ADMAS UNIVERSITY

SCHOOL OF POST GRADUATE STUDIES

DEPARTMENT OF COMPUTER SCIENCE

Automatic detection and recognition of Ethiopian Birr using Deep Learning

A Thesis Submitted to Department of Computer Science of Admas University in Partial Fulfillment of the Requirements for the Degree of Masters of Science in Computer Science

By

Daniel Asnakew

Admas University

Addis Ababa, Ethiopia

July, 2021

ACKNOWLEDGEMENT

First and foremost, praise and thanks to God, the Almighty, for showering His blessings on me during my research work, allowing me to successfully complete the research.

I would like to thank my advisor, Mr. Mulugeta Adbaru, for his support, encouragement, positive supervision, insightful comments, and suggestions during this research project, from whom I have learned a lot. His relentless motivation has been the most motivating factor for me in this research. He taught me how to conduct research and present my findings in the most straightforward and concise manner possible. Working and studying under his direction was a great honor and privilege. I am thankful for everything he has done for me.

This study would not have been possible without the assistance of the National Bank of Ethiopia (NBE), and I am grateful to the NBE's management and staff for providing me with access to the Ethiopian Birr dataset.

I am grateful to my classmates for sharing more knowledge and assisting one another in completing the course and final thesis.

Last but not least, I owe my gratitude to my family and Mr. Solomon Kulchi for their support and encouragement.

Daniel Asnakew

July, 2021

Addis Ababa

Ethiopia

ABSTRACT

In Ethiopia, financial institutions have implemented a variety of automated banking systems that use Birr recognition as their primary operation, making automated Birr recognition a hot topic. However, following a study of the literature on Birr recognition, it was discovered that no methods for recognizing newly published Birrs had been introduced or proposed. Ethiopia's latest Birrs are being held up as a model for intelligent Birr recognition.

In this thesis, I investigated the Ethiopian Birr and created five (5) true Ethiopian Birr (such as 5-Birr, 10- Birr, 50- Birr, 100- Birr, 200- Birr) and three (3) false Ethiopian Birr (such as 50- fake Birr, 100- fake Birr, 200- fake Birr) datasets, which were then divided into three sets: 70 percent training set, 15 percent development set, and 15 percent test set. Then I trained three pre-trained network models (VGG19, ResNet50V2, and Inception-v3) on the training dataset, which accounts for 70 percent of the total dataset, then tuned the hypermeters to find the best performing model on the development dataset, which accounts for 15 percent of the total dataset, and finally tested the model's performance on the test dataset, which also accounts for 15 percent of the total dataset. For the VGG19 network model, I obtained a training accuracy of 98.44 percent, validation accuracy of 100 percent, and test accuracy of 99.65 percent, for the ResNet50V2 network model, I obtained a training accuracy of 98.44 percent, validation accuracy of 99.01 percent, and test accuracy of 97.3 percent, and finally for the Inception-v3 network, I obtained a training accuracy of 96.36 percent, validation accuracy of 99.88 percent, and test accuracy of 99.28 percent.

Keywords: Deep Learning, CNN, ResNet50-v2 networks, Inception-v3 networks, VGG19 networks, Ethiopian Birr

CHAPTER ONE

INTRODUCTION

The study's outline is presented in this chapter. It includes information about the study's background, problem statement, research questions, and objectives, as well as information about the study's significance, scope and limitations, and organization.

1.1 Background

Currency is an important component of our everyday life. How to describe real and false currencies, however, has become the most important problem at present. If we use a device to identify currencies, it will significantly increase the accuracy of identification and effectively reduce the workload of individuals. Automatic recognition of paper currency in its respective denomination, such as paper notes 5,10,50,100 and 200, with the ability to detect fraud currency, is necessary to automate the money transaction system and then to use the self-serving devices such as ATM to their fullest capacity intensively (Asfaw and Million, 2019).

Different researchers studied the banknote in the currency note recognition method by using the banknote as a standalone function or a combination of components such as colors, scale, texture, and shape properties. According to Thomas and Sreekumar (2014), the banknote's color, form, texture and size characteristics were considered to identify the banknote recognition system's nature. But, properly extracting an image's properties and features depends on the quality of the captured image. In most cases, images are affected by lighting in the process of capturing the object, fed up and brown out due to age variables, and lost some of its characteristics due to image degradation and noise that are some of the bottlenecks for the recognition system capabilities (Gogoi et al., 2014).

It is mostly observed that traditional machine learning approach for paper currency recognition system has been used by various researchers. It is most commonly seen in their studies among the learning algorithm to consider artificial neural network (Chambers, 2014). To recognize the currency, these traditional machine learning techniques involve manually extracting features. These functions, which are extracted manually, are subject to human biases and errors.

The structures of currency note identification and classification (Lee et al., 2017), earlier built for different countries, different image datasets are done, but rear research has been done on the Ethiopian currency note called Birr classification because there is no image dataset or model that classifies the Ethiopian banknotes using machine learning. Therefore, the main objective of my paper was to research in depth the Ethiopian currency notes and prepare a machine learning

model that could identify the Ethiopian currency note. We choose the appropriate neural network architecture for the model in the analysis, study and define the hypermeters that match the model and make the model perform effectively, and finally build a dataset, train the model using the dataset, and evaluate the model's output.

1.2 Statement of the problem

The government is currently announcing a new Ethiopian Birr in Ethiopia, although a number of counterfeit currencies have emerged as a result of technological advances. So, a lot of people in Ethiopia are trading commodities for fake Birr.

Currency is accepted almost everywhere (Andrii, 2018). Even though digital payments such as mobile banking and other electronic forms of payment are becoming more common, hand-to-hand cash transactions are still widely used in Ethiopia for day-to-day activities (Mekonnen and Hussien, 2020). As a result, many people are unable to distinguish between genuine and counterfeit Birr.

In addition, there are currently a number of automated transaction facilities in Ethiopia, such as automated teller machines (ATMs) and multifunctional counting machines. All over the world, in county banks, In particular, the automated teller machine is used by every person in the country (ET switch, 2019) in many places and at any time for customers to use its ease of use and accessibility, which makes life simpler for a number of people by allowing them to withdraw cash at any time they want near them, transfer money to the friend, family or business partner easily within sort time, but on the other hand, 80 percent of the country's population has their income from agriculture and small local low-income trades, and these populations are illiterates who do not have that much education, but still most of them use banks because banks are very widespread across the country, making them familiar with banks, especially the Commercial Bank Ethiopian (CBE).but the problem here is that because of not having enough knowledge security futures, these farmers identify currency with color and again the big problem is that most of the bank's ATM machines of all the banks in the country are only used to withdraw money, they are not used as the ATMs here to deposit money. So, it is the bank accountants' responsibility to deposit money into their account number in Ethiopia.

1. 3 Research question

The research question that will be addressed in the study is as follows:

1) Which CNN models (GoogLeNet, VGGNet, or ResNet) perform significantly better at recognizing Ethiopian Birrs?
2) What are the most suitable hyperparameters for designing models?
3) How will the required dataset be prepared for the task of using Birr detection and recognition?

1.4 Objective of the study

There are general and specific objectives of the study.

1.4.1General Objective

The main aim of this research is to design automatic detection and recognition of Ethiopian Birr using deep learning.

1.4.2 Specific objective

We stated the research needs to address these specific objectives described below in order to achieve the general objective.

- To prepare the requisite training, validation and test datasets for Birr detection and recognition.
- To design that detects and recognizes Ethiopian Birr automatically.
- To test the efficiency of automated model detection and recognition.
- To choose modern convolutional neural network architecture that is suitable.

1.5 Scope and limitation of the study

The scope of these study is to study the Ethiopian currency note which is called Birr that contains currently 5 notes which are five(5), ten(10),fifty(50) ,one hundred(100) and two hundred(200) and select the appropriate convolution neural network architecture then prepare the model, develop datasets(training dataset, validation dataset and test dataset), train the model using the training dataset, find the appropriate hypermeters of the model by tuning different values using the validation dataset and finally test the performance of the final model using the appropriate metrics on the test dataset. Since the majority of security futures in Ethiopian Birr are available on the front side of Birr notes, this research focus primarily on the front side of Ethiopian Birr notes [see Appendix A].

1.6 Significance of the study

This thesis work aims to create a prototype that will classify the Ethiopian Birr in its respective denomination as a software solution and can be a great feedback and solution in the struggle against counterfeiting for the banking industry and society.

80 percent of the Ethiopian population lives by agriculture in current technology days a number of transactions are carried out using computers in Ethiopia and almost all of them are illiterate or with no or little education that can get cash and use bank accountants to organize and count their cash for them and all bank ATM machines and automated transaction facilities do not all in order to be incorporated with ATM machines or any automated transaction facility, they allow everyone to deposit cash using ATM machines or any automated transaction facility to identify notes that can help facilitate bank processes.

We want this research work to provide various advantages for the established target populations in the domain field, people involved in the circulation of Ethiopian paper and currency, and institutions that will benefit from self-serving devices such as bankers. In addition, potential researchers in the domain field will refer to the report. In order to satisfy the criteria of a successful carrier in the academic program, the researcher would also be the recipient.

1.7 Organization of the Study

This research is divided into five chapters. In chapter one, the research background, statements of the problem, goals of the study, scope of the study, limitations of the study, and importance of the study are all discussed in detail. The majority of the second chapter is dedicated to a study of the literature. This chapter provides a comprehensive discussion of deep learning and its role in this study. This chapter also contains related works on Ethiopian Birr notes as well as other countries' currency notes. In chapter three, we analyze the methodology we used to complete the analysis. In the fourth chapter, the experimentation and data analysis for the research are discussed. The findings of the classification experiments were also discussed. Finally, chapter five summarizes the study's conclusions and provides ideas for future research.

CHAPTER TWO

LITERATURE REVIEW

This chapter examines various pieces of literature that are relevant to the study's goal. It covers topics such as biological neural networks, artificial neural networks, deep learning, and deep learning architectures, as well as Optimization Algorithms for Deep Learning, Image Augmentation, how a Computer Sees an Image, and related works.

2.1 Biological neural network

The basic operating organ of the nervous system is the neuron. The neuron is a simple cell with specialized structures that enable it to carry out complex functions. The most significant function of a neuron is to transform a chemical signal to an electrical signal. A biological system is depicted in Figure 2.1. Three main components make up a neuron :(1) dendrites, which transport input signals to the cell body which assess the relationship's strength (2) the cell body (soma), which gathers and stores information, and (3) the axon, which transmits the output signal to the axon, which processes the weighted input signals, other neurons that are linked to it (Samarasighe, 2006).

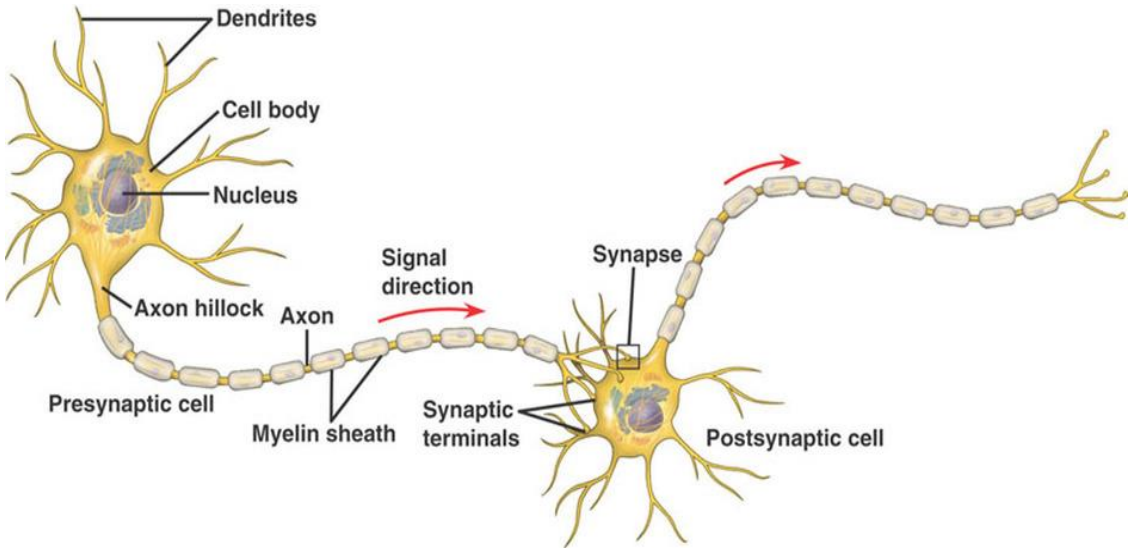


Figure 2.1: Structure of Neuron (https://mikerbio.weebly.com/structure--function.html).

The flow of information in a neural cell:

- Dendrites are activated by the behavior of other neurons.
- The incoming activations are processed by Soma and converted into output activations.
- Axons serve as transmission lines for signal transmission between dendrites and axons.
- Transmission is achieved by the diffusion of chemicals known as neuro-transitters.

2.2 Artificial neural networks (ANN)

One of the most widely used machine learning techniques is artificial neural networks. They are brain-inspired devices that are supposed to replicate how humans learn, as the “neural” half of their name suggests. Each layer is made up of multiple combinations of input, hidden, and output layers. An ANN, as shown in Figure 2.2, is made up of a number of neurons and weighting functions. 2011 (Naim et al.). Artificial neurons are stacked, having one or more hidden layers between the input and output layers, and they "forward" impulses. A number of neurons in each layer are connected to neurons in neighboring layers via unidirectional connections. Throughout the training period, information can only flow in one direction because to connections. To put it another way, a disguised layer is employed from the input layer to the output layer. It's almost certain that something will happen. The design has a number of layers that are hidden. The buried layer contains a synaptic weighting matrix, as do all synaptic weighting matrices. The weights are related by the connections created from the input layer to the hidden layer (Lee et al., 2007; Junfeng and Leping, 2010; Dey et al., 2008).

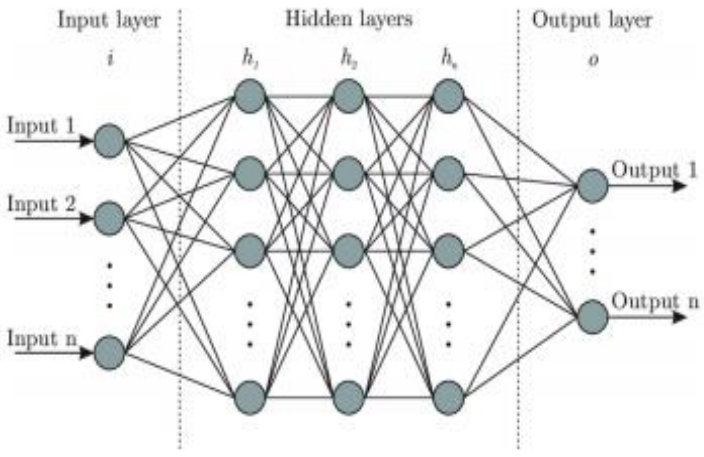


Figure 2.2: the artificial neural network with n inputs (https://www.researchgate.net/publication/321259051_Prediction_of_wind_pressure_coefficients_on_building_surfaces_using_Artificial_Neural_Networks).

2.2.1 Single layer perceptions (SLP)

A Single Layer Perceptron (SLP) is an arrangement of one input layer of neurons that is feed forwarded to one output layer of neurons, as shown in Figure 2.3. In SLP, the inputs are connected to the outputs directly through a series of weights, which is made up of a single layer of weights. Every input is linked to every output through synaptic links that carry weights, but the opposite is not true. A feed-forward network is what this is called (Martin et al., 1996). Networks can only distinguish linearly separable patterns, regardless of the form of pattern. The term "nonlinearity" refers to a form of nonlinearity that is commonly used. The classification of the pattern is needed for linear separation. Linear separation necessitates the classification of the pattern the decision boundaries must be sufficiently isolated from one another to ensure that they are hyper planes (Yeung et al., 2010).

$$f(x) = \sum_{i=1}^{n} w_i x_i + \beta = W^{\mathsf{T}} x_i \tag{2.1}$$

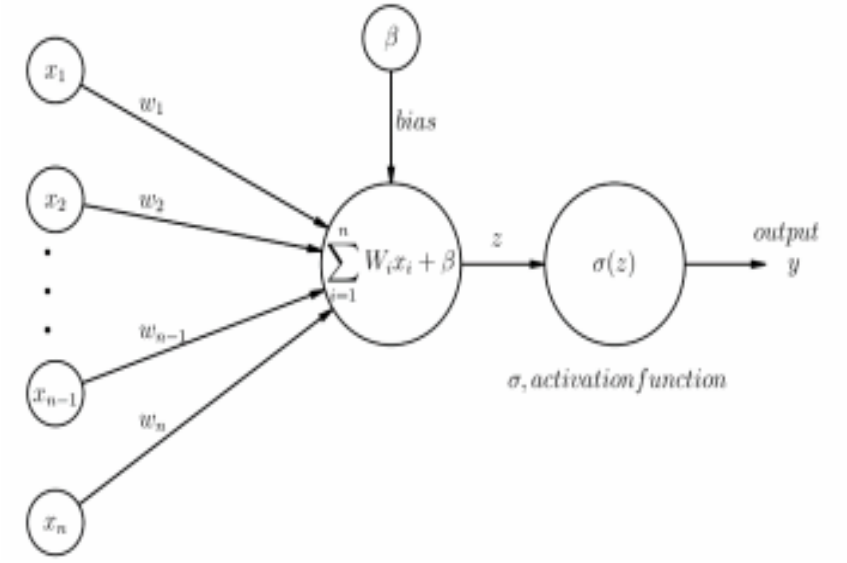


Figure 2.3: A Single Layer Perceptron with n inputs (Paul, 2019).

2.2.2 Multilayer Perceptron (MLP)

This form of network has one or more intermediate layers known as hidden layers in addition to the input and output layers, as shown in Figure 2.4. The Hidden layer's computational units are called hidden neurons (Martin, Howard and Mark 1996). Each neuron is completely connected to all neurons in the layer before it, as well as all neurons in the layer after it (Yeung et al., 2010). MLP will deal with more complicated issues than SLP (Graupe, 2007). On the other side, preparation can be a challenge. However, in some cases, training can be more successful because it is possible to solve a problem that SLP cannot be equipped to solve correctly (Fausett, 1994).

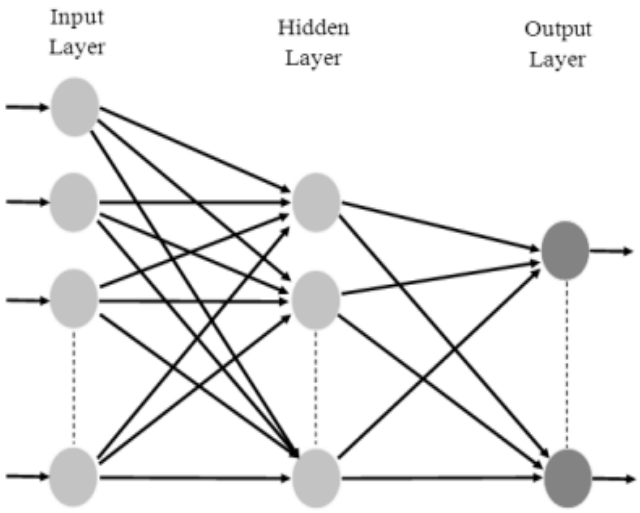


Figure 2.4: Multi-Layer Perceptron (MLP) (Yeung et al., 2010).

2.2.3 Artificial Neural Network Elements

A large number of neuron-like computing elements make up an Artificial Neural Network. There are a lot of weighted relations between all of these processing elements. A distributed representation of data is created by the relations between the elements. To gain information, a learning process is used. Now, let's discuss about ANN elements.

2.2.3.1 Artificial neuron

In an artificial neural network, an artificial neuron is a reference point. Artificial neural networks follow the same layered architecture as biological neural networks in the human body, with each network node (connection point) processing input and forwarding output to other network nodes. An artificial neuron is a connecting point (unit or node) that can process input signals and produce output signals (Brownlee, 2016).

2.2.3.2 Weights and Biases

The parameters w, which are altered during the learning process, dictate the weighting of a neuron's inputs. Each weight gives each neuron's particular input more power. When the weight is low, the genuine input has little impact on the neuron's output. It has a big impact in the opposite circumstance. A bias (β), which is officially the number 1 multiplied by a weight, is an

extra input to a neuron (Figure 2.3). It is a programmable parameter that regulates the total influence of the neuron.

2.2.3.3 Activation Function (AF)

In NNs, AFs are functions that calculate the weighted sum of inputs and biases, which determine whether or not a neuron can fire. It manipulates the presented data using gradient processing, usually gradient descent, to generate an output for the NN that contains the data's parameters. Early research findings by (Karlik and Olgac, 2011), validating categorically that a proper choice of AF enhances results in NN computing, have referred to these AFs as a transfer mechanism in the literature.

