SIGNATURE FORGERY DETECTION SYSTEM – A DEEP LEARNING APPROACH

A PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

in

COMPUTER SCIENCE AND ENGINEERING

BY

JETTIBOINA SWATHI
21331A0569

MUNSHI MOOSA SAIT
21331A05C0

KALYAMPUDI LOKESH
21331A0574

KATARI MURALI MANOHAR
22335A0510

GEMMELA RAMBHA
22335A0507

Under the Supervision of

Mrs. V. Lavanya
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MAHARAJ VIJAYARAM GAJAPATHI RAJ COLLEGE OF ENGINEERING (Autonomous)
(Approved by AICTE, New Delhi, and permanently affiliated to JNTUGV, Vizianagaram),
Listed u/s 2(f) & 12(B) of UGC Act 1956.
Vijayaram Nagar Campus, Chintalavalasa,Vizianagaram-535005, Andhra Pradesh

APRIL, 2024

CERTIFICATE

This is to certify that the project report entitled "**SIGNATURE FORGERY DETECTION – A DEEP LEARNING APPROACH**" being submitted by Jettiboina Swathi (21331A0569), Munshi Moosa Sait (21331A05C0), Kalyampudi Lokesh (21331A0574), Katari Murali Manohar (22335A0510), Gemmela Rambha (22335A0507) in partial fulfillment for the award of the degree of "**Bachelor of Technology" in Computer Science and Engineering** is a record of bonafide work done by them under my supervision during the academic year 2023-2024.

Dr. P. Rama Santosh Naidu
Senior Assistant professor,
Supervisor,
Department of CSE,
MVGR College of Engineering(A),
Vizianagaram.

Dr. T. Pavan Kumar
Associate Professor,
Head of the Department,
Department of CSE,
MVGR College of Engineering(A),
Vizianagaram.

External Examiner

DECLARATION

We hereby declare that the work done on the dissertation entitled "**SIGNATURE FORGERY DETECTION – A DEEP LEARNING APPROACH**" has been carried out by us and submitted in partial fulfillment for the award of credits in Bachelor of Technology in Computer Science and Engineering of MVGR College of Engineering (Autonomous) and affiliated to JNTUGV, Vizianagaram. The various contents incorporated in the dissertation have not been submitted for the award of any degree of any other institution or university.

ABSTRACT

The automated system for detecting forged signatures employs a Convolutional Neural Network (CNN) as its core component. Through extensive analysis of numerous authentic signatures, the CNN effectively learns the distinguishing features of genuine signatures. This learning process enables the CNN to accurately differentiate between authentic and forged signatures, essentially functioning as an expert in signature verification. By leveraging this technology, document security is significantly enhanced, as it becomes more challenging to counterfeit signatures, thus reducing the risk of fraudulent activities.

This project intends to create an automated system utilizing deep learning , especially a simple and efficient Convolutional Neural Network (CNN) and VGG16 that can determine if a given signature is real or faked. The model will take a signature image as input and extract important features through different layers of CNN that will be helpful in distinguishing genuine and forged signatures.

TABLE OF CONTENTS

List of Abbreviations

AI	–	Artificial Intelligence
ML	–	Machine Learning
NN	–	Neural Networks
CNN	–	Convolutional Neural Networks
TP	–	True positive
TN	–	True Negative
FP	–	False Positive
FN	–	False Negative
TPR	–	True Positive Rate
FPR	–	False Positive Rate

CHAPTER - 1

1. INTRODUCTION

In today's interconnected and digitized world, signatures hold paramount importance as a means of authentication and validation in numerous facets of daily life. Whether it's signing contracts, authorizing financial transactions, or endorsing official documents, signatures serve as a symbol of consent, agreement, and legal commitment. However, the proliferation of technology and the advent of digital platforms have brought about both conveniences and challenges, particularly concerning the integrity and security of signatures.

With the advent of digital tools, the risk of signature forgery has reached unprecedented levels. Signature forgery, the unauthorized replication or alteration of a signature, poses significant threats ranging from financial fraud to legal disputes and reputational damage. Fraudsters can exploit vulnerabilities in traditional paper-based signature verification processes or manipulate digital images to create convincing forgeries. These forgeries can lead to severe consequences, including financial losses for individuals and organizations, breach of contractual agreements, and erosion of trust in the integrity of digital transactions.

To address this pressing challenge, the development and implementation of signature forgery detection systems have become imperative. These sophisticated systems leverage cutting-edge technologies such as machine learning, image processing, and pattern recognition to scrutinize and authenticate signatures with unparalleled accuracy and efficiency. By meticulously analyzing the intricate characteristics, strokes, pressure points, and other distinctive features inherent in each signature, these systems can discern genuine signatures from fraudulent ones with a high degree of precision.

Machine learning algorithms play a pivotal role in signature forgery detection systems by continuously learning and adapting to new patterns and variations in forgery attempts. Through extensive training on vast datasets of authentic and forged signatures, these algorithms acquire the capability to identify subtle discrepancies and irregularities indicative of forgery. Moreover, image processing techniques enable the system to enhance the quality of signature images, extract relevant features, and minimize noise or distortions that may obscure vital information crucial for authentication.

Pattern recognition algorithms serve as the backbone of signature forgery detection systems, enabling them to discern recurring patterns and anomalies in signatures that may indicate potential forgery. By employing sophisticated algorithms capable of discerning complex patterns and variations, these systems can swiftly evaluate the authenticity of signatures across diverse contexts and document types.

In essence, signature forgery detection systems represent a formidable defense against the escalating threat of signature fraud in today's digital landscape. By harnessing the power of advanced technologies and algorithms, these systems empower individuals, businesses, and organizations to safeguard the integrity and security of their signatures, thereby mitigating the risks associated with forgery and upholding trust and accountability in digital transactions and documentations.

1.1. Identification of seriousness of the problem

The seriousness of the problem of signature forgery cannot be overstated, as it impacts individuals, businesses, and society on multiple levels. Financial losses, legal disputes, reputational damage, identity theft, and regulatory risks are some of the significant consequences associated with signature forgery. Moreover, the global scope and complexity of the issue further exacerbate its seriousness, requiring comprehensive measures to address it effectively. Overall, the prevalence of signature forgery undermines trust and confidence in transactions, necessitating proactive efforts to combat this pervasive threat.

1.2. Problem definition

Signature forgery involves the unauthorized replication or alteration of signatures, leading to potential fraud, legal disputes, and financial losses. It encompasses various fraudulent activities where individuals or entities manipulate signatures to deceive others, gain unauthorized access to resources, or perpetrate criminal acts. The problem arises from vulnerabilities in traditional signature verification processes and the increasing sophistication of forgery techniques, exacerbated by advancements in technology. Detecting and preventing signature forgery requires robust authentication mechanisms, regulatory compliance, and awareness initiatives to safeguard the integrity of signatures and uphold trust in transactions and documentations.

1.3. Objective

The primary objective of a this project focused on to develop and implement an effective signature forgery detection system that utilizes advanced technologies such as machine learning, image processing, and pattern recognition to accurately identify and authenticate signatures, thereby mitigating the risks associated with forgery, enhancing security in transactions and documentations, and preserving trust and integrity in digital environments.

1.4. Existing models

Indeed, while numerous existing models for signature forgery detection utilize individual algorithms or techniques, the concept of hybrid models that combine multiple approaches is relatively less explored. By building hybrid models, which integrate the strengths of different algorithms, researchers aim to enhance overall detection accuracy and robustness. This can be achieved by leveraging the complementary capabilities of diverse techniques to overcome the limitations of individual methods.

While existing models for signature forgery detection provide valuable insights, the exploration of hybrid models offers a promising avenue for advancing the field and achieving higher detection accuracy and reliability. Through systematic experimentation and innovation, researchers can develop hybrid models that push the boundaries of performance and contribute to the development of more effective signature forgery detection systems.

CHAPTER - 2

2. LITERATURE SURVEY

A signature is a unique and personal way a person writes their name. It is a common biometric used to verify a person's identity on legal documents. Signatures are used to approve financial transactions, contracts, and other documents. They can also be used to show who sent or received a letter or to request information on a form. Issues associated with signatures are numerous since any two signatures may look very similar with little to no differences written by the same person.

Traditional methods of signature verification rely on human experts or simple features extracted from the signature image. However, these methods are prone to errors and limitations. Despite the enormous research efforts, offline signature verification is still difficult, especially when attempting to differentiate between forgeries and authentic signatures. This is because the visual differences between the two can sometimes be less noticeable than between two authentic signatures. New technologies have led to an increase in fraudulent activities, including signature forgery. In order to mitigate these deceptive activities, there is a growing demand for effective and precise signature forgery detection systems. These systems would distinguish between forged and genuine handwritten signatures.

The authors propose a Crest-Trough algorithm with Harris-Surf for forgery detection [2]. Pre-processing techniques including noise removal, scaling, centralization and rotation are used [2]. The CNN model uses a back-propagation method for training and is tested on a dataset of 1320 images for each genuine and forged signature [2]. Finally, False Acceptance Rate (FAR) and False Rejection Rate (FRR) are used to evaluate the performance of the model [2].

In simpler terms, the document explains a system that can automatically distinguish between real and forged signatures. The system uses a special kind of neural network called a CNN, which is trained on a dataset of handwritten signatures. The system first pre-processes the image to remove noise and make sure it is properly aligned, and then uses the CNN to

classify the signature as real or forged. The effectiveness of the system is measured by how often it incorrectly identifies a real signature as a forgery (FAR) and vice versa (FRR).

The document explores a region-based deep neural network (DNN) that utilizes a mutual signature Densenet (MDSD) [3]. In order to learn the feature representation of handwritten signatures, it consists of two convolutional neural network (CNN) branches that share the same structure and parameters [3]. The proposed method analyzes multiple local regions by using the MDSD feature extractor to obtain a similarity score between two signatures [3]. This similarity score is then fed into a Siamese Neural Network for training on the CEDAR database of handwritten signatures along with the GPS dataset [3]. Finally, the performance of the model is compared with other CNN architectures such as VGG-16, AlexNet, and ResNet-18.

