Misr University for Science and Technology
Faculty of Engineering
Department of Computer and Software

Deep Learning Copy-Move Image Forgery Detection

Prepared By:

Abdalrhman Morsi
Khaled Mohamed
Ahmed Abdulfattah
Ahmed Bayoumy
Mohamed Nasser

Supervised By:

Prof. Dr. Heba Elnemr

2022-2023

DECLARATION

We hereby declare that the work presented in this thesis has not been submitted for any other degree or professional qualification, and that it is the result of my own independent work.

Names:

Date:

ACKNOWLEDGEMENT

We would like to thank:

- Dr. Heba Elnemr our supervisor who supports us in all circumstances.
- Dr. Ashraf Mahrous the head of computer and software department.
- Dr. Tamer Nassef Vice Dean of the College of Engineering.
- Dr. Ghada Amer Dean of the College of Engineering.

for their effort and motivational support, you believe in us, encourage us and provide sufficient help, it has been a great honor working under your supervision.

We would like to thank our parents and families who support and motivate us during our long journey. We also want to thank all the people who supported us and believed in us, we couldn't do it without you.

ABSTRACT

The proliferation of images being shared online has led to a pressing demand for automated systems capable of detecting image forgeries. The surge in high-quality fake images on social networks and media platforms has created an urgent need for recognition algorithms specifically designed to identify and detect such content. Among the various techniques used for image and video editing, the copy-move technique, which involves duplicating areas of an image, is particularly common. Traditional image processing methods rely on manual pattern recognition to detect duplicated content, making them less effective for large-scale data classification. In contrast, deep learning approaches have demonstrated superior performance and promising outcomes in this regard. However, these methods face challenges related to generalization, as they heavily rely on training data and require careful selection of hyperparameters.

In our research, we propose a deep learning architecture for copy-move forgery detection (CMFD) that leverages various deep learning approaches, including a custom convolutional neural network (CNN) model, VGG16, ResNet50, and MobileNetV2. Additionally, we employ an object detection approach using YOLOv7 for copy-move forgery detection. To evaluate the effectiveness of our frameworks, we conducted experiments on a large-scale dataset. Our custom CNN model trained on image patches achieved an impressive accuracy of 95%, while the object detection approach achieved a main average precision (mAP) of 83.9%. Furthermore, we developed a web-based application called the CMFD App, which aims to detect image copy forgeries. This application utilizes the proposed deep learning architectures and provides users with a user-friendly interface to identify manipulated images.

TABLE OF CONTENT

TABLE OF ABBREVIATIONS

AI	Artificial Intelligence
CMFD	Copy Move Forgery Detection
CNN	Convolutional Neural Network
CovLSTM	Convolutional Long Short-Term Memory
CX	Customer Experience
FC Layers	Fully Connected Layers
GAN	Generative Adversarial Network
GPU	Graphics processing unit
ITU	International Telecommunication Union
LSTM	Long Short-term Memory Networks
ML	Machine Learning
MLP	Multilayer Perceptron
NLP	Natural Language Processing
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SNN	Shallow Neural Network
TL	Transfer Learning
YOLO	You Only Look Once

LIST OF FIGURES

LIST OF Tables

Chapter 1

INTRODUCTION

In recent years, the expansion of internet services and the proliferation and empowerment of social platforms such as Facebook, Instagram, and Reddit have had a significant impact on the amount of content circulating in digital media. According to the International Telecommunication Union (ITU), 53.6% of the world's population uses the Internet at the end of 2019, which means that approximately 4.1 billion people have access not only to this technology, but also to various tools available online (ITU Statistics, 2018). Although, in most cases, the content shared is original or has been processed for entertainment purposes only. In other cases, the manipulation may be deliberate for disinformation purposes with political and forensic implications, such as the use of fake content as digital evidence in a criminal investigation.