In a linear model, the hidden layers perform the linear mapping of an input function to output before predicting a class scores for each label. In most cases (Chigozie et al., 2020), where the input layer accepts the data for training the NN, it is computed using the affine transformation. Photos, videos, texts, speech, sounds, or numeric data, as well as transformations it is then transformed into x vectors, whose transformation is given by

$$f(x) = w^T + \beta \tag{2.2}$$

The AFs convert these linear outputs into non-linear outputs for further propagation in order to grasp the data patterns, where x represents input data, w represents weights, and β represents biases, and the NNs generate linear outcomes from the equation 2.2 mappings. The outputs are generated using the inputs

$$y = (w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + \beta) \tag{2.3}$$

For multilayered networks like DNN, these outputs are fed into the next layer before the final output is obtained. The expected performance, on the other hand, decides the form of AF to use in a given network. In contrast, since the outputs are linear, a nonlinear AF is required to convert them. They are converted to non-linear outputs. After applying the AF, the non-linear output is given by

$$y = \alpha(w_1 x_1 + w_2 x_2 + .. + w_n x_n + \beta) \quad (2.4)$$

Where α is the AF. For deeper networks, the AF helps with the learning of high order polynomials beyond one degree (Chigozie et al., 2020).

Summary of the major activation functions

This section examines the various forms of AFs as well as how they've evolved over time. Deep architectures, which are used in a number of applications, have been a key research area for AF research and applications. The findings of the SOTA analysis are summarized below; however, it is important to note that this summary does not follow a sequential order, but instead starts with the main functions and their associated changes as variants, followed by the combined derived AFs (Chigozie et al., 2020). The following is a list of these functions:

A. Sigmoid Function

Figure 2.5a depicts the sigmoid function, which is a typical non-linear activation function. This function's performance is bounded, and it was commonly used as the activation function in deep neural networks during deep learning's early days. Even though the in neurology, the sigmoid mechanism has synapses, which are similar to the synapses of neurons. Despite the fact that the derivative of this function is simple to obtain, the function is seldom used nowadays due to its complexity. Because of its weaknesses (Yingying et al., 2020). This function has the characteristic of soft storability based on the sigmoid function's curve. When the input is very high or very small, the slope of the graph tends to be zero. When the function's slope is near zero, the gradient that passed to the underlying network becomes very small, which will make network parameters difficult to be trained effectively. Meanwhile, only one weight path has been changed. The direction of this function's output is always positive, which has an effect on convergence a rate. The sigmoid function has the following formula:

$$f(x) = \frac{1}{(1 + e^{-x})} \quad (2.5)$$

B. Hyperbolic Tangent Function (Tanh)

The tanh function is an improved version of the sigmoid function on the spectrum, which is a symmetric function with a zero-centered middle. It has a bounded output and adds nonlinearity to the neural network. Figure 2.5b depicts the curve of this function. The convergence rate is faster than the sigmoid function, but gradient diffusion is still an issue (Yingying et al., 2020). The tanh function has the following formula:

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2.6}$$

C. Rectified Linear Unit (ReLU) Function

The ReLu function, which is a piecewise function, is the most common activation function in neural networks. This function's curve is depicted in Figure 2.5c. If the value of the input is less than or equal to zero, the output will be zero, as shown by the curve, otherwise the same as the input value. To some degree, the method of explicitly forcing some data to be zero will produce a moderately sparse characteristic. When compared to the previous two functions, the ReLu function is significantly faster. Unlike sigmoid and tanh functions, there is no gradient diffusion problem with ReLu because it is unsaturated. Although the ReLu feature has many benefits, it also has some drawbacks. Since the derivative of the ReLu function is always zero when the input value is negative, when a neuron with a wide gradient passes through the ReLu function, it is likely to cause neuronal necrosis, which will affect the final recognition result (Yingying et al., 2020). This function's equation is as follows:

$$f(x) = max(0, x) = \begin{cases} x_i & if\ x_i \geq 0 \\ 0, & if\ x_i < 0 \end{cases} \tag{2.7}$$

D. Softplus Function

The softplus function is identical to the ReLu function, as seen in Figure 2.5d. The difference between the softplus and ReLu functions is plainly visible in the curve of this function. This feature has some reservations about values less than 0 because it reduces the likelihood of

neuronal death. This function, on the other hand, requires substantially more work than the ReLu function (Yingying et al., 2020). The formula for this function is as follows:

$$f(x) = \log(1 + exp^{x})$$

(2.8)

E. Softmax Function

Another form of activation function used in neural computation is the Softmax function. It's used to make a probability distribution out of a collection of real numbers. The performance of the Softmax function is a set of values between 0 and 1, with the sum of the probabilities equal to 1. The relationship is used to calculate the Softmax function (Goodfellow et al., 2016).

$$f(x_i) = \frac{exp\ (x_i)}{\sum_j exp\ (x_j)}$$

(2.9)

In multi-class models, the Softmax function returns probabilities for each class, with the target class having the highest probability. The Softmax function can be found in the output layer of almost any deep learning architecture (Krizhevsky et al., 2012; Badrinarayanan et al., 2015). The Sigmoid and Softmax AF are distinguished by the fact that the Sigmoid is used for binary classification while the Softmax is used for multivariate classification.

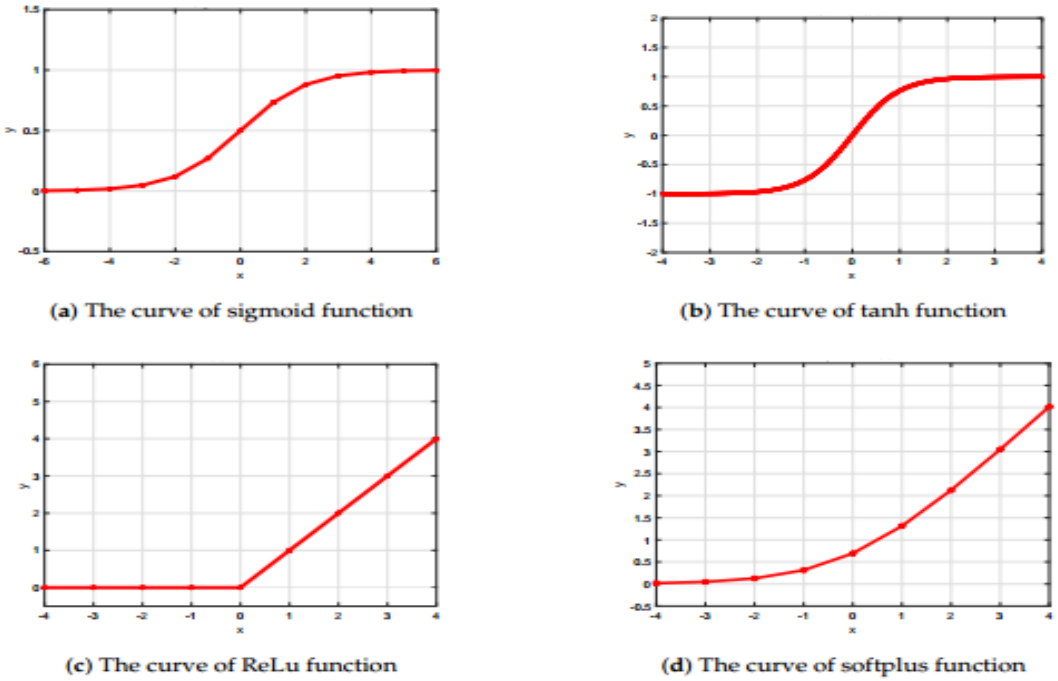


Figure 2.5: The graphs of four distinct activation functions that are commonly used. The sigmoid function, tanh function, ReLu function, and softplus function are the names of the curve functions mentioned above (Yingying et al., 2020).

2.2.4 Type of Artificial Neural Network Propagation

In a neural network, there are two forms of propagation: forward propagation and backward propagation. Let us now focus at the two types of propagation.

2.2.4.1 Forward Propagation

Forward propagation, as the name implies, performs each neuron's function in each layer of the network before we find the Y hat or the output and calculate the cost for m number of examples in the training set. The operation performed by one neuron is shown in Figure 2.6 below.

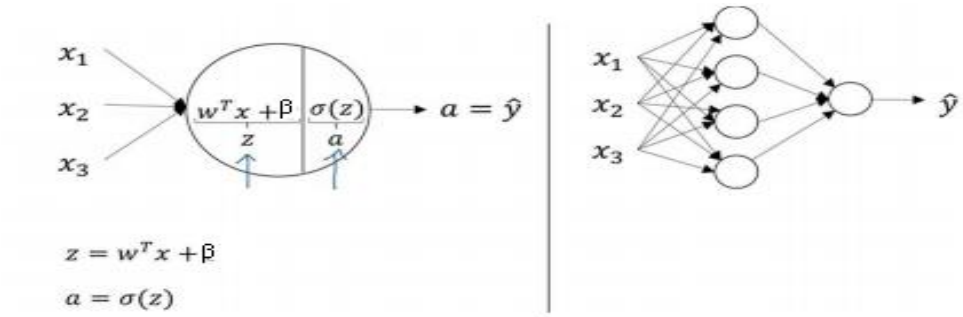


Figure 2.6: Operation performed in a single neuron unit

In Figure 2.6, the left figure depicts operations carried out in a single neuron. The first move is to figure out what $Z=W^TX+ \beta$ is, and the second is to perform the activation. We'll use the sigmoid function as an example, but any activation function, such as tanh, ReLu, or softplus, can be used instead. We can see how a large Neural Network is generated by layering a number of single neurons together to produce an output Y on the right side of the diagram. As a consequence, we use forward propagation to apply the operation we see on a single neuron to every neuron in each layer, going forward until we reach our Y.

The estimation and storage of intermediate variables (including outputs) for a neural network in order from the input layer to the output layer is referred to as forward propagation (or forward pass). We'll now go over the dynamics of a neural network with one hidden layer one by one.

2.2.4.2 Backward Propagation

The method of measuring the gradient of neural network parameters is known as backpropagation. According to the chain rule from calculus, the process traverses the network in reverse order, from the output to the input layer. While calculating the gradient with respect to some parameters, the algorithm stores any intermediate variables (partial derivatives) that are needed.

After the forward transfer, where inputs are passed through the network and output predictions are obtained, backpropagation begins. After getting the final loss value, it operates backwards from the top layers to the bottom layers, using the chain rule to figure out how much each parameter contributed to the loss value (Chollet, 2017). This is the backward pass, in which we

calculate the gradient of the loss function (which includes the loss function and optimizer) at the network's final layer and use the gradient to recursively apply the calculus "chain law" to update the weights layer by layer (Chollet, 2017). This process is also known as the weight update phase (Rosebrock, 2017). Since shifting weight values in the reverse direction has a cascading effect on weights in layers closer to the network's beginning, weights are changed layer by layer. Feature extraction entailed taking the previously-created convolutional base of the previously-created convolutional base of the previously-created convolutional base of the previously running new data through the trained network, and then training a new data classifier on top of it the result The dataset was run through the convolutional foundation, which generated a network feature array. The data was then fed into a stand-alone densely linked classifier (Chollet, 2017). After extracting features from these files; they were flattened before being fed into the densely-connected classifier. The densely-connected classifier was established, and it was trained on the data and labels that had just been registered, using the dropout method for regularization. A forward pass is used to train, and the loss is calculated. Backpropagation is used to update the weights after the loss has been measured.

2.3 Deep learning (DL)

Deep learning is a method for extracting features and patterns from data without the need for manual human intervention. It does so using a hierarchical stack of layers that store information in the form of weights. DL is a mathematical framework focused on the nervous system of a person (Mohammad et al., 2020).Deep learning's findings are so remarkable that we can use it in a number of daily applications such as YouTube, digital assistants such as Google Translate, which can translate text from any language to your native language, video games, medical picture diagnosis that outperforms humans, robotics for automated complex mechanical tasks, and, most importantly, self-driving cars (Shresth and Mahmood, 2019).

Manual feature engineering is used in traditional machine learning approaches, which necessitates the use of domain experts. In raw, unprocessed data, they have limited ability to recognize patterns and inferences (Abhijit, 2019; Arif and Mouchaweh, 2020). In comparison to DL methods, traditional machine learning strategies are labor-intensive, unreliable, and time-consuming. In both performance and accuracy, deep learning has exceeded traditional machine

learning architectures in Computer Vision (CV) and Image Processing (Abhijit, 2019; Zhaoqiang, 2019). The semantic relationship between data and its representation in a computer system can be discovered using DL architectures. Because of their ability to produce high-level data representations automatically, DL techniques are used as feature extractors and classifiers (Arif et al., 2020). Visual media processing and analysis are capable and effective for DLs (Muhammad et al., 2019).

Deep learning methods are classified as supervised, unsupervised, or semi-supervised (Mohamed et al., 2020; Shresth and Mahmood, 2019). In supervised learning, the learning algorithm is fed a pair of named inputs, and the corresponding target output is acquired (Darpan et al., 2019). The training dataset is generated independently of the learning algorithm. The two phases of supervised learning are the preparation phase and the evaluation phase (Sindhu and Suriya, 2019). The training process' goal is to predict and classify target data without knowing its name. The learning algorithm measures and adjusts prediction and classification accuracy until the performance is acceptable. A mechanism known as the objective function or loss function is used to determine such an output threshold. The learning algorithm's prediction accuracy is checked using unlabeled data in the testing process. The loss function aids in the reduction of prediction error (Schmarje et al., 2020). Supervised learning is used in the majority of deep neural network architectures used in image classification (Schmarje et al., 2020; Darpan et al., 2019).

Unsupervised learning does not necessitate the use of labeled input and output data pairs. Instead, the learning algorithm extracts similar patterns and features from unstructured training dataset types (Mohamed et al., 2020). The learning algorithm is trained to discover hidden features of unlabeled data in order to make a decision for the test dataset (Seyedali et al., 2020; Arif et al., 2020).

Semi-supervised learning is a form of learning that falls somewhere between supervised and unsupervised. It's a hybrid learning architecture that blends the best features of different learning architectures (Mohammad et al., 2020). In semi-supervised learning, the training dataset usually includes a small amount of labeled data and a large amount of unlabeled data (Jesper and Holger, 2020). To recognize the unlabeled data, the model must learn the hidden features of the labeled

dataset. Semi-supervised learning is more accurate than unsupervised learning, and it requires less time to complete than supervised learning (Mohammad et al., 2020).

2.3.1 What exactly does “deep” mean in the context of deep learning?

Your neural network's deep is determined by the number of layers it contains; the more layers you have, the deeper your network is; a shallow (simple) neural network has just a few layers. Since there are so many layers, preparation takes a long time.

Because of the increased availability of high-performance computing, deep learning techniques based on deep neural networks have become increasingly popular. When dealing with unstructured data, deep learning achieves greater power and versatility due to its ability to process a large number of features. The data is passed through multiple layers of the deep learning algorithm, each of which is capable of extracting features and passing them on to the next layer. Initial layers extract low-level attributes, which are then combined in subsequent layers to form a complete representation.

The performance of traditional machine learning algorithms and deep learning algorithms is depicted in Figure 2.7 (Palash et al., 2018). When traditional machine learning algorithms cross a certain amount of training data, their output become constant, while deep learning improves as the amount of data increases.

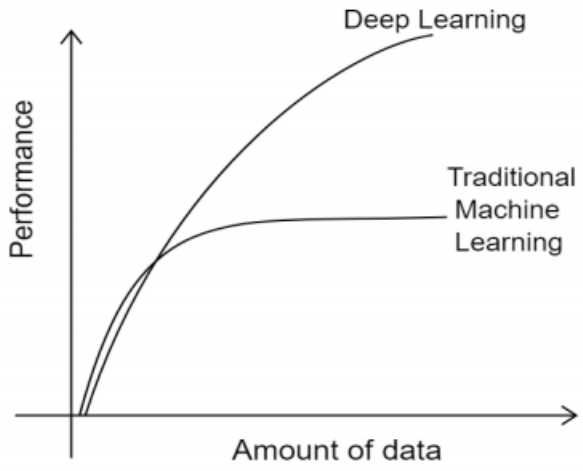


Figure 2.7: Why Deep Learning? (Palash et al., 2018).

2.4 Deep learning architectures

Due to the vanishing gradient problem, deep neural networks are difficult to train with backpropagation, which increases training time and decreases accuracy. The discrepancy between the Neural Network's expected performance and actual output in the training data is calculated as the cost function in Artificial Neural Networks. After each step, the weights and biases are changed based on the cost, the cost is kept to a bare minimum. The rate at which cost changes as a result of weights and biases is known as the gradient. Using deep learning techniques, the issue of vanishing gradient can be avoided. As a consequence, deep learning algorithms outperform traditional algorithms when working with large volumes of data (Savita and Ayesha, 2019). The parts that follow go into some of the most important deep learning architectures.

2.4.1 Recurrent Neural Networks (RNN)

RNNs (Recurrent Neural Networks) are FFNNs with feedback loops that can process sequential tasks (El-Ghazali, 2020). The feedback loops are cyclical in nature (Hojjat et al., 2018). They are supervised learning techniques. RNNs have temporal behavior that captures sequence dynamics through cycles in the computational graph and can adapt to dynamic data changes (Russell and Norvig, 2018). They have internal states that function as memories, allowing them to process time-dependent tasks (Yong et al., 2019). Backpropagation via time is used by RNNs, in which error signals are propagated backward in time (Hojjat et al., 2018). RNNs are used in a variety of activities, including machine translation, image captioning, music generation, video analysis, and natural language processing (Yann et al., 2015), where the processing must be sequential, such as in image captioning and language translation RNN is used (Savita and Ayesha, 2019). Each layer in a standard MLP has its own weights and biases, they can't be combined. The same weights and biases are used to blend these layers (Recurrent layer). This ensures that the neuron remembers its previous state and produces the next output based on that state.

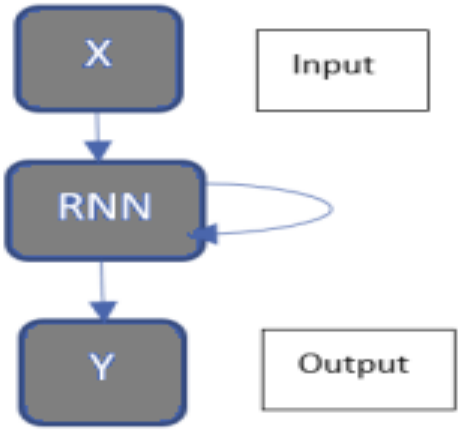


Figure 2.8: A typical RNN architecture

2.4.2 Transformer

The transformer is neural network architecture for language comprehension (Vaswani et al., 2017). For sequence modeling tasks, the transformer is commonly used (Sachin et al., 2020). It draws global dependencies between input and output using an attention mechanism. The word "attention system" was first coined by (Dzmitry et al., 2014). and is now commonly used in a variety of domains, including NLP, machine translation, and CV. Depending on the type of modeling technique used, attention mechanisms may take several different forms. The transformer employs a self-attention mechanism to compute a representation of a single sequence by relating different locations of the sequence. Encoder, decoder, attention system, point-wise feed-forward network, and embedding are the components of the transformer.

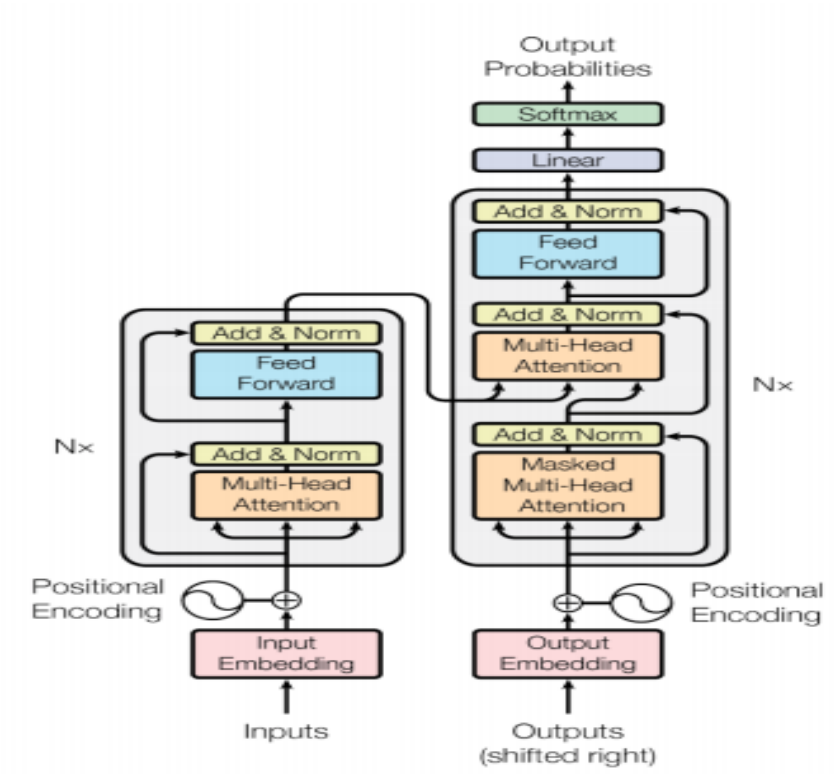


Figure 2.9: The transformer-model Architecture (https://arxiv.org/abs/1706.03762).