The document proposes a method for verifying handwritten signatures using Siamese neural networks [1]. The method includes pre-processing the signature image to convert it to a grayscale image and resize it [1]. Then, morphological erosion is applied to increase the thickness of the signature strokes [1]. The Siamese neural network is used to compare two signature images and determine if they are from the same writer [1]. The performance of the model is evaluated by comparing it to other pre-trained networks such as LeNet and SigNet [1].

The passage proposes a simple CNN architecture for signature forgery detection [51]. The CNN consists of three alternating convolution and max-pooling layers, followed by a flatten layer and a dense layer [51]. The model is trained on a dataset of 5000 images, including original signatures and forgeries created by two different people [51]. The document mentions that all the images are converted from RGB to grayscale format during pre-processing [51].

The authors propose a method to generate a handwritten signature dataset using Optical Mark Recognition (OMR) tools [8]. OMR tools are used to scan attendance sheets which contain a unique barcode to identify the sheet and distinguish between signatures and non-signatures [8]. Then, a deep CNN classifier is used to check the legitimacy of the signatures [8]. The document mentions various hyperparameter tuning techniques were performed to increase the

accuracy of the model, including comparing optimizers like ADAM and SGD [8]. Finally, a confusion matrix is used to evaluate the performance of the neural network model [8].

Though numerous CNN architectures are proposed in literature, the goal of developing a simple and more efficient one always attracts researchers in this field.The document proposes a new approach to automatically detect forgery in handwritten signature images using a custom Convolutional Neural Network (CNN) [1]. The CNN takes signature images as input and extracts important features through different layers to distinguish between genuine and forged signatures [1]. The performance of the CNN is analyzed with different optimizers and training/testing splits [1]. The proposed CNN is also compared to a pre-trained VGG16 model [1]. The study demonstrates that the custom CNN performs well in detecting forged signatures.

CHAPTER - 3

3. THEORETICAL BACKGROUND

3.1 Machine learning Vs Deep learning

3.1.1 What is Machine Learning?

Machine learning is an application of AI that enables systems to learn and improve from experience without being explicitly programmed. Machine learning focuses on developing computer programs that can access data and use it to learn for themselves. Machine learning is something that is capable of imitating the intelligence of human behavior. Machine learning is used to perform complex tasks in a way that humans solve the problems. Machine learning can be descriptive; it uses the data to explain, predictive, and prescription.

3.1.2 Why Machine Learning?

Machine learning involves computers learning from data provided so that they carry out certain tasks. For more advanced tasks, it can be challenging for a human to manually create the needed algorithms. In practice, it can turn out to be more effective to help the machine develop its own algorithm, rather than having human programmers specify every needed step. The discipline of machine learning employs various approaches to teach computers to accomplish tasks where no fully satisfactory algorithm is available. In cases where vast numbers of potential answers exist, one approach is to label some of the correct answers as valid. This can then be used as training data for the computer to improve the algorithms it uses to determine correct answers. The nearly limitless quantity of available data, affordable data storage, and growth of less expensive and more powerful processing has propelled the growth of ML. Now many industries are developing more robust models capable of analyzing bigger and more complex data while delivering faster, more accurate results on vast scales. ML tools enable organizations to more quickly identify profitable opportunities and potential risks.

The practical applications of machine learning drive business results which can dramatically affect a company’s bottom line. New techniques in the field are evolving rapidly and expanding the application of ML to nearly limitless possibilities. Industries that depend on vast quantities of data—and need a system to analyze it efficiently and accurately, have embraced ML as the best way to build models, strategize, and plan.

3.1.3 What is Deep Learning?

Deep learning is the subset of the machine learning technique. It is also known as deep neural network because it uses the architecture of neural networks. Deep learning eliminates some of data preprocessing. This has one input layer and one output layer and more than one hidden layer. This uses labeled data for training and deep learning prediction is more accurate. Deep learning is an element of data science, it includes statistics and predictive modeling. This approach is beneficial to the person whose tasks are related to collecting, analyzing large amounts of data.

3.1.4 Why Deep Learning?

Algorithms used in deep learning learn high level features for data. The more data you feed to deep learning the better the result will be. Deep learning handles large volumes of data and it is kept to the best use when it comes to the large sets of unstructured data. Deep learning has less accuracy when it is fed with less data.

3.2 Machine Learning Approaches

Machine learning approaches are traditionally divided into three broad categories, depending on the nature of the "signal" or "feedback" available to the learning system:

- ✓ Supervised learning
- ✓ Unsupervised learning
- ✓ Reinforcement learning

3.2.1 Supervised learning

Supervised learning is one of the machine learning approaches through which models are trained using perfectly labeled training data and on the basis of that models predicts the output.

Types of supervised learning algorithms include active learning, classification and regression. Classification algorithms are used when the outputs are restricted to a limited set of values, and regression algorithms are used when the outputs may have any numerical value within a range. Similarity learning is an area of supervised machine learning closely related to regression and classification, but the goal is to learn from examples using a similarity function that measures how similar or related two objects are.

3.2.2 Unsupervised learning

Unsupervised learning algorithms take a set of data that contains only inputs, and find structure in the data, like grouping or clustering of data points. The algorithms, therefore, learn from test data that has not been labeled, classified or categorized. Instead of responding to feedback, unsupervised learning algorithms identify commonalities in the data and react based on the presence or absence of such commonalities in each new piece of data. A central application of unsupervised learning is in the field of density estimation in statistics, such as finding the probability density function. Though unsupervised learning encompasses other domains involving summarizing and explaining data features.

3.2.3 Semi-supervised learning

Semi-supervised learning falls between unsupervised learning (without any labeled training data) and supervised learning (with completely labeled training data). Some of the training examples are missing training labels, yet many machine-learning researchers have found that unlabelled data, when used in conjunction with a small amount of labeled data, can produce a considerable improvement in learning accuracy. In weakly supervised learning, the training labels are noisy, limited, or imprecise; however, these labels are often cheaper to obtain, resulting in larger effective training sets.

3.2.4 Reinforcement learning

Reinforcement learning is a feedback-based machine learning technique, and it aims to maximize the rewards by their hit and trial actions. In reinforcement learning the model learns automatically using feedback without any labeled data, unlike supervised learning and since there is no labeled data, the model is used to learn from its experiences only.

Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward.

3.3 Machine Learning Models

Performing machine learning involves creating a model, which is trained on some training data and then can process additional data to make predictions. Various types of models have been used and researched for machine learning systems.

3.3.1 Artificial neural networks

Artificial neural networks (ANN) are a subfield of artificial intelligence which is simply known as neural networks. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes. ANN consists of 3 layers : input layer, output layer and hidden layer. Input layer accepts inputs in several different formats provided by the programmer. The hidden layer presents in between input and output layers. It performs all the operations to find hidden features and patterns. Output layer provides the output based on all the calculations provided by the hidden layer.

3.3.2 Convolution neural networks

Convolutional Neural Network (CNN) is the extended version of artificial neural networks (ANN) which is predominantly used to extract the feature from the grid-like matrix dataset. For example visual datasets like images or videos where data patterns play an extensive role.

3.3.2.1 CNN architecture

Convolutional Neural Network consists of multiple layers like the input layer, Convolutional layer, Pooling layer, and fully connected layers.

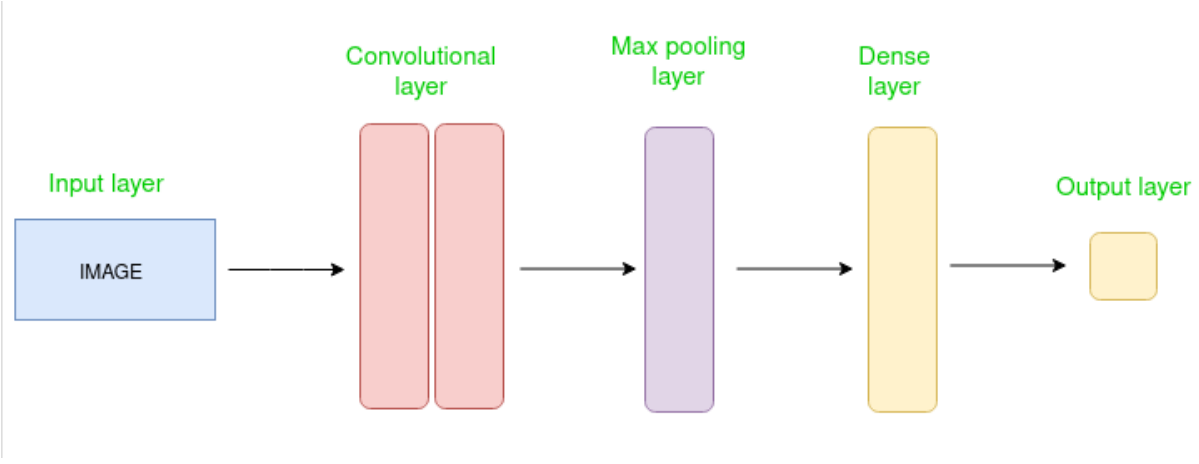
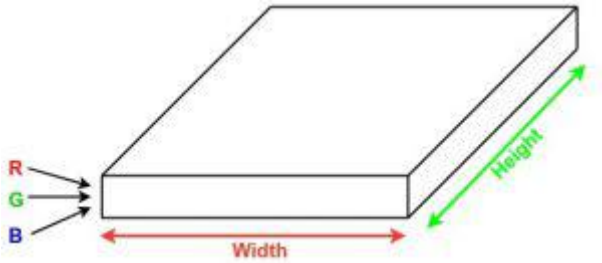


fig-1 : Simple CNN architecture

The Convolutional layer applies filters to the input image to extract features, the Pooling layer downsamples the image to reduce computation, and the fully connected layer makes the final prediction. The network learns the optimal filters through backpropagation and gradient descent.

3.3.2.2 How Convolutional Layers works

Convolutional Neural Networks or covers are neural networks that share their parameters. Imagine you have an image. It can be represented as a cuboid having its length, width (dimension of the image), and height (i.e the channel as images generally have red, green, and blue channels).



Now imagine taking a small patch of this image and running a small neural network, called a filter or kernel on it, with say, K outputs and representing them vertically. Now slide that neural network across the whole image, as a result, we will get another image with different widths, heights, and depths. Instead of just R, G, and B channels now we have more channels but lesser width and height. This operation is called Convolution. If the patch size is the same as that of the image it will be a regular neural network. Because of this small patch, we have fewer weights.