We have many straightforward image editing programs, such as Photoshop. Moreover, forger introduced various methods for image processing to obtain forged images in a tricky way, such as copy-move image forgery that uses the same image, image splicing that uses diverse parts from various images to manipulate pictures, and image retouching that leaves a fine modification in the picture. In image splicing, we optimize areas from various pictures to make a forged one. In copy-move image forgery, regions of the images could be duplicated into the same image to hide an important content in that image.

Since the appropriate elements are like the copied parts, such as color and noise, it is necessary and important to distinguish the manipulated areas from the real ones. In addition, various post-processing techniques such as blurring, anti-aliasing and noise are used by the forger to remove visual traces of image forgery.

This project report concentrates on Copy Move Forgery Detection (CMFD) and try to solve this problem by different ways. An example of a copy- move forgery image is illustrated in Figure 1.1.

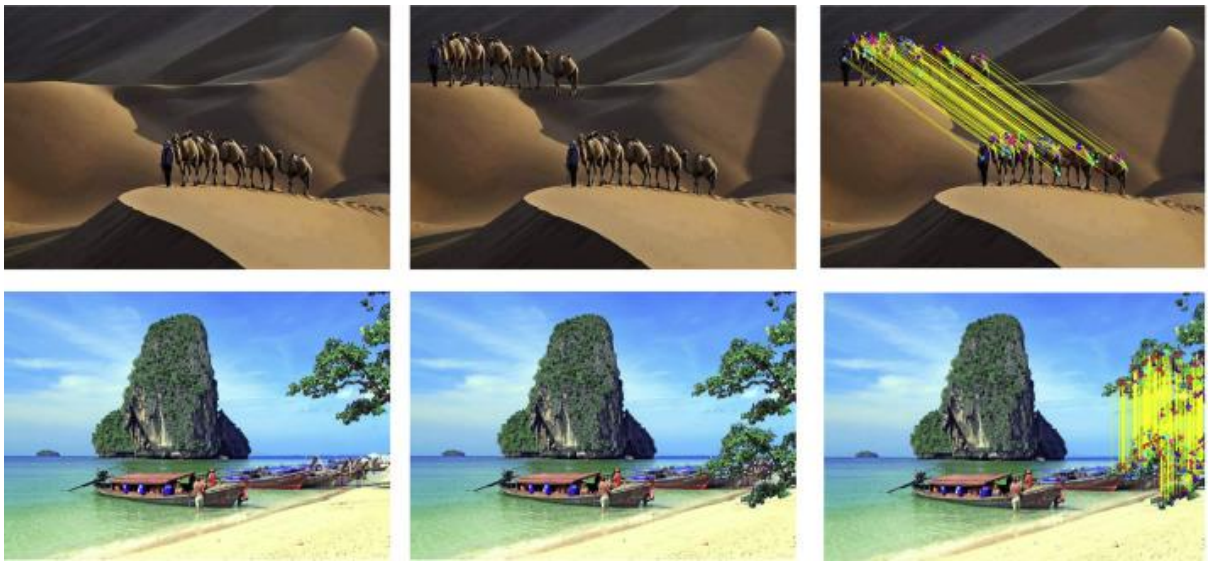

Figure 1. 1 Examples of copy-move forgery

A CMFD system has a wide range of applications in various fields. It can be used in digital forensics to investigate fraudulent activities involving images such as those in legal proceedings, criminal investigations, and intellectual property rights protection. It can also be used in the media and journalism industry to detect image tampering and ensure the authenticity of images before publishing. In addition, the system can be useful in the field of medicine, where medical images are crucial for diagnosis and treatment, and any manipulation can lead to incorrect conclusions. Over and above, CMFD can serve as a means of safeguarding intellectual property by identifying any instances of unauthorized usage or modifications to copyrighted visual content. Additionally, it can help prevent cyberbullying and harassment by detecting and removing doctored images. Overall, the applicability of CMFD systems is vast and can benefit various fields that rely on image authenticity.

