposed part normal Seam carving wouldn't be able to detect them. For eg.

Figure 6.1: *Seam Carving without CNN*

As you can see the image has it's main object under exposed maybe for stylistic purpose. Now obviously our method of detecting important features wouldn't work in this case as we can already see. The object gets distorted because the energy function considers the object useless because the pixels aren't bright enough. So what we can do is manually add a mask. What Seam carving will do is increase the energy of the area inside the mask to max. Making whatever is inside the mask highest priority.

Now we can use any mechanism to find the mask. It's irrelevant to the seam carving algorithm how we find the mask. of course the way we find it might affect the quality of outcome but seam carving algorithm doesn't care. Therefore rather than manually making mask around the object we can just use CNN to create a mask around the object. CNN will try to find the object in image and mask it.Weisstein [8]

6.1 R-CNN

To create mask around image we are using a slightly modified version of CNN that is R-CNN. R-CNN stand for Regional CNN which creates region of various objects inside the image. Girshick [9]

6.2 Masked R-CNN

Masked R-CNN is more modified version of R-CNN that is specifically designed for creating masks around the image objects.

Masked R-CNN really solves a big problem of images that are underexposed or overexposedHe, Gkioxari, Dollár, *et al.* [10]

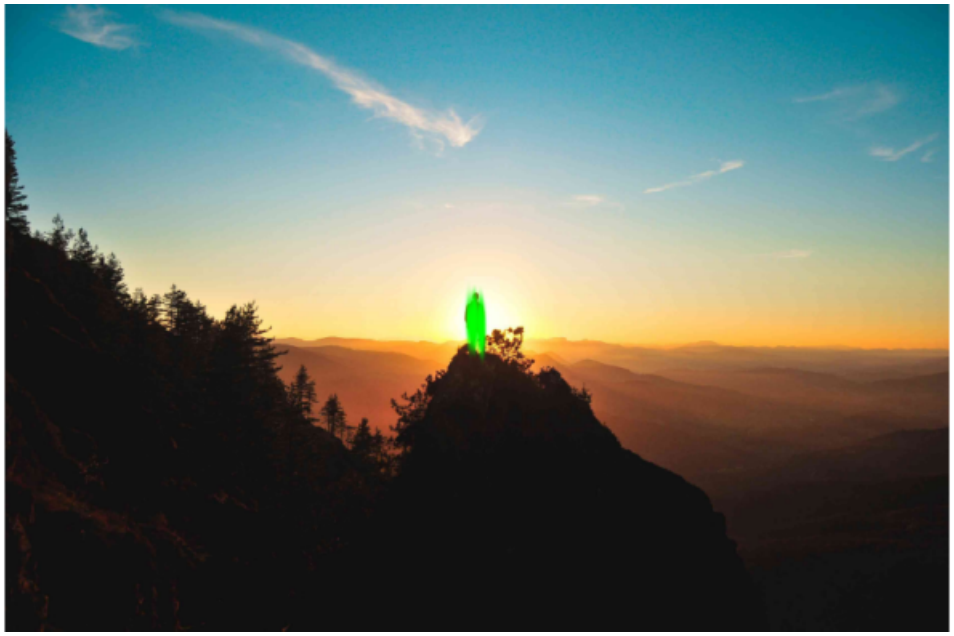

Figure 6.2: *Using Masked R-CNN to mask*

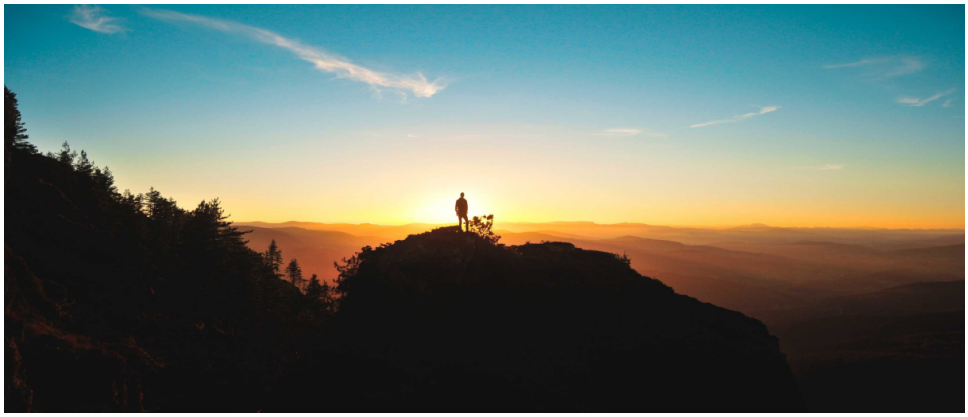

Figure 6.3: *Seam Carving with CNN*

6.3 Picture Representation Digitally

Normally when we look at a picture in digital screen we know that it is comprised of smaller individual units called pixels.Gonzalez [11] These pixels can be said to be a part of a matrix. A cell in a matrix represent the pixels and the value of that cell would represent the intensity of that said pixel. So let's

assume a simple 3x3 matrix that only contains binary values. For eg.

We can take this matrix as a representation of an image that only support two colors. Either black