2.4.3 Deep Generative Models

Deep Generative Models are unsupervised and semi-supervised learning techniques that teach computers to produce new synthetic data (Taesung et al., 2019). To produce new data in any domain, they learn to capture the inner probabilistic distribution of individual groups (Oussidi and Elhassouny, 2018). Unlike FFNNs and CNNs, the aim of generative models is to predict feature maps given a name. They learn how a sample data set is distributed (in the case of an image, the distribution may be pixel intensity and color) and capture the sample's probabilistic distribution to produce similar data sets (Oussidi and Elhassouny, 2018).generative models estimate the distribution of given and the probabilistic distribution of denoted as p(y |x) and p(y), respectively, for a given and as input and target variables (Harshvardhan et al., 2020).Generative models are used in a wide range of data generation such as music, handwriting, faces, sketches, and artifacts of various shapes (Harshvardhan et al., 2020).

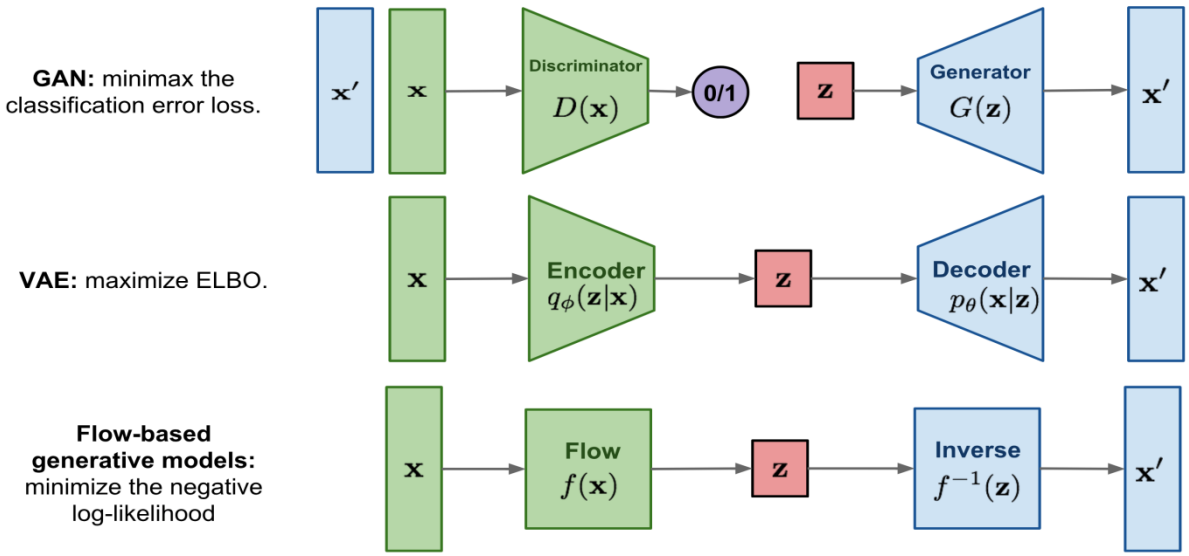


Figure 2.10: Comparison of three categories of generative models (Weng, 2018).

2.4.4 Restricted Boltzmann Machine (RBM)

The Restricted Boltzmann Machine (RBM) is a shallow two-layer neural network in which each neuron in one group is linked to each neuron in the other group without any internal connections. The visible (input) and hidden (output) layers of a neural network are the two classes (Savita and Ayesha, 2019).

The input data is reconstructed using RBMs that have been conditioned. Forward pass, backward pass, compare outcome to input is the RBM training technique. Every input is paired with its own weight and a single bias in forward pass. Each neuron is combined with a weight and an overall bias in the backward pass, and the output is passed to the visible layer for reconstruction. The visible layer's model is compared to the original input. This procedure is repeated with different weights and biases until the input and reconstruction are nearly identical.

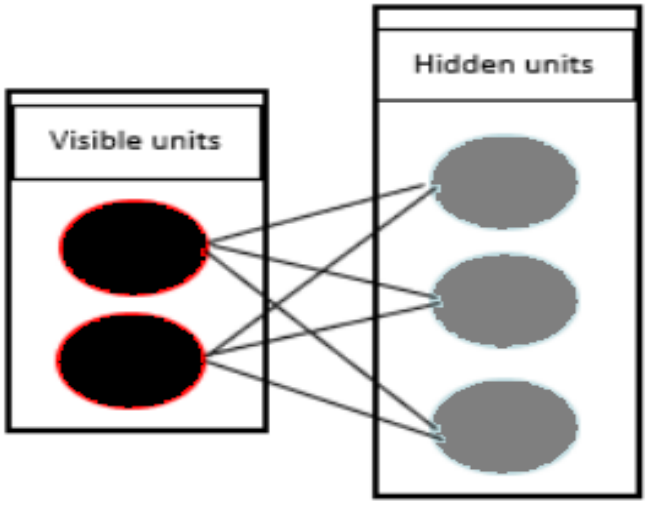


Figure 2.11: Architecture of RBM.

2.4.5 Autoencoders

By training the network to disregard signal noise, an autoencoder learns a representation (encoding) for a set of data. It also attempts to retrieve the original input from the reduced encoding (Savita and Ayesha, 2019). As the machine learns to ignore noise, the mechanism of regenerating the input aids in dimensionality reduction. An autoencoder can have as many hidden layers as it wants. Since both RBMs and Autoencoders attempt to replicate the input, they both promote unsupervised learning and are used in generative models.

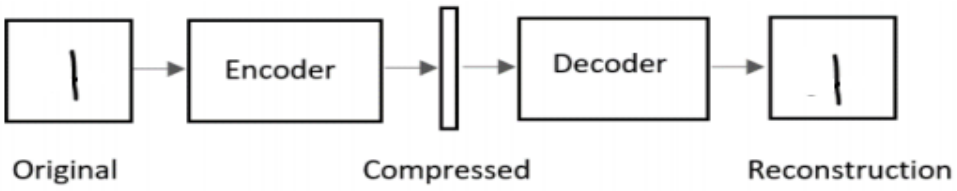


Figure 2.12: Architecture of Autoencoder.

2.4.6 Convolutional Neural Networks (CNN)

One of the most efficient algorithms in use today is convolutional neural networks. They primarily work with image and video data, with the aim of learning the image's "features" for applications like image classification, semantic segmentation, and a variety of others.

A CNN stands for a multi-layer neural network. The front-end input extracts image information using several layers of locally interconnected neurons, which is a key feature. Just a small portion of the image is made up of the feedback of the hierarchical structure's lowest layer neurons. After that, the data is transferred to different layers one by one. The neurons of each layer extract the most significant features of the observed data using a digital filter.

It completely accounts for the invariance of image object translation, rotation, and scaling in space. Its neurons are all the same structure, and it only accepts input from a few neurons in the previous layer's corresponding region. The neural network is designed in such a way that it not only maintains a broad front-end size, but also has a small number of variable adjustment parameters, reducing the computational load and the burden of parameter optimization.

CNN retains a low computational complexity compared to manual image pre-processing filtering and convolution as the input image signal size remains unchanged, and its front-end processing process has been optimized. Its weight-sharing network arrangement reduces the number of weights and the complexity of the network model. Since feature extraction is more specific for image content than manual pre-processing, it outperforms manual pre-processing and has a much higher recognition impact than traditional approaches (Saber, 1998).

CNN is one of the most widely used deep neural networks. Through the design of hidden layers in the network, the function and shape of the layer are changed to better correspond to information processing, and it is widely used in image recognition.

2.4.6.1 Convolutional neural network elements

To get a clear understanding of CNN, we'll start with the basics:

Stride

In fact, CNN has more options, which open up a lot of possibilities for further lowering the parameters while reducing some of the side effects. Stride is one of these options. By looking at the regions, it is clearly presumed that the next layer's node has a lot of overlaps with their neighbors in the example above. Controlling the stride allows one to manipulate the overlap. Figure 2.13 depicts a given 7×7 image. We can only get a 5x5 output if we move the filter one node at a time. It's worth noting that the outputs of the three left matrices in Figure 6 overlap (and three middle ones together and three right ones also). If we switch and make every stride 2, however, the result would be 3x3. Simply stated, not only will there be less duplication, but the output will also be smaller (Teilo, 2016).

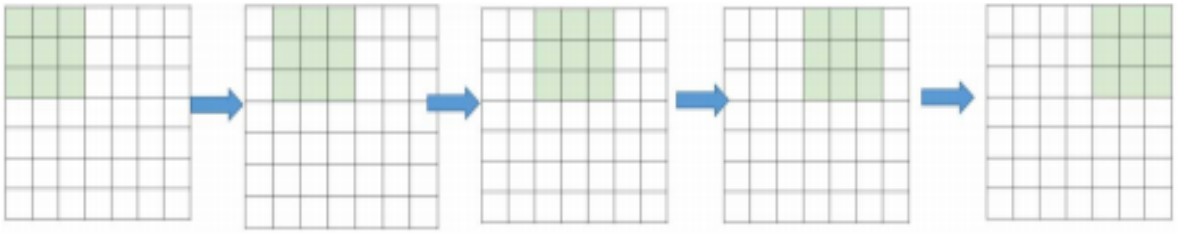


Figure 2.13: Stride 1, the filter window moves only one time for each connection (Antalya and Turkey, 2017).

Equation (2.10), formalize this, given the image $N \times N$ dimension and the filter size of the $F \times F$, the output size O as shown in Figure 2.14 (Antalya and Turkey, 2017).

$$O = 1 + \frac{N-F}{S} \tag{2.10}$$

Where N is the input size, F is the filter size, and S is the stride size.

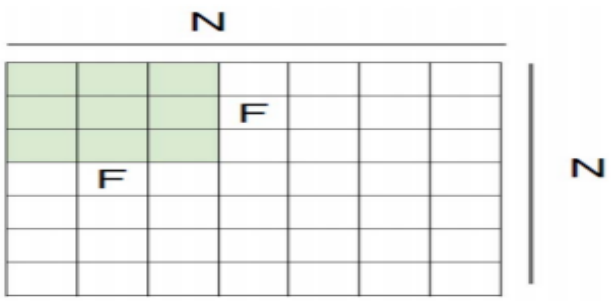


Figure 2.14: The effect of stride in the output (Antalya and Turkey, 2017).

Padding

The loss of information that may occur on the image's border is one of the drawbacks of the convolution phase. They never get a chance to be seen because they are just captured when the filter slides. The use of zero-padding is a very easy but successful way to solve the problem. The other advantage of zero padding is that it allows you to control the production size. For example, in Fig. 6, if N=7, F=3, and stride 1 are used, the output will be 5×5 (reduced from a 7×7 input) (Antalya and Turkey, 2017).

By adding one zero-padding, the output becomes 7×7, which is equivalent to the original input (the real N now becomes 9, use the formula) (2.10). Formula is a modified formula that includes zero padding (2.11).

$$O = 1 + \frac{N + 2P - F}{S} \tag{2.11}$$

This padding concept helps us prevent network output size from shrinking with depth, where P is the number of layers of the zero-padding (e.g. P=1 in Figure 2.15). As a result, any number of deep convolutional networks can be used (Teilo, 2016).

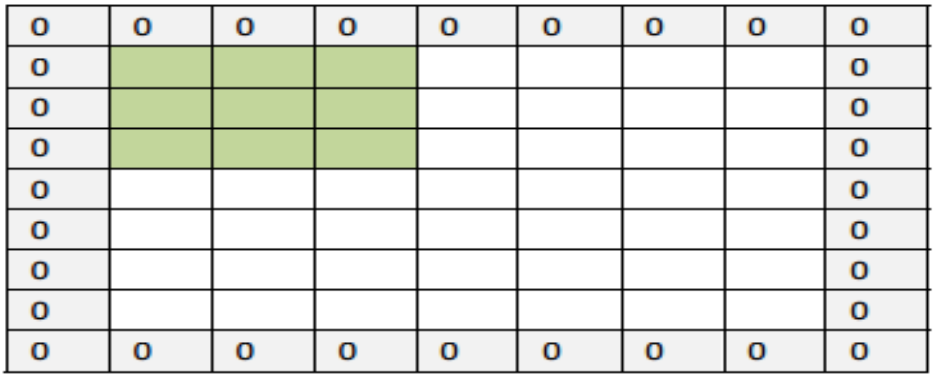


Figure 2.15: Zero-padding (Teilo, 2016).

Convolution

Convolution is the mathematical process of combining two functions to create a third one. Two sets of data are mixed when this happens. CNNs create a function map by adding a convolutional layer (also known as a filter or kernel) to the input data (Figure 2.14) (Saha, 2019).

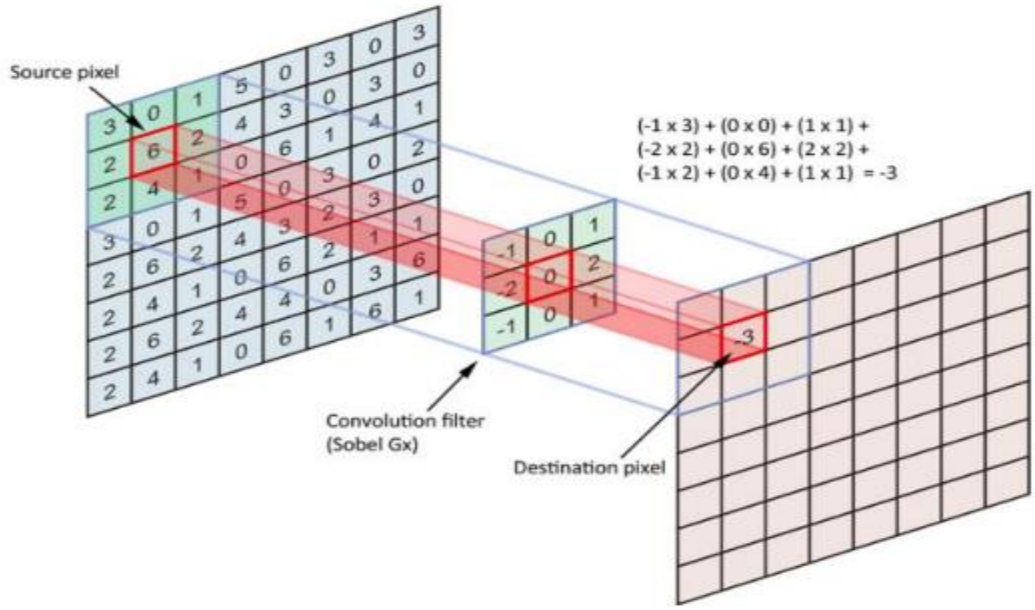


Figure 2.16: The filter slides over the input and performs its output on the new layer (Saha, 2019).

A dot product multiplication is performed in Figure 2.16 between a 3x3 filter matrix and a 3x3 region of the input image's matrix. The output value (“destination pixel”) on the feature map is the number of the elements of the resulting matrix. The filter then slides over the input matrix and completes the function map by repeating the dot product multiplication for each remaining combination of 3x3 sized areas.

2.4.6.2 Convolutional Neural Network Architecture

CNN is now regarded as one of the most widely used machine learning techniques, especially in vision-related applications. CNN can learn representations from grid-like data, and it has recently demonstrated significant performance improvements in a variety of machine learning applications (Asifullah et al., 2020). The architecture of all CNN models is the same (Figure 2.17) (Dertat, 2019) for feature generation and classification, CNN capabilities are used.

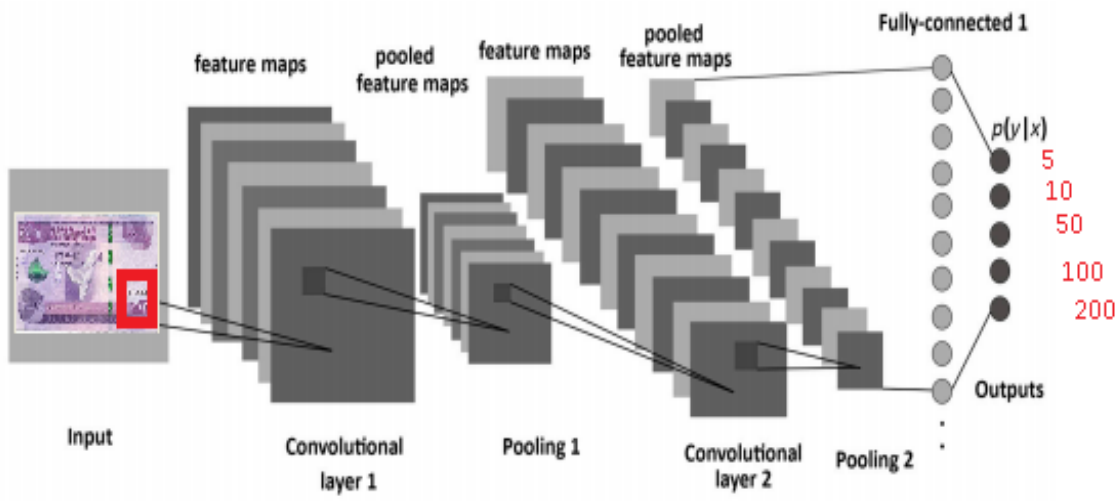


Figure 2.17: CNN architecture

A typical CNN architecture consists of convolution and pooling layers alternated with one or more fully connected layers at the end. A fully connected layer can be replaced with a global average pooling layer in some situations. Different regulatory units such as batch normalization and dropout are also implemented to improve CNN output in addition to different mapping functions (Bouvrie 2006). In order to design new architectures and achieve improved

performance, the arrangement of CNN components is important. The role of these components in CNN architecture is briefly discussed in this section.

Convolutional layer

Each neuron functions as a kernel in the convolutional layer, which is made up of a series of convolutional kernels. The convolution operation becomes a correlation operation if the kernel is symmetric (Ian Goodfellow et al. 2017). The image is divided into small slices, known as receptive fields, by the convolutional kernel. Extracting feature motifs is easier when an image is divided into small blocks. By multiplying its elements with the corresponding elements of the receptive field, the kernel convolves with the images using a particular set of weights (Bouvrie 2006).

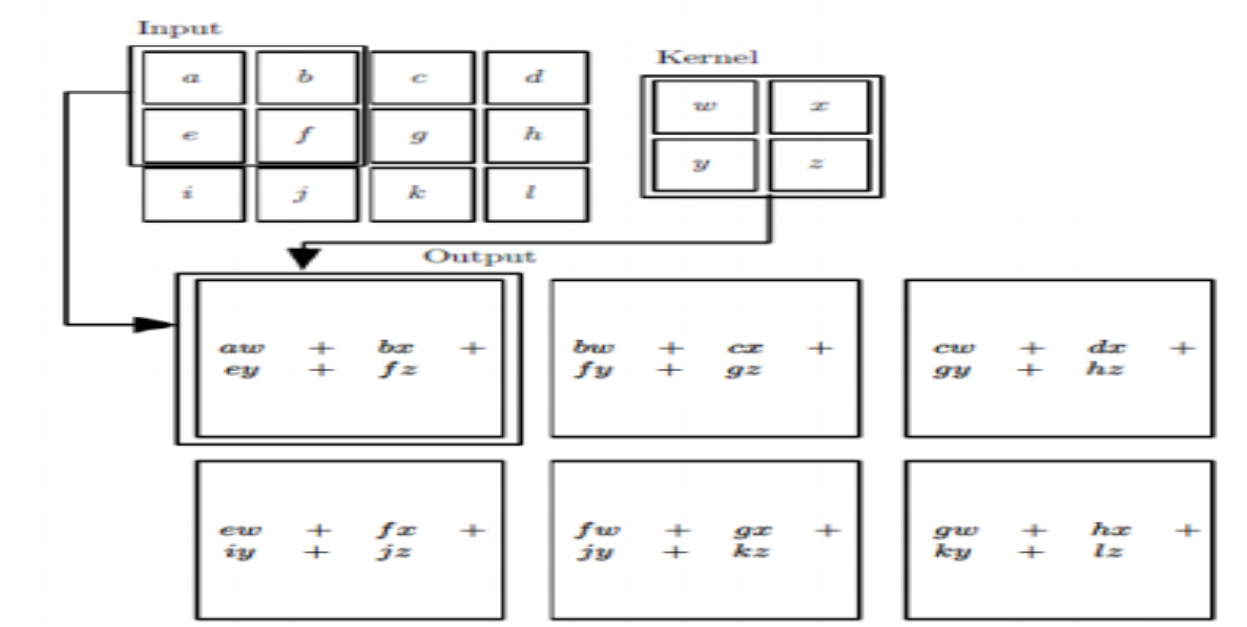


Figure 2.18: Feature Map (https://www.cnblogs.com/pinard/p/6483207.html).

In the convolution layer, the weight of each neuron connecting to the data window is set, and each neuron only looks at one feature. As shown in Figure 2.18, a feature map is the next neuron matrix generated by a receptive field scan with a convolution kernel. The weights and offsets in the convolution nucleus are shared since the neurons on the same feature map use the same convolution nucleus. You'll get different feature maps if you use different convolution kernels.

Pooling Layer

The pooling layer reduces the size of a function map and thus the number of parameters in a CNN model (Umberto, 2019). It also decreases the network's computation time. The pooling layer combines semantically related features into a single layer (Andriy, 2019). Using a down sampling mask, the size of the feature mask is reduced. A 2 x 2 down sampling mask, for example, would reduce the function map dimension by half (Umberto, 2019). There are various forms of pooling. Average pooling computes a filter's average value over a region in the image; whereas max-pooling computes the maximum values (Figure 2.19) (Andriy, 2019).