The enforcement of deep learning techniques for the application of CMFD is an exciting and challenging undertaking. Deep learning can be trained to detect instances of copy-move forgery in digital images, making it a valuable tool for forensic analysis. In this work, we will scrutinize and analyze the usage of various deep-learning techniques for detecting copy-move forgery, opening new possibilities for image analysis and authentication. The deep learning

approaches investigated in this study are a custom-designed CNN model, VGG16, and resnet50. Furthermore, we have developed a CMFD App, which is a web-based enforcement for detecting copy-move forgery in images.

This report is organized as follows. **Chapter 2** presents a comprehensive summary of previous researches on CMDF and the advantages and disadvantages of each. In **Chapter 3** a description of the dataset is reported and discussed. **Chapter 4** introduced a brief description for deep learning techniques. **Chapter 5** demonstrates the methodology used to solve the problem of CMFD and a quick explanation of the methods we will implement to solve this problem. **Chapter 6** portrays the experimental results. **Chapter 7** description of the web application that applies forgery detection. **Chapter 8** the Discussion and comparison between the results. Finally, **Chapter 9** summarizes the suggested work in the conclusion.

Chapter 2

REVIEW OF LITERATURE

This section explores recent developments in deep learning-based algorithms for CMFD and discusses their effectiveness.

2.1 First Paper [2]

This work [2] illustrates the implementation of a deep-learning approach for detecting copy-move forgery. The proposed model depends on applying CNN in addition to Convolutional Long Short-Term Memory (CovLSTM) networks. This method extracts image features by a sequence number of Convolutions (CNVs) layers, (ConvLSTM) layers, and pooling layers, then matching features and detecting copy move forgery. This model had been applied to four aboveboard available databases, MICC-F220, MICC-F2000, MICC-F600, and SATs-130. Moreover, datasets have been combined to build new datasets, for all purposes of generalization testing and coping with an over-fitting problem. In addition, the results of applying the ConvLSTM model only have been added to show the differences in performance between using hybrid ConvLSTM and CNN compared with using CNN only. This technique had proved its robustness, easy, efficiency, and speed in discovering the copy move forgery. The proposed model depends on utilizing CNN and CovLSTM networks.

2.2 Second Paper [3]

E. U. H. Q. &. T. Zia (Zia, 2021) proposed a deep learning-based approach for image forgery detection. The offered model relied on ResNet50v2 architecture, which used residual layers. Thus, using this architecture increases the detection rate of tampered images. This approach also provides the benefit of transfer learning (TL) by using the pre-trained weights of the YOLO-V7 CNN model.

The TL is used to train the model more efficiently, initializing the proposed model by meaningful assigning weights. This reduces the training

time and complexity of the model and makes the architecture more efficient. The proposed architecture is evaluated on benchmark datasets; CASIA_v1 and CASIA_v2. Furthermore, the performance of the system is compared to that without TL.

2.3 Third Paper [4]

In the proposed method (Kumar, 2021), two streams are used to mine features and further combined to generate the most discriminable features.

Stream I: Handcrafted engineering features:

- The image is segmented into blocks of size 8 × 8. The three-color components are extracted and processed separately.
- Treating the picture as Markov model based on treating it in 1d signal.
- In this method state dependencies along with minor and major diagonals are also considered to represent the image better.

Stream II: Extracting off-the-shelf features from RESNET-18:

- Transfer RGB image to YCbCr image: The advantage of YCbCr color space is that it can separate luminance from chrominance more efficiently than RGB
- YCbCr colorspace is utilized to extract the Local Binary Pattern of the image. These local binary feature maps are fed to the pre-trained ResNet-18 model to obtain a 512-D feature vector.

The handcrafted features from Stream I and ResFeats from Stream II are merged. Classification is, finally, achieved by a shallow neural network.

2.4 Fourth Paper [5]

The proposed framework (Bala, 2020) is divided into two parts: the first part performs minimal pre-processing and on-the-fly operations, whereas the second part is the modified CNN architecture that extracts the features from these pre-processed images and performs binary classification of images as original or tampered. The basic block diagram of the framework including the proposed architecture is illustrated in Figure 2.1.