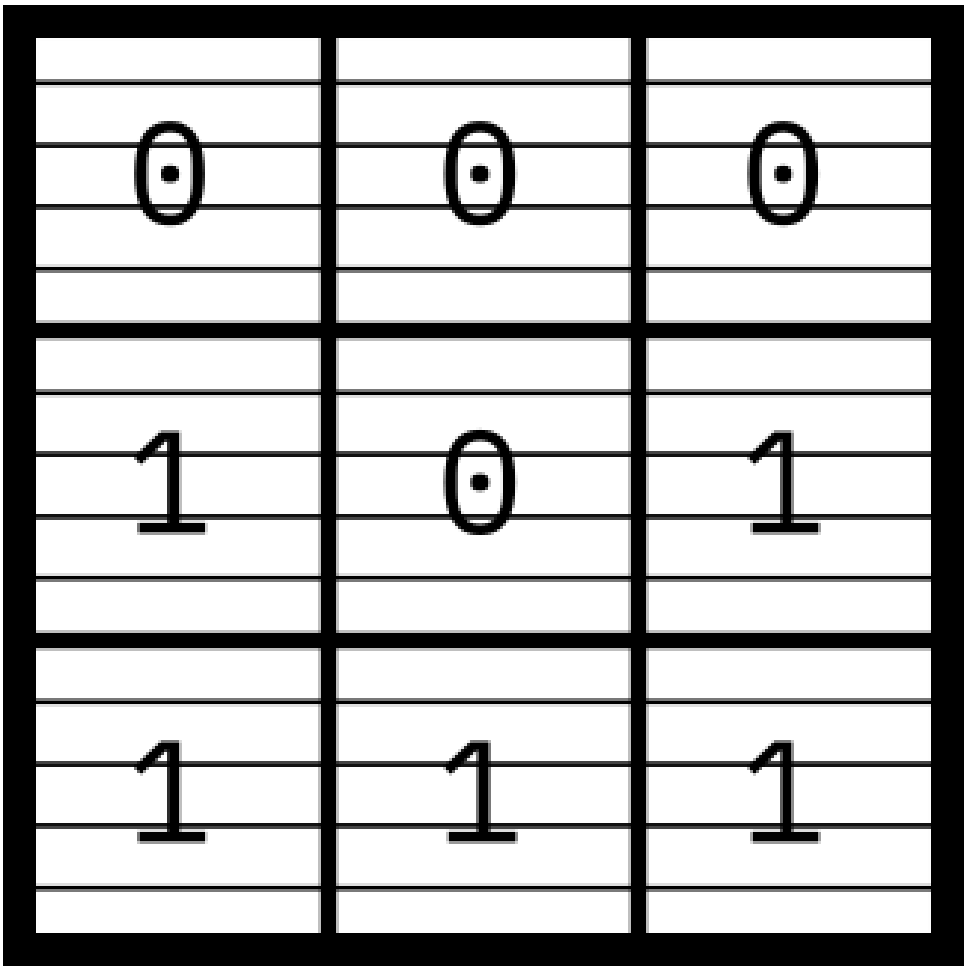

Figure 6.4: *A matrix with binary values*

or white. So wherever there's a 1 in the matrix we represent that part with 1. Wherever there's 0 we represent that part with black. As we can see this might be considered an image of a simple line. Every image we see is just a complex representation of this simple concept. In our case we will get a color image that will usually have RGB value.

RGB Stands for:-

- R:- Red
- G:- Green
- B:- Blue

First imagine a gray scale image. We also need to decide how many bits do we need to represent or information. Normal images for digital screen use 8-bits of information. That would mean 256(2^8)