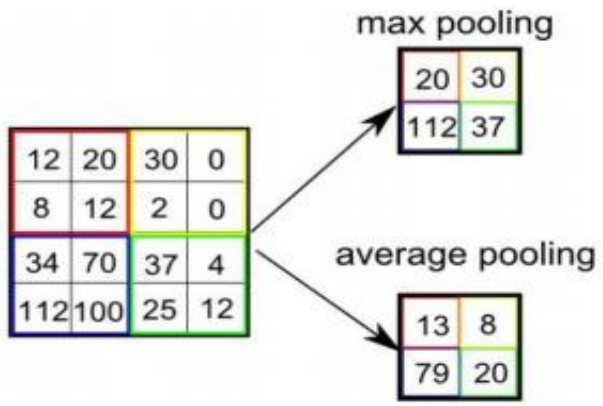


Figure 2.19: Types of pooling (Andriy, 2019).

Fully Connected Layer

FC is the final layer of CNN, and it is used to perform classification tasks. The scores of each class are computed after each unit of the FC layer receives input from the previous layer of feature maps (Ovidiu, 2020; Andriy, 2019). Backpropagation is then used to adjust the network parameters based on the scores.

Each fully connected layer (called a Dense layer) goes through an activation function (such as tanh or ReLu), but the output Dense layer goes through softmax. Cross Entropy (categorical crossentropy in Keras) is the loss function used in Softmax multiclass classification (Dertat, 2019).

2.4.6.3 Modern Convolutional Neural Networks

Convolutional Neural Network (CNN) models are multi-layer neural networks that are dedicated to recognizing the visual features of images based on their pixel structure. AlexNet, VGG16/19, GoogLeNet, and ResNet are some of the most common CNN models today (Ali and Mohamed, 2018). The most common models will be discussed in this section.

A. Deep Convolutional Neural Networks (AlexNet)

AlexNet, which used an 8-layer CNN, blew away the competition in the ImageNet Broad Scale Visual Recognition Challenge 2012. For the first time, this network demonstrated that learning-based features would outperform manually-designed features, breaking the previous paradigm of computer vision. AlexNet has eight layers, including five convolutional layers, two fully connected hidden layers, and one fully connected output layer. Second, AlexNet's activation mechanism was the ReLU rather than the sigmoid (Aston et al., 2020).

Architecture

The convolution window form in AlexNet's first layer is 11 × 11. Since most ImageNet images are ten times higher and wider than MNIST images, objects in ImageNet data appear to take up more pixels. As a result, capturing the object requires a wider convolution window. In the second layer, the convolution window form is reduced to 5×5, followed by 3×3.In addition, the network adds overall pooling layers with a window form of 3× 3 and a stride of 2 after the first, second, and fifth convolutional layers (Aston et al., 2020).

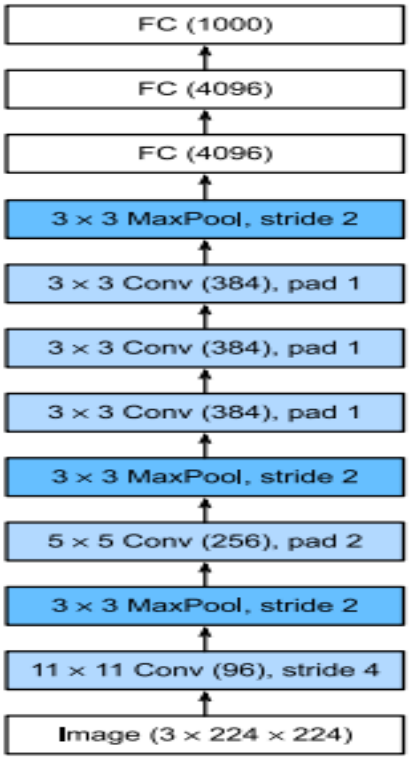


Figure 2.20: A block diagram representation of pre-trained AlexNet architecture (Aston, et al., 2020).

Activation Functions

Furthermore, AlexNet replaced the sigmoid activation function with a more straightforward ReLU activation function. On the one side, the ReLU activation function is easier to calculate. It lacks the exponentiation operation found in the sigmoid activation function, for example. When using different parameter initialization methods, the ReLU activation function, on the other hand, allows model training easier. Since the gradient of these regions is nearly zero when the performance of the sigmoid activation function is very close to 0 or 1, backpropagation cannot proceed to update any of the model parameters. In the positive interval, however, the gradient of the ReLU activation function is always 1. As a consequence, if the model parameters are not properly initialized, the sigmoid function may obtain a gradient of nearly 0 in the positive interval, making it difficult to train the model effectively (Aston et al., 2020).

Capacity Control and Preprocessing

AlexNet uses dropout to keep track of the model complexity of the fully-connected layer. To further supplement the data, AlexNet's training loop incorporated a lot of image augmentation, such as flipping, clipping, and color changes. As a result, the model becomes more stable, and the higher sample size prevents overfitting (Aston et al., 2020).

Reading the Dataset

Despite the fact that AlexNet is trained on ImageNet in the paper, we use Fashion-MNIST here because even on a modern GPU, training an ImageNet model to convergence can take hours or days. One of the issues with using AlexNet on Fashion-MNIST directly is that its images are lower resolution (28 x 28 pixels) than ImageNet images. We resized them to 224 × 224 to make things work (not a good idea in general, but we do it here to stick to the AlexNet architecture) (Aston, et al., 2020).

B. Networks Using Blocks (VGG)

Researchers had progressed from thinking in terms of individual neurons to entire layers, and now to lines, repeating patterns of layers, in the design of neural network architectures. The Visual Geometry Group96 (VGG) at Oxford University came up with the concept of using blocks in their eponymous VGG network. Using loops and subroutines, any current deep learning system can easily implement these repeated structures in code.

VGG Blocks

A series of the following is the fundamental building block of classic CNNs: (i) a convolutional layer with resolution padding, (ii) a non-linearity such as a ReLU, and (iii) a pooling layer such as a max pooling layer. A VGG block consists of a series of convolutional layers followed by a spatial down sampling max pooling layer. The authors used convolutions with 3×3 kernels with padding of 1 (keeping height and width) and 2× 2 max pooling with stride of 2 in the original VGG paper (Simonyan & Zisserman, 2014) (halving the resolution after each block).

VGG Network

The VGG Network, like AlexNet, can be divided into two parts: the first, which is mostly made up of convolutional and pooling layers, and the second, which is made up of fully-connected layers. This is depicted in Figure 2.21.

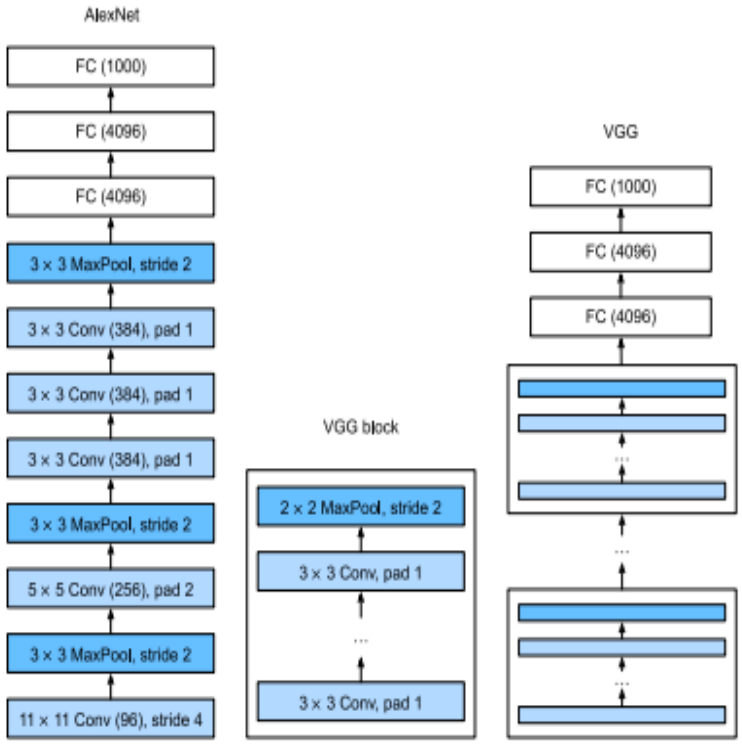


Figure 2.21: From AlexNet to VGG that is designed from building blocks (Aston, et al., 2020).

The original VGG network had five convolutional blocks, the first two of which each have one convolutional layer and the last three of which each have two convolutional layers. The first block has 64 output channels, and each subsequent block doubles the number until it reaches 512 output channels. VGG-11 is the name given to this network because it has eight convolutional layers and three fully connected layers.

C. Networks with Parallel Concatenations (GoogLeNet)

GoogLeNet won the ImageNet Challenge in 2014, proposing a system that incorporated the advantages of network in network and repeated block paradigms (Szegedy et al., 2015). One of

the paper's main objectives was to find out what size convolution kernels are best. After all, previous popular networks used options as small as 1 × 1 and as wide as 11 × 11 in their programming. The use of a mixture of different-sized kernels may often be beneficial, according to this article. In this segment, we'll implement GoogLeNet, a slightly simplified version of the original model that omits a few ad-hoc features that were introduced to stabilize training but are now obsolete due to the availability of better training algorithms.

Inception Blocks

In GoogLeNet, an Inception block is the most fundamental convolutional block. The inception block, as depicted in Figure 2.22, Inception block is made up of four parallel paths. The first three paths use convolutional layers with window sizes of 1 × 1, 3 × 3, and 5 × 5 to extract information from various spatial scales. In the middle two directions, the input is convolutional one-to-one, reducing the number of channels and hence the model's complexity. The fourth paths use a 3× 3 overall pooling layer followed by a 1 × 1 convolutional layer to adjust the number of channels. Padding is used in each of the four paths to ensure that the input and output are the same height and width. Finally, each path's outputs are concatenated along the channel dimension to form the block's output. The number of output channels per layer is one of the most commonly tuned hyperparameters of the Inception block.

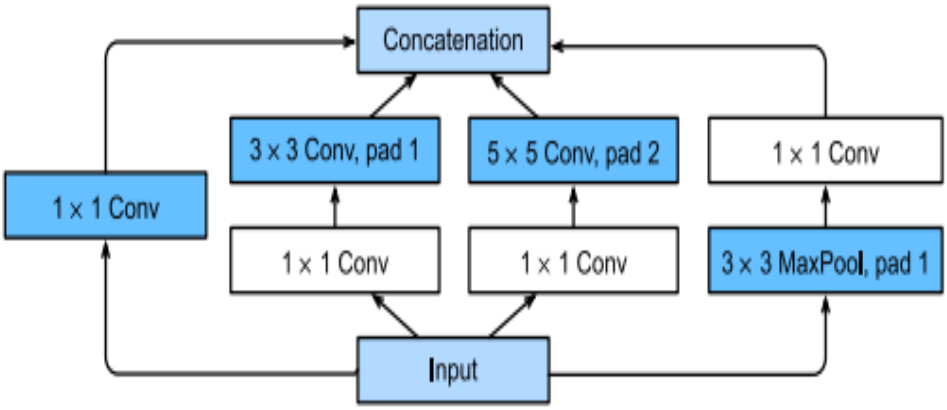


Figure 2.22: Structure of the Inception block (Aston, et al., 2020).

GoogLeNet Model

GoogLeNet generates its estimates using a stack of 9 inception blocks and global average pooling, as shown in Figure 2.23. The dimensionality is reduced by maximum pooling between inception blocks. The first module looks a lot like AlexNet. The stack of blocks comes from VGG, and global average pooling prevents a stack of completely connected layers at the top.

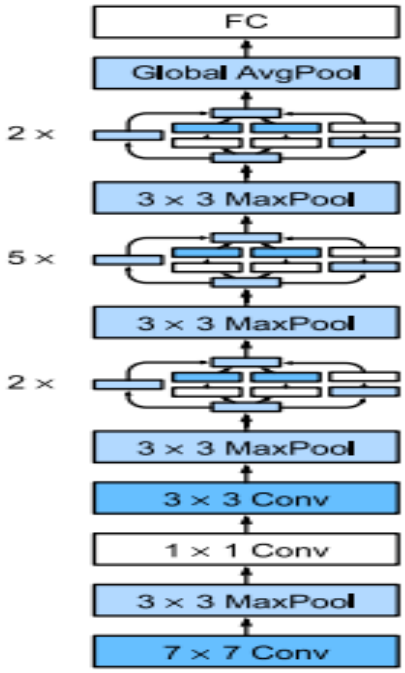


Figure 2.23: The GoogLeNet architecture (Aston, et al., 2020).

2.5 Optimization Algorithms for Deep Learning

In deep learning, adaptive gradient approaches are commonly used. While stochastic gradient descent (SGD) has been one of the most popular algorithms for many years, it struggles to resolve serious issues like ill-conditioning and the time requirement for large-scale datasets while training deep neural networks. It necessitates manual learning rate tuning and is difficult to parallelize (Le et al., 2011). As a result of the SGD issues, more sophisticated algorithms were developed. Deep learning optimization algorithms nowadays adapt their learning speeds during training. Adaptive gradient methods, in essence, change the learning rate for each parameter. As a consequence, when some parameters' gradients are high, their learning rates are reduced, or vice versa (Soydaner, 2020).

One of the most critical steps is determining which algorithm to use to optimize a neural network. There are three types of optimization approaches in machine learning. The first is batch or deterministic gradient methods, which process all training examples in a large batch at the same time. The second form is stochastic or online methods, which only use one example at a time. Most algorithms today are a mix of the two. They only use a portion of the training set at each epoch during training. Minibatch methods are the name for these algorithms. Minibatch methods are widely used in the deep learning period for two main reasons. To begin with, they speed up the training of neural networks. Second, since the minibatches are chosen at random and are unrelated, an unbiased estimation of the predicted gradient can be calculated (Goodfellow et al., 2016). The most commonly used minibatch-based adaptive algorithms are discussed in depth in this section.

2.5.1 Stochastic gradient descent (SGD)

SGD (Soydaner, 2020) basically descends the gradient of randomly chosen minibatches. To train a neural network with SGD, you must first compute the gradient estimate using a loss function. The update from iteration k is then applied to parameters. The following are the calculations for each minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ and their corresponding targets $y^{(i)}$:

$$\hat{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)}) \tag{2.12}$$

$$\theta \leftarrow \theta - \epsilon_k \hat{g} \tag{2.13}$$

The learning rate k is a critical hyperparameter in this case. The size of the update is determined by the learning rate. If it's too big, updates are overly reliant on recent instances. Convergence can necessitate several updates if it is limited. By trial and error, this hyperparameter can be selected. One method is to pick one of the learning rates that produce the smallest loss function value. This is referred to as a line quest. Another choice is to keep track of the first few epochs and use a learning rate that is higher than the best performing one (Soydaner, 2020).

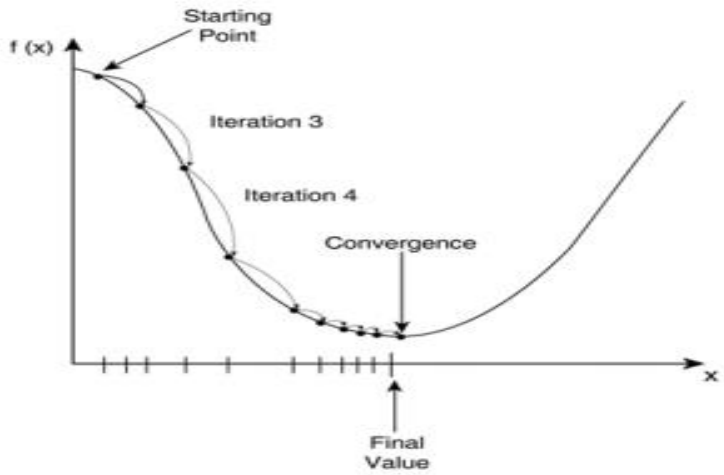


Figure 2.24: The gradient descent (McDonald, 2017)

2.5.2 AdaGrad - adaptive gradient algorithms

AdaGrad is an optimization algorithm that adapts the learning rates of model parameters individually. This is done by using all of the historical squared values of the gradient. Parameters with the largest partial derivative of the loss have a rapid decrease in their learning rate, while

parameters with small partial derivatives have a relatively small decrease in their learning rate (Goodfellow et al., 2016).

For gradient accumulation, AdaGrad adds a variable called r. The gradient accumulation variable is set to zero at the start of this algorithm, and the gradient is computed for a minibatch:

$$g \leftarrow \frac{1}{m}\nabla_\theta \sum_i L(f(x^{(i)};\theta), y^{(i)}) \tag{2.14}$$

The squared gradient is accumulated by employing this gradient. The update is then calculated by scaling all parameter learning rates inversely proportionate to the square root of the sum of all the gradient's historical squared values. Finally, the model parameters are updated as follows:

$$r \leftarrow r + g \odot g \tag{2.15}$$

$$\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g \tag{2.16}$$

$$\theta \leftarrow \theta + \Delta\theta \tag{2.17}$$

The global learning rate is €, and δ is a minor constant for numerical stability. AdaGrad, on the other hand, has significant drawbacks. It works well for small quadratic issues in general, although it frequently stops training neural networks too soon. The learning rate is reduced to the point when the algorithm stops completely before reaching the global optimum (Geron, 2017). Furthermore, while training deep neural networks, the accumulation of squared gradients from the start reduces the effective learning rate excessively. Certain deep learning models perform well with AdaGrad, but not all (Goodfellow et al., 2016).

2.5.3 RMSProp - Root Mean Square Propagation

RMSProp is another AdaGrad-modifying algorithm. It is proposed that changing the gradient accumulation into an exponentially weighted moving average improves performance in the non-convex setting. AdaGrad shrinks the learning rate based on the entire history of the squared gradient, as discussed in Section 3.2. RMSProp, on the other hand, uses an exponentially decaying average to discard history from the distant past, allowing it to converge quickly after

discovering a convex bow (Goodfellow et al., 2016). After computing gradient, the squared gradient is accumulated to implement RMSProp:

$$r \leftarrow \rho r + (1 - \rho) g \odot g \tag{2.18}$$

Where ρ is the rate of decay, then, as follows, the parameter update is computed and applied:

$$\Delta\theta = -\frac{\epsilon}{\sqrt{\delta + r}} \odot g \tag{2.19}$$

$$\theta \leftarrow \theta + \Delta\theta \tag{2.20}$$

2.5.4 Adam - Adaptive Moment Estimation

Adam is a well-known deep learning optimization algorithm. The name Adam comes from adaptive moment estimation, which uses estimates of the first and second moments of the gradients to compute individual adaptive learning rates for various parameters. Adam blends the benefits of AdaGrad for sparse gradients and RMSProp for wide gradients in one bundle, which is appropriate for both stationary and non-stationary settings (Soydaner, 2020).

Adam exhibits a number of significant characteristics. To begin, momentum is directly incorporated as an approximation of the gradient's first-order moment. Adam also applies bias corrections to both the first-order and second-order moments calculations to account for their initialization at the origin (Goodfellow et al., 2016). The algorithm updates the gradient m_t and the squared gradient u_t exponential moving averages, with the hyperparameters 1 and 2 controlling the exponential decay rates of these moving averages. The moving averages are estimates of the gradient's first raw moment (the mean) and second raw moment (the uncensored variance) (Kingma and Ba, 2014).

The first and second moment variables, m and u, are required by the Adam algorithm. At time phase t, biased first and second moment estimates are modified after calculating gradient:

$$m_t \leftarrow \rho_1 m_{t-1} + (1 - \rho_1) g_t \tag{2.21}$$

$$u_t \leftarrow \rho_2 u_{t-1} + (1-\rho_2) g \odot g \tag{2.22}$$

Then, bias is corrected in first and second moments. By using the corrected moment estimates parameter updates are calculated and applied:

$$\hat{m}_t \leftarrow \frac{m_t}{1-\rho_1^t} \tag{2.23}$$

$$\hat{u}_t \leftarrow \frac{u_t}{1-\rho_2^t} \tag{2.24}$$

$$\Delta\theta = -\epsilon \frac{\hat{m}_t}{\sqrt{\hat{u}_t} + \delta} \tag{2.25}$$

$$\theta_t \leftarrow \theta_{t-1} + \Delta\theta \tag{2.26}$$

Adam has a range of benefits. First and foremost, the learning rate must be fine-tuned. It is also an easy-to-use approach that is insensitive to gradient rescaling on the diagonal. It has a low memory footprint and is computationally efficient. Furthermore, Adam is well suited to non-stationary targets as well as problems involving very noisy and sparse gradients (Kingma and Ba, 2014).

2.6 Image Augmentation

Image augmentation technology increases the size of training datasets by making a series of random changes to the training images to create training examples that are identical but different. Another way to understand image augmentation is that randomly shifting training examples will help a model's generalization capacity by reducing its reliance on certain properties. We may crop the images in various ways, for example, so that the objects of interest appear in various locations, minimizing the model's dependency on the location of the objects. To reduce the model's sensitivity to color, we can change the light, color, and other variables (Aston et al., 2020). Let's take a look at a popular image enhancement method:

Flipping and Cropping

The object's category is normally not changed by flipping the picture left and right. This is one of the earliest and most commonly used image enhancement techniques. Furthermore, by randomly cropping the image, we can make objects appear in various positions and proportions in the image. This could also reduce the model's sensitivity to the target location.

