

Heaven's Light is Our Guide



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Rajshahi University of Engineering & Technology, Bangladesh

Surface Crack Detection Using Deep Convolutional Neural Network

Author

Sadman Sakib Radh
Roll No. 1503007

Department of Computer Science & Engineering
Rajshahi University of Engineering & Technology

Supervised by

Prof . Dr. Md. Al Mamun
Professor

Department of Computer Science & Engineering
Rajshahi University of Engineering & Technology

ACKNOWLEDGEMENT

At first, I would like to thank the Almighty Allah for giving us the opportunity and enthusiasm along the way for the completion of our thesis work.

We would like to express our sincere appreciation, gratitude, and respect to our supervisor Prof. Dr. Al Mamun, Professor of Department of Computer Science and Engineering, Rajshahi University of Engineering and Technology, Rajshahi. Throughout the year he has not only given us technical guidelines, advice and necessary documents to complete the work he has also given us continuous encouragement, advice, help and sympathetic co-operation whenever he deemed necessary. His continuous support was the most successful tool that helped us to achieve our result. Whenever we were stuck in any complex problems or situation he was there for us at any time of the day. Without his sincere care, this work has not been materialized in the final form that it is now at the present.

I am also grateful to all the respective teachers of Computer Science and Engineering, Rajshahi University of Engineering and Technology, Rajshahi for good & valuable suggestions and inspirations from time to time.

Finally, I convey my thanks to my parents, friends, and well-wishers for their constant inspirations and many helpful aids throughout this work.

December 6, 2020

RUET,Rajshahi

Sadman Sakib Radh

Heaven's Light is Our Guide



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Rajshahi University of Engineering & Technology, Bangladesh

CERTIFICATE

*This is to certify that this thesis report entitled “**Surface Crack Detection using Deep Convolutional Neural Network**” submitted by **Sadman Sakib Radh, Roll: 1503007** in partial fulfillment of the requirement for the award of the degree of Bachelor of Science in Computer Science & Engineering of Rajshahi University of Engineering & Technology, Bangladesh is a record of the candidate own work carried out by him under my supervision. This thesis has not been submitted for the award of any other degree*

Supervisor
Examine

External

Prof. Dr. Md. Al Mamun
Professor
Department of Computer Science &Engineering
Rajshahi University of Engineering &Technology
Rajshahi-6204

ABSTRACT

Crack detection has a major importance for justifying structural health of buildings and roads. As Bangladesh is a densely populated country with a large number of traffic, it is very important to monitor the roads correctly and reduce the chance of accidents and other uncertainty both in roads and buildings. But the task is challenging for computer vision method as cracks have only low-level features for detection which are easily confused with background texture and irregularities in construction. Also the manual process of crack detection is unfeasible and time consuming. And the result may not also be accurate all the time. That's why we are proposing a Machine Learning based algorithm, Deep Convolutional Neural Networks(CNN) for detecting cracks of roads as well as buildings as Convolutional Neural Network is a promising method with higher accuracy and precision. Quantitative evaluation conducted on a data set of 4000 images of size 227×227 pixels, are classified in two classes based on the crack and non-crack images. The images are reshaped, rescaled to feed the algorithm while necessary. Throughout the paper we will discuss about the comparative study of CNN for detecting cracks.

Key words: Crack detection, Image processing, Deep Convolutional Neural Network

CONTENTS

	Page No.
ACKNOWLEDGEMENT	i
CERTIFICATE	ii
ABSTRACT	iii
CHAPTER 1	
Introduction	1-5
1.1 Introduction	1
1.2 Motivation	2
1.3 Literature Review	2
1.4 Thesis Contribution	4
1.5 Thesis Organization	5
CHAPTER 2	
Background Study	6-14
2.1 Computer Vision	6
2.2 Object Recognition	7
2.2.1 An Example	8
2.2.2 Challenges	8
2.2.3 The Image Classification Pipeline	10
2.3 Linear Classifier	11
2.3.1 Score Function	11
2.3.2 Loss Function: Softmax Function	12
2.3.3 One Hot Encoding	13
2.2 Conclusion	14
CHAPTER 3	
Neural Networks (NN)	15-26
3.1 NN	15

3.2 Biological Motivation and Connections	15
3.3 Neural Network Architectures	17
3.4 Data Preprocessing	18
3.5 Weight Initialization	19
3.6 Regularization	19
3.7 Loss Function	21
3.8 Learning	22
3.8.1 Backpropagation	22
3.8.2 Gradient Checks	24
3.8.3 Learning Rate	25
3.8.4 Per-Parameter Adaptive Learning Rate Methods	25
3.9 Conclusion	26

CHAPTER 4

Convolutional Neural Networks (CNN)	27-39
4.1 CNN	27
4.2 CNN Architecture	28
4.3 Layers in CNN	29
4.3.1 Convolutional Layer	30
4.3.2 Activation Layer	33
4.3.3 Pooling Layer	35
4.3.4 Fully-Connected (FC) Layer	37
4.4 Known Architectures in CNN	38
4.5 Conclusion	39

CHAPTER 5

Methodology	40-48
5.1 Data Source	40
5.2 Data Preprocessing	42
5.2.2 Image Resize	43
5.2.3 Normalization	43
5.3 Model Architecture	43
5.3.1 Features of VGG16	44

5.3.2 Implementation of The Architecture	45
5.3.3 Training The Model	45
5.4 Conclusion	48
CHAPTER 6	
Results and Performance Analysis	49-52
6.1 Environment	49
6.2 Experimental Analysis	49
6.3 Conclusion	52
CHAPTER 7	
Conclusion and Future Works	53-54
7.1 Summary	53
7.2 Limitations	53
7.3 Future Work	53
7.4 Conclusion	54
REFERENCES	55-60

LIST OF TABLES

Table Number	Table Title	Page No.
5.1	Model Summary	47
6.1	Performance of Different Methods	48

LIST OF FIGURES

Figure Number	Figure Caption	Page No.
2.1	Computer Vision Tasks	7
2.2	Object Recognition of Different Class	7
2.3	An Example of Object Recognition	8
2.4	Challenges in Object Recognition	9
2.5	CIFAR 10 Dataset	10
2.6	An Example of a Linear Classifier to Failing Predict Correct Class	12
2.7	Linear Classifier for High Dimensional Points	12
2.8	A One Hot Encoding Example	14
3.1	A Biological Neuron (top) and a Common Mathematical model (bottom)	16
3.2	An example of a 3-layer neural network	17
3.3	Preprocessing The Data	18
3.4	Dropout Used in a Network	21
3.5	The Change to a Hidden to Output Weight Depends on error	22
3.6	Loss Function is Changing with Different Learning Rates	25
4.1	Comparison of an Original Neural Network with a CNN	29
4.2	A Typical CNN Architecture	30
4.3	ConvNet Architecture Example	30
4.4	Convolution layer example	31
4.5	Example of local connectivity	32

4.6	Activation function: tanh	35
4.7	Activation function: ReLU	35
4.8	Example of maxpooling	36
5.1	Crack images	41
5.2	Non Crack Images	42
5.3	VGG16 Architecture	44
6.1	Accuracy Curve for training and validation	50
6.2	Accuracy Curve for training and validation loss	50

CHAPTER 1

Introduction

This chapter begins with the motivation behind this thesis topic. Then the literature reviews are discussed right after. Then in the proposed methodology section, the proposed system is described briefly which will solve the problem that is being dealt with. Then, in thesis contribution, contributions of the thesis are outlined. Finally, the chapter ends with a conclusion.

1.1 Introduction

Crack detection refers to the process of detecting defects in impermeable materials or surfaces like metals, concrete, ceramics etc. It is very crucial in the inspection and maintenance of concrete as well as roads structures. Traditional approach for crack detection is visual inspection by a trained inspector who evaluates the condition of any surface according to the location and width of a crack. This manual process is very time consuming and is also unfeasible since a building or road may have many cracks at a certain time. And the detection result may not also be accurate all the time.

The image-based crack detection system is thoroughly analyzed in view of the human vulnerability of the inspection. Image processing techniques identify cracks from images based on some assumptions that the cracks are slender and connected regions and darker than their backgrounds [1]. Cracks are typically segmented using the proper threshold [2]. General global transformations and local edge detections are used for more robustness of crack detection, [3-5] such as fast Haar transform, fast Fourier transform, Sobel and Canny edge detectors, etc. Fortunately, this method relies heavily on well-chosen techniques for image preprocessing and image edge detection. And the elements on the surface of the structure are complex and influenced by several real-life influences, such as light, shadows, etc.

Machine learning-based techniques are used to more reliably distinguish cracks from photos [6]. Artificial neural networks, supervised machine learning algorithms, are used to classify images that are with or without cracks [7]. In reality, however, only basic architectures of artificial neural networks can be used to detect cracks due

to the constraint of computational capacity. In recent years, deep CNNs have been highlighted in image recognition[9], thanks to the advancement of deep learning and parallel computations using graphic processing units (GPUs) [8].

In comparison to traditional neural networks, due to the partial ties, exchanging weights and pooling mechanism between neurons, CNNs classifying images rely on less computations. Notably, the CNN infrastructure needs to be developed, and a database containing a large number of images should be built to train CNN[10]. With the prevalence of smartphones, smartphones have been used as instruments for tracking the health of structures[11].

In this paper, a deep CNN network is applied as a classifier for detecting surface crack. The CNN based crack detection can verify cracks from images and the detection result will not be affected by noise on surface or road images.

1.2 Motivation

Technological development is going on day by day. The old one is being replaced by the new one. Machine learning based technology is now getting higher popularity. Effective use of such technology is seen in many areas like medical, face recognition etc. As it is a vital issue for safe driving in highways, it is important to ensure the roads are safe. So we need more accuracy with less computational time. And this is only possible through machine based approach like CNN or other models. Some works have been done in this sector and those works have motivated us to develop a classification system that can classify surface crack.

1.3 Literature Review

The section is about reviewing some existing works that are related to crack detection. There have been various studies on the road-traffic conditions, causes and their effects in the socio-economic, physical and mental health the people of Bangladesh [12][13][14]. For pattern recognition, image processing, object identification, semantic and instance segmentation, CNN based machine learning models are very efficiently used. So far, the “AlexNet” architecture introduced by Krizhevsky Et al. [15] has been successfully applied in a number of computer vision

tasks, for example in object detection [6], object tracking [16], segmentation, [17] video classification, [18] human pose estimation [19] and super resolution [20]. Classical deep Neural Networks like CNN and FCN extract image features finding key points and thus are used for 2D image classification. On the other hand, Google Introduced their own residual deep neural network architecture in Inception-v1, they have optimized the network furthermore through Inception-v3 to reduce computational cost [19][20]. The basic difference between Inception-v3 and Inception-v4 is that the latter one has a simpler architecture and more inception modules [21]. The constraints contained in Inception-v3 had come from the need for partitioning model architecture with distributed training using DistBelief [22]. With migration to TensorFlow [23] those constraints have been lifted which allowed the reintroduced architecture to get simplified.

The goal of this research is to classify crack of any surface or road by using deep convolutional neural network and maximizing the accuracy. 40,000 of crack and non crack images are used for this network. Among them 20,000 are positive (crack) and rest 20,000 are negative (non crack). CNN has been used successfully in 2D image recognition and research centered on image processing such as: Food Classification [24] and Gaussian noise detection [25]. For this reason, CNN is used to classify crack images. In this analysis, the qualified model evaluated the pixel strength adjustment for the presence of holes, dumps, surface water and change in road color for hilly tracts and change in soil.

CNN based researches have been performed to detect road, semantic segmentation of road scenes [26-30], road-lane, road area extraction [31], rural roads detection [32], street signs detection and so on. The works are most commonly based on classification, object detection or semantic segmentation. For image recognition and object-detection based works some of the novel approaches have been introduced by Lin et al. [33], Simonyan et al. in VGGNet [34] and GoogleNet (Inception - v1) [35] by Szegedy Et al. Residual connections were introduced in [14]. In civil engineering ventures, deep convolutionary neural network approaches such as bounding box approaches in computer vision-based models have been suggested for surface health monitoring. A model of pothole identification was also proposed[36] based on semantic segmentation on concrete crack images, which is also a fully

connected network (FCN) solution. Several semantic image segmentation models have been introduced there by [37-41].

1.4 Thesis Contribution

- i. An automated system is created using pre-trained Convolutional Neural Network (CNN).
- ii. The system has the ability to distinguish between crack and non-crack images if it is trained with a large enough dataset.
- iii. In the training process, it learns the features of different types of crack and then classifies with the learned knowledge.
- iv. This model can be used in civil engineering sectors.
- v. The model leaves opportunities for further improvement and it can be modified for even better results in the future.

1.5 Thesis Organization

The rest of the thesis is organized as follows:

Chapter 2 - Background Study

This section describes what computer vision and object recognition are and what the challenges and overall computer vision pipeline are. It also describes linear classifier.

Chapter 3 - Neural Networks

This chapter discusses the architecture, data preprocessing, weights initialization, loss function, regularization and how learning is achieved in neural networks.

Chapter 4 - Convolutional Neural Network

This chapter discusses the architecture, layers and some known popular convolutional neural network models available right now. It also distinguishes the differences between an ordinary neural network and a convolutional neural network.

Chapter 5 - Methodology

This chapter describes about the datasets, data preprocessing, the whole work flow of the developed architecture.

Chapter 6 - Result and Performance Analysis

This chapter describes the datasets that are used and the overall result of the developed architecture. It also shows the performance of the architecture.

Chapter 7 - Conclusion and Future Work

This chapter concludes the thesis, describes its limitation and shows a direction of future work.

1.6 Conclusion

Detecting crack is a vital issue in civil engineering. As we are a densely populated country with a large number of vehicles we need to keep the roads safe. The thesis introduces an automated system which can solve this problem.

CHAPTER 2

Background Study

This chapter starts by describing briefly what computer vision is and then later gives an introduction, method and challenges of object recognition. In the latter part of the chapter, the linear classifier for recognizing objects have been discussed. As the research is based on recognizing crack in surfaces this part is pretty significant to the research work.

2.1 Computer Vision

Computer vision is an interdisciplinary field that deals with how computers can be made for gaining high-level understanding from digital images or videos. High level understanding includes recognizing what the object is (e.g. dog, cat, car etc.), semantic segmentation, classifying using localization to detect a single object in a given scene, instant segmentation and many other tasks depicted in figure 2.1. Computer vision is concerned with the theory behind artificial systems that extract information from images. It seeks to automate tasks that an animal visual system can do. The target behind computer vision is to create such a system that would be as efficient in recognizing objects and do further actions depending on what kind of object as the human brain sees and does. Almost 50% neurons of a human's cortex are involved in visual processing [42]. Computer vision aims to mimic the working principles of visual processing in computer processors.