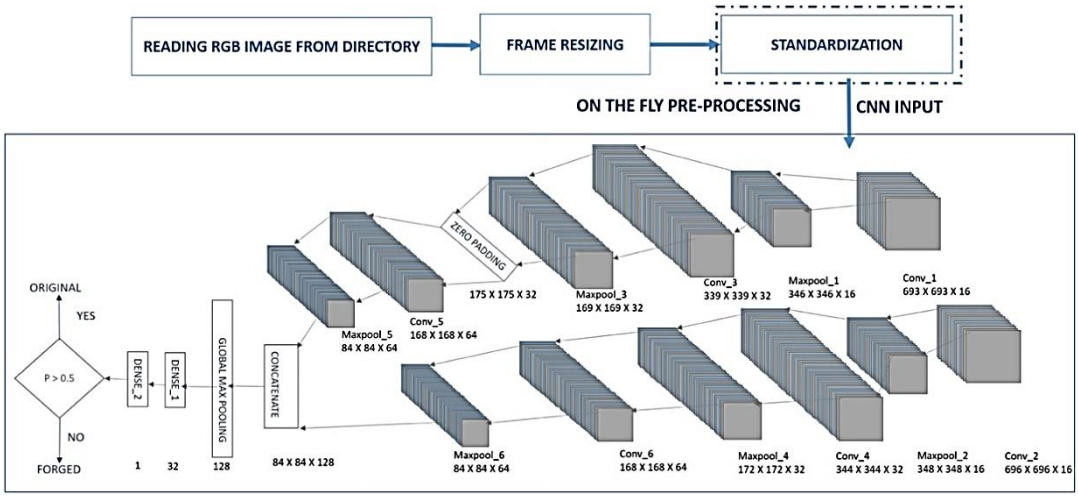

Figure 2. 1 Architecture of fourth paper model

The proposed architecture is dual branch CNN-based architecture, where both the branches are connected to a common input. There are three convolution layers in each branch, with 16, 32 and 64 feature maps for the first, second and third layer, respectively. All the convolutional layers use Relu activation and each convolutional layer is followed by a 2×2 max-pooling layer. To extract multi-scale features from the images, CNN layers in these two branches have different kernel size.

2.5 Conclusion of Literature Review

The issue of CMFD has been addressed by various research. Despite these efforts, several concerns and challenges still need to be addressed, as listed beneath.

1) Most classification models using deep learning have been trained and validated with a unique dataset, limiting their use to other kinds of tampered images. In other words, they have not addressed the problem of generalization.

2) The proposed CNN models did not regard the imbalanced data issue. Hence, they might be biased toward a particular class.

3) Most methods did not report image prediction times, so it is not possible to know if they are suitable in applications that require real-time or massive data analysis.

4) The deep learning-based proffered approaches have used a single design strategy, either a custom model or a TL model, but, to the best of our knowledge, no work has compared the two approaches for the same dataset.

Based on the previously outlined issues, we intend to build two deep-learning CMFD models, one based on a CNN custom design while the other relied on TL. This will be detailed in the following subsections.

Chapter 3

DATA SET

When developing a CMFD system, having a high-quality dataset is crucial for achieving accurate and reliable results. A dataset serves as the foundation for training and testing the proposed model, allowing evaluate its performance and identify areas for improvement. In this section, we will introduce two various datasets used for our CMFD project. Each dataset possesses a diverse collection of digital images, carefully curated and labeled to include instances of copy-move forgery. We will describe the characteristics of the dataset, such as the size, format, and resolution of the images, as well as the labeling methodology that we used. Additionally, we will discuss any preprocessing steps that were taken to prepare the dataset for training, such as data augmentation or normalization.