Changing the Color

Changing colors is another form of augmentation. We can alter the color of an image in four ways: brightness, contrast, saturation, and hue.

Rotation

Rotation is the act of rotating an image at a certain angle. If the image is not square, rotating it will change its dimension, which will aid the network in learning images taken by the user at angles other than horizontal.

2.7 How a Computer See an Image

Each pixel in an image is measured by a number in a computer. A pixel, which is essentially a dot as shown in Figure 2.25, is the smallest element of an image. When an image is defined as having a resolution of 250*350 pixels, it means that it is 250 pixels wide and 350 pixels tall (Pokhrel, 2019). Color and gray scale images are the two types of images that can be classified based on their color (black and white).

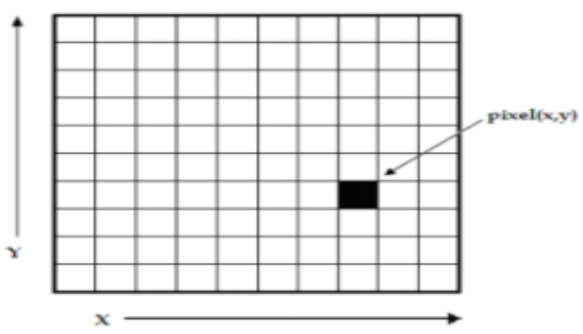


Figure 2.25: Color Pixel Visualization (Pokhrel, 2019).

2.7.1 Colored Images

The RGB color model is used to represent pixels in color images. RGB stands for Red Green Blue and denotes a three-component tuple. Each component has a range of values from 0 to 255, with (0, 0, 0) representing black and (255,255,255) representing white (Rao, 2019). Humans view all colors as different combinations of red, green, and blue. Figure 6 provides a few more color notation examples in the RGB model. When an image is depicted as 250*350*3, the last pixel represents the RGB tuple's three components. By applying transformation to the tuple values, we can convert a colored image to gray scale and vice versa.

2.7.2 Gray Scale Images

A single number represents the amount of light (or intensity) carried by each pixel in a grayscale (black and white) image. Most of the time, a range of intensities from 0 (black) to 255 (white) is used (white). Per number between 0 and 255 represents a different shade of gray. The Figure 2.26 shows a direct image of a gray scale image as shown by a machine. By Colored images can be converted to gray scale and vice versa by changing the RGB tuple components and vice versa (Upadhyay, 2019).

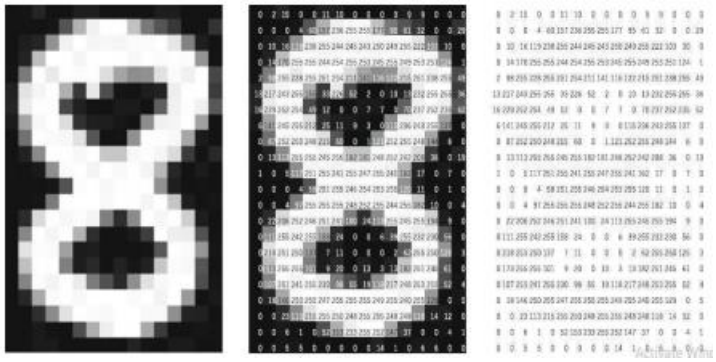


Figure 2.26: Gray Scale Representation of Images (Rao, 2019).

2.8 Related Works

This section's intent is to showcase other people's work that is similar to your own. It's possible Job on which you're basing your work or work that reveals how other people solved the problem

the same issue. We present a summary of the literature on the paper currency of various countries. This portion of the thesis paper addresses recognition mechanisms.

2.8.1 Indian Paper currency recognition

For visually impaired individuals, Anilkumar and Srikanth (2018) developed an Indian paper currency recognition system. Cash recognition, particularly for paper money, is one of the most important problems facing visually impaired individuals. The banknote image size, Euler number, and the correlation coefficient value of the test image with the benchmark image as a feature were considered in this study to be a feature vector and a Nearest Neighbors (k-NN) classifier to recognize Indian banknote image and, 88 percent and 100 percent, respectively, are the correctness of banknote discovery and banknote confidence acknowledgement. Although the work achieves better results in classifying the denomination of the banknote, the features used by the researcher to identify the banknote were not robust. In the case of old banknotes, the scale and Euler number characteristics are not applicable.

Another study conducted by Sawant (2016) implements a method of recognition and identification using Image Processing for Indian currency. The study looked at four significant currency recognition features, such as Dominant Color, Aspect Ratio, ID Mark, and Latent Picture. The work uses RGB color space to extract the dominant color, extract the aspect ratio / which is the width and height ratio of the paper note image, as well as use Fourier Descriptors to extract the shape feature of both the latent and the ID mark section. The calculation of similarity was used by using the Minimum Distance classifier as the classifier. Although the authors found a successful outcome, the function used to research the method of recognition of banknotes was not robust. The authors did not consider the spatial distribution of the paper note as the dominant color function attribute, but instead considered the overall greater color value of the note. Secondly, the paper note's size function attribute, the size value of the old and new note, is not identical, so it may not be strong enough to differentiate between paper notes. Finally, some of the limits are because the RGB color spaces are influenced by noise.

2.8.2 Pakistani paper currency Recognition

The Recognition Scheme for Pakistani paper currency was proposed by Ahmed Ali and Mansoor (2013). The authors aimed to offer an accurate and intelligent recognition system for Pakistani paper currency that uses image processing techniques to recognize distinct denominations and features such as scale, color, and pattern changes. The study's findings were said to prevent the purchase of expensive hardware for recognition and reduce human labor. The suggested system's overall construction was based on personal computers, scanners, and classifiers. In the research work, after the currency banknote image was retrieved from the scanner, pre-processing techniques such as noise reduction, RGB to gray conversion, and gray to binary conversion were performed. As an algorithm for the analysis, one of the instance-based learning algorithms called k-nearest neighbors (KNN) was selected and the number, height, width, aspect ratio, and area of Euler are recognized as features or characteristics to determine the authors' classification. From the training images, these features are extracted and stored as a MAT-le format in the database. The KNN classifier works by looking at the classification of unknown instances that will be performed in line with some distance/likeness output by comparing the unknown to the glorious. The proposed method typically involves four distinct methods, which are image acquisition, preprocessing, extraction of features, and classification. With the help of a scanner, they acquired a total of 100 images, meaning 20 of each currency note (10, 20, 50, 100, 500, and 1000).

2.8.3 Saudi Arabian Paper Currency Recognition System (SPCRS)

Sarfraz (2015) implemented a fully automated method of recognizing Saudi Arabia paper currency using Radial Basis Function Network with no human intervention method. The system proposed includes image acquisition, preprocessing, including elimination of noise, extraction of features, classification, and recognition. The study used 110 RGB input image containing deferred denomination to prepare the data collection. The proposed framework extracts the seven discriminative characteristics of the banknotes, such as the image height (in pixels), the image width (in pixels), the area of the image with the first mask (with the Prewitt Method), the area of the image with the second mask (with the Canny Method), the Euler number, the value of the correlation coefficient found between the sample and the test image/the degree of the image with the second mask (with the Canny Method) A Radial Basis Function Network is used in the

process. In terms of precision, the procedure is very fair. There is space, however, to increase the processing time.

2.8.4 US Dollar Currency Recognition System

(Aljutaili et al., 2018) proposed using the Speeded-up Robust Scale Invariant Feature Transform (SRSIFT) feature descriptor to recognize the US dollar. The authors merged SIFT and SURF to identify paper currency in their research, with the goal of overcoming the shortcomings of both SIFT and SURF algorithms. The SIFT algorithm has a number of drawbacks, including the fact that it is still very slow, takes a long time, and is ineffective for low-power devices. The SURF algorithm is a simplification of the SIFT algorithm. With the same SIFT properties; it is more resilient to scale shifts. The SURF algorithm has the following drawbacks: it is not rotationally stable and does not function properly. The SURF algorithm has the following drawbacks: it is not rotationally stable, and it does not work well with illumination. They suggested a feature detection algorithm in this paper to address the issues with both the SIFT and SURF algorithms. Image retrieval is part of the job. SIFT filter edge, interest point detection, SURF function detection, and matching point location The proposed SR-SIFT algorithm outperforms the SIFT and SURF algorithms, according to simulation results. The SR-SIFT improved the distribution of BKP on the surface and reduced average response time, particularly with a small and minimum number of BKP/ on best key point/. The proposed SR-SIFT algorithm outperforms the SIFT and SURF algorithms, according to simulation results. The SR-SIFT reduced average response time, especially with a small and minimum number of BKP/ on best key point/, and increased the distribution of BKP on the currency's surface area. Furthermore, compared to the other two algorithms, it improves the accuracy of the true BKP distribution within the currency edge.

2.8.5 Mexican Paper Currency Recognition

The main goal of (Garcia-Lamont et al., 2012) research was to propose an artificial vision-based system for classifying Mexican banknotes. Color and texture characteristics were used to classify the items. After the image was converted to grey-scale, the color feature was modeled using the RGB color space, and the texture feature was modeled using the Local Binary Patterns (LBP) process. The dominant color of each banknote was determined and interpreted using a unit

vector. LBP was a greyscale invariant texture primitive statistic that was very robust in terms of greyscale variations and computational simplicity, and was used to model the texture function.

The dominant color was extracted and interpreted in a vector after pre-processing deteriorated banknotes with a Wiener filter. Thresholding was applied to the central pixel and its neighbors for texture extraction, accompanied by histogram representation of the various local binary patterns. The histogram was transformed into a vector and normalized. The classifiers were given a combination of the color and texture features as input. A LVQ network classifier, which is a supervised version of vector quantization, was the first classifier. A classification model based on G statistics was used as the second classifier. G statistics is a non-parametric pseudo-metric that calculates the probability that a given sample belongs to one of many texture groups based on the exact probabilities of feature values from previously classified texture models.

The proposed model was said to have a short processing time and to fit well with highly deteriorated banknotes. Furthermore, the identification was invariant of image rotation, meaning that the banknote could be identified correctly regardless of image orientation.

2.8.6 Chinese Paper Currency Recognition

The main goal of (Zhu and Ren 2012)'s research was to suggest a method for recognizing RMB word numbers based on character features. The RMB is China's legal currency, and each banknote has a unique crown word number that identifies each note. Picture pre-processing, number region location, character segmentation, feature extraction, and character recognition are all part of the proposed identification scheme. As a pre-process operation, grey-scale scaling, image de-noising with a 5x5 median filter, and image binarization or thresholding were performed to reduce the processing time in the recognition process. The number position was omitted by using horizontal and vertical projection. The number position was obtained by using horizontal and vertical projection, and the picture tilt was corrected by determining the tilt angle. After that, the characters were normalized and segmented. The 8-direction gradient feature extraction method and the nearest neighbor algorithm were used for feature extraction and character recognition, respectively.

2.8.7 Ethiopia Paper Currency Recognition

Now let's see studies on Ethiopian banknotes (Asfaw, 2019) suggested in their research they suggested to automatically classify and recognize Ethiopia banknote Deep Learning Method for Ethiopian Banknote Denomination Classification and Fake Detection System, the study follows the key steps of image processing such as image acquisition, preprocessing, feature extraction and recognition. It captures and pre-processes the input banknote image. The banknote regions in the captured image are resized in this pre-processing step. The brightness normalization and noise filtering tasks were performed following image resizing. In addition, the conversion tasks for the gray scale were done. In the proposed CNN model, the preprocessed image of the banknote is fed in. In this study, we considered the banknote classification system to evaluate the performance of the feed-forward ANN, SVM, and KNN classifiers for the classification and verification outcome and analysis of banknotes. In this research, we examined the convolutional neural network as a feature extraction technique and as a classifier for the design of the Ethiopian banknote recognition system, the feed-forward artificial neural network. Using FFANN as a convolutional neural network classifier as a classifier with real scene images taken from the scanner, a high banknote recognition and classification rate was achieved. For the designation of the banknote denomination and 96 percent precision for fake currency identification, the study found an overall accuracy of 99.4 percent. However, the main goal of the study is to build a model using traditional machine learning algorithms such as SVM, KNN, and ANN.

In another study on' Automatic Recognition and Counterfeit Detection of Ethiopian Paper Currency' by Jegnaw and Yaregal (2016), they suggest hardware and software solutions for the recognition of Ethiopian paper currency in their study. There are three key components of the Ethiopian paper currency recognition system: currency image acquisition, currency denomination and currency verification. To capture the images of the currency using the scanner and camera, currency image acquisition is necessary. The part of the currency denomination is responsible for classifying the paper currency into the respective denomination based on the scanner-captured image of the currency, while currency verification checks whether the

particular paper currency is genuine or counterfeit based on the camera-captured image of the currency. The acquisition of currency images attributes the hardware solution to the problem as we use this part by using the scanner and camera to attenuate characteristic features of Ethiopian banknotes. On the other hand, currency denomination and verification components are software solutions to the problem and considered the four characteristic features of the banknotes as the discriminative features of banknotes, such as the dominant color, the distribution of the dominant color, the hue value, and speeded robust features (SURF) were extracted. Those features were involved in a four-level classification process in conjunction with local feature descriptors, as a classification task was performed each time one of the four features was extracted. Final verification tasks were carried out to verify the originality of the paper notes by segmenting test results showed that for genuine Ethiopian currency, the suggested design had an average recognition rate of 90.42 percent, with an average processing time of 1.68 seconds per banknote.

SIFT, GLCM, color momentum, CNN and a combination of SIFT, GLCM and color momentum techniques were proposed in another research study investigating the case of feature extraction, and Feed-Forward ANN as a classifier for the design of the Ethiopian paper currency recognition system Asfaw and Million (2019). The key phases of image processing, such as image acquisition using a scanner, pre-processing that is responsible for eliminating noise, converting RGB to gray scale and normalizing the size of the input banknote image to minimize the effect of the noise on the recognizer, This research study was followed by feature extraction that is responsible for extracting the descriptive characteristics from the given banknotes using the convolutional Neural Network (CNN) model and classification that was carried out using the feed-forward artificial neural network classifier. A total of 2400 banknote images were collected via the scanner in this research work to train and evaluate the proposed model and 70 percent, 15 percent and 15 percent were used for training, validation and testing, respectively.

But there are also a number of researches, not just the above studies, conducted in different countries around the world. Since many of the research works are trained and evaluated using a small number of datasets, here they have their limitations/gaps, and most of the classification studies of banknote recognition that are done use traditional methods of machine learning and some of them use neural networks to classify their country banknotes. And the Ethiopian

researcher also used old currency datasets but now the government declared a new Ethiopian currency.

CHAPTER THREE

METHODOLOGY

A summary of the research methodology is provided in this chapter. Research Methodology, Proposed System Architecture, Research Design, Dataset, Preprocessing, Model Utilization, Data sources, Model Evaluation, and Tools and Experiment Setup *are all included.*

3.1 Research Methodology

Study methodology is a way of systematically addressing the issue of research (Kumar and Ranjit, 2011). It allows researchers to understand which methods or techniques are important and which are not, what they would mean, what they would suggest and why. It also allows researchers to understand the principles underlying the different techniques and the parameters under which they can determine that a certain issue is important to certain techniques and procedures. In order to complete this research job, we follow the following procedures and methods.

3.2 Proposed system Architecture

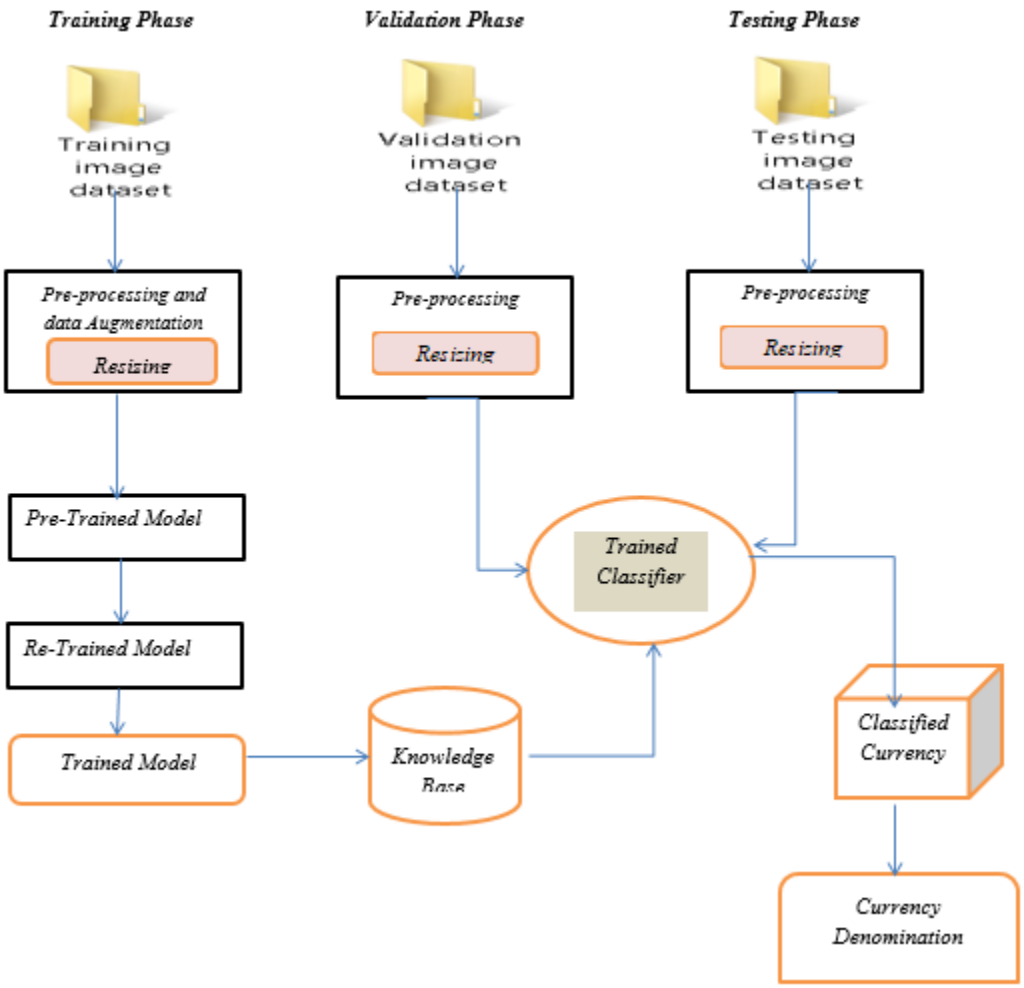


Figure 3.1: The proposed architecture of Automatic detection and recognition of Ethiopian Birr

3.3 Research Design

An experimental research design follows this research review. Since a scientific approach is accompanied by experimental testing, which involves a hypothesis, a variable that the researcher can control, and a variable that can be measured, estimated, and compared (Cash et al., 2016). In order to pick the descriptive currency note function, the analysis aims to classify the optimal descriptive currency note feature and classification techniques, giving the optimal recognizer obtained through experimental studies a better result. These also involved a thorough experiment, which includes analyzing the currency note by its texture, color, and the descriptive hierarchical function that gives a better result in classifying and recognizing the uniqueness of the currency note. We follow the step-by-step method in image processing for thorough experimentation, such as image acquisition, preprocessing, extraction of features and classification.

3.4 Dataset of the study

For training, testing and validation, the deep learning model needs a large amount of data. The information we collect is split into training, testing and validation, and all data passes through the preprocessing phase. The dataset is divided according to a standard law. The general rule is that if we have a large dataset (in the millions), we divide it into training, test, and validation using a ratio of 98 percent, 1 percent, and 1 percent, respectively, or even less for test and validation datasets (Moindrot and Genthial, 2018) Since we have big data, 1 percent of a big dataset is enough for testing and validation, but if we don't have big data, the percentages for training, test, and validation would be different, such as 60 percent, 20 percent, 20 percent, or 70 percent, 15 percent, 15 percent, respectively. Since we do not have a million data points in our dataset, but rather a few thousand, we will divide it into three parts: 70 percent for training, 15 percent for testing, and 15 percent for validation. The model is initially fitted into a training dataset, which is a collection of examples used to match the model's parameters (e.g. connection weights between neurons in convolutions neural networks). The fitted model is then used to predict responses in a second dataset for the observations called the validation dataset. The validation dataset allows for an objective assessment of a model that fits into the training dataset while tuning the model's

hyperparameters (for example, learning rate, batch size, epoch). Finally, the test dataset is used to provide objective measurements of a final model that fits into the training dataset.

3.5 Preprocessing

The recognition rate of the identifier can be significantly improved by the preprocessing phase. Unwanted defects occurring on the image are deleted in this portion of the proposed model. For the recognition of dirty, new, and very dirty currency images, the preprocessing steps of the suggested model are essential. The currency note regions in the captured image are resized in this pre-processing step, because the size of the CNN input image should be the same. The image augmentation and pixel normalization tasks were performed following image resizing. In addition, the pixel conversion to a numpy array tasks were completed.