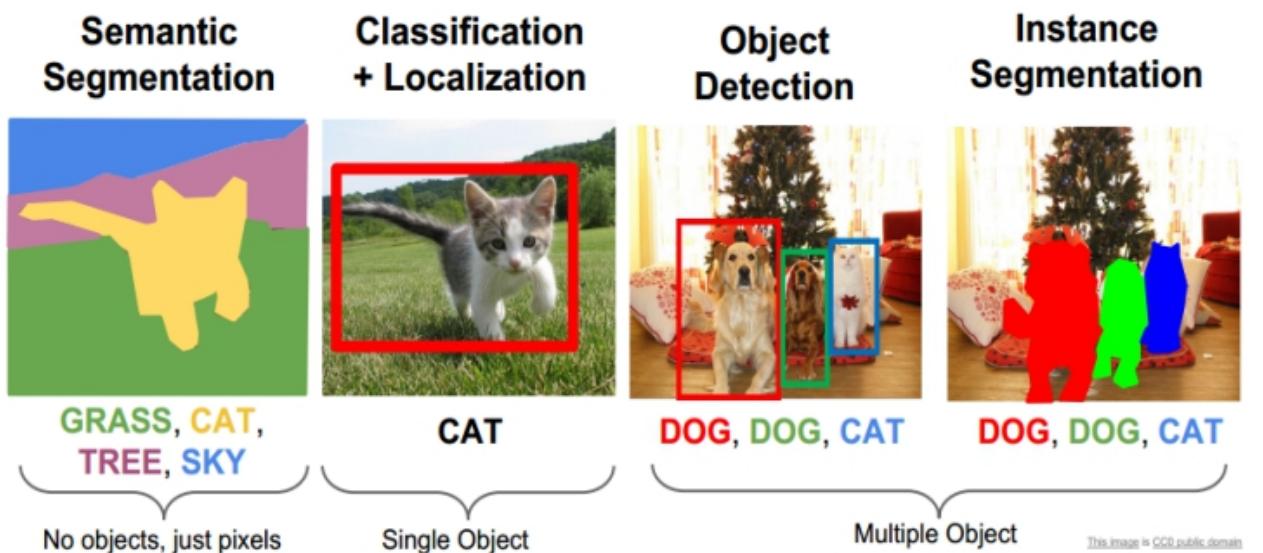


Figure 2.1: Computer Vision Tasks [42]

2.1 Object Recognition

Object recognition is the task of assigning an input image one label from a fixed set of categories. It creates a model that most accurately classifies objects in the given images of the dataset. This is one of the core problems in Computer Vision that, despite its simplicity, has a large variety of practical applications. Figure 2.2 depicts the recognition of different objects.



Figure 2.2: Object recognition of different class objects [42]

2.2.1 An Example

In the image of figure 2.3 an object recognition model takes a single image and assigns probabilities to 4 labels, {cat, dog, hat, mug}. As shown in the image, an image is represented as one large 3-dimensional array of numbers. In this example, the cat image is 248 pixels wide, 400 pixels tall, and has three color channels Red, Green, Blue (or RGB). Therefore, the image consists of $248 \times 400 \times 3$ numbers or a total of 297,600 numbers. Each number is an integer that ranges from 0 (black) to 255 (white). The task will be to turn this quarter of a million numbers into a single label, such as “cat”.

The task in object recognition is to predict a single label (or a distribution over labels as shown here to indicate our confidence) for a given image. Images are 3-dimensional arrays of integers from 0 to 255, of size Width x Height x 3. The 3 represents the three-color channels Red, Green, Blue.

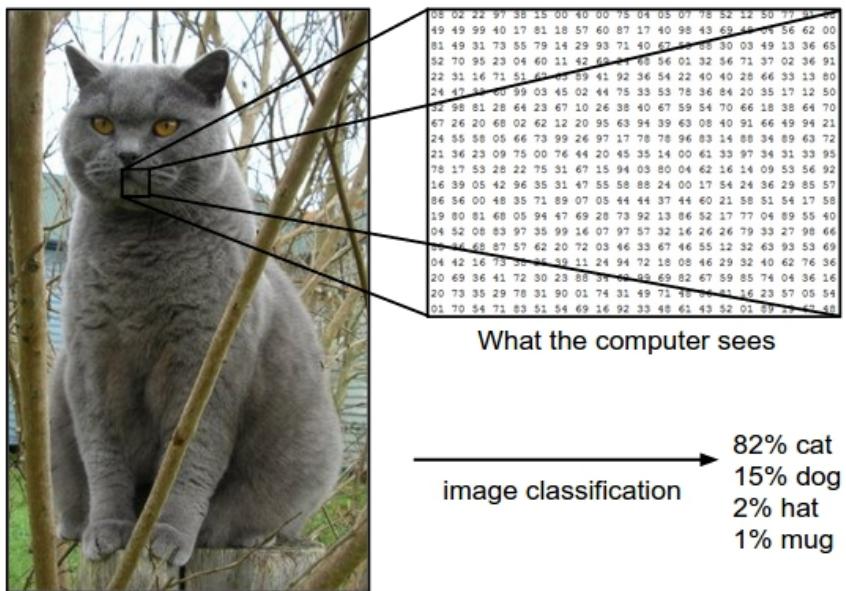


Figure 2.3: An example of object recognition [42]

2.2.2 Challenges

Since this task of recognizing a visual concept (e.g. cat) is relatively trivial for a human to perform, it is worth considering the challenges involved from the

perspective of a Computer Vision algorithm. A list of challenges for the computer in the raw representation of images as a 3-D array of brightness values:

- **Viewpoint variation.** A single instance of an object can be oriented in many ways with respect to the camera.
- **Scale variation.** Visual classes often exhibit variation in their size (size in the real world, not only in terms of their extent in the image).
- **Deformation.** Many objects of interest are not rigid bodies and can be deformed in extreme ways.
- **Occlusion.** The objects of interest can be occluded. Sometimes only a small portion of an object (as little as few pixels) could be visible.
- **Illumination conditions.** The effects of illumination are drastic on the pixel level.
- **Background clutter.** The objects of interest may blend into their environment, making them hard to identify.
- **Intra-class variation.** The classes of interest can often be relatively broad, such as chair. There are many different types of these objects, each with their own appearance.

A good image classification model must be invariant to the cross product of all these variations, while simultaneously retaining sensitivity to the inter-class variations. Figure 2.4 shows the different challenges for a specific class.

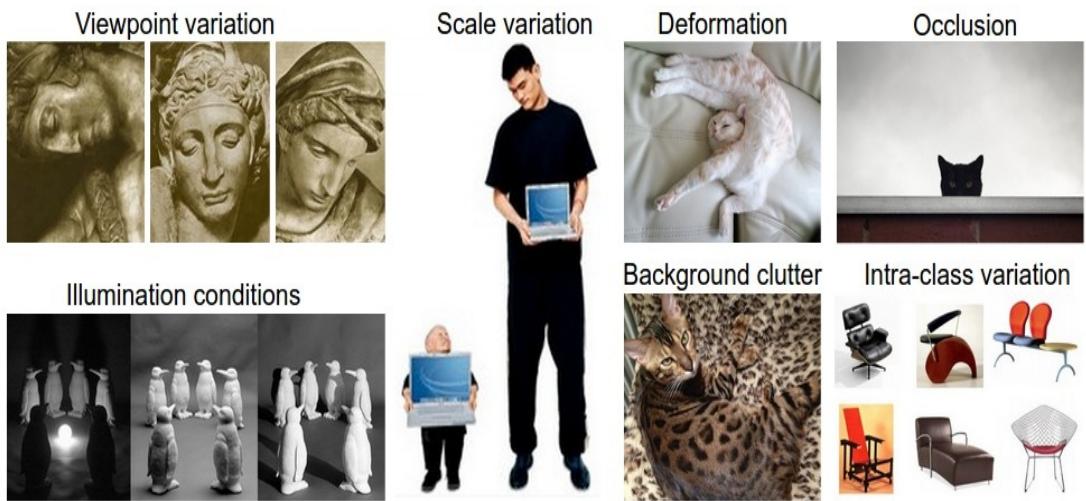


Figure 2.4: Challenges in object recognition [42]

The challenge is to write an algorithm that can classify images into distinct categories. Instead of trying to specify what every one of the categories of interest looks like directly in code, the method is to provide the computer with many examples of each class and then develop learning algorithms that look at these examples and learn about the visual appearance of each class. This approach is referred to as a data-driven approach since it relies on first accumulating a training dataset of labeled images. An example of this kind would be the CIFAR10 dataset which consists of 10 classes of 50,000 training images and 10,000 testing images shown in figure 2.5.



Alex Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", Technical Report, 2009.

Figure 2.5: CIFAR 10 dataset

2.2.3 The Image Classification Pipeline

The task in object recognition is to take an array of pixels that represents a single image and assign a label to it. The complete pipeline can be formalized as follows:

- **Input:** The input consists of a set of N images, each labeled with one of K different classes. This data is referred to as the training set.
- **Learning:** The learning uses the training set to learn what every one of the classes looks like. This step is referred to as training a classifier or learning a model.
- **Evaluation:** At the last step, the quality of the classifier is evaluated by asking it to predict labels for a new set of images that it has never seen before. Then it

will be compared with the true labels of these images to the ones predicted by the classifier.

2.3 Linear Classifier

An approach to object recognition that we will eventually naturally extend to entire Neural Networks and Convolutional Neural Networks. The approach will have two major components: a score function that maps the raw data to class scores, and a loss function that quantifies the agreement between the predicted scores and the ground truth labels.

2.3.1 Score Function

A training dataset of images $y_i \in R^D$, each associated with a label y_i . Here $i = 1...N$ and $y_i \in 1...N$. Total of N examples (each with a dimension of D) and K distinct categories. Then a linear mapping can be defined as:

$$f(x_i, W, b) = Wx_i + b \quad (1)$$

In the above equation, the image x_i has all of its pixels flattened out to a single column vector of shape $[D \times 1]$. The matrix W (of size $[K \times D]$), and the vector b (of size $[K \times 1]$) are the parameters of the function. The parameters in W are weights, and b is called the bias vector because it influences the output scores, but without interacting with the actual data x_i . The weights are essential in order to classify the object. In Figure 2.6 explains a linear classifier for an image only having 4 pixels and having only 3 classes (red (cat), green (dog), blue (ship) class). In this classifier, dog score has the highest, meaning it is classifying a cat as dog because of the weights which need to be changed to get the actual results. The next section, it is shown how a loss function changes these weights to get a better result.

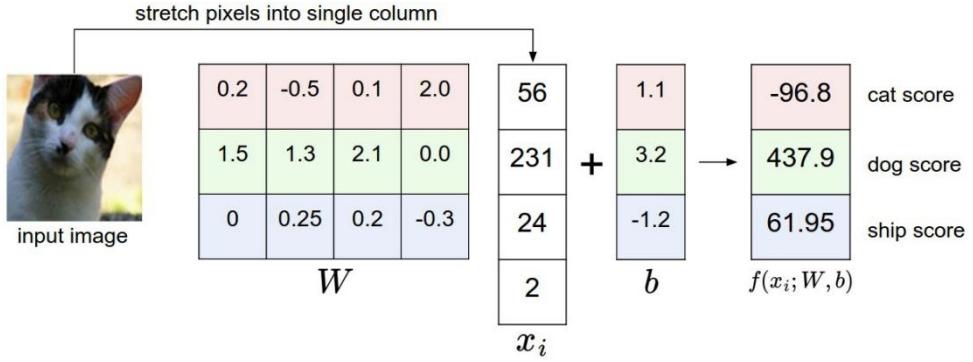


Figure 2.6: An example of a linear classifier failing to predict correct class [42]

And figure 2.7 depicts the analogy of images as high-dimensional points.

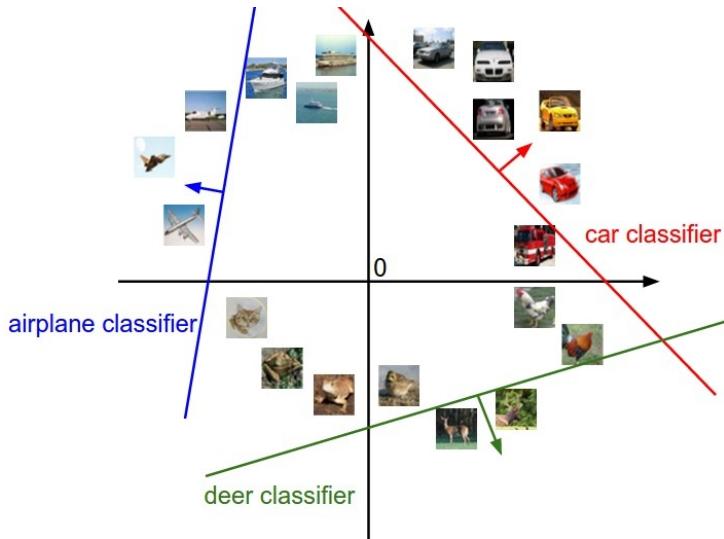


Figure 2.7: Linear classifier for high dimensional points [42]

2.3.2 Loss Function: Softmax Function

In the previous section, a function from the pixel values to class scores, which was parameterized by a set of weights W . As the data cannot be changed, the weights are changed to minimize the loss of the predicted class with the actual class. As seen in the previous section how the classifier failed to predict the correct class. The loss function will determine this outcome to predict the amount of loss in a particular prediction. The loss function softmax is discussed here as it is used in the research.

The Softmax classifier computes “probabilities” for all labels. For example, given an image, the softmax classifier compute the probabilities of the three labels as [0.9, 0.09, 0.01] for the classes “cat”, “dog” and “ship” which interpret its confidence in

each class. In the softmax classifier, we interpret the scores as the non normalized log probabilities for each class and calculate loss with a cross-entropy loss that has the form:

$$L_i = -\log \left(\frac{e^{f_{y_j}}}{\sum_j e^{f_j}} \right) \quad (2)$$

The notation f_j to means the j th element of the vector of class scores f . The full loss for the dataset is the mean of L_i overall training examples together. The function $f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$ is called the softmax function: It takes a vector of arbitrary real-valued scores (in z) and squashes it to a vector of values between zero and one that sum to one. The cross-entropy between a “true” distribution p and an “estimated” distribution q is defined as:

$$H(p, q) = -\sum p(x) \log q(x) \quad (3)$$

The Softmax classifier is hence minimizing the cross-entropy between the estimated class probabilities ($q = \frac{e^{f_{y_j}}}{\sum_j e^{f_j}}$ as seen above) and the “true” distribution, which in this interpretation is the distribution where all probability mass is on the correct class (i.e. $p = [0, \dots, 1, \dots, 0]$ contains a single 1 at the y_i position.).

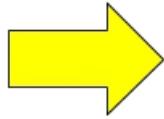
2.3.2 One Hot Encoding

Many machine learning algorithms cannot work directly on label knowledge. They include the numericity of both input variables and output variables. In general, this is often a restriction on the implementations themselves of the effective execution of machine learning algorithms rather than hard constraints. That implies that it is important to transform categorical data into a numerical form. If the categorical variable is an output variable, in order to present it or use it in any application, we may also want to transform model predictions back into a categorical type.

One hot encoding[43] is a method by which categorical variables are translated into a form that could be provided to machine learning algorithms in order to achieve better prediction results. In order to perform "binarization" of the type, we use one hot encoder and include it as a function for model training. The first condition is for integer values to be converted to categorical values. Each integer value is then

represented as a binary variable, all of which are zero values, with the exception of the integer index marked with 1.

New (binary) columns are generated by one hot encoding, showing the inclusion of each potential value from the original data. Figure 2.8 gives an example of one hot encoding, with the values in the original data being Red, Yellow, and Green. We create a separate column for each possible value. Wherever the original value was Red, we put a 1 in the Red column.



Color	Red	Yellow	Green
Red	1	0	0
Red	1	0	0
Yellow	0	1	0
Green	0	0	1
Yellow			

Figure 2.8: A one hot encoding example [44]

2.4 Conclusion

Object recognition is a part of a greater field i.e. computer vision which the process of imitating the vision system of humans. A good classifier is needed for the computer to classify objects with the minimum of loss from the actual results. While working with categorical data, we need a way to turn this into a numerical value.