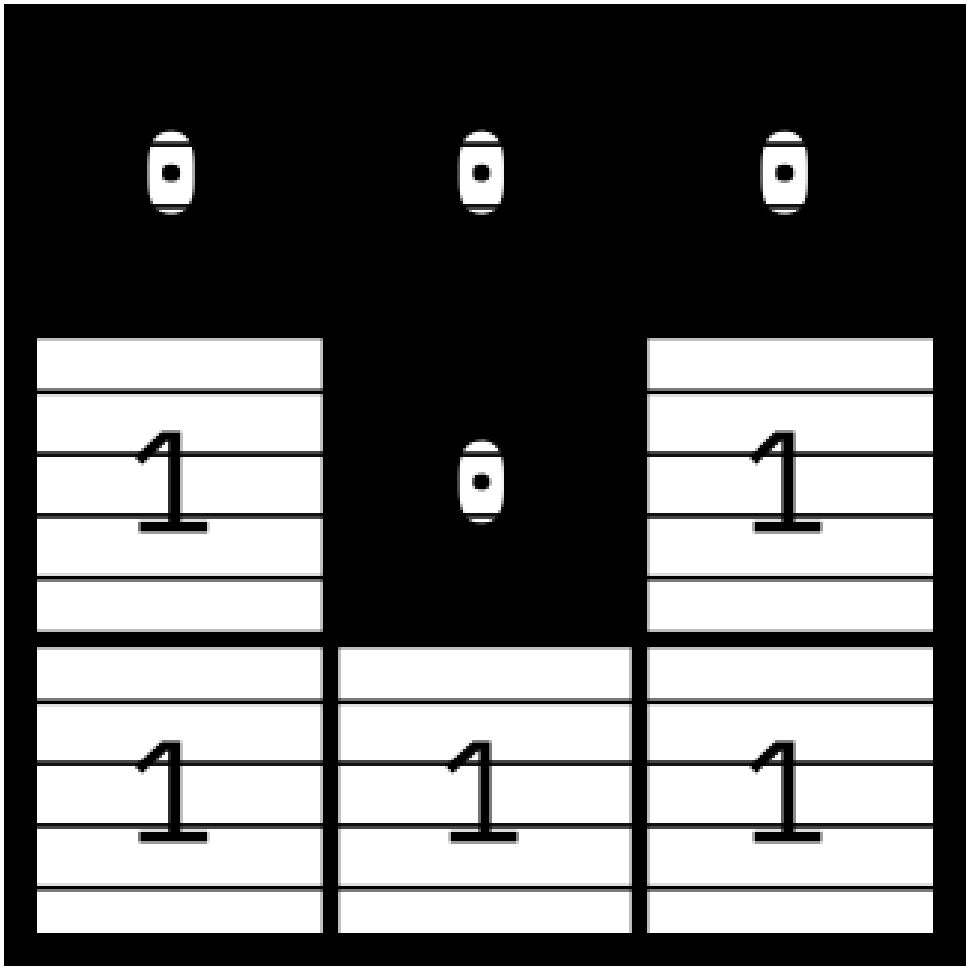

Figure 6.5: *Representing the color of matrix according to its value*

different intensity or pixel values we can use starting from 0 to 255.Saravanan [12] As the name is grayscale we would represent 0 as the absence of black and gradually adding white as the value increases. If we want to increase information we can always increase the bits but since our purpose is to seam carving on digital screen and for images that are used everyday, we will have no problem with 8-bits of information. Resolution a term commonly used with images mostly used to define quality of the image.Den Dekker and Van den Bos [13] However that's not necessarily true. The reason we consider resolution important is because we as humans are better able to understand information when many pixels get combined and we can see 'something' that represents real world in image. For that we would definitely need high amount of pixels as the requirement for high resolution image and screens would suggest.

The bit is as important as resolution. Imagine looking at an image with only two color black and white like the example before. There would be not depth to the image. The colors wouldn't be accurate or even close to what we see in real world. Let alone colors even the simple information wouldn't be able to represented in the image.

Now getting back to the grayscale image. Consider grayscale a channel or a layer. In a grayscale image we would only have one channel

For eg. So this single channel will create a new image but only has one color.

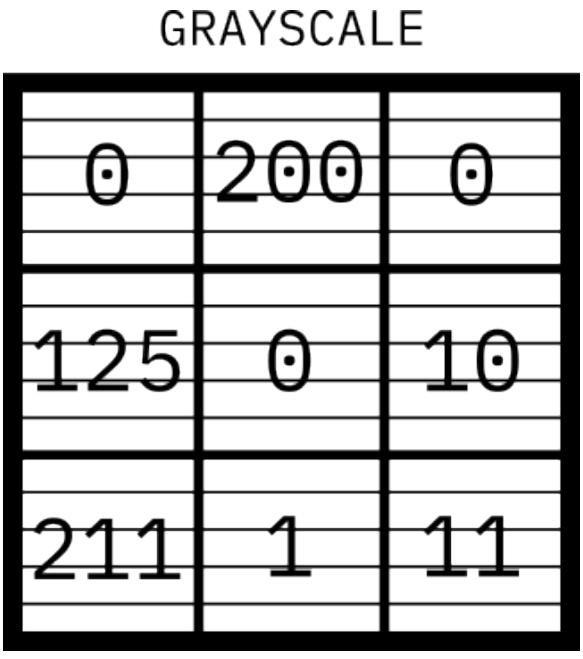

Figure 6.6: *A single gray scale channel image matrix.*

However we know that mixing few color can create multiple colors. We use the same principle in images. The colors we see are usually combined of three colors or layers of three colors or matrix of three colors. Which combine to give use visually vivid images.