3.1 DEFACTO dataset:

The DEFACTO Copy-Move Dataset is a large-scale dataset of images that have been tampered with using the copy-move forgery technique. The dataset contains over 19000 images (defacto-copymove, 2022). The dataset was automatically generated utilizing Microsoft Common Object in Context Database (MSCOCO) to construct semantically meaningful forgeries. Two binary masks are provided for every copy-move instance. The first mask, located in the probe mask subdirectory, identifies the location of the forgery. While the second mask, presented in the donor mask, highlights the source location within the image.

The copy-move forgery technique is a common way to tamper with images. It involves copying a portion of an image and then pasting it into another location in the same image. Figure 3.1 displays samples of the DEFACTO dataset.

Figure 3. 1 Examples of DEFACTO dataset

(a) Copy-move of the kite on y axis (b) Copy-move of the bird on x axis

The DEFACTO Copy-Move Dataset is a valuable resource for researchers who are working on developing methods to detect copy-move forgeries. The dataset can be used to train ML models that can automatically detect copy-move forgeries.

3.2 Flickr Image Dataset

The Flickr image dataset is a collection of over 28,000 images from the Flickr photo sharing website, along with their corresponding image tags and metadata (dataset, 2018).

The images in the dataset cover a wide range of topics, including nature, animals, people, and landscapes. The images are available in different formats, including JPEG and PNG, with varying resolutions.

The dataset also includes image tags, which are descriptive keywords or phrases associated with each image. These tags provide additional information about the content of the image and can be used for tasks such as image classification and retrieval.

In addition to the images and tags, the dataset includes metadata such as the date the image was taken, the camera used to capture the image, and the location where the image was taken (if available).

The Flickr Image Dataset can be used for a variety of computer vision tasks, such as image classification, object detection, and semantic segmentation. It can also be used for training and testing ML models. Flickr data is utilized to provide us with the original images.

Chapter 4

DEEP LEARNING

This section depicts a brief description of the deep learning concepts and applications. Furthermore, different state-of-the-art deep learning architectures are delivered.

4.1 Definition of Deep Learning

Deep learning is a subset of ML, thus, is essentially a neural network with three or more layers: the input layer, the hidden layer(s), and the output layer. These neural networks attempt to simulate the behavior of the human brain, albeit far from matching its ability allowing it to “learn” from substantial amounts of data. While a neural network with a single layer can still make approximate predictions, additional hidden layers can help to optimize and refine for accuracy (Team I. , 2020).

Deep learning drives many artificial intelligence (AI) applications and services that improve automation, performing analytical and physical tasks without human intervention. Deep learning technology lies behind everyday products and services (such as digital assistants, voice-enabled TV remotes, and credit card fraud detection) besides emerging technologies (such as self-driving cars).

4.1.1 Applications of Deep Learning

Deep learning is currently used in most common Image recognition tools, natural language processing (NLP), and speech recognition software. These tools are starting to appear in applications as diverse as self-driving cars and language translation services (Brush, 2022). Use cases today for deep learning include all types of big data analytics applications, especially those focused on NLP, language translation, medical diagnosis, stock market trading signals, network security, and image recognition.

Specific fields in which deep learning is currently being used include the following:

- Customer experience (CX). Deep learning models are already being used for chatbots. Also, as it continues to mature, deep learning is expected to be implemented in various businesses to improve CX and increase customer satisfaction.
- Text generation. Machines are being taught the grammar and style of a piece of text and are then use this model to create automatically completely new text matching the proper spelling, grammar and style of the original text.
- Aerospace and military. Deep learning is being used to detect objects from satellites that identify areas of interest, as well as safe or unsafe zones for troops.
- Industrial automation. Deep learning is improving worker safety in environments like factories and warehouses by providing services that automatically detect when a worker or object is getting too close to a machine.
- Adding color. Color can be added to black-and-white photos and videos using deep learning models. In the past, this was an extremely time-consuming, manual process.
- Medical research. Cancer researchers have started implementing deep learning into their practice as a way to automatically detect cancer cells.
- Computer vision. Deep learning has greatly enhanced computer vision, providing computers with extreme accuracy for object detection and image classification, restoration, and segmentation.