CHAPTER 3

Neural Networks (NN)

This chapter discusses different sections of the neural network. Neural network's motivation and its architecture are briefly explained here. The latter part discusses how the model gets to build and how it is learned through the process.

3.1 NN

Also for the human visual system, image recognition based on any basic visual details has always been a hard challenge. Owing to their complicated geometric form as well as the inter-intraclass heterogeneity in the images, they are much more difficult to identify because of the microscopic images from histopathological sections[44]. A deep learning architecture is a trainable neural network. This was promoted by the animal optical system[45] that explicitly learns features from the input data presented, eliminating the hand-crafted function. In several steps, it consists of several nonlinear transformations and works to reduce the error to be correctly classified in each step.

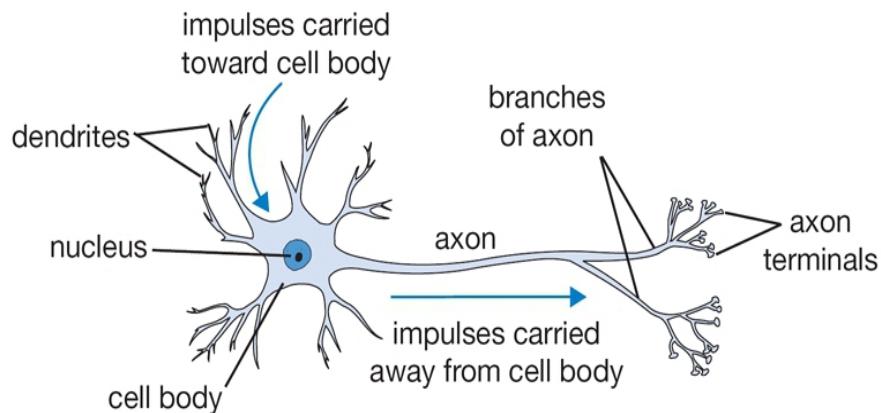
3.2 Biological Motivation and Connections

To develop an artificial intelligence, through using the combination of basic computational components that approximate the operating process of a human brain, researchers developed a highly integrated device. Researchers originally sought to mimic the neurophysiology of the brain[45] and these components acted as the neurons of the brain. As modern neural networks are now combined with numerous methods of numerical analysis, they will make predictions about various real-world problems[45]. A neuron is the fundamental computational unit of the brain. In the human nervous system, approximately 86 billion neurons can be identified and are connected to approximately $10^{14} - 10^{15}$ synapses[45]. Figure 3.1 shows a drawing of a biological neuron (top) and a common mathematical model (bottom).

From its dendrites, each neuron receives input signals and generates output signals along its (single) axon. Eventually, the axon branches out and binds to other neurons' dendrites through synapses. In the computational model of a neuron, the

signals that travel along the axons (e.g. x_0) interact multiplicatively (e.g. w_0x_0) with the dendrites of the other neuron based on the synaptic strength at that synapse (e.g. w_0).

The theory is that the synaptic strengths (weights w) are learnable and that the force of one neuron's effect on another is regulated. The dendrites take the signal to the cell body in the fundamental model, where they are all rounded up. The neuron may fire if the final amount is above a certain level, sending a spike down its axon. We believe in the theoretical model that the accurate timings of the spikes do not matter and that information is only transmitted by the frequency of the firing. We model the firing rate of the neuron with an activation function f , which describes the frequency of the spikes along the axon, on the basis of this rate code understanding. Historically, the sigmoid function σ is a typical activation function preference, since it takes a real-valued input (the signal intensity after the sum) and squashes it to a range between 0 and 1. Thus, with the input and its weights, each neuron performs a dot product, adds the bias and applies the non-linearity (or activation function).



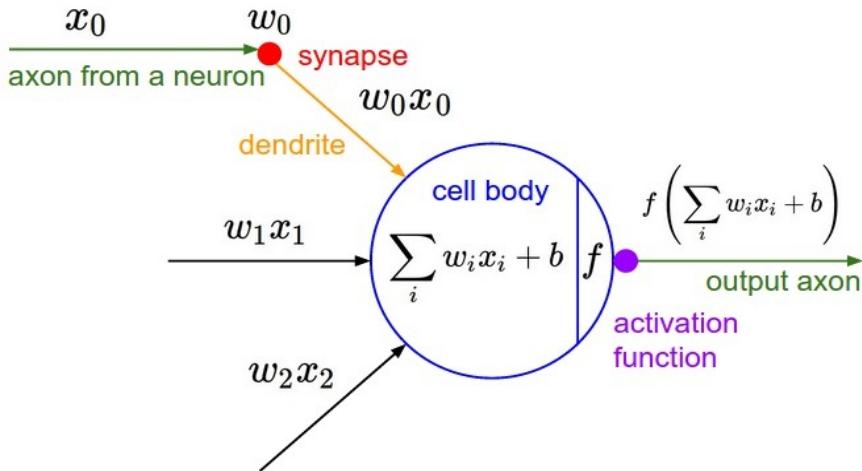


Figure 3.1: A biological neuron (top) and a common mathematical model (bottom)

[42]

3.3 Neural Network Architectures

In an acyclic graph, neural networks are modeled as neuron collections that are connected. In other words, certain neurons' outputs can become inputs to other neurons. Cycles are not permitted and an infinite loop in a network's forward pass would mean that. Versions of the Neural Network are also structured into individual layers of neurons instead of an amorphous blob of connected neurons. The most common layer type for standard neural networks is the fully-connected layer in which nodes are fully pairwise connected between two adjacent layers, but neurons share no connections within a single layer. Figure 3.2 shows a 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer. In both cases there are connections (synapses) between neurons across layers, but not within a layer.

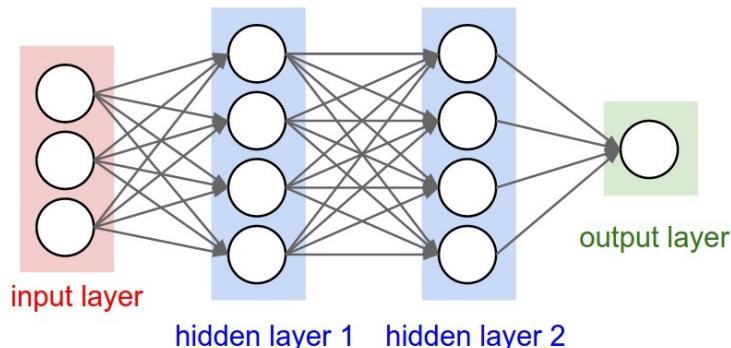


Figure 3.2: An example of a 3-layer neural network [42]

3.4 Data Preprocessing

In neural networks, some computer preprocessing methods are used to turn raw data into a comprehensible format. The most popular method of preprocessing is Mean Subtraction. It requires subtracting the mean across each of the data's individual features and has the geometric interpretation of centering the data cloud in any axis from the origin.

Normalization refers to the normalization of the proportions of the data such that they are exactly the same height. Two common ways of achieving this normalization are available. One is to divide each dimension until it has been zero-centered, by its standard deviation. Each dimension is normalized by another method of this preprocessing, such that the min and max along the dimension are -1 and 1 respectively. It is only used when there are different scales (or units) of different input features, but they should be of about equal importance to the learning algorithm.

It is only used when there are different scales (or units) of different input features, but they should be of about equal importance to the learning algorithm. Figure 3.3 shows this process in three steps. the red lines in the figure indicate the extent of the data - they are of unequal length in the middle, but of equal length on the right.

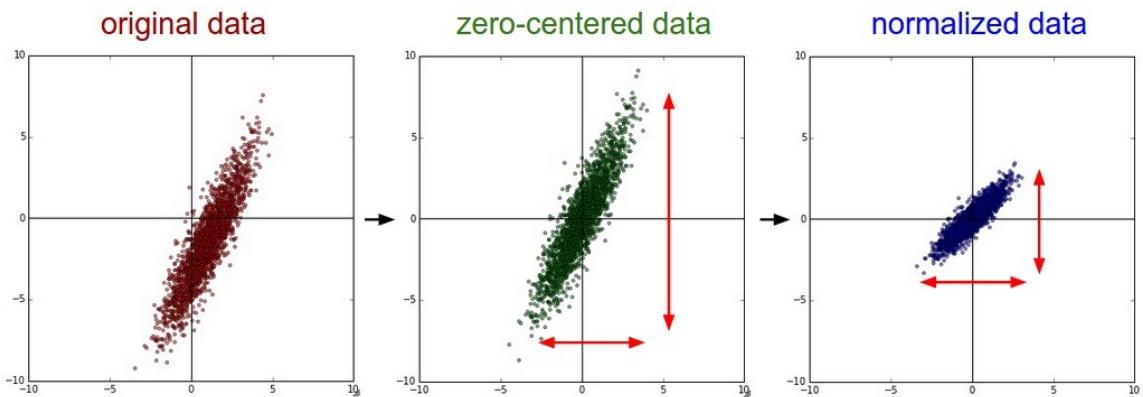


Figure 3.3: Preprocessing the data [42]

3.5 Weight Initialization

The convention is to initialise its parameters before training the network. Setting all initial weights to zero, which is the assumption of "best guess" in expectation, may be a reasonable-sounding idea. But if every neuron in the network computes the same

output, then during backpropagation they will also all compute the same gradients and undergo the exact same parameter updates. In other words, if their weights have been initialized to be the same, there is no cause of asymmetry between neurons. As a solution, initializing the weights of the neurons to small numbers is usual and refers to doing so as breaking symmetry. The idea is that at the beginning, the neurons are all random and distinct, so they will compute different updates and integrate themselves as diverse components of the entire network. With this formulation, the weight vector of a neuron is initialized as a random vector sampled from a multi-dimensional Gaussian, so the neurons point into the input space in the random direction. Since the asymmetry break is given by the small random numbers in the weights, it is possible and normal to initialize the biases to be nil.

The he_uniform initializer [46] method for weight initialization is an important tool for initializing the weights which are used in this research. The he_uniform initialization makes sure the weights are appropriate, keeping the signal in a reasonable range of values through many layers. It tries to ensure zero mean and unit variance is the distribution of the inputs to each activation function. It implies that the input data has been normalized to the same distribution in order to do this. In order to compensate for the amount of inputs, the lower the initial weights should be, the greater the number of inputs a neuron has. In a word, the he_uniform initialization method tries to initialize weights with a smarter value, such that neurons won't start training in saturation.

3.6 Regularization

Some regularization techniques are implemented in neural networks to minimize overfitting. L2 regularization is one of them. Perhaps the most common type of regularization is L2 regularization. It can be enforced by explicitly penalizing the square magnitude of all parameters in the target. For every weight w in the network, the term $\frac{1}{2}\lambda w^2$ is added to the objective, where λ is the regularization strength. It is common to see the factor of $\frac{1}{2}$ in front because then the gradient of this term with respect to the parameter w is simply λw instead of $2\lambda w$. The L2 regularization has the intuitive interpretation of heavily penalizing peaky weight vectors and preferring diffuse weight vectors.

L1 regularization is another relatively common form of regularization, where for each weight w we add the term $\lambda|w|$ to the objective. It is possible to combine the L1 regularization with the L2 regularization: $\lambda_1 w + \lambda_2 w^2$ (this is called Elastic net regularization [47]). The L1 regularization has the intriguing property that it leads the weight vectors to become sparse during optimization (i.e. very close to exactly zero). In other words, neurons with L1 regularization end up using only a sparse subset of their most important inputs and become nearly invariant to the “noisy” inputs. In comparison, final weight vectors from L2 regularization are usually diffuse, small numbers.

Another method of regularization is to impose an absolute upper limit for each neuron on the magnitude of the weight vector and use the predicted gradient descent to enforce the limitation. This is known as the restrictions of max norm.

Dropout is an extremely effective, simple and recently introduced regularization technique [48]. During preparation, dropout is applied by only having a neuron alive with a certain probability (a hyperparameter), or otherwise setting it to zero. During preparation, it is only feasible to view Dropout as sampling a Neural Network within the complete Neural Network and to change the sampled network parameters depending on the input data. (However, the exponential number of possible sampled networks are not independent because they share the parameters.) There is no dropout implemented during research, with the interpretation of measuring an average estimate across all sub-networks' exponentially-sized ensemble. Figure 3.4 illustrates the dropout idea where cross means that the neurons are set to zero (randomly).

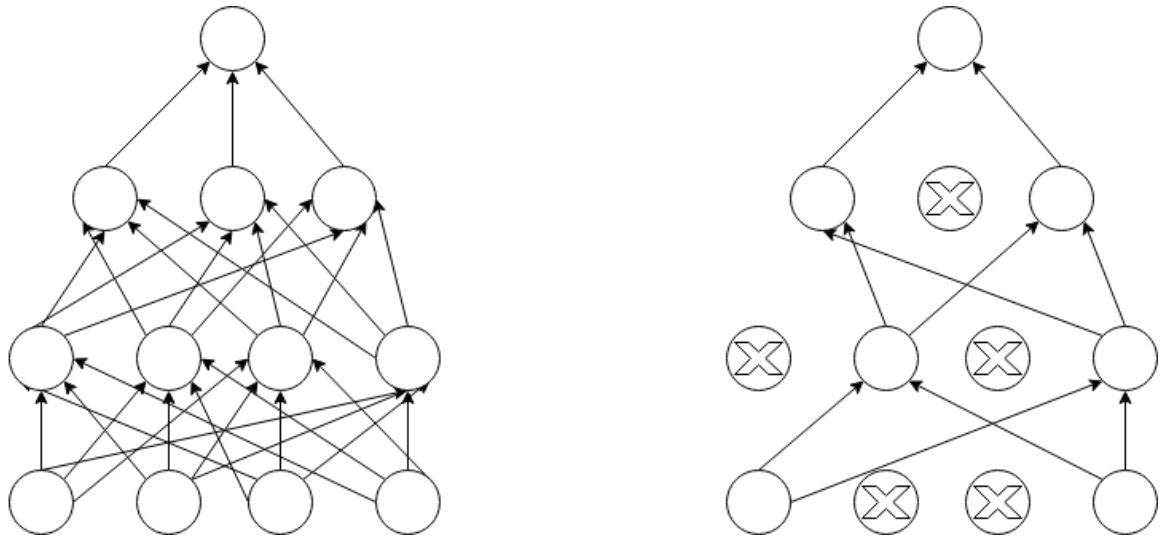


Figure 3.4: Dropout used in a network

3.7 Loss Function

The data loss, which is a supervised learning problem measures the compatibility between a prediction (e.g. the class scores in classification) and the ground truth label. The data loss takes the form of an average over the data losses for every individual example. That is, $L = \frac{1}{N} \sum L_i$ where N is the number of training data and $f = f(x_i; W)$.

If the correct output y_i is a binary vector where every example may or may not have a certain attribute. In this case, the way is to build a binary classifier for every single attribute independently. For example, a binary classifier for each category independently would take the form:

$$L_i = \sum \max(0, 1 - y_{ij}f_j) \quad (4)$$

where the sum is over all categories j , and y_{ij} is either +1 or -1 depending on whether the i th example is labeled with the j th attribute, and the score vector f_j will be positive when the class is predicted to be present and negative otherwise. Notice that loss is accumulated if a positive example has a score less than +1, or when a negative example has a score greater than -1.

3.8 Learning

How the neural network learns through iteration and updates its weights to minimize the loss function will now be discussed.