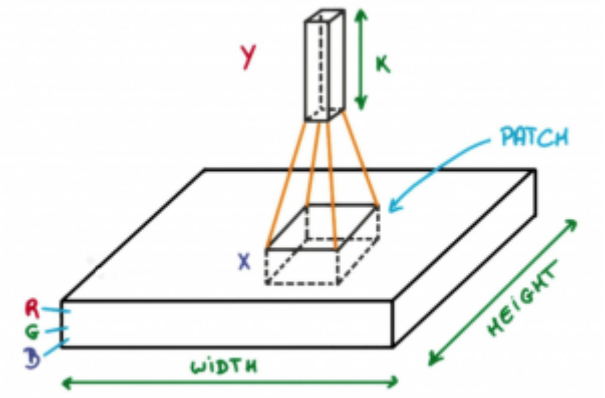


Image source: Deep Learning Udacity

3.3.2.3 Layers used to build ConvNets

A complete Convolution Neural Networks architecture is also known as covnets. A covnets is a sequence of layers, and every layer transforms one volume to another through a differentiable function.

Types of layers: datasets

Let's take an example by running a covnets on an image of dimension 32 x 32 x 3.

- **Input Layers**: It's the layer in which we give input to our model. In CNN, Generally, the input will be an image or a sequence of images. This layer holds the raw input of the image with width 32, height 32, and depth 3.
- **Convolutional Layers:** This is the layer, which is used to extract the feature from the input dataset. It applies a set of learnable filters known as the kernels to the input

images. The filters/kernels are smaller matrices, usually 2×2, 3×3, or 5×5 shapes. it slides over the input image data and computes the dot product between kernel weight and the corresponding input image patch. The output of this layer is referred to as feature maps. Suppose we use a total of 12 filters for this layer we'll get an output volume of dimension 32 x 32 x 12.

- **Activation Layer:** By adding an activation function to the output of the preceding layer, activation layers add nonlinearity to the network. it will apply an element-wise activation function to the output of the convolution layer. Some common activation functions are RELU: max(0, x), Tanh, Leaky RELU, etc. The volume remains unchanged hence output volume will have dimensions 32 x 32 x 12.
- **Pooling layer:** This layer is periodically inserted in the covnets and its main function is to reduce the size of volume which makes the computation fast, reduces memory and also prevents overfitting. Two common types of pooling layers are max pooling and average pooling. If we use a max pool with 2 x 2 filters and stride 2, the resultant volume will be of dimension 16x16x12.

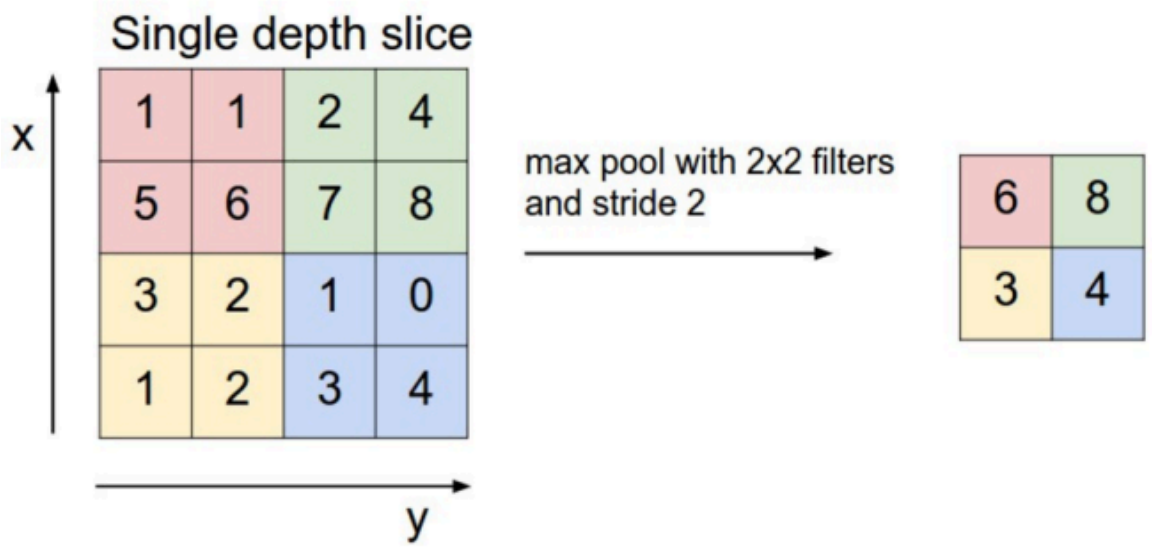


Image source: cs231n.stanford.edu

- **Flattening**: The resulting feature maps are flattened into a one-dimensional vector after the convolution and pooling layers so they can be passed into a completely linked layer for categorization or regression.
- **Fully Connected Layers:** It takes the input from the previous layer and computes the final classification or regression task.

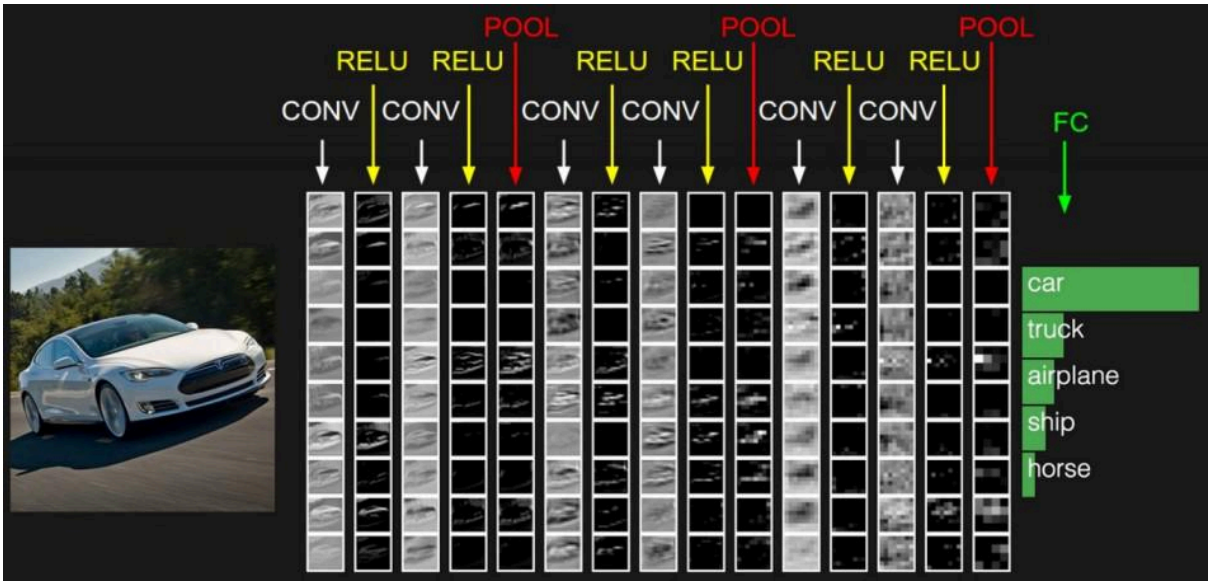


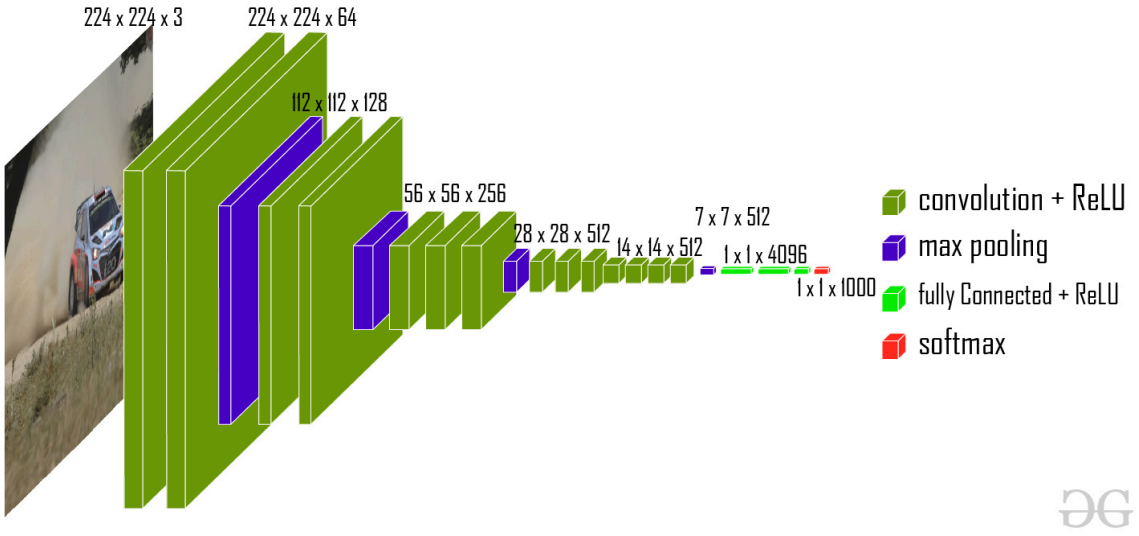
Image source: cs231n.stanford.edu

- **Output Layer:** The output from the fully connected layers is then fed into a logistic function for classification tasks like sigmoid or softmax which converts the output of each class into the probability score of each class.

3.3.3 VGG-16 | CNN model

The VGG-16 model is a convolutional neural network (CNN) architecture that was proposed by the Visual Geometry Group (VGG) at the University of Oxford. It is characterized by its depth, consisting of 16 layers, including 13 convolutional layers and 3 fully connected layers. VGG-16 is renowned for its simplicity and effectiveness, as well as its ability to achieve strong performance on various computer vision tasks, including image classification and object recognition. The model's architecture features a stack of convolutional layers followed by max-pooling layers, with progressively increasing depth. This design enables the model to learn intricate hierarchical representations of visual features, leading to robust and accurate predictions. Despite its simplicity compared to more recent architectures, VGG-16 remains a popular choice for many deep learning applications due to its versatility and excellent performance.