3.6 Model Utilization

The time needed for training is one of the main drawbacks of convolutions neural networks for large-scale problems. It can take several weeks for the model to converge, even on modern GPUs. Fine-tuning based on the pre-trained network is among the solutions to this problem. For the new domain, a network which was previously fully trained for the specific dataset is reused and re-trained. The network such as VGG networks (Simonyan and Zisserman, 2014) or GoogLeNet (Szegedy et al., 2014) that is already trained on broad ILSVRC2014 datasets (Russakovsky et al., 2014) is very popular to take and fine-tune for another problem. In comparison to scratch training, this approach gives much faster results and a slightly lower error.

3.7 Data sources

For these studies, as a primary and secondary data source, we have two data sources as a primary data source. We will take a number of different pictures of all the Ethiopian Birr notes, so we have to first collect the Birr notes to take pictures of all those Birr notes, so we should collect various Ethiopian Birr notes from the Ethiopian Bank and my friends around the sub city of Akaki Kality. We went to collect Birr notes like dirty, new and really dirty, since my primary

data source is not enough, we will also use my secondary data sources here and we need different distribution of Birr notes.

3.7.1 Primary data sources

The consistency of the dataset is the major factor that influences the model from performing effectively. That means that the model's output is highly dependent on the consistency of the dataset that we will be using to train the model. We need a dataset for the distribution to have high quality dataset that we really want to implement the model; it's like setting our objective that we want to reach. We have to first collect the real and fake Birr notes with different characteristics such as very dirty, new and dirty using 3 mobile devices with different capture ability, dimension and image size as shown in Table 3.1 and Table 3.2.

No	Device	True Birr note type															Total
		5-Birr			10-Birr			50-Birr			100-Birr			200-Birr			
		Quality			Quality			Quality			Quality			Quality			
		Dirty	very dirty	new	dirty	very dirty	new	dirty	very dirty	new	dirty	very dirty	new	dirty	very dirty	New	
1	Device 1	550	550	550	550	550	550	550	550	550	550	550	550	550	550	550	8250
2	Device 2	550	550	550	550	550	550	550	550	550	550	550	550	550	550	550	8250
3	Device 3	550	550	550	550	550	550	550	550	550	550	550	550	550	550	550	8250
Total		1650	1650	1650	1650	1650	1650	1650	1650	1650	1650	1650	1650	1650	1650	1650	24750

Table 3.1: Total sample True Birr note images for the experimentation sample.

Fake birr dataset samples			
NO	Fake birr type	Amount in number	
1	50-birr	46	
2	100-birr	206	
3	200-birr	528	
Total samples			780

Table 3.2: Total sample Fake Birr note images for the experimentation sample.

3.7.2 Secondary data sources

As my secondary data source, we use two techniques: one is to collect high-quality images of currency notes from the internet, which distributes our dataset over various types of sources, thereby raising the quality of the dataset, and the second is that there are different techniques that help to increase the dataset used without collecting any dataset. These methods are referred to as image augmentation (Flip, Rotation, Scale and Crop), which is important for improving the performance of the model.

3.8 Model Evaluation

We evaluated the performance of the model using various possible inputs of Ethiopian Birr notes with various paper quality statuses in this study. The different hypermeter values we tuned up will be evaluated using scalar integer values for each hypermeter, and the output of the final model's cost function will be shown using graphs, and finally the tuned hypermeters for each model will be shown.

3.9 Tools and Experiment Setup

We will use Phyton programming language as a development method for these studies to build the model, NumPy library to perform complex mathematical operations and Neural Network System, which is the TensorFlow framework to develop and train the established Convolution

Neural Network model, and I also write some small programs to preprocess data such as the software that performs those data So the program must make the dataset images the same size because the dataset is a selection of images, however the network may not allow such image datasets that must be transformed to some readable type and those operations are often performed, however the usage of the basic software at the top is mentioned below.

- **Python:** -is an object oriented high-level programming language and is an integrated dynamic semantic developed by Guido Van Rossum in 1991, mainly for app and web creation. Here I used Python as a programming language for my model creation because python is the popular machine learning language for machine learning because most machine learning problems require strong math skills and because python is simple to learn than other programming languages, any trained professional can learn the language and solve the problem of machine learning.
- **TensorFlow: -** is also an open source library built for numerical operations by Google's brain team and is an easy-to-use Application Programing Interface (API) that can help us implement machine learning algorithms and enable us to run both the Central Processing Unit (CPU) and Graphics Processing Unit (GPU).
- **NumPy: -** is a python module that allows users to use the library to save development time and computational time by performing calculations performed on large multidimensional arrays and computing elements of each complex matrix.
- **Keras: -**is also a Python API specific to the high-level Neural Network and has the ability to run over other APIs or libraries such as TensorFlow. Its main objective is to provide fast testing and allows you to prototype quickly and easily.
- **Jupyter Notebook: -** is a web-based open source framework consisting of two types, such as an executable document or a computer program (code) written in programming languages such as python, which can be executed or executed to carry out a particular action and easily readable documents such as texts, paragraphs, headings, calculations, graphs, links, etc.
- **Web Browser:** -I need a web browser to open the Jupyter notebook program. The browser uses localhost to run on the local computer.

- **Computer Devices:** - Deep learning experimentations require high processor and GPU supported computing. In this study, we used a computer with a memory capacity of 8 GB RAM, 2.5 GHz processor, and 64-bit Windows 10 operating system.

CHAPTER FOUR

EXPERIMENTATION AND DATA ANALYSIS

The experimental data will be thoroughly investigated and explained in this chapter. This chapter will evaluate all models used for automatic Ethiopian Birr detection and recognition, as well as topics such as dataset preparation, pre-trained network model development, training, validation, and testing on pre-trained model and results, hyper-parameter fine-tuning, learning curve, and discussion.

4.1 Preparing the Dataset

To conduct the experimentation and achieve a better result and well-performing model for the classification of Ethiopian Birr, a large amount of data is needed. Deep learning methods, by their very nature, necessitate a large amount of data and are compatible with large datasets. Five (5) true Ethiopian Birr (such as 5- Birr, 10- Birr, 50- Birr, 100- Birr, 200- Birr) and three (3) fake Ethiopian Birr (such as 50- fake Birr, 100- fake Birr, 200- fake Birr) were used in this study [see Appendix B and C]. As shown in Figure 4.1, after the data is obtained, it is labelled with the corresponding class directory. The Ethiopian Birr images are saved as JPG files with a resolution of 3264 x 2448 pixels. Our datasets were divided into three categories: train, validation, and test. As shown in table 4.1, we used 25,530 images divided into 17,876 for preparation, 3,827 for validation, and 3,827 for testing with 75:15:15 ratios respectively. Create a separate directory for 5-Birr with only 5-Birr images, a separate directory for 10-Birr with only 10-Birr images, a separate directory for 50-Birr with only 50-Birr images, a separate directory for 100-Birr with only 100-Birr images, and a separate directory for 200-Birr with only 200-Birr images in each of the directories.

No	Dataset	Birr Note type								total
		Fake Birr Note			Real Birr Note					
		50- Birr	100- Birr	200- Birr	5- Birr	10- Birr	50- Birr	100- Birr	200- Birr	
1	Training set (70%)	32	144	370	3466	3466	3466	3466	3466	17876
2	Validation set (15%)	7	31	79	742	742	742	742	742	3827
3	Testing set (15%)	7	31	79	742	742	742	742	742	3827
total		46	206	528	4950	4950	4950	4950	4950	25530

Table 4.1 Collected data sampled

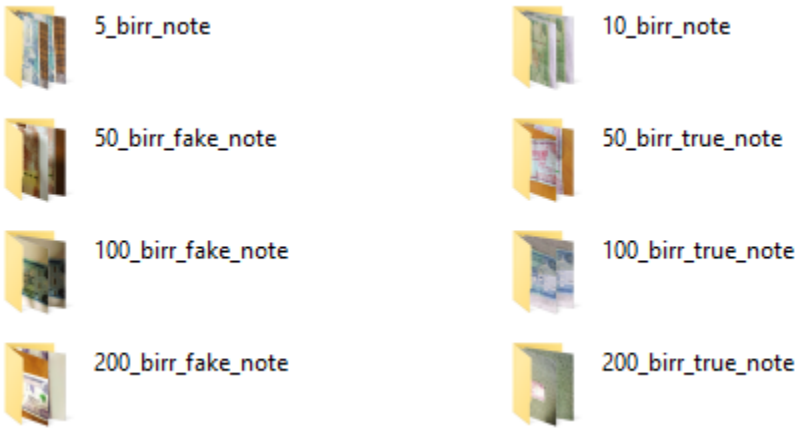


Figure 4.1 each class directory sampled dataset

4.2 Developing pre-trained network model

A pre-trained network model is one that has already been trained to solve a similar problem. Instead of starting from scratch to solve a similar problem, you start with a model that has already been trained on another problem. We can directly use the weights and architecture obtained and apply the learning to our problem statement by using pre-trained network models that have been previously trained on large datasets. This is referred to as "Transfer learning." The pre-trained network model's learning is "transferred" to our particular problem statement.

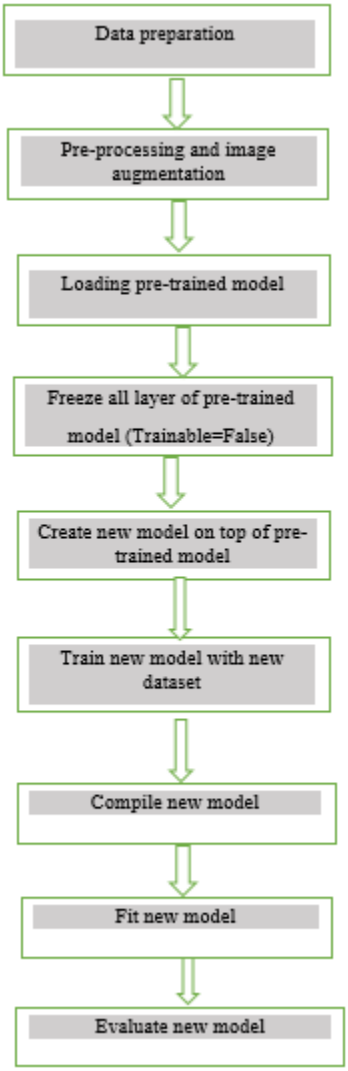


Figure 4.2 Workflow of pre-trained network model

Implementation of VGG19 Network Model

VGG19 is a VGG variant with 19 layers (16 convolution layers, 3 fully connected layers, 5 MaxPool layers, and 1 SoftMax layer), in short. The VGG team used this Keras model of the 19-layer network in the ILSVRC-2014 competition. The first step in implementing a VGG-19 pre-trained network model is to obtain pre-trained VGG-19 weights and architecture from the internet. Then, load a VGG-19 convolutional neural network that has already been trained and analyze the layers and classes.

```
In [3]: from tensorflow.keras.applications.vgg19 import VGG19
        base_model = VGG19(input_shape = (224, 224, 3),
                           include_top = False,
                           weights = 'imagenet')
```

```
In [5]: base_model.summary()
```

```
Model: "vgg19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
=================================================================
Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0
```

Figure 4.3 Model summary of VGG19 network model

Implementation of ResNet50_V2 Network Model

ResNet50V2 is a variant of ResNet50 that has been modified. It consists of a 50-layer convolutional neural network. The first step in implementing the ResNet50V2pre-trained network model is to obtain pre-trained ResNet50V2weights and architecture from the internet. Then load a ResNet50V2 convolutional neural network that has already been trained and analyze the layers and classes.

```
In [3]: from tensorflow.keras.applications import ResNet50V2

        base_model = ResNet50V2(input_shape=(224, 224,3),
                                include_top=False,
                                weights="imagenet")
```

```
In [4]: base_model.summary()
```

conv5_block3_2_conv (Conv2D)	(None, 7, 7, 512)	2359296	conv5_block3_2_pad[0][0]
conv5_block3_2_bn (BatchNormali	(None, 7, 7, 512)	2048	conv5_block3_2_conv[0][0]
conv5_block3_2_relu (Activation	(None, 7, 7, 512)	0	conv5_block3_2_bn[0][0]
conv5_block3_3_conv (Conv2D)	(None, 7, 7, 2048)	1050624	conv5_block3_2_relu[0][0]
conv5_block3_out (Add)	(None, 7, 7, 2048)	0	conv5_block2_out[0][0] conv5_block3_3_conv[0][0]
post_bn (BatchNormalization)	(None, 7, 7, 2048)	8192	conv5_block3_out[0][0]
post_relu (Activation)	(None, 7, 7, 2048)	0	post_bn[0][0]

```
Total params: 23,564,800
Trainable params: 23,519,360
Non-trainable params: 45,440
```

Figure 4.4 Model summary of ResNet50V2 network model

Implementation of Inception_V3 Network Model

Inception-v3 is a 48-layer deep convolutional neural network. The first step in implementing a pre-trained Inception-v3 model is to obtain pre-trained Inception-v3 weights and architecture

from the internet. Then explore the layers and classes of a pre-trained Inception-v3 convolutional neural network.

```
In [5]: from tensorflow.keras.applications.inception_v3 import InceptionV3
        base_model = InceptionV3(input_shape = (224, 224, 3),
                                 include_top = False,
                                 weights = 'imagenet')
```

```
In [6]: base_model.summary()
```

activation_85 (Activation)	(None, 5, 5, 320)	0	batch_normalization_85[0][0]
mixed9_1 (Concatenate)	(None, 5, 5, 768)	0	activation_87[0][0] activation_88[0][0]
concatenate_1 (Concatenate)	(None, 5, 5, 768)	0	activation_91[0][0] activation_92[0][0]
activation_93 (Activation)	(None, 5, 5, 192)	0	batch_normalization_93[0][0]
mixed10 (Concatenate)	(None, 5, 5, 2048)	0	activation_85[0][0] mixed9_1[0][0] concatenate_1[0][0] activation_93[0][0]

```
Total params: 21,802,784
Trainable params: 21,768,352
Non-trainable params: 34,432
```

Figure 4.5 Model summary of Inception_V3 network model

4.3 Training on pre-trained network model and results

The training stage of deep learning is when the model is gradually refined or when the dataset is learned. The aim is to learn enough about the training dataset's structure to make assumptions about data that hasn't been seen yet. Transfer learning is used to train the prepared Ethiopian Birr custom dataset using the pre-trained network models ResNet50V2, VGG19, and Inception V3. The same amount of custom dataset and hyper-parameter was used in each pre-trained network model. The dataset was trained using the Tensorflow framework, as stated in the software tools selection section. Before starting the actual training phase, a number of prerequisite steps must be completed, including image augmentation, pixel resizing, pixel normalization, pixel standardization, and pixel conversion to a numpy array. If you learn too much about the training dataset, your predictions will only work with the data you've seen so far and will be useless. Overfitting is the term for this problem, and image augmentation was used to solve it. ImageDataGenerator creates tensor image data in batches with real-time data augmentation. While the neural network model is learning on the training data, the flow_from_directory method allows you to read images directly from the directory and supplement them.

```
In [7]: train_generator = train_datagen.flow_from_directory(train_dir,
                                                          batch_size = 32,
                                                          class_mode = 'categorical',
                                                          target_size = (224, 224))
```

```
In [*]: train_datagen = ImageDataGenerator(rescale = 1./255.,
                                          rotation_range = 40,
                                          width_shift_range = 0.2,
                                          height_shift_range = 0.2,
                                          shear_range = 0.2,
                                          zoom_range = 0.2,
                                          horizontal_flip = True,
                                         featurewise_center=True,
                                          featurewise_std_normalization=True)
```

Training on ResNet50_V2 Network Model and Results

The training can be launched on the ResNet50V2 pre-trained network model after ensuring that the above essential steps have been completed correctly. As shown in Figure 4.6, the step count, loss and accuracy value for each step can be displayed on the screen. It should be remembered that the classification loss starts out high and progressively decreases as the algorithm learns over time. As you can see, with just 10 epochs, we were able to achieve a training accuracy of 98.44 percent.

Training on ResNet50V2 Network Model

```
In [9]: train_resnet = resnet_model.fit(train_generator, batch_size = 25, epochs = 10)

Epoch 1/10
645/645 [==============================] - 10941s 17s/step - loss: 0.9486 - accuracy: 0.8870
Epoch 2/10
645/645 [==============================] - 10551s 16s/step - loss: 0.0495 - accuracy: 0.9874
Epoch 3/10
645/645 [==============================] - 10165s 16s/step - loss: 0.0762 - accuracy: 0.9835
Epoch 4/10
645/645 [==============================] - 10312s 16s/step - loss: 0.0440 - accuracy: 0.9900
Epoch 5/10
645/645 [==============================] - 10449s 16s/step - loss: 0.1928 - accuracy: 0.9719
Epoch 6/10
645/645 [==============================] - 10490s 16s/step - loss: 0.1018 - accuracy: 0.9832
Epoch 7/10
645/645 [==============================] - 10366s 16s/step - loss: 0.0394 - accuracy: 0.9898
Epoch 8/10
645/645 [==============================] - 10483s 16s/step - loss: 0.0135 - accuracy: 0.9965
Epoch 9/10
645/645 [==============================] - 10569s 16s/step - loss: 0.0377 - accuracy: 0.9924
Epoch 10/10
645/645 [==============================] - 10015s 16s/step - loss: 0.0893 - accuracy: 0.9844
```

Figure 4.6: training performance with in 10 epochs of ResNet50v2 network model

Training on Inception_V3 Network Model and Results

The same thing happened when training custom dataset on Inception V3 model as it did with ResNet50_V2 model. On the Inception V3 pre-trained network model, the training can be launched. As shown in Figure 4.7, the step count, accuracy and loss value for each step can be displayed on the screen. It should be remembered that the classification loss starts out high and progressively decreases as the algorithm learns over time. As you can see, with just 10 epochs, we were able to achieve a training accuracy of 96.36 percent.

Training on Inception_V3 Network Model

```
In [34]: inception_history = model.fit(train_generator,  epochs = 10, batch_size = 25)

         Epoch 1/10
         645/645 [==============================] - 3112s 5s/step - loss: 3.2094 - accuracy: 0.7499
         Epoch 2/10
         645/645 [==============================] - 3605s 6s/step - loss: 0.2085 - accuracy: 0.9290
         Epoch 3/10
         645/645 [==============================] - 2577s 4s/step - loss: 0.1632 - accuracy: 0.9458
         Epoch 4/10
         645/645 [==============================] - 2916s 5s/step - loss: 0.1376 - accuracy: 0.9534
         Epoch 5/10
         645/645 [==============================] - 2751s 4s/step - loss: 0.1251 - accuracy: 0.9592
         Epoch 6/10
         645/645 [==============================] - 2560s 4s/step - loss: 0.1161 - accuracy: 0.9615
         Epoch 7/10
         645/645 [==============================] - 2890s 4s/step - loss: 0.1122 - accuracy: 0.9599
         Epoch 8/10
         645/645 [==============================] - 2893s 4s/step - loss: 0.0907 - accuracy: 0.9691
         Epoch 9/10
         645/645 [==============================] - 16986s 26s/step - loss: 0.0877 - accuracy: 0.9713
         Epoch 10/10
         645/645 [==============================] - 2989s 5s/step - loss: 0.1081 - accuracy: 0.9636
```

Figure 4.7: training performance with in 10 epochs of Inception V3 network model

Training on VGG19 Network Model and Results

The same thing happened when training own dataset on VGG19 model as it did with ResNet50 V2 and Inception V3 model. VGG19 pre-trained network model can be used to launch the training. As shown in Figure 4.8, the step count, accuracy and loss value for each step can be displayed on the screen. It should be remembered that the classification loss starts out high and progressively decreases as the algorithm learns over time. As you can see, with just 10 epochs, we were able to achieve a training accuracy of 98.44 percent.

Training on VGG19 Network Model

```
In [8]: train_vgg19 = vgg_model.fit(train_generator, batch_size = 25, epochs = 10)

Epoch 1/10
645/645 [==============================] - 6174s 10s/step - loss: 0.6826 - accuracy: 0.8007
Epoch 2/10
645/645 [==============================] - 6184s 10s/step - loss: 0.1174 - accuracy: 0.9642
Epoch 3/10
645/645 [==============================] - 5834s 9s/step - loss: 0.0872 - accuracy: 0.9712
Epoch 4/10
645/645 [==============================] - 5801s 9s/step - loss: 0.0909 - accuracy: 0.9695
Epoch 5/10
645/645 [==============================] - 6404s 10s/step - loss: 0.0657 - accuracy: 0.9775
Epoch 6/10
645/645 [==============================] - 6547s 10s/step - loss: 0.0716 - accuracy: 0.9751
Epoch 7/10
645/645 [==============================] - 5818s 9s/step - loss: 0.0545 - accuracy: 0.9816
Epoch 8/10
645/645 [==============================] - 6435s 10s/step - loss: 0.0544 - accuracy: 0.9797
Epoch 9/10
645/645 [==============================] - 6337s 10s/step - loss: 0.0610 - accuracy: 0.9797
Epoch 10/10
645/645 [==============================] - 11404s 18s/step - loss: 0.0449 - accuracy: 0.9844
```

Figure 4.8: training performance with in 10 epochs of VGG19 network model

4.3 Validating on pre-trained network model and results

Since the validation set is contained in a single folder, one of the predefined methods for loading datasets in tensorflow (specifically, flow_from_directory method) were used. Deep learning relies on model validation as a fundamental technique. This aids in determining which algorithm and parameters to use, as well as preventing overfitting during training. This dataset is used to assess the model's output when fine-tuning its hyperparameters. Since this data is used for more regular evaluation and to update hyperparameters, the validation collection has an indirect impact on the model. While tuning the hyperparameters of a model is not strictly required, it is usually recommended. Data augmentation is limited to the training set because it aids the model's generalization and robustness. As a result, adding to the validation collection is pointless. Adding augmented data to the validation would not increase its accuracy. Preprocessing, on the other hand, was carried out.