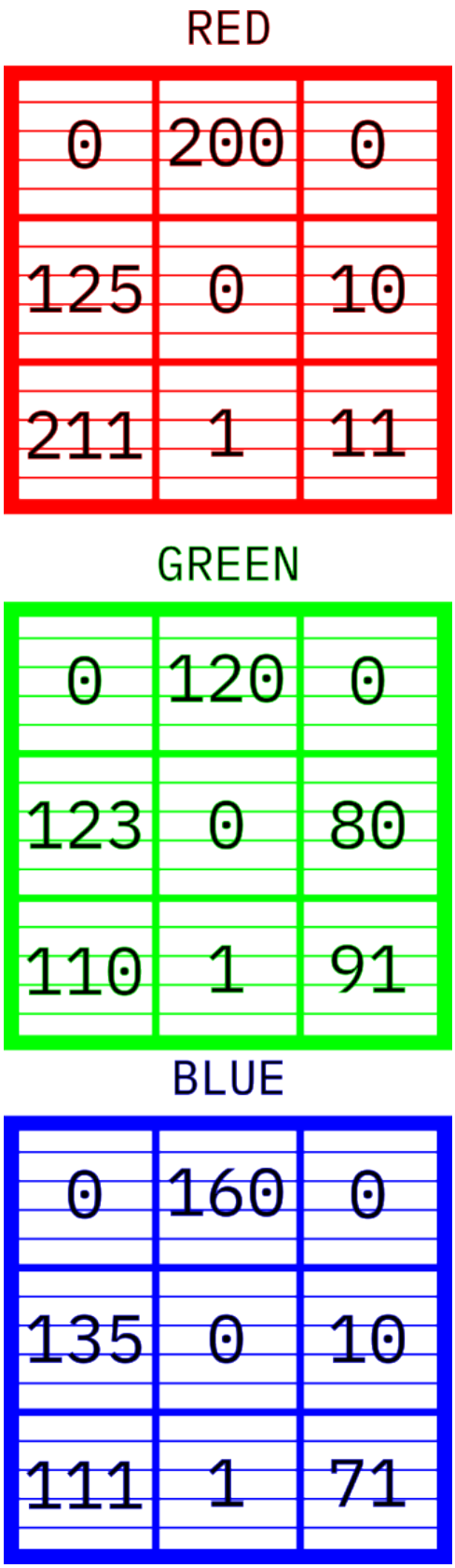

Figure 6.7: *A three channel image matrix.*

6.4 Convolution

Mathematically it can be defined as integral of product of 2 functions after we change property of one function by reversing and shifting it.Weisstein [14] As we discussed before images are just a matrix. We can perform various mathematical operation on a matrix. Thus altering properties of that matrix thereby also changing properties of the image. In this we will be using the convolution. Simply put

$$(f * g)[n] = \sum_{k=-\infty}^{\infty} f[k]g[n-k]$$

we will take another matrix usually 3x3 which will be called kernel. Using kernel is how we will