3.8.1 Backpropagation

When training the network there are mostly two passes which need to be completed one after another. The forward pass measures values from inputs to output, and the backward pass then performs backpropagation that begins at the end and applies the chain rule recursively to measure the gradients all the way to the circuit inputs. To learn the weight of a multilayer neural network with a fixed architecture, the Backpropagation algorithm is used. It performs gradient descent to try to minimize the sum squared error between the network's output values and the given target values.

Figure 3.5 explains (depicted as a lined pattern) at the output node and activation (depicted as a solid pattern) at the hidden node. Although the adjustment to the hidden weight input depends on the hidden node error (which in turn depends on the error on all the output nodes) and on the input node activation.

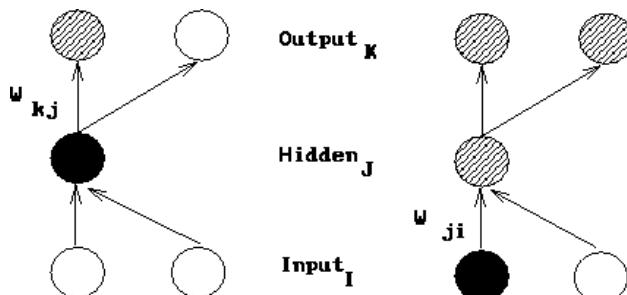


Figure 3.5: The change to a hidden to output weight depends on error [42]

For the purpose of this derivation:

- The subscript k denotes the output layer.
- The subscript j denotes the hidden layer.
- The subscript i denotes the input layer.
- W_{kj} denotes a weight from the hidden to the output layer.
- W_{ji} denotes a weight from the input to the hidden layer.

- a denotes an activation value (sigmoid function).
- t denotes a target value.
- net denotes the net input.

The total error in a network is given by the following equation:

$$E = \frac{1}{2} \sum (t_k - a_k)^2 \quad (5)$$

We want to adjust the network's weights to reduce this overall error.

$$\Delta W \propto -\frac{\partial E}{\partial W} \quad (6)$$

We will begin at the output layer with a particular weight.

$$\Delta W_{kj} \propto -\frac{\partial E}{\partial W_{kj}} \quad (7)$$

However, the error is not directly a function of a weight. We expand this as follows.

$$\Delta W_{kj} = -\epsilon \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial net_k} \frac{\partial net_k}{\partial W_{kj}} \quad (8)$$

Let's consider each of these partial derivatives in turn. Note that only one term of the E summation will have a non-zero derivative: the one associated with the particular weight we are considering.

Derivative of the error with respect to the activation:

$$\frac{\partial E}{\partial a_k} = \frac{\partial (\frac{1}{2} \sum (t_k - a_k)^2)}{\partial a_k} = -(t_k - a_k) \quad (9)$$

Derivative of the activation with respect to the net input:

$$\frac{\partial a_k}{\partial net_k} = \frac{\partial (1 + e^{-net_k})^{-1}}{\partial net_k} = \frac{e^{-net_k}}{(1 + e^{-net_k})^2} \quad (10)$$

Using the activation function's equation, we can rewrite the result of the partial derivative as:

$$a_k(1 - a_k) \quad (11)$$

Derivative of the net input with respect to a weight:

$$\frac{\partial net_k}{\partial W_{kj}} = \frac{\partial (\partial net_k a_j)}{\partial W_{kj}} = a_j \quad (12)$$

Now substituting these results back into our original equation (8), we have:

$$\Delta W_{kj} = \epsilon(t_k - a_k)a_k(1 - a_k)a_j \quad (13)$$

This equation is typically simplified as shown below where the δ term represents the product of the error with the derivative of the activation function.

$$\Delta W_{kj} = \epsilon \delta_k a_j \quad (14)$$

Now we have to determine the appropriate weight change for an input to hidden weight. This is more complicated because it depends on the error at all of the nodes this weighted connection can lead to.

$$\begin{aligned}\Delta W_{ji} &\propto \left(\sum \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial net_k} \frac{\partial net_k}{\partial a_j} \right) \frac{\partial a_j}{\partial net_j} \frac{\partial net_j}{\partial W_{ji}} \\ &= \epsilon (\sum (t_k - a_k) a_k (1 - a_k) W_{kj}) a_j (1 - a_j) a_i \\ &= \epsilon (\sum \delta_k W_{kj}) a_j (1 - a_j) a_i\end{aligned}$$

So, the final form it takes:

$$\Delta W_{ji} = \epsilon \delta_j a_i \quad (15)$$

3.8.2 Gradient Checks

Performing a gradient check is as simple as comparing the analytic gradient to the numerical gradient. The centered difference formula of the form or the finite difference approximation when evaluating the numerical gradient looks as follows:

$$\frac{df(x)}{dx} = \frac{f(x+h) - f(x-h)}{2h} \quad (16)$$

This requires us to evaluate the loss function twice to check every single dimension of the gradient (so it is about 2 times as expensive), but the gradient approximation turns out to be much more precise. To see this, Taylor expansion of $f(x + h)$ and $f(x - h)$ is used which also verifies that the first formula has an error on order of $O(h)$, while the second formula only has error terms on order of $O(h^2)$.

For comparing the numerical gradient f'_n and analytic gradient f'_a , introducing the relative error:

$$\frac{|f'_a - f'_n|}{\max(|f'_a|, |f'_n|)} \quad (17)$$

Which considers their ratio of the differences to the ratio of the absolute values of both gradients. In practice:

- relative error $> e^{-2}$ usually means the gradient is probably wrong
- $e^{-2} >$ relative error $> e^{-4}$ should be okay.
- $e^{-4} >$ relative error is usually okay for objectives with kinks (Kinks refer to non-differentiable parts of an objective function). But if there are no kinks (e.g. use of tanh nonlinearities and softmax), then e^{-4} is too high.
- e^{-7} and less should be great.

3.8.3 Learning Rate

The rate the weights will be updated is known as the learning rate for the model. The learning rate needs to be controlled as the effects of different learning rates can produce different results and the goal is to minimize the loss function. This is depicted in figure 3.5 as with low learning rates the improvements will be linear. They will start to look more exponential, with high learning rates. Higher learning rates can decrease the loss more efficiently, but they are left with poorer loss values (green line). This is because the optimization involves so much "energy" and the parameters are chaotically jumping around, unable to settle in a pleasant position in the optimization landscape.

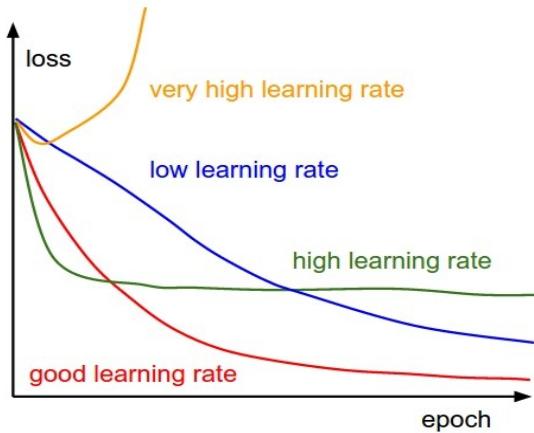


Figure 3.6: Loss function is changing with different learning rates [42]

3.8.4 Per-Parameter Adaptive Learning Rate Methods

All previous methods that we have addressed so far have manipulated the learning rate for all parameters globally and similarly. Tuning the learning rates is an expensive process, so much effort has gone into creating techniques that can change the learning rates adaptively, and even do so per parameter. Other hyperparameter settings can also require many of these techniques, but the point is that they are well-behaved for a wider variety of hyperparameter values than the raw learning rate. We highlight some common adaptive methods in this section.

Adagrad:

Adagrad is an adaptive learning rate method originally proposed by Duchi et al [49]. In the Adagrad method, the denominator accumulates the squared gradients from each iteration starting at the beginning of training. Since each term is positive, this

accumulated sum continues to grow throughout training, effectively shrinking the learning rate on each dimension. In this method, the variable cache has the size equal to the size of the gradient and keeps track of the per-parameter sum of squared gradients. This is then used to normalize the parameter update step, element-wise. The weights that receive high gradients will have their effective learning rate reduced, while weights that receive small or infrequent updates will have their effective learning rate increased. The smoothing term “*eps*” (usually set somewhere in the range from e^{-4} to e^{-8}) avoids division by zero. A downside of Adagrad is that in the case of Deep Learning, the monotonic learning rate usually proves too aggressive and stops learning too early.

Adadelta:

Adadelta is a more robust extension of Adagrad that adapts learning rates based on a moving window of gradient updates, instead of accumulating all past gradients [50]. This way, Adadelta continues learning even when many updates have been done. The method dynamically adapts over time using only first order information and has minimal computational overhead beyond vanilla stochastic gradient descent.

After many iterations, this learning rate will become infinitesimally small. The method requires no manual tuning of a learning rate and appears robust to noisy gradient information, different model architecture choices, various data modalities, and selection of hyperparameters.

The two main drawbacks of Adagrad which were improved in Adadelta:

- The continual decay of learning rates throughout training
- The need for a manually selected global learning rate.

3.9 Conclusion

Neural network tries to mimic the construction of the human optical system. It is generally consisted of three layers. At first, some minimal data processing is done and the weights are initialized of the network. Then the network is ready to train and the gradient is calculated so the weights can be changed in such a way that the loss of the network will be minimized.

CHAPTER 4

Convolutional Neural Networks (CNN)

This chapter discussed in detail about convolutional neural network. It explores the architecture of CNN and how it works to solve a problem.

4.1 CNN

Convolutional Neural Networks (CNN) are very close to the previously described ordinary neural networks. Neurons that have learnable weights and prejudices are often made up of CNN. Each neuron receives some inputs, performs a dot product, and with a non-linearity optionally follows it. The whole network expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they shave a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer just like an ordinary neural network. The key difference is that ConvNet architectures specifically presume that the inputs are images, so that such properties can be encoded into the architecture by the model. This then allow the forward function more powerful to enforce the number of parameters in the network and reduce them dramatically.

CNN is an enabling architecture of deep learning. This was promoted by the animal optical system[45] that directly learns features from the input data provided, avoiding the hand-crafted function. In several steps, CNN consists of several nonlinear transformations and works to reduce the error to be correctly classified in each step. In short, a CNN is composed of several trainable stages composed of a stack of data tops, followed by a supervised classifier. Each stage has a set of arrays called feature vectors that represent the vectors of input and output respectively[45]. To train the network, CNN needs a large number of labeled samples and computational resources. Due to the growth of accessible digital data and powerful computing tools, i.e. the Graphics Processing Unit (GPU)[51], the time needed during network training is substantially reduced than before. This convenience will enable us in our environment to train deeper CNN architectures to achieve a higher outcome.

4.2 CNN Architecture

Let us clarify how the CNN network converges before going to the info. It is a structured neural network that needs minimal preprocessing to process visual information, where the first couple of layers are sequentially interconnected. Since the output of the existing layer becomes the input of the next layer, CNN is feed-forwarding in nature. There are learnable parameters such as weights and preferences in the CNN neurons. Through a forward move, the network begins teaching itself. The input volume is converted into an output volume by a list of connected layers. In the output layer, the prediction is computed as a probability reflecting class scores. In order to compute the error, the expected outcome is then compared with the actual result. The computed error in backpropagation produces the gradient that flows in the backward direction. The parameters are tuned in such a direction at each stage that the previously produced error [52][53][54] is attempted to decrease. Until the model converges, this process proceeds in an iterative fashion.

Convolutional Neural Networks take advantage of the fact that images are the input and they limit the architecture in a more rational manner. In particular, the layers of a ConvNet have neurons organized in 3 dimensions, unlike a standard Neural Network: distance, height, depth. (Note that the word depth here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network.) Instead of all neurons in a fully-connected way, the neurons in a layer would only be connected to a small region of the layer before it. A ConvNet is generated by layers. Each layer has a simple API: it converts a 3D volume input to a 3D volume output with some distinguishable feature that may or may not have parameters. Figure 4.1 depicts this ConvNet process. From the figure, (Top) A regular 3-layer Neural Network. (Bottom) A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

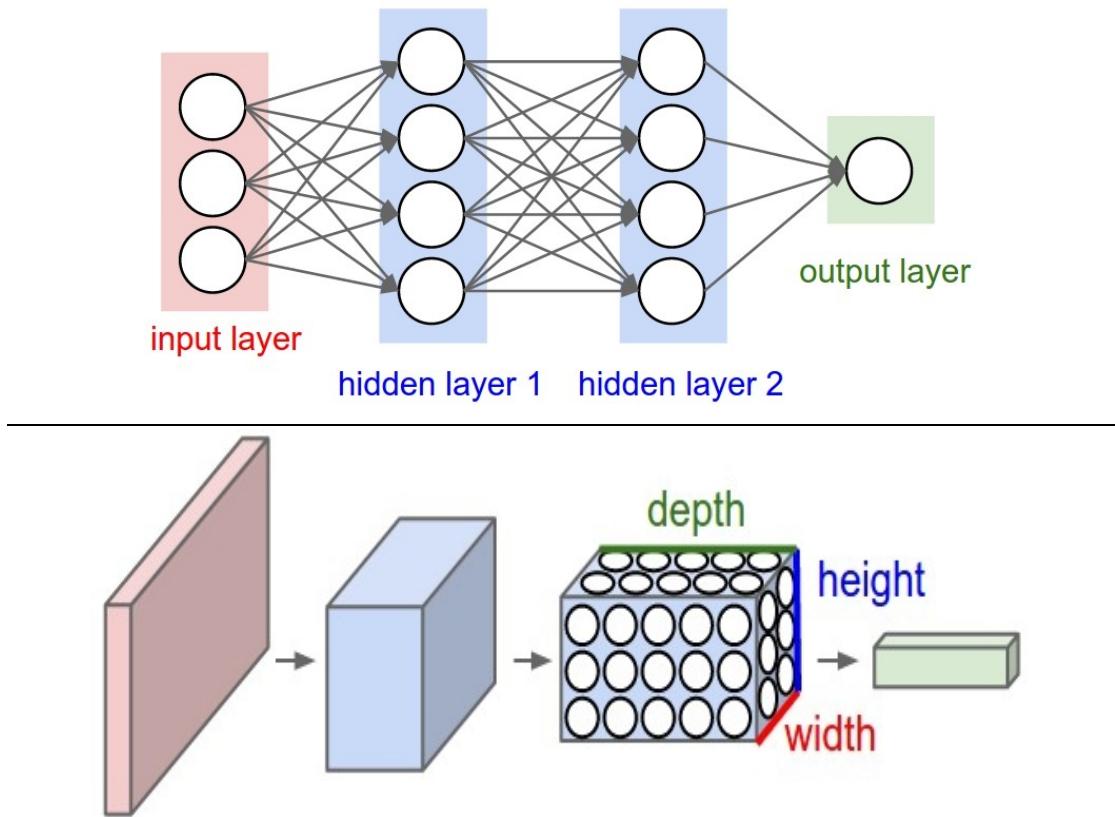


Figure 4.1: Comparison of an original Neural Network with a CNN [42]

4.3 Layers in CNN

A simple ConvNet is a sequence of layers, and through a distinctive feature, each layer of a ConvNet transforms one volume of activations into another. To build ConvNet architectures, three main layer types are required: Convolutional Layer, Pooling Layer, and Fully-Connected Layer (exactly as seen in regular Neural Networks). To form a complete ConvNet architecture, stack these layers together. Figure 4.2 shows how a typical CNN architecture looks like.