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual competition in computer vision where teams tackle tasks including object localization and image classification. VGG16, proposed by Karen Simonyan and Andrew Zisserman in 2014, achieved top ranks in both tasks, detecting objects from 200 classes and classifying images into 1000 categories.



VGG-16 architecture

This model achieves *92.7% top-5* test accuracy on the ImageNet dataset which contains *14* million images belonging to 1000 classes.

VGG-16 Model Objective:

The ImageNet dataset contains images of fixed size of *224*224* and have RGB channels. So, we have a tensor of *(224, 224, 3)* as our input. This model process the input image and outputs the a vector of *1000* values:

$$\hat{y} = \begin{bmatrix} \hat{y_0} \\ \hat{y_1} \\ \hat{y_2} \\ . \\ . \\ . \\ \hat{y_{999}} \end{bmatrix}$$

This vector represents the classification probability for the corresponding class. Suppose we have a model that predicts that the image belongs to class 0 with probability *1*, *class 1* with probability *0.05*, *class 2* with probability *0.05*, class *3* with probability *0.03*, *class 780* with probability *0.72*, class *999* with probability *0.05* and all other class with *0*.

so, the classification vector for this will be:

$$\hat{y} = \begin{bmatrix} \hat{y_0} = 0.1 \\ 0.05 \\ 0.05 \\ 0.03 \\ . \\ . \\ . \\ \hat{y_{780}} = 0.72 \\ . \\ . \\ \hat{y_{999}} = 0.05 \end{bmatrix}$$

To make sure these probabilities add to *1*, we use the softmax function.

This softmax function is defined as follows:

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

After this we take the 5 most probable candidates into the vector.

$$C = \begin{bmatrix} 780 \\ 0 \\ 1 \\ 2 \\ 999 \end{bmatrix}$$

and our ground truth vector is defined as follows:

$$G = \begin{bmatrix} G_0 \\ G_1 \\ G_2 \end{bmatrix} = \begin{bmatrix} 780 \\ 2 \\ 999 \end{bmatrix}$$

Then we define our Error function as follows:

$$E = \frac{1}{n} \sum_k min_i d(c_i, G_k)$$

It calculates the minimum distance between each ground truth class and the predicted candidates, where the distance function d is defined as:

- d=0 if $c_i = G_k$
- d=1 otherwise

So, the loss function for this example is :

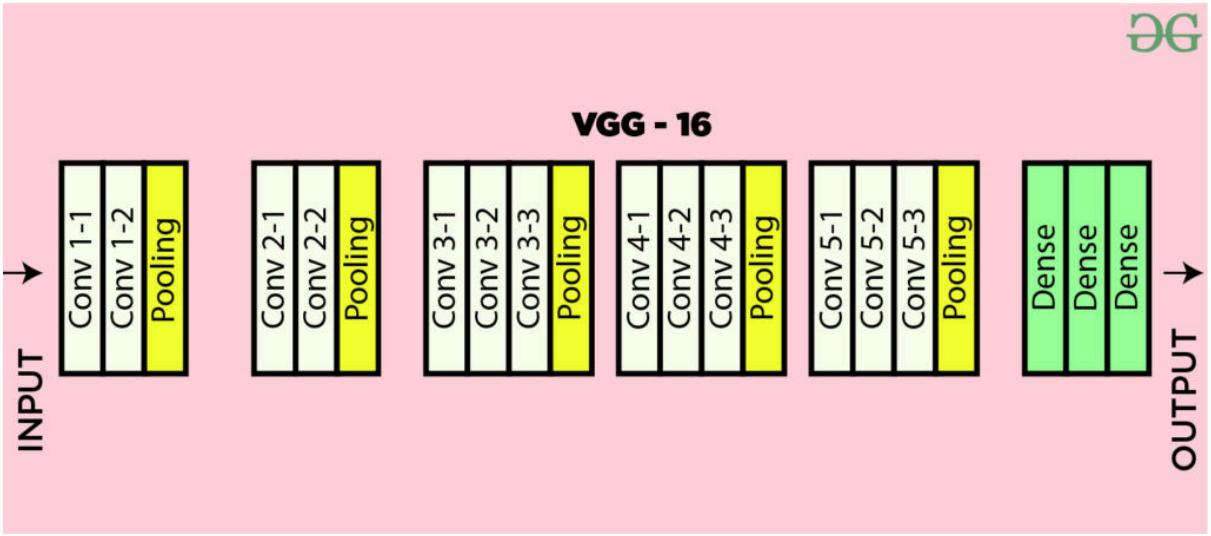
$$E = \frac{1}{3}\left(min_i d(c_i, G_1) + min_i d(c_i, G_2) + min_i d(c_i, G_3)\right)$$
$$= \frac{1}{3}(0 + 0 + 0)$$
$$= 0$$

Since, all the categories in ground truth are in the Predicted top-5 matrix, so the loss becomes 0.

3.3.3.1 VGG -16 architecture

The VGG-16 architecture is a deep convolutional neural network (CNN) designed for image classification tasks. It was introduced by the Visual Geometry Group at the University of Oxford. VGG-16 is characterized by its simplicity and uniform architecture, making it easy to understand and implement.

The VGG-16 configuration typically consists of 16 layers, including 13 convolutional layers and 3 fully connected layers. These layers are organized into blocks, with each block containing multiple convolutional layers followed by a max-pooling layer for downsampling.



VGG-16 architecture Map

3.3.3.2 Layers used to build VGG16 network

Here's a breakdown of the VGG-16 architecture based on the provided details:

1. **Input Layer:**

- Input dimensions: (224, 224, 3)

2. **Convolutional Layers (64 filters, 3×3 filters, same padding):**
    - Two consecutive convolutional layers with 64 filters each and a filter size of 3×3.
    - Same padding is applied to maintain spatial dimensions.
3. **Max Pooling Layer (2×2, stride 2):**
    - Max-pooling layer with a pool size of 2×2 and a stride of 2.
4. **Convolutional Layers (128 filters, 3×3 filters, same padding):**
    - Two consecutive convolutional layers with 128 filters each and a filter size of 3×3.
5. **Max Pooling Layer (2×2, stride 2):**
    - Max-pooling layer with a pool size of 2×2 and a stride of 2.
6. **Convolutional Layers (256 filters, 3×3 filters, same padding):**
    - Two consecutive convolutional layers with 256 filters each and a filter size of 3×3.
7. **Convolutional Layers (512 filters, 3×3 filters, same padding):**
    - Two sets of three consecutive convolutional layers with 512 filters each and a filter size of 3×3.
8. **Max Pooling Layer (2×2, stride 2):**
    - Max-pooling layer with a pool size of 2×2 and a stride of 2.
9. **Stack of Convolutional Layers and Max Pooling:**
    - Two additional convolutional layers after the previous stack.
    - Filter size: 3×3.
10. **Flattening**:
    - Flatten the output feature map (7x7x512) into a vector of size 25088.
11. **Fully Connected Layers:**
    - Three fully connected layers with ReLU activation.
    - First layer with input size 25088 and output size 4096.
    - Second layer with input size 4096 and output size 4096.
    - Third layer with input size 4096 and output size 1000, corresponding to the 1000 classes in the ILSVRC challenge.

- Softmax activation is applied to the output of the third fully connected layer for classification.

This architecture follows the specifications provided, including the use of ReLU activation function and the final fully connected layer outputting probabilities for 1000 classes using softmax activation.

Limitations Of VGG 16:

- It is very slow to train (the original VGG model was trained on the Nvidia Titan GPU for 2-3 weeks).
- The size of VGG-16 trained imageNet weights is *528* MB. So, it takes quite a lot of disk space and bandwidth which makes it inefficient.
- 138 million parameters lead to exploding gradients problem.

3.4 Confusion matrix and its metrics

3.4.1 Confusion Matrix

A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known. All the measures can be calculated by using the remaining four parameters. So, let's talk about those four parameters first.

	Predicted class		
Actual Class		Class = Yes	Class = No
	Class = Yes	True Positive	False Negative
	Class = No	False Positive	True Negative

fig : Confusion matrix with 4 parameters

True positives and true negatives are the observations that are correctly predicted. We want to minimize false positives and false negatives. These terms are a bit confusing. So, let's take each term one by one and understand it fully.

True Positives (TP) - These are the correctly predicted positive values which means that the value of actual class is yes and the value of predicted class is also yes.

True Negatives (TN) - These are the correctly predicted negative values which means that the value of actual class is no and value of predicted class is also no.

False positives and false negatives, these values occur when your actual class contradicts with the predicted class.

False Positives (FP) – When actual class is no and predicted class is yes.

False Negatives (FN) – When actual class is yes but predicted class in no.

3.4.2 AUC- ROC curve

The **Receiver Operator Characteristic (ROC)** curve is an evaluation metric for binary classification problems. It is a probability curve that plots the **TPR** against **FPR** at various threshold values and essentially **separates the 'signal' from the 'noise'**. The **Area Under the Curve (AUC)** is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.