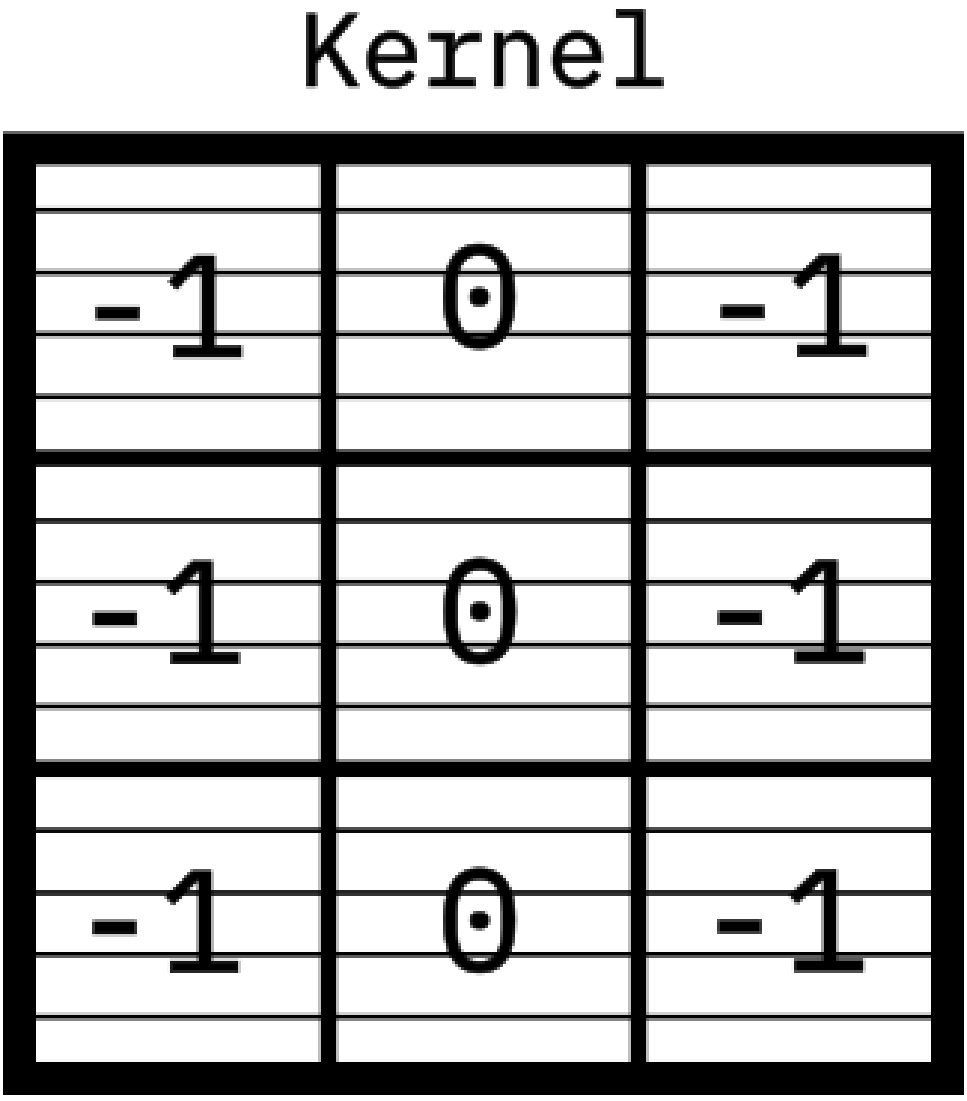

Figure 6.8: *An vertical edge detection kernel*

extract various features from image and find various objects. We use kernel and convolve it over the image to edge detect in this case.

6.5 Summary

These were the basics of Convolutional Neural Net. We saw how convolution work. The layers will convolve the image to extract feature. That is precisely why the CNN is used rather than any other type of neural network. Since it processes images natively.

Chapter 7

RESULTS

What we can see is that algorithms like seam carving are going to play very important role in near future. Since they allow live content editing which can be used in many places.

7.1 Results

As we can see in the image below the image has been resized but the important content that is the car has not been changed at all. Which is what we were expecting.

Figure 7.1: *Original*

Figure 7.2: *After Seam Carve*

The results before were achieved without using the CNN because they weren't required in the image as the object wasn't underexposed. However the image in the next case is underexposed

Figure 7.3: *Original*

We then create a mask around the object using CNN

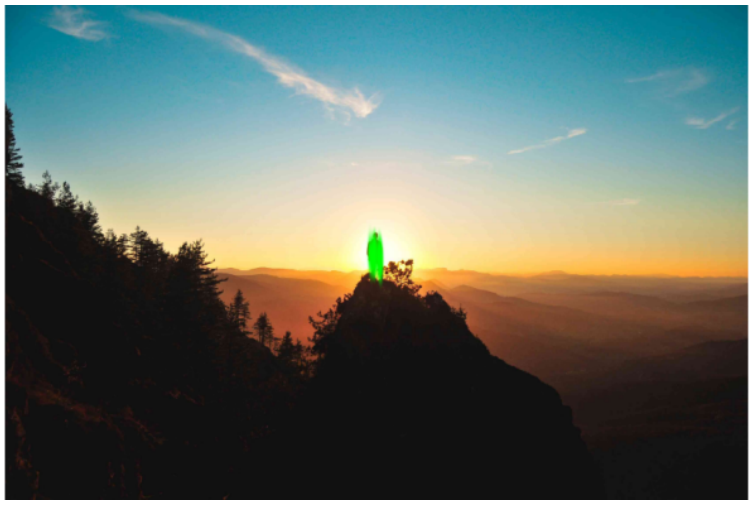

Figure 7.4: *Creating a Mask*

Then we simply apply the seam carving algorithm and we get the results i.e. an expanded image.