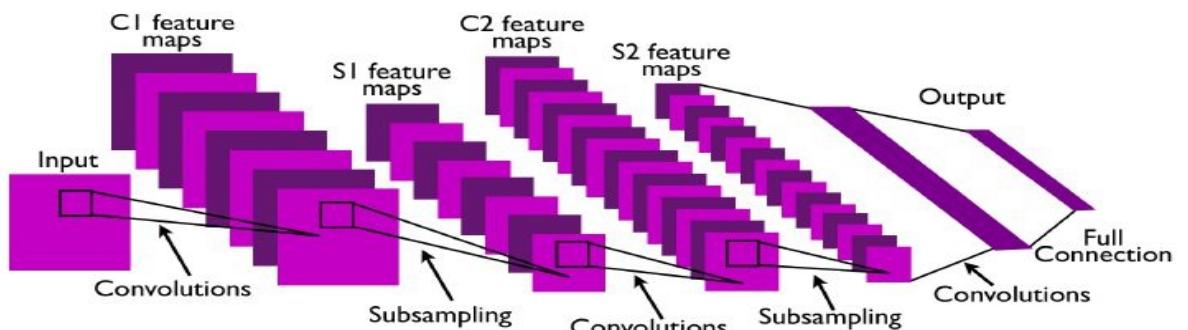


Figure 4.2: A typical CNN architecture [45]

Figure 4.3 describes the activations of an example ConvNet architecture. The raw image pixels are stored in the initial volume (left), and the last volume stores the class scores (right). As a column, each volume of activations along the processing path is shown. Since visualizing 3D volumes is hard, we spread out the slices of each volume in rows. For each class, the last layer volume contains the scores, but here we only visualize the top 5 scores sorted and print the labels of each one.

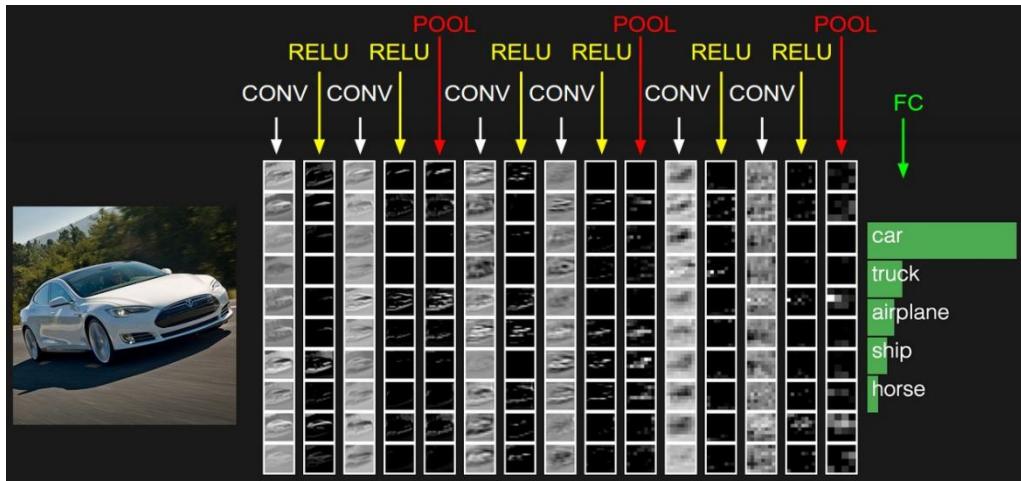


Figure 4.3: ConvNet architecture example [42]

4.3.1 Convolutional Layer

The Conv layer is a Convolutional Network's main building block that performs much of the heavy computational lifting. The parameters of the CONV layer consist of a series of learnable filters. Each filter is spatially small (along with width and height), but it extends to the maximum depth of the input volume. For example, a typical filter on the first layer of a ConvNet might have size $5 \times 5 \times 3$ (i.e. 5 pixels width and height, and 3 because images have depth 3, the color channels). During the forward pass, each filter is slid i.e. convoluted across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position.

As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position shown in figure 4.4. Intuitively, the network will learn filters that

activate when they see some type of visual features such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. Now, we will have an entire set of filters in each CONV layer (e.g. 12 filters), and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume.

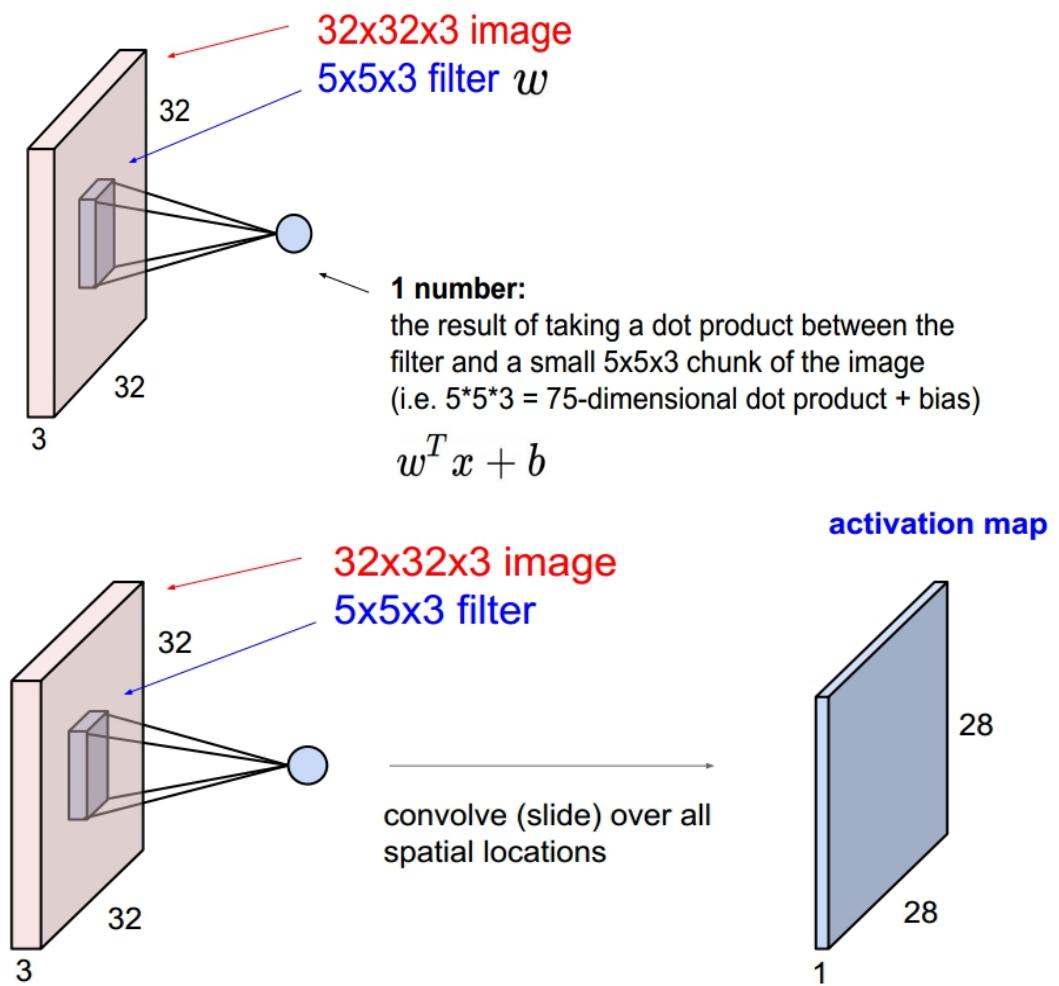


Figure 4.4: Convolution layer example [42]

An example in figure 4.5 shows (Top) an input volume in red (e.g. a $32 \times 32 \times 3$ CIFAR-10 image), and an example volume of neurons in the first Convolutional layer. Each neuron in the convolutional layer is connected only to a local region in the input volume spatially but to the full depth (i.e. all color channels). There are multiple neurons (5 in this example) along the depth, all looking at the same region in the input. (Bottom) The neurons from the Neural Network chapter remain unchanged: They still

compute a dot product of their weights with the input followed by a non-linearity, but their connectivity is now restricted to be local spatially.

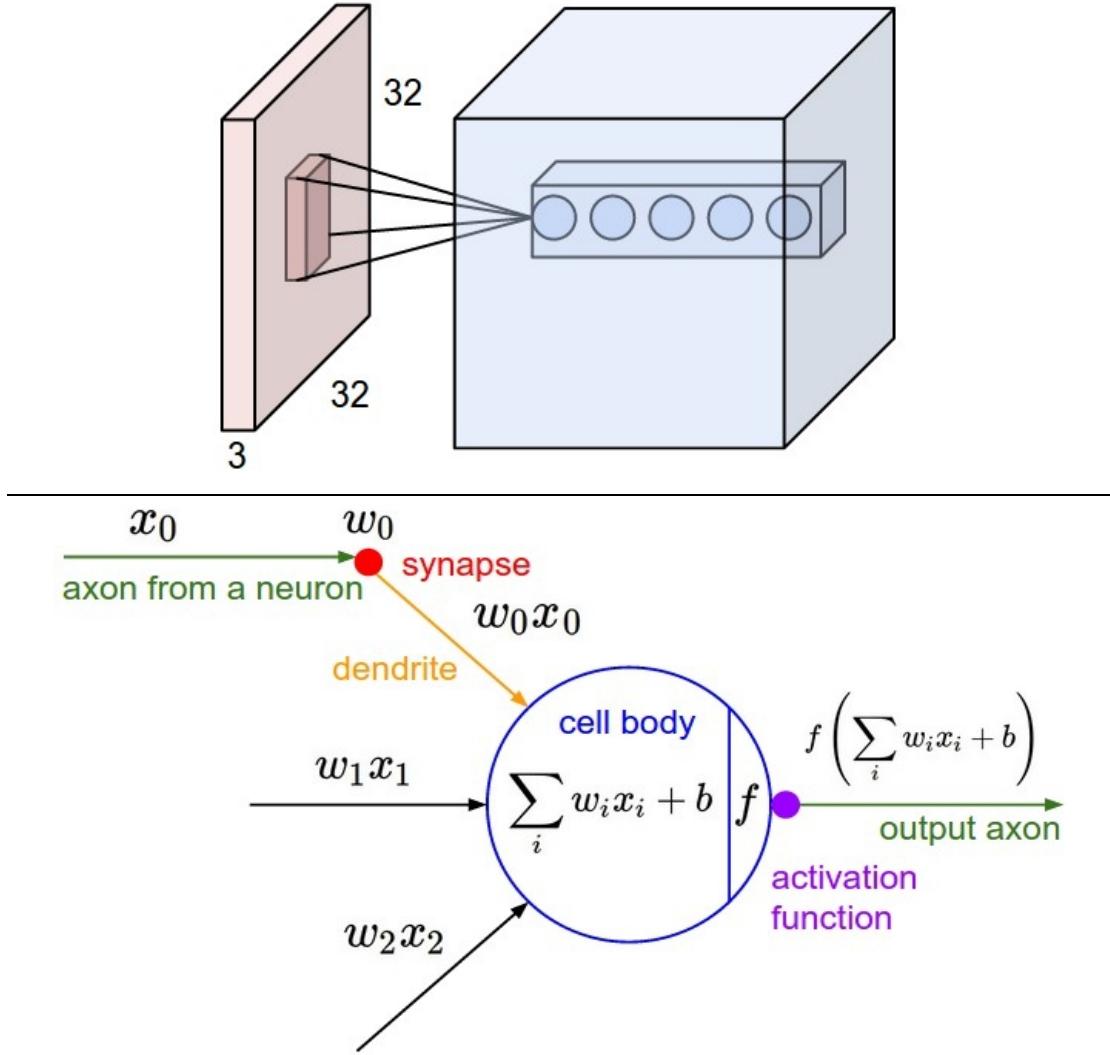


Figure 4.5: Example of local connectivity [42]

Three hyperparameters control the size of the output volume: the depth, stride, and zero-padding. The depth of the output volume is a hyperparameter: it corresponds to the number of filters that are being used to each learning to look for something different in the input. The number of the stride must be specified with which we slide the filter. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 (or uncommonly 3 or more, though this is rare in practice) then the filters jump 2 pixels at a time as we slide them around. This will produce smaller output volumes spatially. Finally, sometimes it will be convenient to pad the input volume

with zeros around the border. The size of this zero-padding is a hyperparameter. The nice feature of zero padding is that it will allow us to control the spatial size of the output volumes.

The spatial size of the output volume can be computed as a function of the input volume size (W), the receptive field size of the Conv Layer neurons (F), the stride with which they are applied (S) and the amount of zero padding used (P) on the border. In general, setting zero padding to be $P = \frac{F-1}{2}$ when the stride is $S = 1$ ensures that the input volume and output volume will have the same size spatially.

4.3.2 Activation Layer

The activation layer introduces non-linear properties to the network followed by convolution layers. Their main purpose is to convert an input signal of a node into an output signal. The generated output will be passed towards the next layer. Every activation function (or non-linearity) takes a single number and performs a certain fixed mathematical operation on it. There are many activation functions to choose from. Two activation functions are discussed here.

Sigmoid: The sigmoid non-linearity has the mathematical form $\sigma(x) = \frac{1}{1+e^{-x}}$ and is shown in figure 4.6. It takes a real-valued number and “squashes” it into a range between 0 and 1. In particular, large negative numbers become 0 and large positive numbers become 1. The sigmoid function has seen frequent use historically since it has a nice interpretation as the firing rate of a neuron: from not firing at all (0) to fully-saturated firing at an assumed maximum frequency (1).

- (i) In practice, the sigmoid non-linearity has recently fallen out of favor and it is rarely ever used. It has two major drawbacks:
 - (ii) A very undesirable property of the sigmoid neuron is that when the neuron’s activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero. Therefore, if the local gradient is very small, it will effectively “kill” the gradient and almost no signal will flow through the neuron to its weights and recursively to its data.

- (iii) Sigmoid outputs are not zero-centered. This is undesirable since neurons in later layers of processing in a Neural Network (more on this soon) would be receiving data that is not zero-centered.

ReLU: The Rectified Linear (ReLU) Unit has become very popular in the last few years. It is computationally efficient and converges much faster than most other activation functions [14]. It computes the function $f(x) = \max(0, x)$ showed in figure 4.7. In other words, the activation is simply being the threshold at zero (see image above on the left). There are several pros and cons to using the ReLUs:

(i) It was found to greatly accelerate the convergence of stochastic gradient descent compared to the sigmoid functions.

(ii) Compared to sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero.

(iii) Unfortunately, ReLU units can be fragile during training and can “die”.

For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on. That is, the ReLU units can irreversibly die during training since they can get knocked off the data manifold.

(iv) Compared to sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero.

(v) Unfortunately, ReLU units can be fragile during training and can “die”.

For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on. That is, the ReLU units can irreversibly die during training since they can get knocked off the data manifold.