Validating on ResNet50_V2 Network Model and Results

On the ResNet50V2 pre-trained network model, the validating set can be launched. As shown in Figure 4.9, the step count, accuracy and loss value for each step can be displayed on the screen. As you can see, with just 10 epochs, we were able to achieve a validation accuracy of 99.01 percent.

Validating on ResNet50v2 Network Model

```
In [35]: validation_data = validation_generator
         validation_history = resnet_model.fit(validation_data,  epochs = 10 ,batch_size = 25)

         Epoch 1/10
         138/138 [==============================] - 2369s 17s/step - loss: 0.0107 - accuracy: 0.9974
         Epoch 2/10
         138/138 [==============================] - 2273s 16s/step - loss: 0.0106 - accuracy: 0.9980
         Epoch 3/10
         138/138 [==============================] - 2207s 16s/step - loss: 0.0028 - accuracy: 0.9991
         Epoch 4/10
         138/138 [==============================] - 2113s 15s/step - loss: 0.0011 - accuracy: 0.9994
         Epoch 5/10
         138/138 [==============================] - 2109s 15s/step - loss: 6.9379e-04 - accuracy: 0.9997
         Epoch 6/10
         138/138 [==============================] - 2109s 15s/step - loss: 5.2075e-04 - accuracy: 1.0000
         Epoch 7/10
         138/138 [==============================] - 2110s 15s/step - loss: 0.0306 - accuracy: 0.9957
         Epoch 8/10
         138/138 [==============================] - 2108s 15s/step - loss: 0.0657 - accuracy: 0.9875
         Epoch 9/10
         138/138 [==============================] - 2362s 17s/step - loss: 0.0260 - accuracy: 0.9951
         Epoch 10/10
         138/138 [==============================] - 2313s 17s/step - loss: 0.0709 - accuracy: 0.9901
```

Figure 4.9: validation performance with in 10 epochs of ResNet50v2 network model

Validating on Inception_V3 Network Model and Results

On the Inception V3 pre-trained network model, the validating set can be launched. As shown in Figure 4.10, the step count, accuracy and loss value for each step can be displayed on the screen. As you can see, with just 10 epochs, we were able to achieve a validation accuracy of 99.88 percent.

validating on Inception_V3 Network Model

```
In [56]: validation_data = validation_generator
         validation_history = model.fit(validation_data,  epochs = 10 ,batch_size = 25)

         Epoch 1/10
         138/138 [==============================] - 537s 4s/step - loss: 0.0727 - accuracy: 0.9794
         Epoch 2/10
         138/138 [==============================] - 497s 4s/step - loss: 0.0289 - accuracy: 0.9884
         Epoch 3/10
         138/138 [==============================] - 496s 4s/step - loss: 0.0136 - accuracy: 0.9959
         Epoch 4/10
         138/138 [==============================] - 497s 4s/step - loss: 0.0109 - accuracy: 0.9977
         Epoch 5/10
         138/138 [==============================] - 496s 4s/step - loss: 0.0061 - accuracy: 0.9980
         Epoch 6/10
         138/138 [==============================] - 496s 4s/step - loss: 0.0048 - accuracy: 0.9988
         Epoch 7/10
         138/138 [==============================] - 495s 4s/step - loss: 0.0057 - accuracy: 0.9977
         Epoch 8/10
         138/138 [==============================] - 495s 4s/step - loss: 0.0069 - accuracy: 0.9974
         Epoch 9/10
         138/138 [==============================] - 496s 4s/step - loss: 0.0072 - accuracy: 0.9980
         Epoch 10/10
         138/138 [==============================] - 495s 4s/step - loss: 0.0069 - accuracy: 0.9988
```

Figure 4.10: validation performance with in 10 epochs of Inception V3 network model

Validating on VGG19 Network Model and Results

On the VGG19 pre-trained network model, the validating set can be launched. As shown in Figure 4.11, the step count, accuracy and loss value for each step can be displayed on the screen. As you can see, with just 10 epochs, we were able to achieve a validation accuracy of 100 percent.

Validating on VGG19 Network Model

```
In [10]: validation_data = validation_generator
         validation_history = vgg_model.fit(validation_data,  epochs = 10 ,batch_size = 25)

         Epoch 1/10
         138/138 [==============================] - 1239s 9s/step - loss: 0.0366 - accuracy: 0.9910
         Epoch 2/10
         138/138 [==============================] - 1239s 9s/step - loss: 0.0158 - accuracy: 0.9954
         Epoch 3/10
         138/138 [==============================] - 1350s 10s/step - loss: 0.0084 - accuracy: 0.9983
         Epoch 4/10
         138/138 [==============================] - 1350s 10s/step - loss: 0.0115 - accuracy: 0.9965
         Epoch 5/10
         138/138 [==============================] - 1273s 9s/step - loss: 0.0054 - accuracy: 0.9977
         Epoch 6/10
         138/138 [==============================] - 1285s 9s/step - loss: 0.0074 - accuracy: 0.9974
         Epoch 7/10
         138/138 [==============================] - 1270s 9s/step - loss: 0.0044 - accuracy: 0.9991
         Epoch 8/10
         138/138 [==============================] - 1318s 10s/step - loss: 0.0043 - accuracy: 0.9991
         Epoch 9/10
         138/138 [==============================] - 1363s 10s/step - loss: 0.0017 - accuracy: 1.0000
         Epoch 10/10
         138/138 [==============================] - 1366s 10s/step - loss: 0.0012 - accuracy: 1.0000
```

Figure 4.11: validation performance with in 10 epochs of VGG19 network model

4.4 Testing on pre-trained network model and results

The testing collection is used to assess the model's performance. It must also represent the model's complexity and diversity. This dataset is used to provide an unbiased assessment of the training set's final model fit. This set is only used after the model has been fully trained, it has no effect on the model and is only used to measure results. Adding augmented data to the test would not increase its accuracy. Preprocessing, on the other hand, was carried out.

Testing on ResNet50_V2 Network Model and Results

A test dataset differs from the training dataset ResNet50_V2 model. A model that suits the training dataset well and also fits the test dataset well, without overfitting (see figure below). Overfitting is normally indicated by a better fit of the training dataset compared to the test dataset. As you can see, we were able to achieve a test accuracy of 97.3 percent.

```
Testing on ResNet50v2 Network Model

In [58]: test_resnet=resnet_model.evaluate(test_generator)
         108/108 [==============================] - 566s 5s/step - loss: 0.1510 - accuracy: 0.9730
```

Figure 4.12: Testing performance of ResNet50v2 network model

Testing on Inception_V3 Network Model and Results

A test set is a collection of examples used solely to evaluate a fully defined classifier's output (i.e. generalization). A model that suits the training dataset well and also fits the test dataset well, without overfitting (see figure below). As you can see, we were able to achieve a test accuracy of 99.28 percent.

```
Testing on Inception_V3 Network Model

In [98]: test_inception=model.evaluate(test_generator)
         138/138 [==============================] - 541s 4s/step - loss: 0.0247 - accuracy: 0.9928
```

Figure 4.13: Testing performance of Inception V3 network model

Testing on VGG19 Network Model and Results

The final model is used to predict how the test set's examples would be classified. To test the model's accuracy, these predictions are compared to the true classifications of the examples. As shown in Figure 4.14, a model fits the training dataset well and also fits the test dataset well, without overfitting. As you can see, we were able to achieve a test accuracy of 99.65 percent.

```
Testing on VGG19 Network Model

In [19]: test_vgg19=vgg_model.evaluate(test_generator)
         108/108 [==============================] - 1270s 12s/step - loss: 0.0134 - accuracy: 0.9965
```

Figure 4.14: Testing performance of VGG19 network model

Summary of three pre-trained network model loss value				
NO	Pre-trained net model	Training-loss	Validation-loss	Testing-loss
1	ResNet50_V2	0.0893	0.0709	0.1510
2	Inception_V3	0.1081	0.0069	0.0247
3	VGG19	0.0449	0.0012	0.0134

Table 4.2: Summary of three pre-trained network model loss value

Summary of three pre-trained network model accuracy value				
NO	Pre-trained net model	Training- accuracy	Validation- accuracy	Testing- accuracy
1	ResNet50_V2	0.9844	0.9901	0.9730
2	Inception_V3	0.9636	0.9988	0.9928
3	VGG19	0.9844	1.0000	0.9965

Table 4.3: Summary of three pre-trained network model accuracy value

The comparison is made using a pre-trained network model that uses the same dataset and parameters but has a different architectural design. In conclusion, the VGG19 network model outperformed ResNet50v2 and Inception v3 network models in terms of accuracy and loss value, as demonstrated in Tables 4.2 and 4.3.

Time required to complete 10 epochs of three pre-trained network model				
NO	Pre-trained net model	Training-time(hour)	Validation-time(hour)	Testing-time(hour)
1	ResNet50_V2	28.9836	6.1314	0.1572
2	Inception_V3	12.0219	1.3888	0.1503
3	VGG19	18.5938	3.6268	0.3528

Table 4.4: The time required to complete 10 epochs of three pre-trained network model

The Inception v3 network model outperformed ResNet50v2 and VGG19 network models in terms of time required to complete training, validation, and testing activities, as shown in Tables 4.4.

4.5 Hyper-parameter Fine-tuning

A hyperparameter is an external to the model parameter that is defined before the training or learning process begins. Finding the best model for your data is a collaborative process that entails experimenting with various algorithms in order to reduce model error. Hyperparameters are the parameters that govern the behavior of a deep learning algorithm. We arrived at our conclusions empirically, by trial and error, and by testing them against real-world evidence. Let's look at some popular deep learning hyperparameters:

a) **Learning rate:** A hyperparameter that regulates how quickly or slowly a neural network model learns a problem is called learning rate. It has an effect on how close our model will get to a local minimum. A tuning parameter in an optimization algorithm that decides the phase size at each iteration when moving toward a loss function minimum is called learning rate.

b) **Batch size:** The batch size is a gradient descent hyperparameter that determines how many training samples must be processed before the model's internal parameters are modified. It refers to the total number of training examples used in a single iteration. In modern deep learning systems, batch size is one of the most critical hyperparameters to tune.

> batch size = the number of training examples in one forward/backward pass.

c) **Epoch:** The number of epochs is a gradient descent hyperparameter that determines how many complete passes through the training dataset are made. An epoch is a term used in deep learning that refers to the number of passes the deep learning algorithm has made over the entire training dataset.

> one epoch = one forward and backward pass across all of the training examples.

d) **Regularization:** Regularization is a hyperparameter that prevents neural networks from overfitting and, as a result, improves the accuracy of a Deep Learning model when faced with entirely new data from the problem domain. It is the method of adding information to a problem in order to solve it or to avoid overfitting.

e) **Loss function:** The loss function is a hyperparameter function that can calculation neural network error. It's a metric for determining how close an approximate value is to its true value. A loss function, also known as a cost function, is a function that converts an event or the values of one or more variables into a real number that represents the cost of the event.

f) **Activation function:** An activation function is a hyperparameter function that is applied to an artificial neural network to aid it in learning complex patterns in data. An activation function is a function that is applied to the output of a neural network layer before being passed on to the next layer as input. In neural network models that predict a multinomial probability distribution, the softmax function is used as the activation function in the output layer. When making multi-class predictions, Softmax is used in the output layer.

g) **Optimizer:** Optimizers are hyperparameter algorithms or methods that adjust the characteristics of your neural network, such as weights and learning rate, to minimize losses. Optimizers aid in obtaining quicker results. An optimizer is a process or algorithm for updating various parameters in order to minimize loss with minimal effort.

h) **Metric function:** A metric is a hyperparameter function that is used to assess the model's efficiency. Metric functions are similar to loss functions, except that the effects of measuring a metric aren't included in the model's preparation.

No	Hyper-parameter	Value
1	Learning rate	0.0003
2	Batch size	25
3	Epoch	10

4	Regularization	Dropout(0.2)
5	Loss function	Categorical_crossentropy
6	Activation function	softmax
7	optimizer	Adam
8	Metric function	accuracy

Table 4.5: The tuning hyper-parameter values

4.6 Learning curve

Data visualization is one of the most effective ways to personalize data and make it easier to evaluate and extract key trends from it. The performance of a pre-trained network model can be visualized, a means to make sense of the data coming out of the model and make an informed decision about what to do with it the changes that must be made to the parameters or hyperparameters that have an impact on the model that has been pre-trained.

A learning curve is a graph in which epochs are represented on the horizontal axis and learning or improvement is represented on the vertical axis. Visualizing the training loss vs validation loss, or training accuracy vs validation accuracy, is a helpful way to see if the model has been properly trained. This is crucial to prevent the model from underfitting or overfitting to the point where it memorizes the training data and loses its capacity to predict accurately. There were three types of learning curves in deep learning:

To begin with, an underfitting model is unable to learn the training dataset. It happens when On the training set, the model fails to attain a low enough error value. Only the training loss's learning curve can reveal an underfit model. It could have a flat line or values that are noisy. The model was unable to learn the training dataset at all, resulting in a significant loss. In the event that the training loss is either constant or decreases until the end of the training period. Underfitting is demonstrated by a plot of learning curves during training.

Second, overfitting is when a model has learned the training dataset too well, taking into account all of the variables. statistical noise and random fluctuations into account the problem with overfitting is that it causes the model to be inaccurate. When it becomes more specialized to training data, it loses the ability to generalize to new input, resulting in resulting in a rise in generalization error on the validation dataset, the model's performance may be evaluated. This increase in generalization error was measured using this method. If the training loss plot continues to fall, with practice, or when the plot of validity loss reaches a certain point and then starts to rise again, Overfitting can be shown in the plot of learning curves.

Third, as indicated in figure 4.4, the learning algorithm's goal is a good fit, which exists between an overfit and underfit model. A good fit model's learning curve starts off with a somewhat high training and validation loss, but when more training instances are added, the loss lowers and the curve flattens out. The training and validation losses are nearly identical, with the validation loss somewhat higher than the training loss.

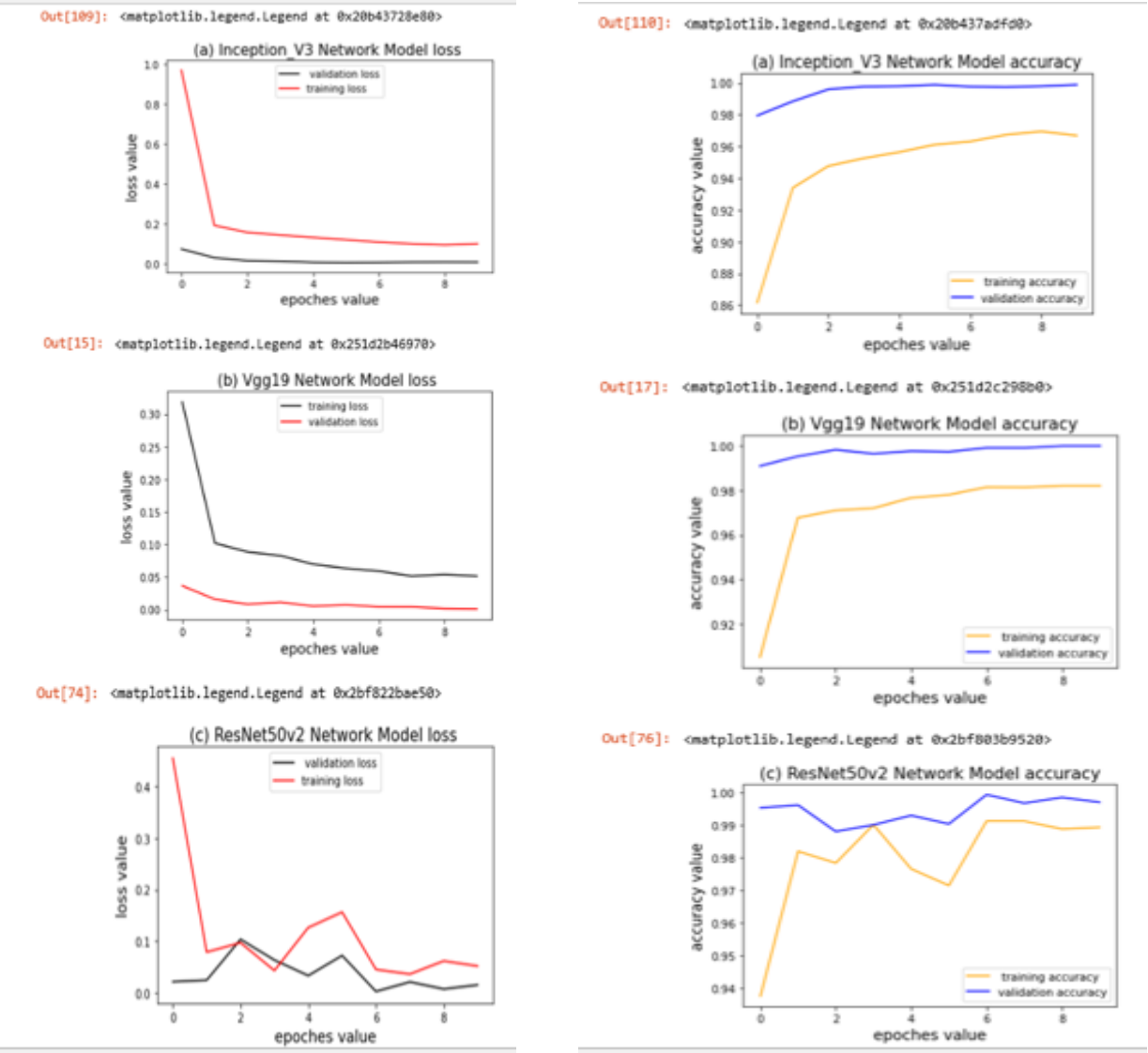


Figure 4.15: Learning curve three pre-trained network model

4.7 Discussion

In terms of accuracies, the VGG19 network model outperforms the ResNet50v2 and inception v3 network models significantly. The reasons for this are that the VGG19 network model achieved 98.44 percent, 100 percent, and 99.65 percent training, validation, and test accuracies within 10 epochs, respectively. However, within 10 epochs, training, validation and test accuracies in ResNet50v2 network model registered are 98.44 percent, 99.01 percent, and 97.3 percent, respectively, and within 10 epochs, training, validation and test accuracies in inception v3 network model registered are 96.36 percent, 99.88 percent, and 99.28 percent, respectively. Due to a lack of hardware resources, no more study was conducted, and this is the final model. It's probable that if we train the models for more epochs, the score will improve.

As shown in Table 4.4, in terms of computational cost, the inception v3 network model outperforms the VGG19 and ResNet50v2 network models significantly. The reason for this is that training time for 10 epochs in inception v3 is 13 hours, but training time for 10 epochs in VGG19 and ResNet50v2 is 19 hours and 29 hours, respectively. The number of hidden layers, dropout values, learning rate, Bach size, and other hyper-parameters affect how long it takes each pre-trained network model to complete one epoch.

We see from the model's performance that the performance in terms of accuracy metrics or loss metrics doesn't always decrease for the loss or doesn't always increase for the accuracy; rather, their values always increase and decrease, indicating that it oscillates in the process of moving the loss to the point of minimal point and the accuracy to the point of maximal point, as shown in Figure 15.

The model is now performing exceptionally well. As seen in Figure 15, there is neither overfitting or underfitting because of the image augmentation and dropout approach. The number of layers in the network model drops, the model loss value lowers, and the time required to complete the epochs lowers as the value of dropout increases. The higher the number of weighted layers in the model, the higher the computational cost, like in the ResNet50v2 network model.

CHAPTER FIVE

CONCLUSION AND FUTURE WORK

We will describe the contents of the whole text in this chapter, and then introduce the work that we will continue to investigate in the future, as well as the research contributions.

5.1 Conclusion

We explored the general concepts of deep learning models in this thesis work, which laid the groundwork for advanced image manipulations. Pre-trained VGG19, ResNet50V2, and Inception-v3 network models are used, and these models are also trained using a custom dataset, with each model's accuracy and loss value evaluated. Both real and fake Birr detection and recognition methods are used in this study. The recognition method takes an image captured with a mobile phone camera and attempts to identify it as 5-Birr, 10-Birr, 50-Birr, 100-Birr, or 200-Birr, but also includes 50-fake Birr, 100-fake Birr, and 200-fake Birr.

High performance on the model's loss value or training accuracy value does not necessarily imply high performance on the test dataset. The training process takes a long time since the model is run on a Central Processing Unit (CPU). The loss value didn't always go down, and the training accuracy didn't always go up, it changes depending on the type of hyper-parameters.

The model's performance is deeply influenced by a variety of factors, including dataset size, hyper-parameter tuning ability, and computer capability. The model learns faster on GPU processors than CPU processors, but it takes hours to learn the data. However, as a developer's experience grows, he or she will be able to do all of the above except for the processor capacity.