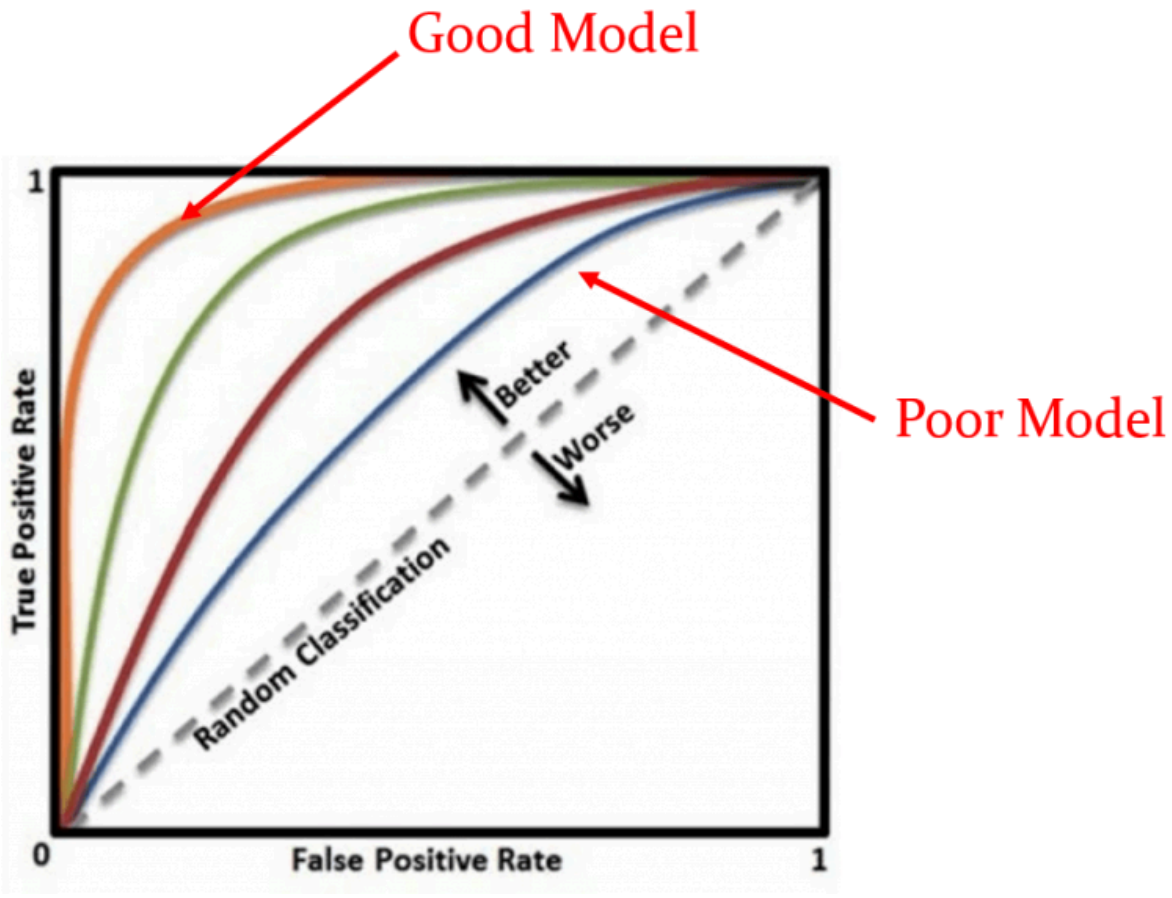


fig : AUC-ROC curve

When AUC = 1, then the classifier is able to perfectly distinguish between all the Positive and the Negative class points correctly. If, however, the AUC had been 0, then the classifier would be predicting all Negatives as Positives, and all Positives as Negatives. When 0.5<AUC<1, there is a high chance that the classifier will be able to distinguish the positive class values from the negative class values. When AUC=0.5, then the classifier is not able to distinguish between Positive and Negative class points.

3.4.3 Metrics

Accuracy:

Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations.

Accuracy = TP+TN/TP+FP+FN+TN

Precision:

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.

Precision = TP/TP+FP

Recall/ Sensitivity/ True Positive Rate (TPR):

Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes.

Recall = TP/TP+FN

F1 Score:

F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

False Positive Rate (FPR):

FPR tells us what proportion of the negative class got incorrectly classified by the classifier.

FPR=FP/TN+FP

3.5 Exploratory Data Analysis

Exploratory data analysis is an approach of analyzing data sets to summarize their main characteristics, often using statistical graphics and other data visualization methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task. EDA is different from initial data analysis (IDA) which focuses more narrowly on checking assumptions required for model fitting and hypothesis testing, and handling missing values and making transformations of variables as needed. EDA encompasses IDA. Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypotheses and to check assumptions with the help of summary statistics and graphical representations.

CHAPTER - 4

4. APPROACH DESCRIPTION

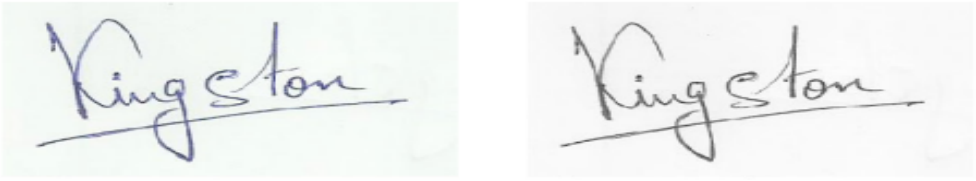
4.1 Approach Flow

1.Data Acquisition:

- Handwritten signatures are collected and some unique features are extracted to create a knowledge base for each and every individual.

2.Pre-processing:

- The first step in preprocessing a signature is to convert it from RGB to grayscale.



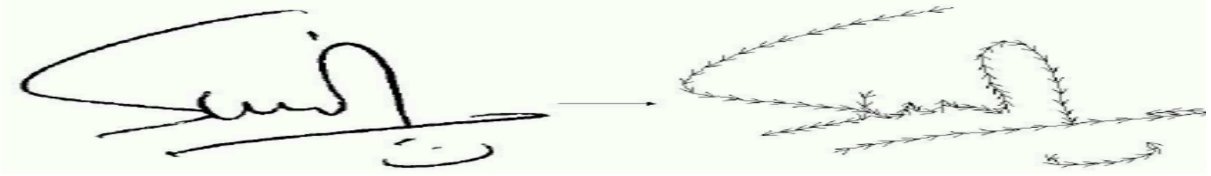
3.Noise Removal:

- Noise in images can come from many sources,including the camera,scanner,and transmission.
- To remove noise,a small amount of known noise is added to image,which makes the real noise easier to detect and remove.
- Averaging and mean filters are used to remove noise from the image.
- The addition of Known noise and the use of filters helps to improve the quality of the image.

- Noise removal process: a)Addition of salt and pepper noise. b)Removal of all Noise.

4.Grayscale to Bitmap:

- The Grayscale image is converted into bitmap when image file format is used to store digital images.
- This is because bitmap images store each pixel as a single bit,either 0 or 1,Which can represent black or white in a grayscale image.



5.Resizing:

- The system must be able to maintain high performance regardless of the size and slant of the signature.

- It is important that the system is insensitive enough for correction of the signature image.
- The image matrix is rescaled to a standard resolution of 256 x 256.

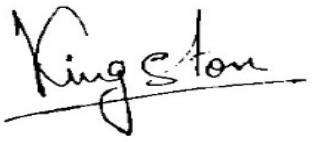


Fig.3. Grayscale without noise, to bitmap.

Fig.4. Resized bitmap.

6.File Management:

- **Data Sourcing & Preprocessing:** Organizing Diverse handwriting samples(real and forged),apply consistent formatting, and potentially augment for model robustness.
- **Model Training & Evaluation:** Tracking model versions,training parameters,and performance metrics for analysis and reproducibility.
- **Scalability & Security:** Considering the cloud storage and secure access control for large datasets and privacy concerns.

7.Training The Model

- In this application we use CNNs.(Convolutional Neural Networks).
- CCN is a class of deep,feed forward artificial neural networks that has successfully been applied to analyzing visual imagery.
- CNNs were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex.
- In our work,we use the Keras library with the TensorFlow backend to implement CNN and VGG16.
- The directory of images is loaded, preprocessed and then we train the model with training-testing split ratios to evaluate the performance.

CHAPTER - 5

5. DATA EXPLORATION

5.1 Signature Dataset

The signature forgery detection dataset is organized within a main folder, featuring two distinct subfolders: "forged" and "real." These subfolders categorize the signature images based solely on their authenticity, with the "forged" directory housing images of forged signatures, while the "real" directory contains images of genuine signatures. Each subfolder contains a collection of signature images, directly stored without any further subdivision. Every image within these subfolders is labeled to indicate whether it represents a genuine or forged signature. This straightforward organization streamlines data access and management, facilitating efficient preprocessing and model development for signature forgery detection.

5.2 Dataset Manipulation

A. Folder Organization:
   a. Initially structured the dataset into two main folders: "genuine" and "forged," segregating genuine and forged signature images.

B. File Traversal:
   a. Utilized file manipulation methods to navigate through the "genuine" and "forged" folders, accessing individual signature images.

C. Resizing Operations:
   a. Implemented resizing operations on each signature image to ensure uniform dimensions.
   b. Employed image processing libraries or functions to perform resizing while maintaining aspect ratio.

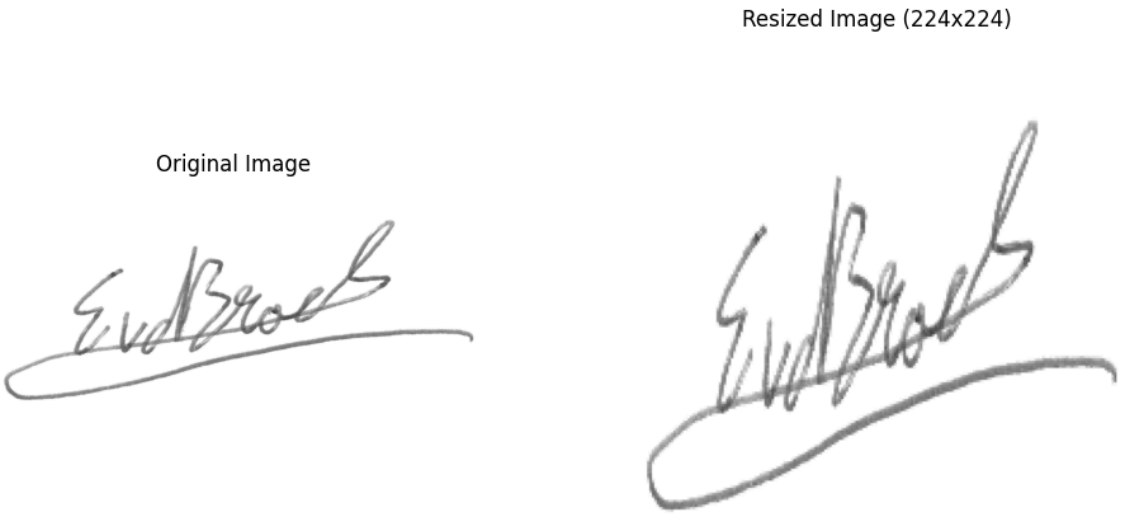
D. Consistency Enforcement:
   a. Ensured consistent image sizes across the dataset by resizing all signature images to the same dimensions.

E. Additional Manipulations:
   a. Conducted various file manipulation tasks, such as renaming files, copying images to new directories, or organizing images into subfolders based on specific criteria.