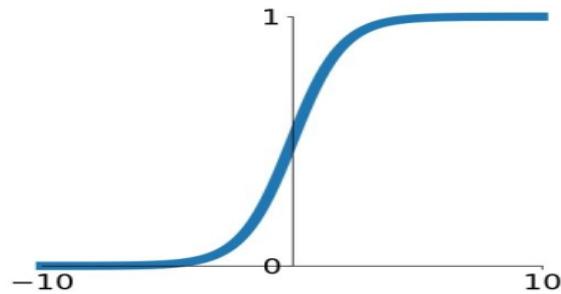


Fig 4.6: Activation function: tanh

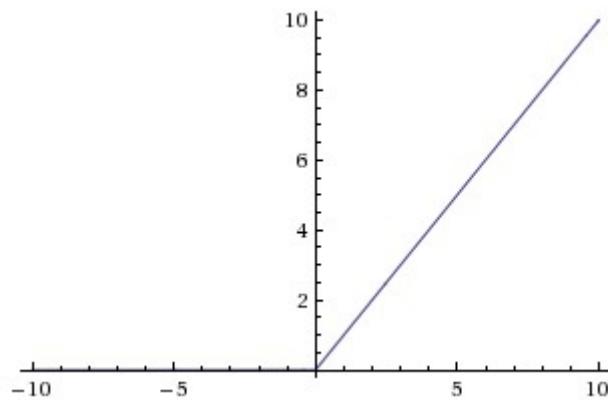


Figure 4.7: Activation function: ReLU

4.3.3 Pooling Layer

It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture. Its function is to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation. The most common form is a pooling layer with filters of size 2×2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations. Every MAX operation would, in this case, be taking a max over 4 numbers (little 2×2 region in some depth slice). The depth dimension remains unchanged. More generally, the pooling layer:

- Accepts a volume of size $W_1 * H_1 * D_1$
- Requires two hyperparameters:
 - their spatial extent F ,
 - the stride S ,

- Produces a volume of size $W_2 * H_2 * D_2$ where:

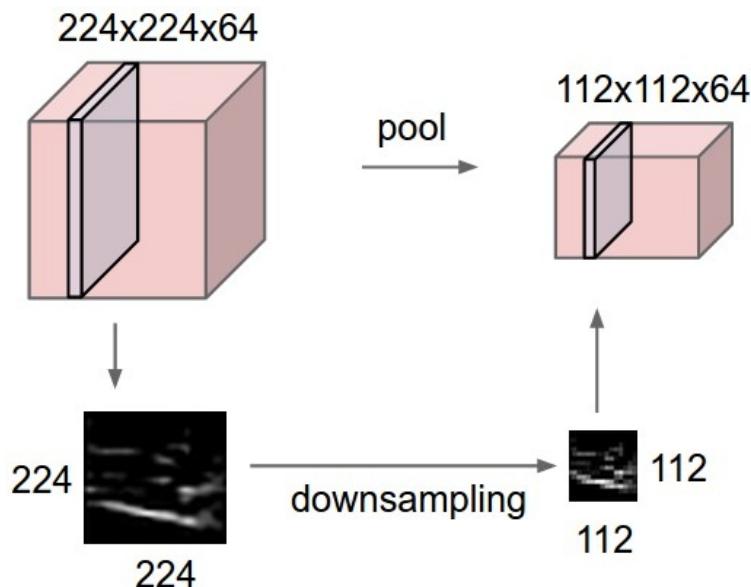
$$\circ W_2 = \frac{W_1 - F}{S} + 1$$

$$\circ H_2 = \frac{H_1 - F}{S} + 1$$

$$\circ D_2 = D_1$$

- Introduces zero parameters since it computes a fixed function of the input
- For Pooling layers, it is not common to pad the input using zero-padding.

Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. In this example of figure 4.8, (top) the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. (Bottom) The most common downsampling operation is max, giving rise to max pooling, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).



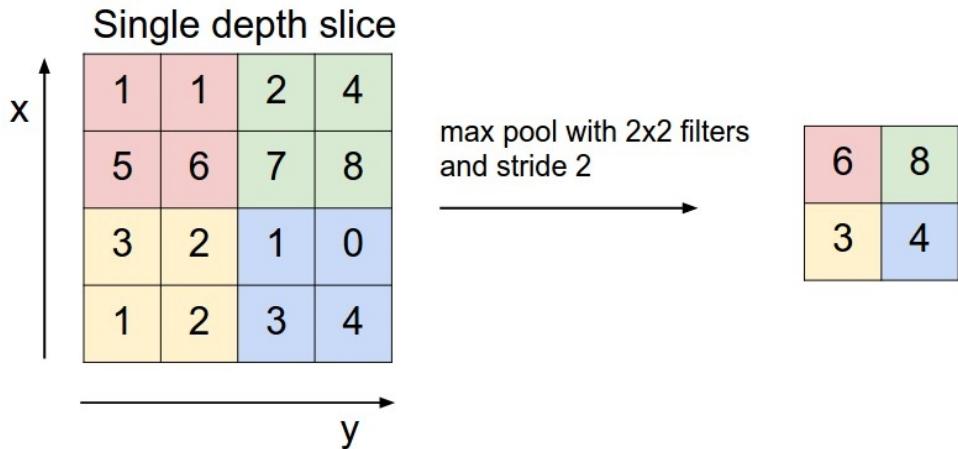


Figure 4.8: Example of maxpooling [42]

4.3.4 Fully-Connected (FC) Layer

Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset. It is worth noting that the only difference between FC and CONV layers is that the neurons in the CONV layer are connected only to a local region in the input and that many of the neurons in a CONV volume share parameter. However, the neurons in both layers still compute dot products, so their functional form is identical. Therefore, it turns out that it's possible to convert between FC and CONV layers:

- For any CONV layer, there is an FC layer that implements the same forward function. The weight matrix would be a large matrix that is mostly zero except for at certain blocks (due to local connectivity) where the weights in many of the blocks are equal (due to parameter sharing).

Conversely, any FC layer can be converted to a CONV layer. For example, an FC layer with $K=4096$ that is looking at some input volume of size $7 \times 7 \times 512$ can be equivalently expressed as a CONV layer with $F=7, P=0, S=1, K=4096$. In other words, we are setting the filter size to be exactly the size of the input volume, and hence the output will simply be $1 \times 1 \times 4096$ since only a single depth column “fits” across the input volume, giving an identical result as the initial FC layer.

The final fully connected layer of the network produces the net output with an activation function i.e. softmax function depending on the number of classes in the classification problem.

A typical CNN architecture consists of these types of several components. An example of a typical architecture would be:

$$[(\text{CONV-RELU})^N \text{-POOL?}]^M - (\text{FC-RELU})^K, \text{SOFTMAX}$$

where N is usually up to ~ 5 , M is large, $0 \leq K \leq 2$.

4.4 Known Architectures in CNN

There are several architectures in the field of Convolutional Networks that have a name. The most common are:

- **LeNet:** The first successful applications of Convolutional Networks were developed by Yann LeCun in 1990's. Of these, the best known is the LeNet architecture [34] that was used to read zip codes, digits, etc.
- **AlexNet:** The first work that popularized Convolutional Networks in Computer Vision was the AlexNet [14], developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton. The Network had a very similar architecture to LeNet, but was deeper, bigger, and featured Convolutional Layers stacked on top of each other (previously it was common to only have a single CONV layer always immediately followed by a POOL layer).
- **GoogLeNet:** The ILSVRC (Large Scale Visual Recognition Challenge) 2014 winner was a Convolutional Network from Szegedy et al. [47] from Google. Its main contribution was the development of an Inception Module that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M). Additionally, this paper uses Average Pooling instead of Fully Connected layers at the top of the ConvNet, eliminating a large number of parameters that do not seem to matter much.
- **VGGNet:** The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman that became known as the VGGNet [48]. Its main contribution was in showing that the depth of the network is a critical component for good performance. Their final best network contains 16 CONV/FC layers and, appealingly, features an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end.
- **ResNet:** Residual Network developed by Kaiming He et al. [49] was the winner of ILSVRC 2015. It features special skip connections and a heavy use

of batch normalization. The architecture is also missing fully connected layers at the end of the network. The reader is also referred to Kaiming's presentation (video, slides), and some recent experiments that reproduce these networks in Torch. ResNets are currently by far state of the art Convolutional Neural Network models and are the default choice for using ConvNets in practice.

4.5 Conclusion

CNN is another form of the ordinary neural network but it takes images as inputs. CNN architectures are mainly constructed by some major layers and they are stacked one after another.

Chapter 5

Methodology

This chapter consists of the work flow throughout the research. It includes data collection, building databank for training and validation set, developing deep CNN for training, classifying crack and non crack images.

5.1 Data Source

The datasets are collected [55]. The datasets include pictures of different textures of concrete with and without cracking. The image data is split into two in a separate folder for image classification as negative (without crack) and positive (with crack). With a total of 40000 images of 227 x 227 pixels with RGB channels, each class has 20000 images. The dataset is created from 458 (4032x3024 pixel) high-resolution images using the method proposed by Zhang et al (2016). High-resolution photographs have been shown to have high differences in terms of surface finish and state of lighting. No data augmentation is implemented in terms of random rotation or flipping or tilting.

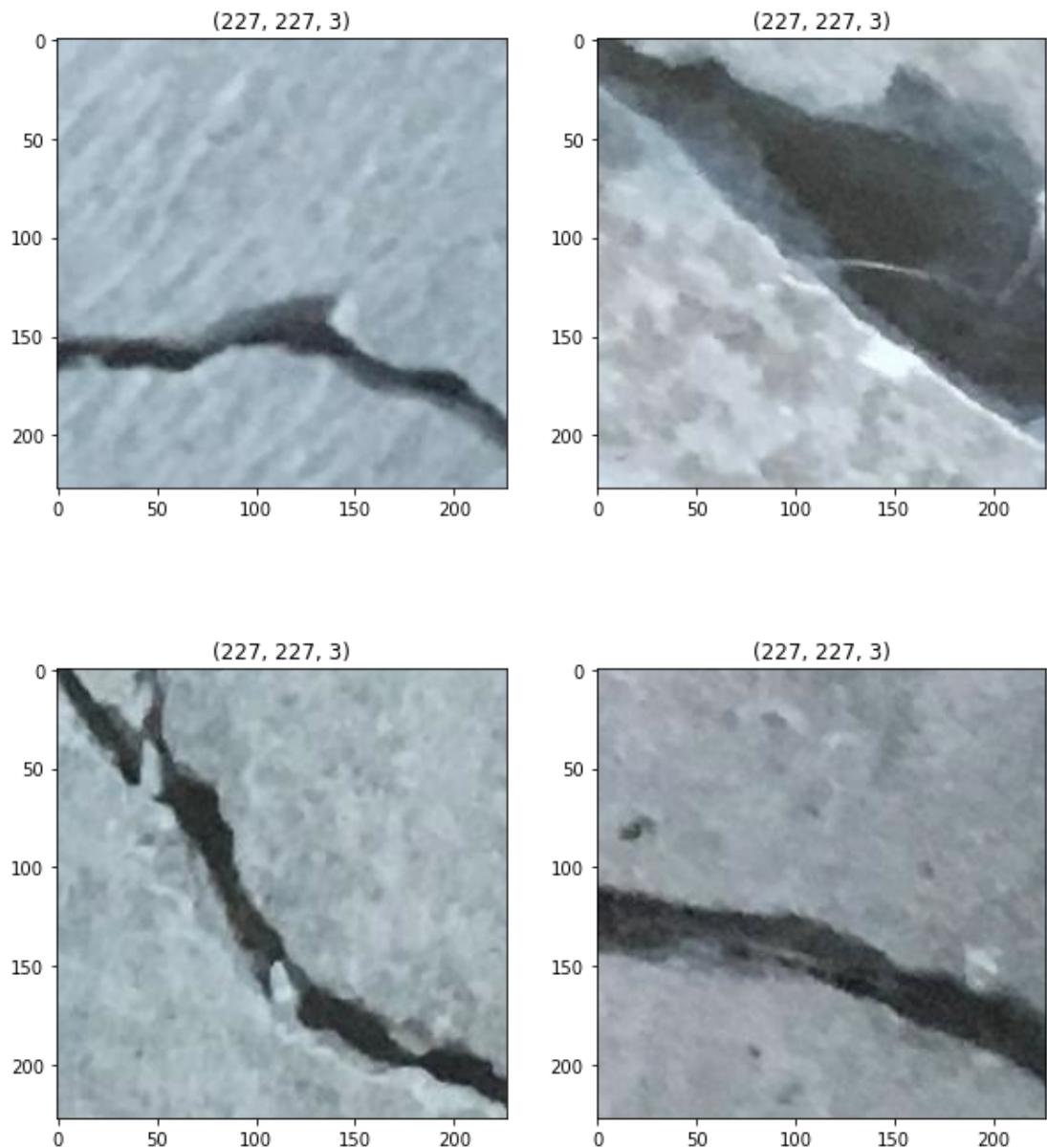


Fig 5.1: Crack Images[55]

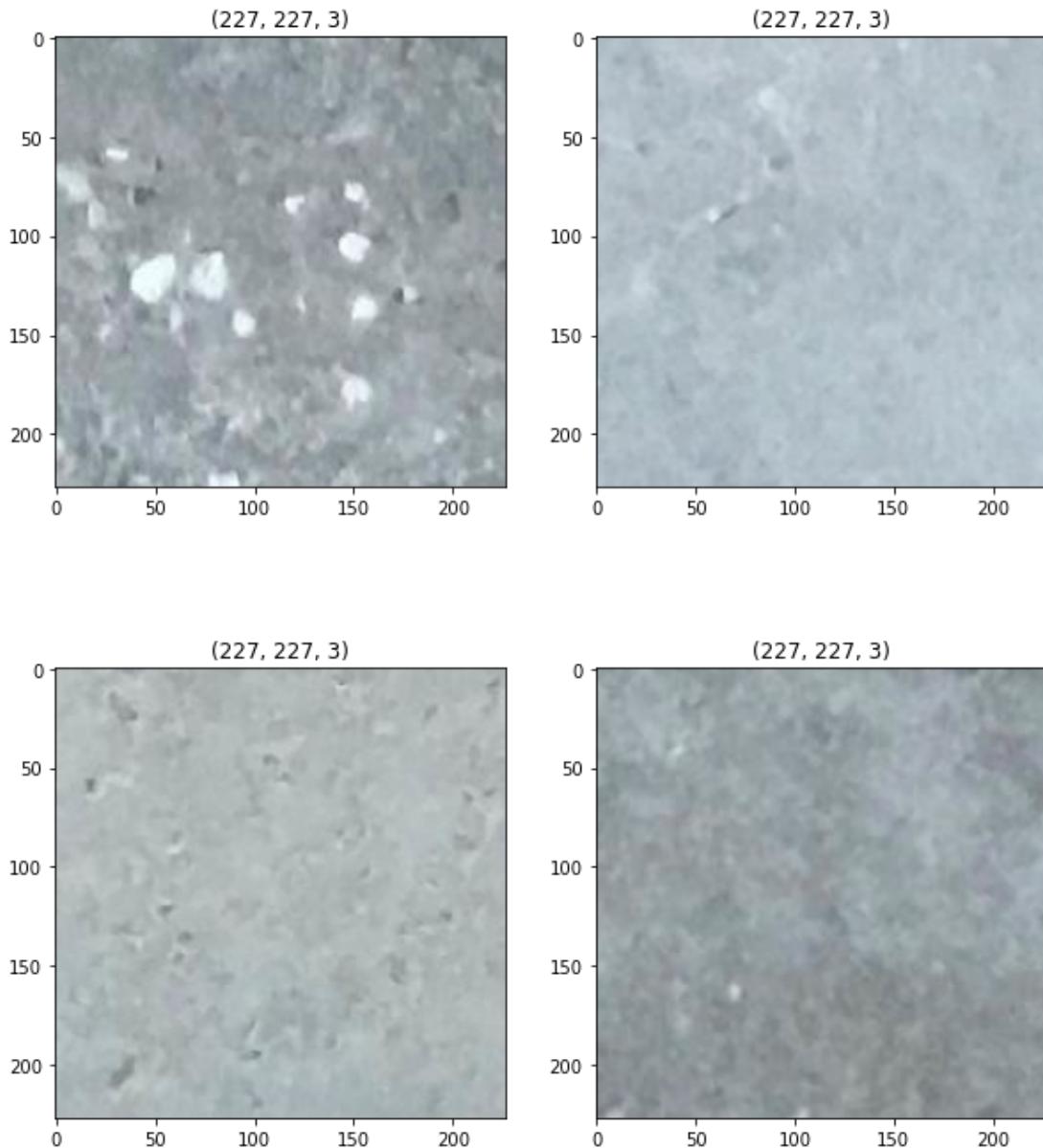


Fig 5.2: Non-Crack Images[55]

All the images are in JPG Format. This a balanced dataset with equally distributed crack and non-crack images.