Figure 7.5: *After Seam Carving*

Appendix A

MAIN

```
mod seam;
mod input;

fn main() {
    seam::seam();
}
```

Appendix B

INPUT

```
use std::path::PathBuf;
use std::env::args;

pub struct Parameters {
    pub file_path: PathBuf,
    pub new_width: u32,
    pub new_height: u32,
}

pub fn user_input() -> Result<Parameters, Box<dyn std::error::Error>> {
    let input_arguments: Vec<String> = args().collect();
    if input_arguments.len() > 4 || input_arguments.len() < 4 {
        println!("{}", "Usage: seam [FILE] new_width new_height");
        std::process::exit(0);

    }
    let para = Parameters {
        file_path: input_arguments[1].parse()?,
        new_width: input_arguments[2].parse()?,
        new_height: input_arguments[3].parse()?,
    };
    return Ok(para);
}
```

Appendix C

SEAM

```
use extern::crate::imagemagick
use super::input;
use image::io::Reader;
use image::DynamicImage;
use image::GrayImage;
use imageproc::edges::canny;
use imageproc::seam_carving::*;
use imageproc::histogram;

pub fn seam() -> Result<(), Box<dyn std::error::Error>> {
let dim = match input::user_input(){
    Ok(x) => x,
    Err(_) => std::process::exit(0),
};
let img = Reader::open(dim.file_path)?.decode()?.to_rgb8();
let mut img_clone = img.clone();
let (width, height) = img.dimensions();
let img_luma = DynamicImage::ImageRgb8(img).into_luma8();
let l_threshold: f32 = 1f32;
let img_edge = canny(&img_luma, l_threshold, l_threshold*2.0);

let mut img_energy = GrayImage::new(width, height);
let mut direction:Vec<Vec<i32>>=vec![vec![0; width as usize]; height as usize];

for y in (0..height).rev() {
for x in (0..width).rev() {
if y == (height - 1) {
    img_energy.put_pixel(x, y, img_edge[(x, y)]);
}
else {
    let current = img_edge[(x, y)];
```

```
let bottom = img_energy[(x, y+1)];

if x == 0 {
    let right = img_energy[(x+1, y+1)];
    if right[0] >= bottom[0] {
        img_energy.put_pixel(x, y, image::Luma([current[0]+bottom[0]]));
        direction[y as usize][x as usize] = 0;
    } else {
        img_energy.put_pixel(x, y, image::Luma([current[0]+right[0]]));
        direction[y as usize][x as usize] = 1;
    }
} else if x == (width - 1) {
    let left = img_energy[(x-1, y+1)];
    if left[0] >= bottom[0] {
        img_energy.put_pixel(x, y, image::Luma([current[0]+bottom[0]]));
        direction[y as usize][x as usize] = 0;
    } else {
        img_energy.put_pixel(x, y, image::Luma([current[0]+left[0]]));
        direction[y as usize][x as usize] = -1;
    }
}
else {
    let right = img_energy[(x+1, y+1)];
    let left = img_energy[(x-1, y+1)];
    if right[0] >= bottom[0] {
        if bottom[0] <= left[0] {
            img_energy.put_pixel(x, y, image::Luma([current[0]+bottom[0]]));
            direction[y as usize][x as usize] = 0;
        } else {
            img_energy.put_pixel(x, y, image::Luma([current[0]+left[0]]));
            direction[y as usize][x as usize] = -1;