Since the built model is a software-based and completely automated solution, it can be easily accessed by anyone who wants to use it and can be installed on an individual level with a low-powered stand-alone microcomputer. Furthermore, it can be a useful tool for banks and other financial institutions to use their Automated Teller Machines (ATMs) for both cash in and cash out transactions. This is possible because the machine's built model would allow it to distinguish various Birr denominations.

5.2 Contribution

The following are the major contributions of this research:

- Situation in the real world our main contribution is the Ethiopian Birr dataset.
- In order to create an Ethiopian Birr detection and recognition model, the research used transfer learning techniques and approaches.
- We put our new model to the test on Ethiopian Birr detection and recognition, and we came out on best.

5.3 Future work

In the future, the research could be improved in the following areas to enable it to function in more complicated situations.

Firstly, extend the design so that Ethiopian Birr can be denominated and recognized from either side and in any direction.

Second, the model we use can be applied to other deep learning models such as ResNet101 V2, ResNet152 V2, EfficientNet, DenseNet, and others.

Thirdly, the Tensorflow framework, which we used to create the MobileNet model, is ideal for designing deep learning mobile applications, so I strongly encourage other researchers to use these models and datasets to create a mobile application for these problems.

Last but not least, we will pick the currencies of several different countries for recognition and use the recognition results to classify the countries' names and other information

Bibliography

A. Rao (2019), “Convolutional Neural Network Tutorial (CNN) – Developing An Image Classifier In Python Using TensorFlow”.

Abhijit G. (2019),” Deep learning with R”, *Springer Nature*, Singapore.

Achraf O. and Azeddine E. (2018), "Deep Generative Models", *Survey International Conference on Intelligent Systems and Computer Vision (ISCV), IEEE*, pp. 1-8.

Adice M. (2019),” Basic Structure of the Human Nervous System”, accessed on March 2020,Available:https://mikerbio.weebly.com/structure--function.html.

Ahirwar, K. (2017),” Everything you need to know about Neural Networks”, accessed on March 2020,Available:https://hackernoon.com/everything-you-need-to-know-about-neuralnetworks-8988c3ee4491.

Ahmed, A. and Manzoor (2013), "Recognition System for Pakistani Paper Currency", *World Applied Sciences Journal*, vol. 28, pp. 2069-2075.

Ajay S. and Ausif M. (2019),” "Review of Deep Learning Algorithms and Architectures", *IEEE Access*, vol. 7, pp. 53040-53065.

Ali Abd Almisreb, Mohamed A. Saleh (2018, January),” Transfer Learning Utilization for Banknote Recognition: a Comparative Study Based on Bosnian Currency”, *Southeast Europe Journal of Soft Computing*, Available online: http://scjournal.ius.edu.ba

Alloghani, Mohamed, Dhiya Al-Jumeily, Jamila Mustafina, Abir Hussain and Ahmed J. Aljaaf(2020),”A Systematic Review on Supervised and Unsupervised Machine Learning

Algorithms for Data Science", in *Supervised and Unsupervised Learning for Data Science, ,* Springer Nature, Switzerland, pp. 3-21.

Aljutaili, Daliyah S., Redna A. Almutlaq, Suha A. Alharbi, Dina M. Ibrahim (2018),"A Speeded up Robust Scale-Invariant Feature Transform Currency Recognition Algorithm", *International Journal of Computer and Information Engineering*.12

Amitha M. (2020, August)," Deep Learning Techniques": An Overview, *research gate*. Available: https://www.researchgate.net/publication/341652370.

Andrii Kyrychok (2018), "The overview of investigation in the field of banknote design for visually impaired people", National Technical University of Ukraine, Kyiv, *Reports on research 3*.

Antalya and Turkey (2017, August)," Understanding of a Convolutional Neural Network", *research Gate.*

Asfaw, S. and Million, M. (2019), "Ethiopian Paper Currency Recognition System: An Optimal Feature Extraction", *IEEE-SEM*, vol. **7,** no. 8, pp. 130 - 137.

Asfaw, A. (2019), "Deep Learning Approach for Ethiopian Banknote Denomination Classification and Fake Detection System", *International Journal of Computer Science and Control Engineering.* Vol. **7**, No. 4, pp. 30-37.

Asifullah Khan, Anabia Sohail, Umme Zahoora1, and Aqsa Saeed Qureshi (2020),"A Survey of the Recent Architectures of Deep Convolutional Neural Networks", *Artificial Intelligence Review*, DOI: Available online: https://doi.org/10.1007/s10462-020-09825-6

Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola (2020, October)," Dive into Deep Learning", *Release 0.15.0*.

Avinash S. (2017),” Understanding Activation Functions in Neural Networks”, URL: https://medium.com/the-theory-of-everything/ understanding - activation - functions - in - neural - networks - 9491262884e0

Ayalew, T., E., Ramani, B., and Kebede, B., T. (2018), “Ethiopian Banknote Recognition and Fake Detection Using Support Vector Machine”, *Proceedings of the International Conference on Inventive Communication and Computational Technologies, ICICCT 2018,https://doi.org/10.1109/ICICCT.2018.8473013*

Berhane, Y., Worku, A., Bejiga, A., Adamu, L., Alemayehu, W., Bedri, A., and Kebede, T. (2008). “National Survey on Blindness, Low Vision and Trachoma in Ethiopia: Methods and Study Clusters Profile”, *Ethiopian Journal of Health Development*, 21(3), 1-4. doi:10.4314/ejhd. v21i3.10049.

Bre, F., Giminez, J. (2017),” Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks”, Cited6.12.2019. Available: https://www.researchgate.net/publication/321259051_Prediction_of_wind_pressure_coefficients_on_building_surfaces_using_Artificial_Neural_Networks.

Brownlee, J. (2016.),” Crash Course on Multi-Layer Perceptron Neural Networks”, Date of retrieval 8.12.2019. Available: https://machinelearningmastery.com/neural-networks-crash-course/

Bouvrie J. (2006),”Introduction Notes on Convolutional Neural Networks”, doi: http://dx.doi.org/10.1016/j.protcy.2014.09.007

Christoph (2016),”Perceptrons - the most basic form of a neural network.”, URL: https://appliedgo.net/perceptron/

Chigozie E. Nwankpa, Winifred l.Ijomah and Anthony Gachagan (2020, December), "Activation Functions: Comparison of Trends in Practice and Research for Deep Learning", *2nd International Conference on Computational Sciences and Technologies*

Chollet, F. (2017), "Deep Learning with Python", Manning Publications. www. Manning.com (accessed 28 April 2018).

D. Upadhyay (2019), "How a computer looks at pictures: Image Classification".

D. Kingma and J. Ba (2014)," Adam: A method for stochastic optimization", *arXiv preprint arXiv: 1412.6980.*

Darpan P., Kamal N. and Bharti C. (2019, February), "Machine Learning Algorithms: A Review*," International Research Journal of Engineering and Technology, vol. 06, no. 02,* pp.916-922,.

De, S., Mukherjee, A., Ullah, E. (2018)," Convergence guarantees for RMSProp and ADAM in non-convex optimization and an empirical comparison to Nesterov acceleration", *arXiv*, arXiv: 1807.06766.

Dertat, A. (2017), "Applied Deep Learning - Part 1: Artificial Neural Networks", Date of retrieval 16.12.2019.Available:https://towardsdatascience.com/applied-deep-learning-part-1-artificialneural-networks-d7834f67a4f6?#860e

Derya Soydaner (2020, May), "A Comparison of Optimization Algorithms for Deep Learning**",** *International Journal of Pattern Recognition and Artificial Intelligence.*

Dey, R.; Bajpai, V.; Gandhi, G. and Dey, B. (2008),"Application of Artificial Neural Network (ANN) technique for Diagnosing Diabetes Mellitus*", IEEE Region 10 and the Third Int. Conf. on Industrial and Information Systems (ICIIS),* pp. 1 – 4.

Dzmitry Bahdanau, Kyunghyun Cho and Yoshua Bengio (2014, May), "Neural Machine Translation by Jointly Learning to Align and Translate", *arXiv: 1409.0473*.

E. Saber, A.M. Tekalp (*1998*),*"*Integration of color, edge and texture features for automatic region-base image annotation and retrieval*", Electronic Imaging, 7, pp. 684–700,*

El-Ghazali Talbi (2020, June), "Optimization of deep neural networks: a survey and unified taxonomy", hal-02570804v2 .

Ejaz Ali, Mriganka Gogoi, and Subra Mukherjee (2014),"Challenges in Indian currency denomination recognition and authentication", *International Journal of Research in Engineering and Technology*, 03(11), 477.

F. Garcia-Lamont, J. Cervantes and A. Lopez (2012),"Recognition of Mexican Banknotes via their colour and texture features", *Expert Systems with Applications*. 39.

Fausett L. (1994)," *Fundamentals of Neural Networks: Architectures, Algorithms and Applications", Florida Institute of Technology: Prentice Hall.*

Goodfellow, Y. Bengio, and A. Courville (2016), "Deep learning." *MIT Press*. [Online]. Available: http://www.deeplearningbook.org.

Graupe, D. (2007),"Principles of Artificial Neural Networks*", 2nd ed*. Chicago, USA: University of Illinois.

Harshvardhan GM, Mahendra Kumar Gourisaria, Manjusha Pandey and Siddharth Swarup Rautaray Valaee (2020, November),"A comprehensive survey and analysis of generative models in machine learning", *Computer Science Review*, vol. 38.

Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak and Shahrokh Valaee (2018, February),"Recent Advances in Recurrent Neural Networks", arXiv:1801.01078v3.

Jegnaw, F. and Yaregal, .A. (2016),"Automatic Recognition and Counterfeit Detection of Ethiopian Paper Currency", *International Journal of Image, Graphics and Signal Processing (IJIGSP), vol. 8*, pp. 28-36.

Jesper E. van Engelen and Holger H. Hoos (2020), "A survey on semi-supervised learning," *Machine Learning, vol. 109*, pp. 373-440.

Junfeng, Z. and Leping, X. (2010)," The Fault Diagnosis of Marine Engine Cooling System Based on Artificial Neural Network (ANN)",*IEEE The 2nd Int. Conf. on Computer and Automation Engineering (ICCAE),* pp. 186 – 189.

K. Sindhu Meena and S. Suriya(2019), "A Survey on Supervised and Unsupervised Learning Techniques," In *Proceedings of International Conference on Artificial Intelligence, Smart Grid and Smart City Applications*, Switzerland, *Springer Nature*, pp. 627-642.

K. Teilo (2016), "An Introduction to Convolutional Neural Networks", pp. 0–11.

Kedar Sawant, C. M. (2016),"Currency Recognition Using Image Processing and Minimum Distance Classifier Technique", *International Journal of Advanced Engineering Research and Science.*

Kohl, N. (2010)," Answer to Role of Bias in Neural Networks on Stackoverflow",Date of retrieval 16.12.2019,Available:https://stackoverflow.com/questions/2480650/role-of-bias-in-neuralnetworks.

Krizhevsky, I. Sutskever, and G. E. Hinton (2012), "Imagenet classification with deep convolutional neural networks", [Online].Available: http://citeseerx.ist.psu.edu/viewdoc/summary.

Kumar and Ranjit (2011)."Research Methodology : A step by step guide for beggner", New Delhi: SAGE Publications India Pvt Ltd.

Lan Goodfellow, Bengio Y. and Courville A. (2017),"Deep learning",Nat Methods 13:35. doi: 10.1038/nmeth.3707

Lars S., Monty S., Simon-Martin S. and Reinhard K., (2020, July),"A Survey on Semi-, Self- and Unsupervised Learning for Image Classification".

Lee, J., Hong, H., Kim, K., and Park, K. (2017)," A survey on banknote recognition methods by various sensors",Sensors. 2017; 17:313 doi: 10.3390/s17020313.

Lee, H. W., Syono, M. I. and Azid, I. H. A. (2007),"Application of Artificial Neural Network (ANN) for Predicting The Behaviour of Micromachined Diaphragm Actuated Electrostatically", *IEEE Int. Conf SENSORS (ICSENS),* pp. 316 – 319.

Lilian Wing (2018, October),"Flow-based Deep Generative Models **",**https://lilianweng.github.io/lil-log/

M. Arif Wani, Farooq Ahmad Bhat, Saduf Afzal and Asif Iqbal Khan (2020)," Advances in DeepLearning", Singapore: *Springer Nature*.

M. Arif Wani, Mehmed Kantardzic and Moamar Sayed-Mouchaweh. (2020), "Trends in Deep Learning Applications," *Deep Learning Applications,* vol. 1098, pp. 1-7.

Martin, H. T., Howard, D. H. and Mark, B. (1996)," *Neural network design*", *Boston: PWS Publ, Company*.

McDonald, C. (2017),"Machine learning fundamentals (I): Cost functions and gradient descent", Date of retrieval18.12.2019.Available:https://towardsdatascience.com/machine-learning-fundamentals-vialinear-regression-41a5d11f5220

Md Zahangir Alom, Tarek M. Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Mahmudul Hasan, Brian C. Van Essen, Abdul A. S. Awwal and

Vijayan K. Asari (2019, March),"A State-of-the-Art Survey on Deep Learning Theory and Architectures," *Electronics*, vol. 8, no. 3, p. 292, .

Mekonnen, L. and Hussien, S. (2020),” Real-Time Ethiopian Currency Recognition for Visually Disabled Peoples Using Convolutional Neural Network”,*Research Square.*

Mohammad-Parsa Hosseini, Senbao Lu, Kavin Kamaraj, Alexander Slowikowski and Haygreev C. Venkatesh (2020),"Deep Learning Architectures," in *Deep Learning: Concepts and Architectures*, vol. 866, Switzerland, *Springer Nature*, pp. 1-24.

Moindrot, O., and Genthial, G. (2018, January 24),”Splitting into train, dev and test sets”, Retrieved March 12, 2019, from *https://cs230-stanford.github.io/train-dev-test-split.html*

Muhammad Shahroz Nadeem, Virginia N. L. Franqueira, Xiaojun Zhai and Fatih Kurugollu(2019, May),"A Survey of Deep Learning Solutions forMultimedia Visual Content Analysis," *IEEE Access*, vol.7.

Naim, N. F., Yassin, A. I. M., Zakaria, N. and Ab Wahab, N. (2011),”Classification of Thumbprint using Artificial Neural Network (ANN*)”, Int. Conf. on System Engineering and Technology (ICSET)*, pp. 321 – 234.

Niall O' Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco-Hernandez, Lenka Krpalkova, Daniel Riordan and Joseph Walsh (2019),"Deep Learning vs. Traditional Computer Vision" ,in *Proceedings of the 2019 Computer Vision Conference (CVC)*, Las Vegas.

Ovidiu Calin (2020),” Deep Learning Architectures a Mathematical Approach”, Switzerland: *Springer Nature.*

Palash G., Sumit P. and Karan J. (2018),"Introduction to natural language processing and deep learning. In *Deep Learning for Natural Language Processing*", *Springer*,. doi: 10.1007/978-1-4842-3685-7 1, pages 1–74.

Paul Wilmott (2019),"Machine Learning an Applied Mathematics Introduction". *Panda Ghana Publishing.*

Q.V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow and A.Y. Ng, (2018)," On optimization methods for deep learning", I*n Proc. 28th International Conference on Machine Learning*,Bellevue,Washington,USA,2011,pp.265–272.

Rosebrock, A. (2017), "Deep Learning for Computer Vision with Python", Pyimagesearch, https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2014)," Imagenet large scale visual recognition challenge", *International Journal of Computer Vision*, pages 1–42.

S.Pokhrel (2019)," How does Computer understand Images?" Available: https://towardsdatascience.com/how-does-computer-understand-images

Sachin Mehta, Marjan Ghazvininejad, Srinivasan Iyer, Luke Zettlemoyer and Hannaneh Hajishirzi, (2020, August), "DeLighT: Very Deep and Light-weight Transformer" ,arXiv:2008.00623v1.

Saha, S. (2018)," A Comprehensive Guide to Convolutional Neural Networks",Available:https://towardsdatascience.com/a-comprehensive-guide-to-convolutionalneural-networks-the-eli5-way-3bd2b1164a53

Samarasighe, S. (2006)," Neural Network for Applied Science and Engineering: From Fundamentals to Complex Pattern Recognition", New York: Auerbach Publication.

Sarfraz and Muhammad. (2015),"An intelligent paper currency recognition system",*Science Direct International Conference on Communication, Management and Information Technology.*

Savita K Shetty, and Ayesha Siddiqa (2019, July)," deep learning algorithms and applications in computer vision**",** *International journal of computer sciences and engineering*, Available online at: www.ijcseonline.org

Seyedali Mirjalili, Hossam Faris and Ibrahim Aljarah (2020),"Introduction to Evolutionary Machine Learning Technique", In *Evolutionary Machine Learning Techniques Algorithms and Applications*, Singapore, *Springer Nature*, pp. 1-7.

Ahish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser and Illia Polosukhin, (2017)," Attention Is All You Need", in *31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, USA.

Simonyan, K. and Zisserman, A. (2014),"Very deep convolutional networks for large-scale image recognition", *CoRR*, abs/1409.1556

Stuart J. Russell and Peter Norvig (2020)," Artificial Intelligence A Modern Approach Fourth Edition",Pearson.

Sutskever, Martens and Hinton (2011)," Generating text with recurrent neural networks." *In 28th International Conference on Machine Learning (ICML-11)*, Bellevue, WA, USA.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014),"Going deeper with convolutions", *arXiv preprint arXiv:1409.4842*.

Taigman, Y., Yang, M., Ranzato, M., and Wolf, L. (2014),” Deepface: Closing the gap to human-level performance in face verification”, In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1701–1708. *IEEE.*

Taesung Park, Ming-Yu Liu, Ting-Chun Wang and Jun-Yan Zhu(2019),” Semantic Image Synthesis with Spatially-Adaptive Normalization", arXiv:1903.07291v2.

Thomas A., and Sreekumar K. (2014),”A survey on image feature descriptors color, shape and texture”, *International Journal of Computer Science and Information Technologies*, 5(6), 7847–7850.

Umberto M. (2019)” Advanced Applied Deep Learning: Convolutional Neural Networks and Object Detection”, *Apress.*

V. Badrinarayanan, A. Kendall, and R. Cipolla(2015), “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation” ,*arXiv*,. [Online]. Available: http://arxiv.org/abs/1511.00561

X. Zhu, M. Ren (2014, May),” A Recognition Method of RMB Numbers Based on Character Features”, *2nd International conference on Information, Electronics and Computer.*

Yann L., Yoshua B. and Geoffrey H. (2015, May),”Deep learning" ,*Nature*, vol. 521, 28

Yeung, D. S., Cloete, I., Shi, D. and Ng, W. W. Y. (2010),”*Sensitivity Analysis for Neural Network”,* London: Springer.

Yingying Wang, Yibin Li and Xuewen Rong (2020, February),” The Influence of the Activation Function in a Convolution Neural Network Model of Facial Expression Recognition”, *applied sciences.*

Yong Yu, Xiaosheng Si, Changhua Hu and Jianxun Zh(2019, July),” A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures" ,*Neural Computation*, vol. 31, no. 7, pp. 1235 – 1270.

Zhaoqiang Xia (2019),” An Overview of Deep Learning in *Deep Learning in Object Detection and* Recognition", Singapore, Springer Nature, pp. 1-18.

.

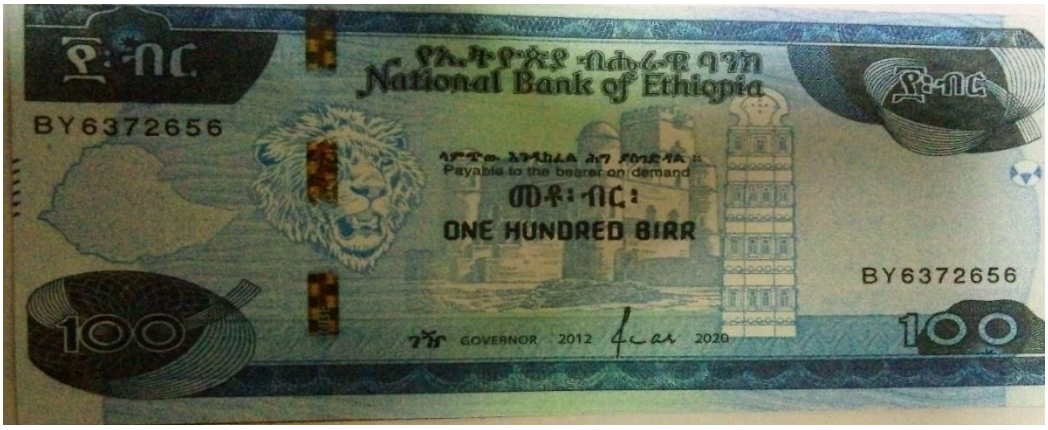
Appendices A

Security future

Appendices B

Ethiopian Fake Birr note

50-fake Birr note



100-fake Birr note

200-fake Birr note

Appendices C

Ethiopian True Birr note

5-Birr note

10-Birr note

50-Birr note

100-Birr note

200-Birr note