5.2 Data Preprocessing

I have spent a fair amount of time on strategies for data preprocessing typically used for image processing. This is because in most deep learning applications, preprocessing takes a large amount of time. As the dataset contains 40,000 images with RGB channel, I've performed two image processing techniques.

5.2.1 Image Resize

The first step in data pre-processing is the creation of images of the same size to feed our pre-trained CNN model. For this I've used keras built in function[56]. The images are re-sized into 150 x 150.

5.2.2 Normalization

Image normalization is a method often used in the preparation of machine learning data sets in which multiple images are positioned in a standard statistical distribution in terms of scale and pixel values; but within themselves, a single image may also be normalized. Typically, the phase entails both spatial and intensity normalization.

This is the most crucial step in image pre-processing. This corresponds to rescaling the values of the pixels so that they lie within a confined range. One of the reasons for doing so is to help with the challenge of gradient propagation. For normalization I've divided each pixel by 255 and re-scaled them between 0 to 1.

5.3 Model Architecture

For the classification I've used pre-trained CNN model VGG16 [57]. VGG is an acronym for the Oxford University Visual Geometric Group, and VGG-16 is a network suggested by the Visual Geometric Group with 16 layers. The trainable parameters are found in these 16 layers and there are other layers, such as the Max pool layer, but there are no trainable parameters. This architecture was the first heir to the 2014 Visual Perception Competition, i.e. ILSVRC-2014 which was founded by Zisserman and Simonyan. Figure 5.3 shows a VGG16 architecture.

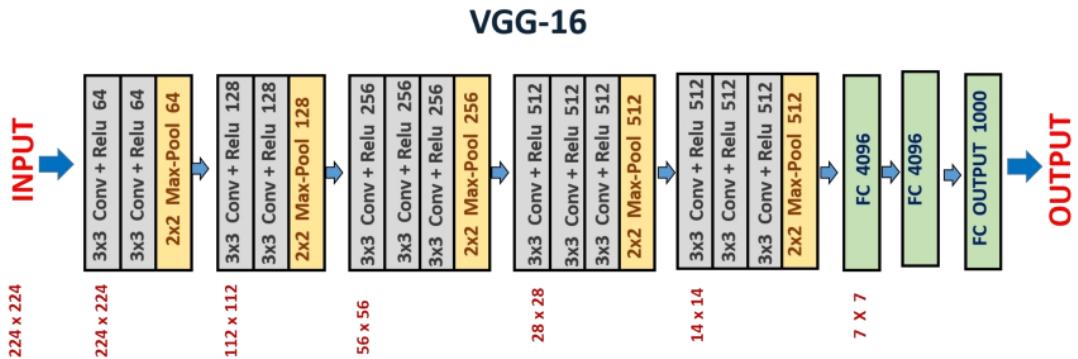


Fig 5.3: VGG16 Architecture[57]

The architecture is simple. It has 2 contiguous blocks of 2 convolution layers, followed by max-pooling, then it has 3 contiguous blocks of 3 convolution layers, followed by max-pooling, and we eventually have 3 thick layers. In different architectures, the last 3 layers of convolution have different depths.

5.3.1 Features of VGG16 Network

- Input Layer:** It accepts color images as an input with the size 224×224 and 3 channels i.e. Red, Green, and Blue.
- Convolution Layer:** The images pass through a stack of convolution layers where every convolution filter has a very small receptive field of 3×3 and stride of 1. Every convolution kernel uses row and column padding so that the size of input as well as the output feature maps remains the same or in other words, the resolution after the convolution is performed remains the same.
- Max pooling:** It is performed over a max-pool window of size 2×2 with stride equals to 2, which means here max pool windows are non-overlapping windows.
- Not every convolution layer is followed by a max pool layer as at some places a convolution layer is following another convolution layer without the max-pool layer in between.
- The first two fully connected layers have 4096 channels each and the third fully connected layer which is also the output layer have 1000 channels, one for each category of images in the imangenet database.

6. The hidden layers have ReLU as their activation function.

5.3.2 Implementation of The Architecture

The input to the conv1 layer is a fixed image size of 224 x 224 RGB. The picture is passed through a stack of convolutional (conv.) layers where the filters have been used with a very narrow receptive field: 3 x 3 (which is the smallest size to catch the left/right, up/down, center). It also uses 1 x 1 convolution filters in one of the setups, which can be used as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1; conv's spatial padding. The layer input is such that after convolution, the spatial resolution is retained, i.e. the padding for 3 x 3 conv is 1-pixel. Of textures. Five max-pooling layers, which obey some of the conv, perform spatial pooling. Of textures (not all the conv. layers are followed by max-pooling). Max-pooling, with stride 2, is done over a 2 to 2 pixel window.

A stack of convolutional layers (which has a different depth in different architectures) is preceded by three Fully-Connected (FC) layers: the first two have 4096 channels each, the third has 1000-way ILSVRC classification and thus includes 1000 channels (one for each class). The last layer is soft-max layer. The configuration of fully linked layers on all networks is the same.

Both hidden layers are fitted with non-linearity for rectification (ReLU). Furthermore, it is remembered that none of the networks (except for one) contain Local Response Normalization (LRN), which does not boost the efficiency of the ILSVRC dataset, but results in increased memory usage and computation time.

5.3.3 Training The Model

As it is mentioned earlier about the proposed CNN model. For training the model the validation set was splitted from entire dataset. A validation dataset is a dataset of examples used for tuning a classifier's hyperparameters. Often it is also referred to as the "Development set" or the "dev set". The number of hidden units in each layer includes an example of a hyperparameter for machine learning.

For proposed method about 30 percent of the total dataset was used for validation. The validation set was selected randomly. So 12000 images were used for validation and rest 28000 images were used for training containing both crack and non crack images. After that the training and validation data were feed into VGG16 model. For VGG16 the weights are initialized with “imagenet” [58] a pretrained model weight initialization. The batch size is 64 and epoch is 20. VGG16 has 5 block with 13 convolutional layer. So the training and validation dataset are fast go through 5 blocks of VGG16. After that the output is flattened from $4 \times 4 \times 512$ to 8192×1 which is a one dimensional vector. Three fully connected layers follow the stack of convolutional layers. The first two has 1024 units and final layer is ‘softmax’ layer with output node one.

Validation and testing were performed in training period. ‘ReLU’ function was used in each hidden layers. It is effective in computing and converges much faster than most other activation functions. “ReLU” converts the negative calculated value to 0 and no negative value passes to next layer. Hidden layers are the feature extracting layer. So using “ReLU” function in hidden layers doesn’t effect the output much because of the ignorance of the negative values. “Sigmoid” is used in final layer. The sigmoid non-linearity has the mathematical form $\sigma(x) = \frac{1}{1+e^{-x}}$. It takes a real-valued number and “squashes” it into a range between 0 and 1. In particular, large negative numbers become 0 and large positive numbers become 1. As the input dataset is binary class sigmoid function merges the output into crack or non crack images. 0 for non crack image and 1 for crack image.

“Dropout” is used before the output layer to overcome the overfitting problem in the network [48]. Overfitting is' the development of a study that correlates too closely or precisely to a specific data set and may therefore struggle to match additional data or accurately predict future findings.' An overfit model is a mathematical model containing more parameters than the data can explain [59]. To overcome this problem dropout is used. Dropout is a method where randomly selected neurons are dropped during training. They are “dropped-out” arbitrarily. This infers that their contribution to the activation of downstream neurons is transiently evacuated on the forward pass

and any weight refreshes are not applied to the neuron on the backward pass. The dropout is selected 20 percent of total nodes.

The network is trained all over using Adam optimizer with initial standard parameters [60]. During training the network weights may stack into local minima problem. Adam helps to find the global minima when the network stacks into local minima problem. Adam updates network weights iterative based in training data.

The model has two classes, crack and non crack. Regarding the problem “binary_crossentropy” function was used to minimize the error [56]. The binary crossentropy is very convenient to train a model to solve many classification problems at the same time, if each classification can be reduced to a binary choice. The crack detection is a binary classification task. The output is a binary label $y \in \{0,1\}$; representing the absence of crack or not respectively. In the training set, weighted binary cross entropy loss is optimized. A table of the summary of the whole network is given below:

Table 5.1 Model Summary

Layer(Type)	Output Shape	No of Parameter
input	(150, 150, 3)	0
block1_conv1	(150, 150, 64)	1792
block1_conv2	(150, 150, 64)	36928
block1_pool	(75, 75, 64)	0
block2_conv1	(75, 75, 128)	73856
block2_conv2	(75, 75, 128)	147584
block2_pool	(37, 37, 128)	0
block3_conv1	(37, 37, 256)	295168
block3_conv2	(37, 37, 256)	590080

block3_conv3	(37, 37, 256)	590080
block3_pool	(18, 18, 256)	0
block4_conv1	(18, 18, 512)	1180160
block4_conv2	(18, 18, 512)	2359808
block4_conv3	(18, 18, 512)	2359808
block4_pool	(9, 9, 512)	0
block5_conv1	(9, 9, 512)	2359808
block5_conv2	(9, 9, 512)	2359808
block5_conv3	(9, 9, 512)	2359808
block5_pool	(4, 4, 512)	0
flatten	(None, 8192)	0
Dense	(None, 1024)	8389632
Dropout	(None, 1024)	0
Dense	(None, 1)	1025

5.4 Conclusion

The chapter is about the methodology of building the architecture. VGG16 is a pretrained architecture with total number of five blocks containing thirteen convolutional layer distributed in each block. Each block has a maxpooling layer following the convolutional layers.

CHAPTER 6

Results and Performance Analysis

This is the result analysis chapter. Through out the chapter the result of the method will be discussed.

6.1 Environment

The VGG16 model was built in kaggle notebook [57]. The notebook editing session allows 9 hours execution time. It has 20 Gigabytes of auto saved disk including 4 CPU cores, 16 Gigabytes of RAM. And the GPU specs with 2 CPU cores and 13 Gigabytes of RAM. It provides NVIDIA Tesla P100. Keras API on top of TensorFlow (CUDA toolkit 9.0, cuDNN SDK v7 and python 3.6) were used [58].

6.2 Experimental Analysis

The data set is divided into training and validation set randomly. 12000 images are selected for validation and the rest for training. Testing was performed during training period. The results presented in this work is based on accuracy and f1 score [59] which are described by the following equations:

$$\text{accuracy} = \frac{tp+tn}{tp+tn+fp+fn} \quad (20)$$

$$f1\ score = 2 * \frac{\text{precision}*\text{recall}}{\text{precision}+\text{recall}} \quad (21)$$

Where tp, tn, fp, fn represent true positive, true negative, false positive and false negative respectively. Recall is defined as the fraction of the relevant instances in a dataset that is successfully retrieved and precision expresses the proportion of the data points the model says is relevant actually are relevant.

CNN has the advantage of learning features automatically instead of manual feature extraction techniques. The self-learning ability of CNN model makes it more convenient than the traditional learning system.

Figure 6.1 represents the training and validation accuracy and figure 6.2 shows the training and validation loss.

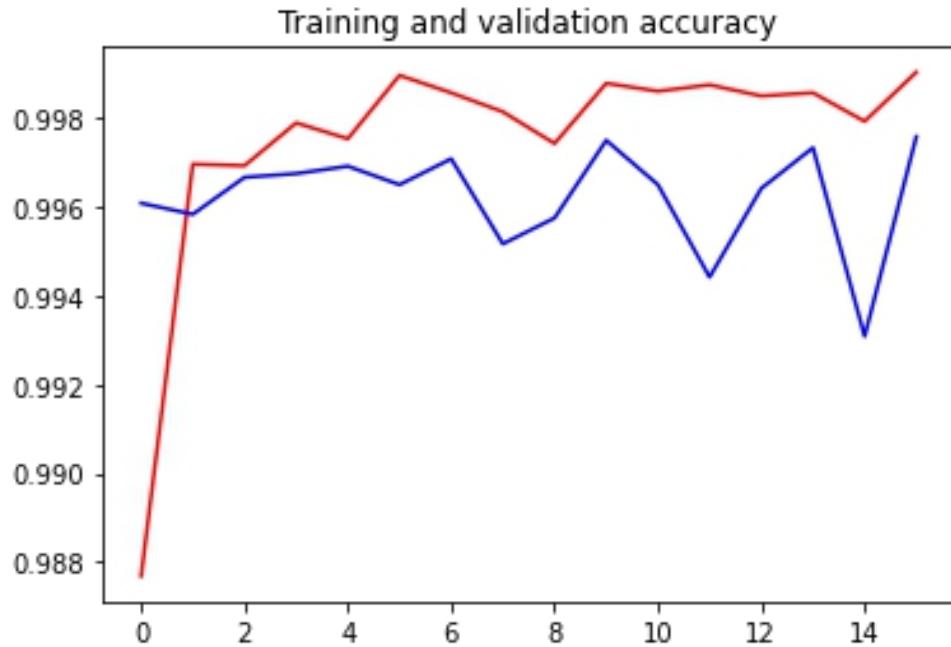


Figure 6.1: Accuracy curve for training and validation

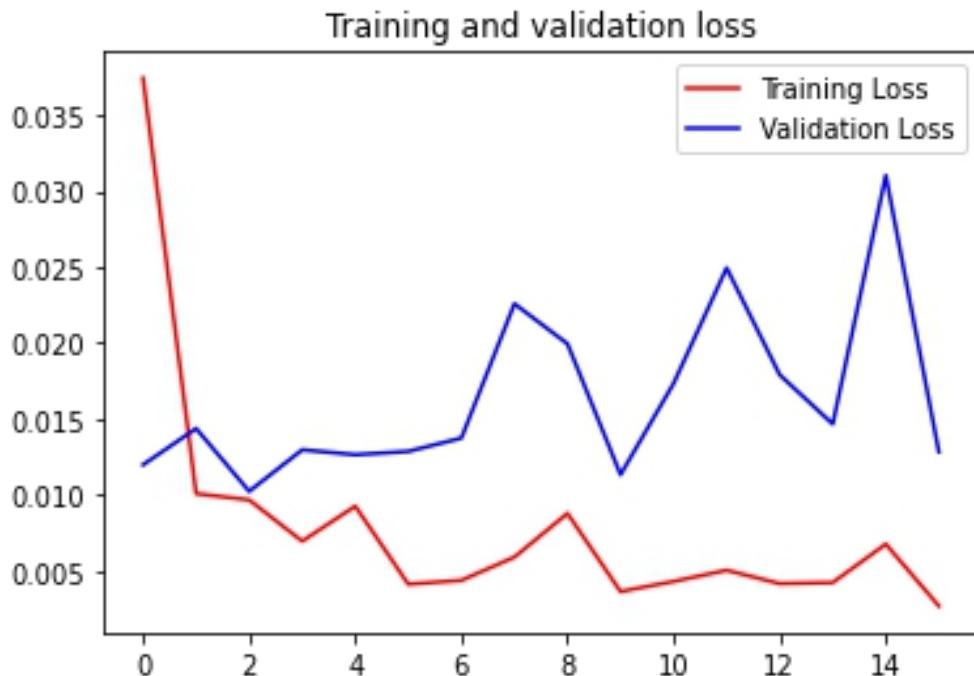


Figure 6.2: Accuracy curve for training and validation loss

From the figure we can see, by the increasing number of epochs the proposed model gets more accuracy with a decreasing factor of loss. It took around 233s for first epoch and gradually the time was minimized. Using equations (20) and (21), the

proposed system achieved 99.90 percent accuracy, 0.990 f1 score and 0.9992 precision which outperformed the previous state of the art. A recent research [30] had the accuracy of 97.5 percent accuracy using ResNet based classifier. Though VGG16 got 99.8 percent accuracy but needed 50 epoch to acquire this accuracy which was very time consuming. Proposed method got almost same accuracy performing only 20 epoch which was very much time friendly. Performance of different of CNN architecture is shown in the table below comparing the proposed model also:

Method	Accuracy	Recall	Precision	F1 Score
VGG16	99.8%	0.999	0.998	0.998
Inception V3	99.7%	0.997	0.998	0.997
ResNet	97.5%	0.945	0.994	0.968
Proposed VGG16	99.9%	0.9988	0.9992	0.9990

Table 6.1: Performance of Different Methods

From the table we can see proposed method has outperformed in all the field including accuracy, precision, recall and f1 score.