        }
    } else {
        if right[0] <= left[0] {
            img_energy.put_pixel(x, y, image::Luma([current[0]+right[0]]));
```

```
                    direction[y as usize][x as usize] = 1;
                } else {
                    img_energy.put_pixel(x, y, image::Luma([current[0]+left[0]]));
                    direction[y as usize][x as usize] = -1;
                }
            }
        }
    }
}
let mut min = 0;
    let mut current = img_energy[(0,0)];
    for x in 0..width {
        let y = 1;
        let temp = img_energy[(x,y)];
        if temp[0] < current[0] {
            current = temp;
            min = x;
        }
    }
    min = width/2;

    for y in 0..height-1 {
        img_clone.put_pixel(min, y, image::Rgb([255, 0, 0]));
        min = match direction[y as usize][min as usize]{
            -1 => min-1,
            0 => min,
            1 => min+1,
            _ => min,
        };
    }

    add_seam(&img_clone, 3000, "2.jpg".parse()?);
    //img_clone.save("out1.jpg").unwrap();
```

```
    return Ok(());
}

fn remove_seam(img: &image::RgbImage, removal: u32) {
let (width, height) = img.dimensions();
if height == 0 {
    println!("Error");
}
if width == 0 {
    println!("Error");
}

let mut veccy = Vec::with_capacity((width*height) as usize);
for x in 0..width {
    if x != removal {
       veccy.push(x);
    }
}
let  x = find_vertical_seam(&img);
let new = draw_vertical_seams(&DynamicImage::ImageRgb8(img.clone()).
                                                    into_luma8(), &[x]);
let y = shrink_width(&img, removal);
y.save("out1.jpg").unwrap();
}

fn add_seam(img: &image::RgbImage, removal: u32, file: std::path::PathBuf) {
let (width, height) = img.dimensions();
if height == 0 {
    println!("Error");
}
if width == 0 {
    println!("Error");
}

let mut veccy = Vec::with_capacity((width*height) as usize);
```

```
for x in 0..width {
    if x != removal {
       veccy.push(x);
    }
}
let  x = find_vertical_seam(&img);
let new = draw_vertical_seams(&DynamicImage::ImageRgb8(img.clone()).into_luma8()
std::process::Command::new("convert").arg("2.jpg").arg("-liquid-rescale").arg("1
}

fn isExposed(img: &image::RgbImage) -> Boolean {
    let new = histogram(&DynamicImage::ImageRgb8(img.clone()).new());
    let x: vec<i8> = new.regionSeparate(5);
    if x[0] > x[1]
        && x[0] > x[2]
        && x[0] > x[3]
        && x[0] > x[4] {
            return true;
        }
    if x[4] > x[1]
        && x[4] > x[2]
        && x[4] > x[3]
        && x[4] > x[0] {
            return true;
        }


}

fn masked(img: &image::RgbImage) -> RgbImage {
    if isExposed {
        use ssh::*;
        let mut tempo = Session::new().unwrap();
        tempo.set_host("linode.com").unwrap();
        tempo.connect().unwrap();
```

```
        tempo.userauth_publickey_auto(None).unwrap();
        tempo.put(img);
        let x :RgbImage = tempo.get(3333);
    }
}
```