5.3 Data Preprocessing

5.3.1. Resizing

- Each signature image is resized to ensure uniform dimensions suitable for processing by the VGG16 model, which typically requires input images of size 224x224 pixels. Resizing the images to this standard size ensures consistency and compatibility with the model architecture.



5.3.2. Color conversion

- Signature images are checked for their color format. If the images are in grayscale or BGR format, they are converted to RGB format. This step ensures uniformity in color representation across all images, as the VGG16 model expects input images in RGB format.

5.3.3. Normalisation

- Pixel values in the resized and converted images are normalized to a range between 0 and 1. Normalization involves dividing each pixel value by 255.0, which scales the pixel intensities to a range suitable for model training. This step ensures numerical stability and aids in faster convergence during model training.

CHAPTER - 6

6. MODELING

6.1 Model Development

We developed three distinct models for signature forgery detection:

1. Basic Convolutional Neural Network (CNN) model
2. VGG16 model
3. Hybrid model combining elements of both CNN and VGG16 architectures

Each model is designed to analyze signature images and classify them as either genuine or forged, contributing to the overall goal of enhancing security and preventing fraud in document verification processes.

6.1.1. Basic Convolutional Neural Network (CNN) model

In the process of developing a Convolutional Neural Network (CNN) model for signature forgery detection, the following steps were undertaken:

6.1.1.1 Model Architecture

- A Sequential model was initialized.
- Two Convolutional layers were added with 32 and 64 filters respectively, followed by ReLU activation functions.
- MaxPooling layers were added after each Convolutional layer to downsample feature maps.
- The output was flattened to prepare for fully connected layers.
- A Dense layer with 128 units and ReLU activation function was added for feature aggregation.
- A Dropout layer with a rate of 0.5 was incorporated to prevent overfitting.
- Finally, an Output layer with a single unit and sigmoid activation function was added for binary classification.

```
# Create a Sequential model
model = Sequential()

# Add convolutional layers
model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(128,
128, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Flatten the output
model.add(Flatten())
```

```
# Fully connected layers
model.add(Dense(units=128, activation='relu'))
model.add(Dropout(rate=0.5))

# Output layer
model.add(Dense(units=1, activation='sigmoid'))
```

6.1.1.2 Model Compilation

- The model was compiled using the Adam optimizer and binary cross-entropy loss, suitable for binary classification tasks.

```
# Compile the model
model.compile(optimizer='adam',loss='binary_crossentropy', metrics=['accuracy'])
# Train the model
history = model.fit(X_train, y_train, epochs=30, batch_size=32, validation_data=(X_test, y_test))
```

The resulting CNN model is expected to effectively capture and learn discriminative features from signature images, enabling it to distinguish between genuine and forged signatures accurately.

6.1.2. VGG16 model

In the process of developing a Convolutional Neural Network (CNN) model for signature forgery detection, the following steps were undertaken:

6.1.2.1 Model Architecture

Base Model Initialization:

- The VGG16 model pre-trained on the ImageNet dataset was utilized as the base model.
- The weights parameter was set to 'imagenet' to load the pre-trained weights.
- The include_top parameter was set to False to exclude the fully connected layers at the top of the network.
- The input_shape parameter was set to (224, 224, 3) to match the input image dimensions.

Freezing Layers:

- All layers in the base VGG16 model were set to non-trainable to prevent their weights from being updated during training.
- This step allows us to leverage the pre-trained feature extraction capabilities of VGG16 while customizing the classification layers for our specific task.

Feature Extraction and Classification Layers:

- A Flatten layer was added to the output of the base model to convert the 3D feature maps into a 1D vector.

- A Dense layer with 256 units and ReLU activation function was added to perform feature aggregation and abstraction.
- A Dropout layer with a dropout rate of 0.5 was included to reduce overfitting by randomly dropping neurons during training.
- Finally, an Output layer with a single unit and sigmoid activation function was added for binary classification.

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

for layer in base_model.layers:
    layer.trainable = False

flatten_layer = Flatten()(base_model.output)

dense_layer = Dense(units=256, activation='relu')(flatten_layer)
dropout_layer = Dropout(0.5)(dense_layer)
output_layer = Dense(units=1, activation='sigmoid')(dropout_layer)

model = Model(inputs=base_model.input, outputs=output_layer)
```

6.1.2.2 Model Compilation

- The model was compiled using the Adam optimizer with a learning rate of 1e-4.
- Binary cross-entropy loss was chosen as the loss function, and accuracy was used as the evaluation metric.

```
model.compile(optimizer=Adam(lr=1e-4), loss='binary_crossentropy',
metrics=['accuracy'])
```

6.1.3. Hybrid Model (VGG16 + CNN)

For the VGG16 model with custom top layers developed for signature forgery detection, the following steps were undertaken:

6.1.3.1 Model Architecture

Base Model Initialization:

- The VGG16 model pre-trained on the ImageNet dataset was utilized as the base model.
- The weights parameter was set to 'imagenet' to load the pre-trained weights.
- The include_top parameter was set to False to exclude the fully connected layers at the top of the network.
- The input_shape parameter was set to (224, 224, 3) to match the input image dimensions.

Freezing Layers:

- All layers in the base VGG16 model were set to non-trainable to prevent their weights from being updated during training.

- This step allows us to leverage the pre-trained feature extraction capabilities of VGG16 while customizing the classification layers for our specific task.

Custom Top Layers:

- A new Sequential model was created.
- The base VGG16 model was added as the first layer of the Sequential model.
- A Flatten layer was added to the output of the base model to convert the 3D feature maps into a 1D vector.
- A Dense layer with 256 units and ReLU activation function was added to perform feature aggregation and abstraction.
- A Dropout layer with a dropout rate of 0.5 was included to reduce overfitting by randomly dropping neurons during training.
- Finally, an Output layer with a single unit and sigmoid activation function was added for binary classification.

```
# Load pre-trained VGG16 model without top layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freeze the VGG16 layers
for layer in base_model.layers:
    layer.trainable = False

# Create a new model by adding custom layers on top of VGG16
model = Sequential([
    base_model,
    Flatten(),
    Dense(units=256, activation='relu'),
    Dropout(rate=0.5),
    Dense(units=1, activation='sigmoid')
])
```

6.1.3.2 Model Compilation

- The model was compiled using the Adam optimizer with a learning rate of 1e-4.
- Binary cross-entropy loss was chosen as the loss function, and accuracy was used as the evaluation metric.

```
# Compile the model
model.compile(optimizer=Adam(lr=1e-4), loss='binary_crossentropy',
metrics=['accuracy'])
```

6.2. Model Evaluation & Classification Report

The classification report provides a comprehensive summary of the model's performance on the test set:

6.2.1. Basic Convolutional Neural Network (CNN) model

```
Test Loss: 1.6341
Test Accuracy: 74.35%
```

6.2.2. VGG16 model

```
Classification Report:
              precision    recall  f1-score   support

   Genuine 0       0.85      0.81      0.83       140
   Forged  1       0.73      0.78      0.75        90

    accuracy                           0.80       230
   macro avg       0.79      0.80      0.79       230
weighted avg       0.80      0.80      0.80       230`
```

6.2.3. Hybrid Model (VGG16 + CNN)

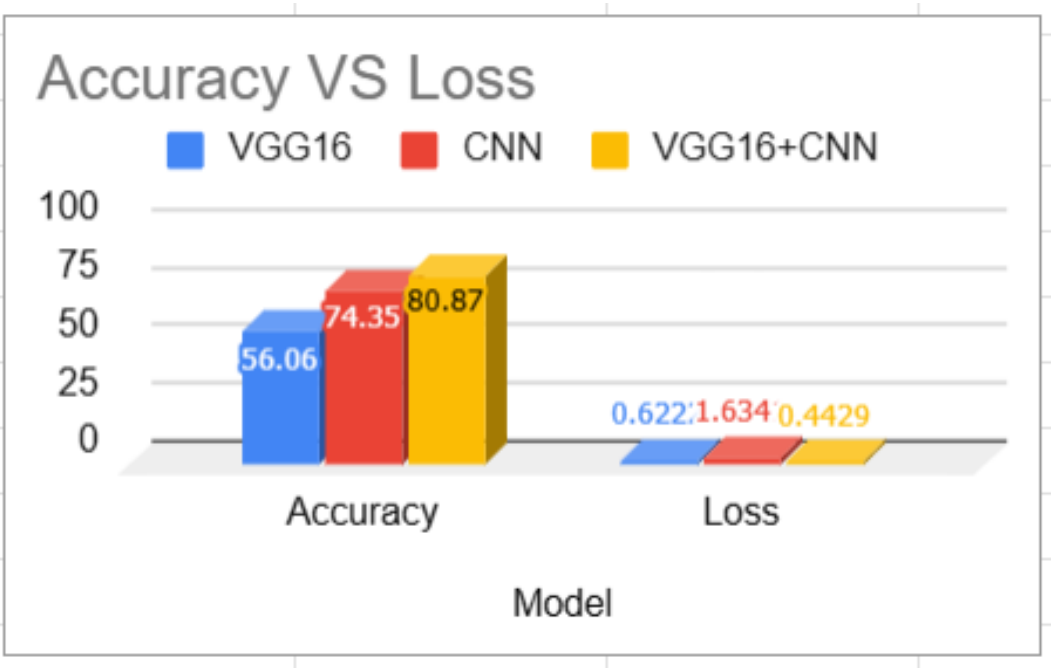
```
Classification Report:
              precision    recall  f1-score   support