The accuracy of the network highly depends on the depth of CNN architecture [61]. VGG16 is a deep convolutional neural network with 13 convolutional layer and three fully connected layer. The hidden layers of CNN extract the features. That's why VGG16 performs better for feature extracting methods.

“ReLU” is the activation function of VGG16 in hidden layers. “ReLU” is computationally efficient and converges much faster than most other activation functions [14]. It computes the function $f(x) = \max(0, x)$ and the activation being threshold at zero. Compared to ‘Sigmoid’ function which has exponential operations ‘ReLU’ can be implemented by simply thresholding a matrix of activation at zero.

The kernels are initialized with ‘imagenet’ initializer [58]. This is pretrained weight for VGG16. Three ‘fully connected’ layers were used in the network. The

network was trained with Adam optimizer with initial standard parameters. Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data. Previous methods used ‘RMSProp’, ‘AdaGrad’[30] etc optimization tools. But among them Adam showed the best optimization in proposed method.

6.3 Conclusion

The proposed method achieved a quite good accuracy detecting crack images. This methodology can contribute a lot in detecting cracks in civil engineering sectors.

CHAPTER 7

Conclusion and Future Works

This chapter discusses the summary of thesis work, its limitation and shows future work direction.

7.1 Summary

The motivation of this research work is to build a machine learning image classification model that have a decent accuracy to classify the crack on surface or road. It is very important to identify the cracks of a building or road in civil engineering sectors. Manually detecting cracks is very time consuming and may not also accurate all the time. The proposed method can detect the crack with a decent accuracy. A 40,000 images of crack and non crack were taken for the method. Some of the images were noisy, some of them consists of dust, shadows etc. The proposed method can handle those problems. The images were 227x227 pixels of size with RGB channel. It was reduced to 150x150 pixels with RGB channel to feed the network and reduce the computational time and complexity. Six layers of convolution were used attaching maxpooling layer in each. And three fully connected layers were used. Finally the result of the architecture came quite good with decent accuracy. Future work can explore different architectures using optimized hyperparameters to improve system accuracy.

7.2 Limitations

We worked with only one data source to train our model. With a bigger dataset and more data sources, we would have more variant data samples where the model can be more generalized for detecting cracks. Besides we did not develop any mobile applications for detecting cracks. With the proposed method mobile application can be developed and cracks can be identified.

7.3 Future Work

The plan is to keep working with object recognition and explore other domains related to this. The plan includes:

- We want to work with other object recognition challenges and try to achieve the state of art accuracy in the respective domain.
- We also want to develop:
 - A new hybrid model architecture which would work to recognize objects much better.
 - Ensemble between different models to achieve greater results.
 - Extract additional features and fusion with CNN architecture for better accuracy.
 - More optimized Hyperparameters.

7.4 Conclusion

In the present study, a pretrained model of CNN is proposed for concrete crack detection. The model performs better than ResNet classifier and gives almost same performance against VGG16 classifier and Inception V3 classifier. The model also needs less time than any other model because of using only 20 epoch. Though the proposed method capture cracks in path, quantifying crack size autonomously is still challenging when an image contains so much noises. Therefore, future studies should focus on how to improve the proposed method more robust to make autonomous crack detection.

References

- [1] Yamaguchi, T., Nakamura, S., Saegusa, R., and Hashimoto, S., “Image-based crack detection for real concrete surfaces,” IEEJ T. Electr. Electr. 3(1), 128-135 (2010).
- [2] Oliveira, H. and Lobato Correia, P. “Automatic road crack segmentation using entropy and image dynamic thresholding,” Proc. EUSIPCO 2009, 622-626 (2009).
- [3] Santhi B., Krishnamurthy G., Siddharth S., Ramakrishnan P.K., “Automatic detection of cracks in pavements using edge detection operator,” J. Theor. App. Inf. Technol. 36(2), 199-205 (2012).
- [4] Abdelqader, I., Abudayyeh, O., and Kelly, M. E., “Analysis of edge-detection techniques for crack identification in bridges,” J. Comput. Civil Eng. 17(4), 255-263 (2003).
- [5] Yeum, C. M. and Dyke, S. J., “Vision-based automated crack detection for bridge inspection,” Comput.-Aided Civ. Infrastruct. Eng. 30(10), 759-770 (2015).
- [6] Liu, S. W., Huang, J. H., Sung, J. C., and Lee, C. C., “Detection of cracks using neural networks and computational mechanics,” Comput. Methods Appl. Mech. Engrg. 191(25–26), 2831-2845 (2002).
- [7] Mosehli, O. and Shehab-Eldeen, T., “Classification of defects in sewer pipes using neural networks,” J. Infrastruct. Syst. 6(3), 97-104 (2000).
- [8] Lecun, Y., Bengio, Y., and Hinton, G., “Deep learning,” Nature. 521(7553), 436 (2015).
- [9] Krizhevsky, A., Sutskever, I., and Hinton, G. E., “ImageNet classification with deep convolutional neural networks,” International Conference on Neural Information Processing Systems, 60, 1097-1105 (2012).

- [10] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., and Li, F. F., “ImageNet: A large-scale hierarchical image database,” Proc. CVPR 2009, 248-255 (2009).
- [11] Zhao, X. F., Han, R. C., Yu, Y., Hu, W. T., Jiao, D., Mao, X. Q., Li, M. C, and Ou, J. P., “Smartphone-based mobile testing technique for quick bridge cable-force measurement,” J. Bridge Eng. 22(4), 06016012 (2016).
- [12] A. A. Haider, “Traffic jam: The ugly side of Dhaka's development,” [Online], Available: <https://www.thedailystar.net/opinion/society/traffic-jam-the-ugly-side-dhakas-development-1575355>. [Accessed Dec 14, 2019].
- [13] K. Mahmud, K. Gope, S. Mustafizur, and S. Chowdhury, “Possible Causes & Solutions of Traffic Jam and Their Impact on the Economy of Dhaka City” Journal of Management and Sustainability, Volume 2, 2012.
- [14] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition”, arXiv preprint arXiv:1512.03385, 2015.
- [15] R. Shamsher, and M. Abdullah, “Traffic Congestion in Bangladesh- Causes and Solutions: A study of Chittagong Metropolitan City,” Asian Business Review, Volume 2, March 2015.
- [16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks”, In Advances in neural information processing systems, pages 1097–1105, 2012
- [17] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation”, In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.
- [18] N. Wang and D.-Y. Yeung, “Learning a deep compact image representation for visual tracking”, In *Advances in Neural Information Processing Systems*, pages 809–817, 2013.
- [19] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation”, In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3431–3440, 2015.

- [20] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, “Large-scale video classification with convolutional neural networks”, In Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, pages 1725–1732. IEEE, 2014.
- [21] A. Toshev and C. Szegedy, “Deeppose: Human pose estimation via deep neural networks”, In Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, pages 1653–1660. IEEE, 2014.
- [22] C. Dong, C. C. Loy, K. He, and X. Tang, “Learning a deep convolutional network for image super-resolution”, In Computer Vision–ECCV 2014, pages 184–199. Springer, 2014.
- [23] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision”, arXiv preprint arXiv:1512.00567, 2015.
- [22] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan, “The pothole patrol: using a mobile sensor network for road surface monitoring,” In Proceedings of the 6th international conference on Mobile systems, applications, and services (MobiSys'08), New York, USA, June 17 - 20, pp. 29-39.
- [23] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [24] A. Goswami and H. Liu, "Deep Dish: Deep Learning for Classifying Food Dishes,"(2017) [Online], Available: <https://www.semanticscholar.org/paper/Deep-Dish-%3A-DeepLearning-for-Classifying-Food-GoswamiMicrosoft/2e596abd9d4616f75c9da45d88787cb2fd5bee81>. [Accessed April 29, 2019].

- [25] J. H. Chuah, H. Y. Khaw, F. C. Soon and C. Chow, "Detection of Gaussian Noise and Its Level using Deep Convolutional Neural Network", Proc. of the 2017 IEEE Region 10 Conference (TENCON), Malaysia, November 5-8, 2017.
- [26] M. Teichmann, M. Weber, M. Zöllner, R. Cipolla and R.Urtasun, "MultiNet: Real-time Joint Semantic Reasoning for Autonomous Driving", IEEE Intelligent Vehicles Symposium, IV 2018, Changshu, China, June 26-30, 2018, pp. 1013-1020.
- [27] Y. Lyu, and X. Huang, "Road Segmentation Using CNN with GRU", 22nd International Conference on Information Visualization, 2018.
- [28] Z. Zhang, Q. Liu, Y. Wang. "Road extraction by deep residual U-net", IEEE Geosci. Remote Sens. Lett., vol. 15, no. 5, pp. 749-753, May 2018.
- [29] J. G. Haran, J. Dillenburg, and P. Nelson, "Realtime Image Processing Algorithms for the Detection of Road and Environmental Conditions", Ninth International Conference on Applications of Advanced Technology in Transportation, AATT 2006, Chicago, Illinois, August 13-15, 2006, pp. 55-60.
- [30] C. V. Dung and L. D. Anh, "Autonomous concrete crack detection using deep fully convolutional neural network", Automation in Construction, Volume 99, Pages 52-58, March 2019.
- [31] J. Zhao, H. Wu, and L. Chen, "Road Surface State Recognition Based on SVM Optimization and Image Segmentation Processing," Journal of Advanced Transportation, Vol 2017, pp. 1-21. 2017.
- [32] Z. Wan-zhi, W. Zeng-cai, "Rural Road Detection of Color Image in Complicated Environment", International Journal of Signal Processing, Image Processing and Pattern Recognition, Vol. 6, No. 6(2013), p.p. 161-168.
- [33] M. Lin, Q. Chen, and S. Yan, "Network in network", arXiv preprint arXiv:1312.4400, 2013.
- [34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", arXiv preprint arXiv:1409.1556, 2014.

- [35] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions”, In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1–9, 2015.
- [36] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, et al. “Large scale distributed deep networks”, In Advances in Neural Information Processing Systems, pages 1223–1231, 2012.
- [37] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan, “The pothole patrol: using a mobile sensor network for road surface monitoring,” In Proceedings of the 6th international conference on Mobile systems, applications, and services (MobiSys'08), New York, USA, June 17 - 20, pp. 29-39.
- [38] H. Zhao, J. Shi, X. Qi, X. Wang, J. Jia, “Pyramid Scene Parsing Network”, arXiv preprint arXiv:1612.01105v2[cs.CV] 27 Apr 2017.
- [39] V. Badrinarayanan, A. Kendall, & R. Cipolla (2017), Segnet: A deep convolutional encoder-decoder architecture for image segmentation, IEEE transactions on pattern analysis and machine intelligence, 39(12), 2481-2495.
- [40] M. Teichmann, M. Weber, M. Zoellner, R. Cipolla, & R. Urtasun, (2018, June), Multinet: Real-time joint semantic reasoning for autonomous driving. In 2018 IEEE Intelligent Vehicles Symposium (IV) (pp. 1013-1020). IEEE.
- [41] T. Pohlen, A. Hermans, M. Mathias, B. Leibe, “Full Resolution Residual Networks for Semantic Segmentation in Street Scenes”, arXiv:1611.08323v2 [cs.CV] 6 Dec 2016.
- [42] Moran, Jeffrey, and Robert Desimone. "Selective attention gates visual processing in the extrastriate cortex." Science 229.4715 (1985): 782-784.
- [42] Fei-Fei Li, Justin Johnson and Serena Yeung. CS231n. Class Lecture, Topic: “Convolutional Neural Networks for Visual Recognition.” Stanford University, Stanford, California 94305, Spring 2017.
- [43] Beck, Joseph E., and Beverly Park Woolf. "High-level student modeling with machine learning." International Conference on Intelligent Tutoring Systems. Springer, Berlin, Heidelberg, 2000.

- [44] Spanhol, Fabio Alexandre, et al. "Breast cancer histopathological image classification using convolutional neural networks." *Neural Networks (IJCNN)*, 2016 International Joint Conference on. IEEE, 2016.
- [45] LeCun, Yann, Koray Kavukcuoglu, and Clément Farabet. "Convolutional networks and applications in vision." *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*. IEEE, 2010.
- [46] https://www.tensorflow.org/api_docs/python/tf/keras/initializers/HeUniform
- [47] Zou, Hui, and Trevor Hastie. "Regularization and variable selection via the elastic net." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67.2 (2005): 301-320.
- [48] Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." *The Journal of Machine Learning Research* 15.1 (2014): 1929-1958.
- [49] Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of Machine Learning Research* 12.Jul (2011): 2121-2159.
- [50] Zeiler, Matthew D. "ADADELTA: an adaptive learning rate method." *arXiv preprint arXiv:1212.5701* (2012).
- [51] Hey, Anthony JG, and Anne E. Trefethen. "The data deluge: An e-science perspective." (2003): 809-824.
- [52] Hecht-Nielsen, Robert. "Theory of the backpropagation neural network." *Neural networks for perception*. 1992. 65-93.
- [53] Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." *nature* 323.6088 (1986): 533.
- [54] LeCun, Yann, et al. "Backpropagation applied to handwritten zip code recognition." *Neural computation* 1.4 (1989): 541-551.
- [55] Lei Zhang , Fan Yang , Yimin Daniel Zhang, and Y. J. Z., Zhang, L., Yang, F., Zhang, Y. D., & Zhu, Y. J. (2016). Road Crack Detection Using Deep Convolutional Neural Network.
- [56] <https://keras.io/api/preprocessing/image/>
- [57]<https://medium.com/towards-artificial-intelligence/the-architecture-and-implementation-of-vgg-16-b050e5a5920b>
- [58] <https://en.wikipedia.org/wiki/ImageNet>
- [59] <https://en.wikipedia.org/wiki/Overfitting>

[60]<https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c>

[61] S Maitra, RK Ojha and K Ghosh. ‘Impact of Convolutional Neural Network Input Parameters on Classification Performance’ 2018 4th International, 2018.