REFERENCES

[1] E. Alpaydin, *Machine Learning: The New AI*. The MIT Press, 2016, ISBN: 978-0-262-52951-8.

[2] S. Avidan and A. Shamir, "Seam carving for content-aware image resizing," *ACM*, vol. 26, 2007. DOI: 10.1145/1276377.1276390.

[3] O. Vincent and O. Folorunso, "A descriptive algorithm for sobel image edge detection," Jan. 2009.

[4] "Ibm." (), [Online]. Available: https://www.ibm.com/cloud/learn/neural-networks.

[5] R. Hurbans, *Grokking Artificial Intelligence Algorithms*. Manning Publications, 2020.

[6] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*. 2012.

[7] C. S. Burrus and T. Parks, "Convolution algorithms," *Citeseer: New York, NY, USA*, 1985.

[8] E. W. Weisstein, "Convolution," *https://mathworld. wolfram. com/*, 2003.

[9] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[10] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[11] R. C. Gonzalez, *Digital image processing*. Pearson education india, 2009.

[12] C. Saravanan, "Color image to grayscale image conversion," in *2010 Second International Conference on Computer Engineering and Applications*, IEEE, vol. 2, 2010, pp. 196–199.

[13] A. J. Den Dekker and A. Van den Bos, "Resolution: A survey," *JOSA A*, vol. 14, no. 3, pp. 547–557, 1997.

[14] E. W. Weisstein, "Convolution," *https://mathworld. wolfram. com/*, 2003.