   Genuine 0       0.55      0.70      0.61        33
    Forged 1       0.58      0.42      0.49        33

    accuracy                           0.56        66
   macro avg       0.57      0.56      0.55        66
weighted avg       0.57      0.56      0.55        66
```

CHAPTER - 7

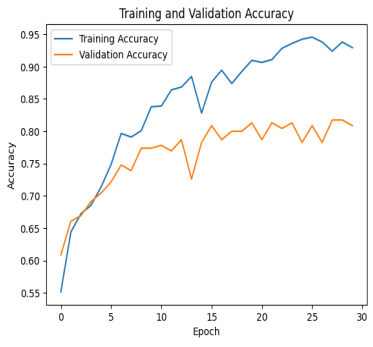
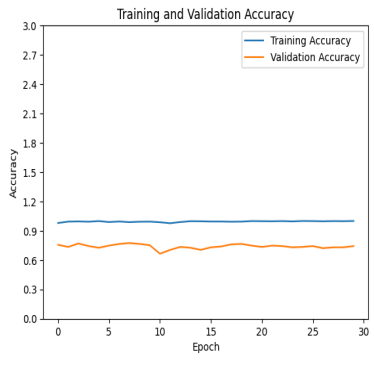
7. RESULTS AND CONCLUSIONS

7.1. Model Comparison Table

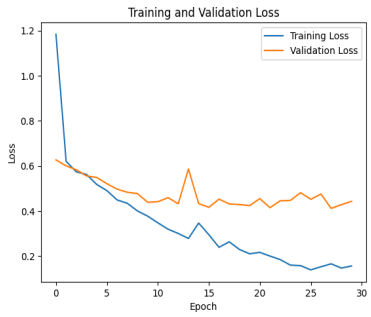
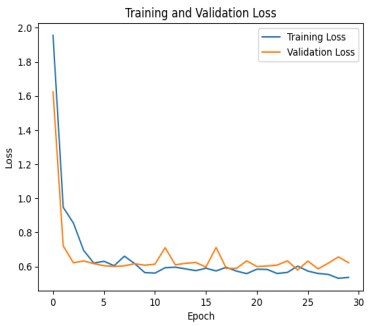
Model	Accuracy	Loss
VGG16	56.06	0.6222
CNN	74.35	1.6341
VGG16+CNN	80.87	0.4429



7.2. Training & Validation Accuracy Comparison

Hybrid Algorithm (CombinationVGG16,CNN)	Basic CNN Algorithm	VGG16 Algorithm
		

7.3. Training & Validation Loss Comparison

Hybrid Algorithm (CombinationVGG16,CNN)	Basic CNN Algorithm	VGG16 Algorithm
		

7.4.Conclusion

✓ **VGG16 Model**: Achieved an accuracy of 56.06% with a loss of 0.6222. While the accuracy is relatively low, the loss is moderate. This indicates that the VGG16 model struggled to effectively learn the features necessary for accurate classification, resulting in a lower accuracy.

✓ **CNN Model:** Achieved an accuracy of 74.35% with a loss of 1.6341. The CNN model performed better than the VGG16 model, demonstrating higher accuracy. However, the loss is relatively high, suggesting that the model may be overfitting to the training data.

✓ **Hybrid VGG16 + CNN Model:** Achieved the highest accuracy of 80.87% with a loss of 0.4429. The hybrid model outperforms both the VGG16 and CNN models in terms of accuracy and loss. Combining the strengths of both architectures likely contributed to the improved performance of the hybrid model.

REFERENCES

- *M. Subramaniam, E. Teja, & A. Mathew, Signature forgery detection using machine learning. Int. Res. J. Modernization Eng. Technol. Sci., 4(2), pp. 479-483, 2022.*
- *J. Poddar, V. Parikh, & S. K. Bharti, Offline signature recognition and forgery detection using deep learning. Procedia Computer Science, 170, pp. 610-617, 2020.*
- *S. Jain, M. Khanna, & A. Singh, Comparison among different CNN Architectures for Signature Forgery Detection using Siamese Neural Network. In 2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), pp. 481-486, 2021.*
- *E., Teja, A. Mathew, & M., Pragathi, Comparison of signature forgery detection architectures, International Journal for Research in Applied Science & Engineering Technology (IJRASET), 10(06), pp. 4124-4134, 2022.*
- *E. Alajrami, B. A. Ashqar, B. S. Abu-Nasser, A. J. Khalil, M.M. Musleh, A.. M., Barhoom, & S. S. Abu-Naser, Handwritten signature verification using deep learning, International Journal of Academic multidisciplinary Research, 3(12), pp. 39-44, 2019.*
- *K. Kancharla, V. Kamble, and M. Kapoor, "Handwritten signature recognition: a convolutional neural network approach," in 2018 International Conference on Advanced Computation and Telecommunication(ICACAT), pp. 1–5, IEEE, 2018.*
- *F. Noor, A. E. Mohamed, F. A. Ahmed, and S. K. Taha, "Offline handwritten signature recognition using convolutional neural network approach," in 2020 International Conference on Computing, Networking, Telecommunications Engineering Sciences Applications (CoNTESA),*
  pp. 51–57, IEEE, 2020.
- *J. Poddar, V. Parikh, and S. K. Bharti, "Offline signature recognition and forgery detection using deep learning," Procedia Computer Science,vol. 170, pp. 610–617, 2020.*
- *V. L. Souza, A. L. Oliveira, and R. Sabourin, "A writer-independent approach for offline signature verification using deep convolutional neural networks features," in 2018 7th Brazilian Conference on Intelligent Systems (BRACIS), pp. 212–217, IEEE, 2018.*
- *S. M. Sam, K. Kamardin, N. N. A. Sjarif, N. Mohamed, et al., "Offline signature verification using deep learning convolutional neural network(cnn) architectures googlenet inception-v1 and inception-v3," Procedia Computer Science, vol. 161, pp. 475–483, 2019.*

Appendix-A: Libraries, Tools used & Working Process

A.1. Python Programming language

Python is a high-level Interpreter based programming language used especially for general-purpose programming. Python features a dynamic type of system and supports automatic memory management.

It supports multiple programming paradigms, including object-oriented, functional and Procedural and also has a large and comprehensive standard library. Python is of two versions. They are Python 2 and Python 3.

This project uses the latest version of Python, i.e., Python 3. This python language uses different types of memory management techniques such as reference counting and a cycle-detecting garbage collector for memory management. One of its features is late binding (dynamic name resolution), which binds method and variable names during program execution.

Python's offers a design that supports some of the things that are used for functional programming in the Lisp tradition. It has vast usage of functions for faster results such as filter, map, split, list comprehensions, dictionaries, sets and expressions. The standard library of python language has two modules like itertools and functools that implement functional tools taken from Standard machine learning.

A.2. OS

- Python has a built-in os module with methods for interacting with the operating system, like creating files and directories, management of files and directories, input, output, environment variables, process management, etc.

A.3. CV2

- OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as Numpy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e whatever operations one can do in Numpy can be combined with OpenCV.

A.4.Numpy

- NumPy stands for Numerical Python, supports large arrays and matrices and can write advanced arithmetic operations that operate on arrays. NumPy's powerful implementation in C and Fortran makes NumPy the library of choice for computing in Python. It is a Python library that helps us work with the arrays. It also has functions for working in the domain of linear algebra, Fourier transform, and matrices.

A.5. Matplotlib

- Python is the most used language for Matplotlib is a plotting library for creating static, animated, and interactive visualizations in Python. Matplotlib can be used in Python scripts, the Python and IPython shell, web application servers, and various graphical user interface toolkits like Tkinter, awxPython, etc.

A.6. From sklearn.model_selection import train_test_split

- The train_test_split() method is used to split our data into train and test sets.
- First, we need to divide our data into features (X) and labels (y). The dataframe gets divided into X_train,X_test , y_train and y_test. X_train and y_train sets are used for training and fitting the model. The X_test and y_test sets are used for testing the model if it's predicting the right outputs/labels. we can explicitly test the size of the train and test sets. It is suggested to keep our train sets larger than the test sets.

A.7. From tensorflow.keras.applications import VGG16

- The VGG16 model from tensorflow.keras.applications is a deep convolutional neural network architecture that was developed and trained by the Visual Graphics Group (VGG) at Oxford. It's well-known for its performance in image recognition tasks and was a top contender in the 2014 ImageNet Large Scale Visual Recognition Challenge (ILSVRC). The model is characterized by its simplicity, using only 3x3 convolutional layers stacked on top of each other in increasing depth. Reducing volume size is handled by max pooling. ReLU activation functions are used throughout the network.

A.8. From tensorflow.keras.models import Sequential

- The Sequential model in Keras is a linear stack of layers. It's a simple yet powerful way of building neural network models in TensorFlow. The Sequential model API allows you to create models layer-by-layer for most problems. It is limited in that it does not allow you to create models that share layers or have multiple inputs or

outputs, which are possible in the more flexible Functional API. However, for many tasks, the Sequential model is straightforward and easy to use.

A.9. From tensorflow.keras.layers import Flatten, Dense, Dropout

- The Flatten layer transforms its input into a one-dimensional array (flattens it). This layer does not affect the batch size. It is often used when transitioning between convolutional layers (which operate on 3D tensors) and dense layers (which operate on 1D vectors) in a neural network.
- The Dense layer is a standard fully connected neural network layer where each input node is connected to each output node. It is used to change the dimensions of your vector. Mathematically, it applies a linear operation on the input data (followed by an optional activation function).
- The Dropout layer randomly sets input units to 0 with a frequency of the rate parameter at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged.

A.10. From tensorflow.keras.optimizers import Adam

- To use the Adam optimizer in TensorFlow/Keras, you can import it from tensorflow.keras.optimizers and specify it in the compile method of your model. You can use Adam with its default parameters, but it also allows you to customize several options, including the learning rate (lr), which is one of the most commonly adjusted parameters.

Appendix-B: Sample Source Code with Execution

B.1. Sample Code(Hybrid Algorithm)

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
# Function for preprocessing signature images
def preprocess_image(image):
    # Resize the image to match VGG16 input size
    image_resized = cv2.resize(image, (224, 224))
    # Convert the image to RGB (if it's in BGR format)
    if len(image_resized.shape) == 2:
        image_resized = cv2.cvtColor(image_resized,
cv2.COLOR_GRAY2RGB)
    else:
        image_resized = cv2.cvtColor(image_resized,
cv2.COLOR_BGR2RGB)
    # Normalize the pixel values
    image_normalized = image_resized / 255.0
    return image_normalized

# Load genuine and forged images
real_path = '/content/drive/MyDrive/JETTI PROJECTS/MINI project/final
collected data set/genuine'
forge_path = '/content/drive/MyDrive/JETTI PROJECTS/MINI
project/final collected data set/forged'

real_images = [preprocess_image(cv2.imread(os.path.join(real_path,
img_name))) for img_name in os.listdir(real_path)]
forge_images = [preprocess_image(cv2.imread(os.path.join(forge_path,
img_name))) for img_name in os.listdir(forge_path)]


# Create labels for genuine (0) and forged (1) signatures
real_labels = np.zeros(len(real_images))
```

```
forge_labels = np.ones(len(forge_images))

# Combine data and labels
X_data = np.concatenate((real_images, forge_images), axis=0)
y_labels = np.concatenate((real_labels, forge_labels), axis=0)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_data, y_labels,
test_size=0.2, random_state=42)

# Load pre-trained VGG16 model without top layers
base_model = VGG16(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))

# Freeze the VGG16 layers
for layer in base_model.layers:
    layer.trainable = False

# Create a new model by adding custom layers on top of VGG16
model = Sequential([
    base_model,
    Flatten(),
    Dense(units=256, activation='relu'),
    Dropout(rate=0.5),
    Dense(units=1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer=Adam(lr=1e-4), loss='binary_crossentropy',
metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=30, batch_size=32,
validation_data=(X_test, y_test))

# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy*100:.2f}%")
```

B.2. Output

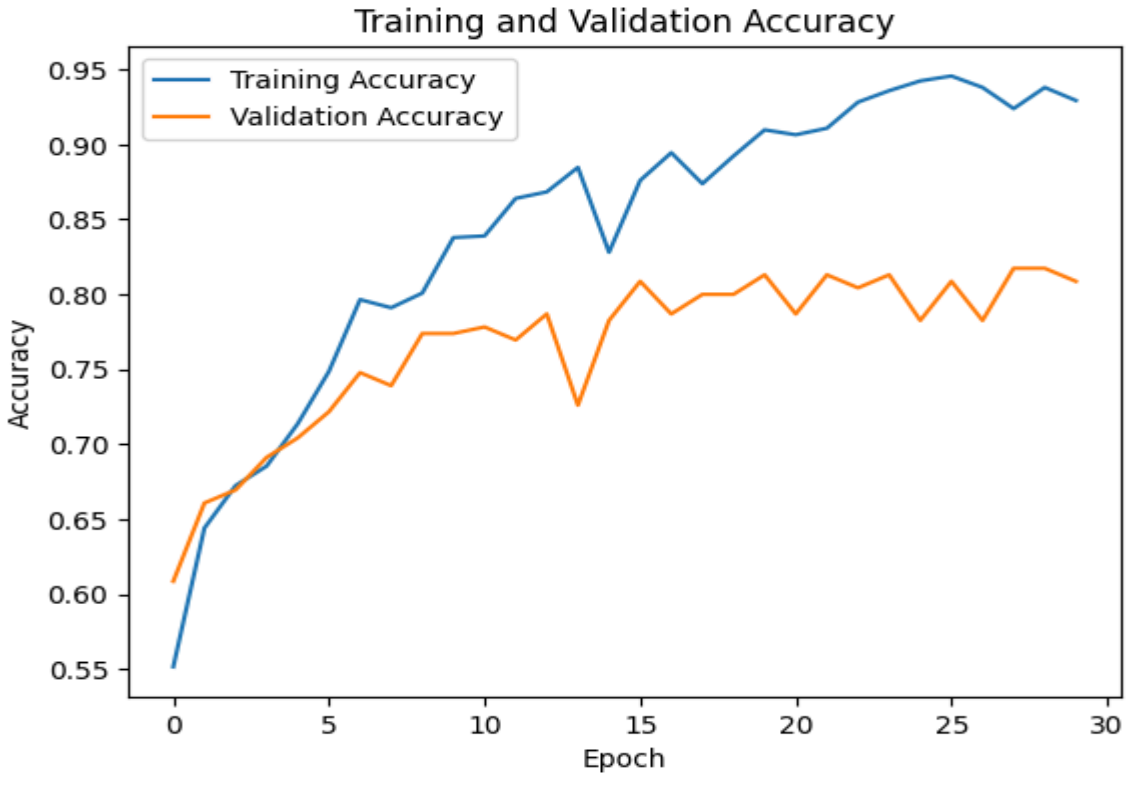
```
Epoch 24/30
29/29 [==============================] - 5s 159ms/step - loss: 0.1603 - accuracy: 0.9358 - val_loss: 0.4471 - val_accuracy: 0.8130
Epoch 25/30
29/29 [==============================] - 5s 158ms/step - loss: 0.1579 - accuracy: 0.9423 - val_loss: 0.4813 - val_accuracy: 0.7826
Epoch 26/30
29/29 [==============================] - 5s 173ms/step - loss: 0.1392 - accuracy: 0.9456 - val_loss: 0.4521 - val_accuracy: 0.8087
Epoch 27/30
29/29 [==============================] - 5s 159ms/step - loss: 0.1529 - accuracy: 0.9380 - val_loss: 0.4755 - val_accuracy: 0.7826
Epoch 28/30
29/29 [==============================] - 5s 159ms/step - loss: 0.1659 - accuracy: 0.9238 - val_loss: 0.4118 - val_accuracy: 0.8174
Epoch 29/30
29/29 [==============================] - 5s 175ms/step - loss: 0.1472 - accuracy: 0.9380 - val_loss: 0.4281 - val_accuracy: 0.8174
Epoch 30/30
29/29 [==============================] - 5s 160ms/step - loss: 0.1564 - accuracy: 0.9293 - val_loss: 0.4429 - val_accuracy: 0.8087
8/8 [==============================] - 1s 114ms/step - loss: 0.4429 - accuracy: 0.8087
Test Loss: 0.4429
Test Accuracy: 80.87%
```

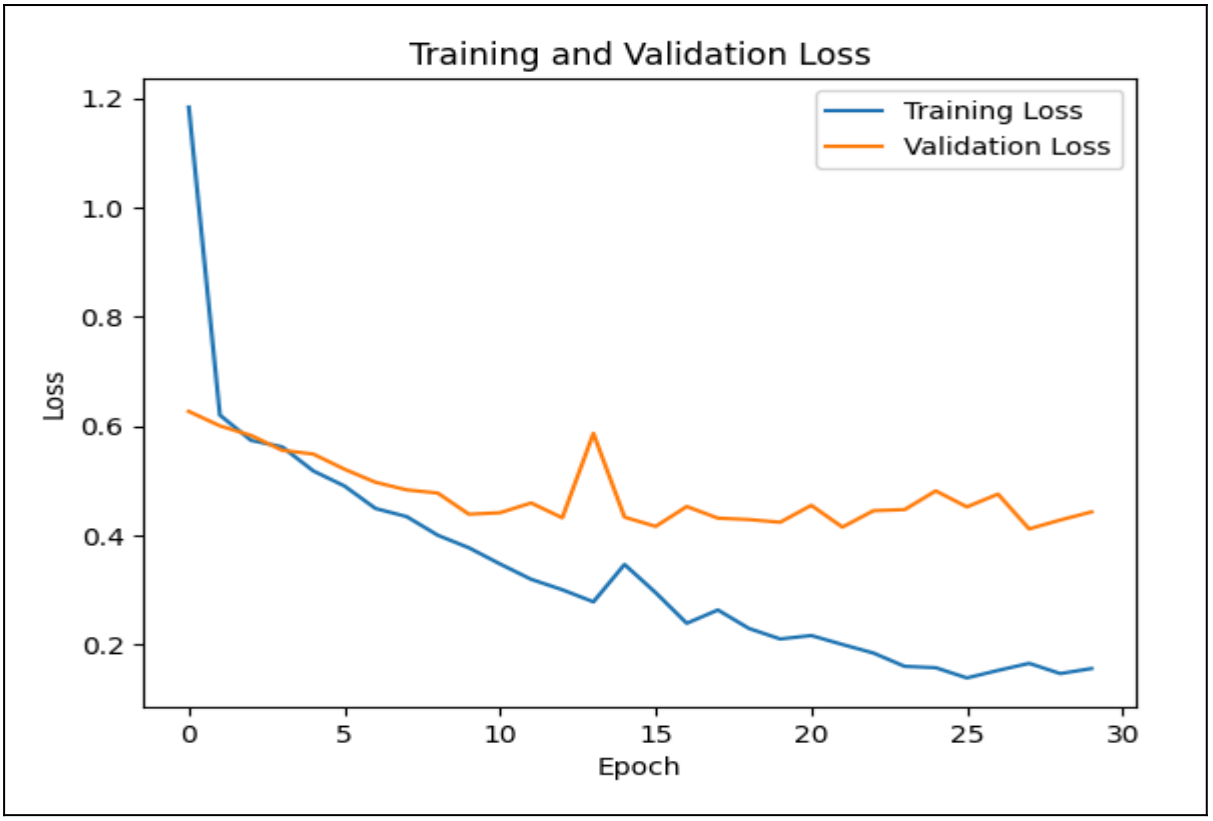
```python
import cv2
import numpy as np

# Function to preprocess a single image
def preprocess_image(image_path):
    img = cv2.imread(image_path)
    img = cv2.resize(img, (224, 224))  # Adjust the size accordin
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = img.astype('float32') / 255.0
    img = np.expand_dims(img, axis=0)  # Add batch dimension
    return img

image_path = '/content/drive/MyDrive/JETTI PROJECTS/MINI project/
image = preprocess_image(image_path)
prediction = model.predict(image)
result = "Forged" if prediction[0][0] > 0.5 else "Genuine"
print(f"The signature is predicted as: {result}")
```

```
1/1 [==============================] - 0s 34ms/step
The signature is predicted as: Genuine
```

```python
import cv2
import numpy as np

# Function to preprocess a single image
def preprocess_image(image_path):
    img = cv2.imread(image_path)
    img = cv2.resize(img, (224, 224))  # Adjust the size acc
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = img.astype('float32') / 255.0
    img = np.expand_dims(img, axis=0)  # Add batch dimension
    return img

image_path = '/content/drive/MyDrive/JETTI PROJECTS/MINI pro
image = preprocess_image(image_path)
prediction = model.predict(image)
result = "Forged" if prediction[0][0] > 0.5 else "Genuine"
print(f"The signature is predicted as: {result}")
```

```
1/1 [==============================] - 0s 18ms/step
The signature is predicted as: Forged
```





B.3. GitHub Repository

https://github.com/Swathijetti2004/Signature-Foregery-Detection