4.1.2 Deep Learning Method

Deep learning networks learn by discovering intricate structures in the data they experience. The networks can create multiple levels of abstraction to represent the data by building computational models composed of multiple processing layers.

For example, a deep learning model known as a CNN can be trained using large numbers (as in millions) of images, such as those containing cats. This type of neural network typically learns from the pixels contained in the images it acquires. It can classify groups of pixels that are representative of a cat's features, with groups of features such as claws, ears, and eyes indicating the presence of a cat in an image. Deep learning is fundamentally different from conventional ML. In this example, a domain expert would need to spend considerable time engineering a conventional ML system to detect the features that represent a cat. With deep learning, all that is needed is to supply the system with a very large number of cat images, and the system can autonomously learn the features that represent a cat.

For many tasks, such as computer vision, speech recognition (also known as NLP), machine translation, and robotics, the performance of deep learning systems far exceeds that of conventional ML systems. This is not to say that building deep learning systems is relatively easy compared to conventional ML systems. Although feature recognition is autonomous in deep learning, thousands of hyperparameters (knobs) need to be tuned for a deep learning model to become effective (Brush, 2022), (NG, 2020).

4.1.3 Deep learning Algorithms

1. Convolutional Neural Networks (CNNs)

A (CNN or ConvNet) is a network architecture for deep learning that learns directly from data. CNNs are particularly useful for finding patterns in images to recognize objects, classes, and categories. They can also be quite effective for classifying audio, time-series, and signal data.

2. Recurrent Neural Networks (RNNs)

RNN is a type of artificial neural network which uses sequential data or time series data. These deep learning algorithms are commonly used for ordinal or temporal problems, such as language translation, NLP, speech recognition, and image captioning; they are incorporated into popular applications such as Siri, voice search, and Google Translate. Like feedforward and CNNs, RNNs utilize training data to learn.

3. Long Short-term Memory Networks (LSTMs)

Long Short-Term Memory (LSTM) networks are a type of RNN capable of learning order dependence in sequence prediction problems. This is a behavior required in complex problem domains like machine translation, speech recognition, and more.

4. Generative Adversarial Networks (GANs)

A Generative Adversarial Networks (GAN) is a ML model in which two neural networks compete with each other to be more accurate in their predictions. GANs typically run unattended and use a zero-sum cooperative game framework for learning.

The two neural networks that make up a GAN are called the generator and the discriminator. The generator is a CNN, and the discriminator is a deconvolutional neural network. The purpose of the generator is to artificially fabricate results that could easily be mistaken for real data. The purpose of the discriminator is to identify which outputs it receives have been artificially created.

Basically, GANs create their own training data. As the feedback loop between the antagonistic networks continues, the generator will start to produce higher quality results and the discriminator will get better at flagging the data that has been artificially created.

5. Multilayer Perceptron (MLPs)

Multilayer Perceptron (MLPs) are an excellent place to start learning about deep learning technology. MLPs belong to the class of feedforward neural networks with multiple layers of perceptron that have activation functions. MLPs consist of an input layer and an output layer that are fully connected. They have the same number of input and output layers but may have multiple hidden layers and can be used to build speech-recognition, image-recognition, and machine-translation software [6], (Biswal, 2022).

4.2 CNN

4.2.1 Definition of CNN

CNN is a special type of multilayer neural network or deep learning architecture inspired by the visual system of living beings. The foundation of CNN started with the discovery of Hubel and Wiesel in 1959 [10]. CNN, also called ConvNet, is a type of Artificial Neural Network.

A deep CNN model consists of a finite set of processing layers that can learn various features of input data (e.g., image) with multiple levels of abstraction.

The initiatory layers learn and extract the high-level features (with lower abstraction), and the deeper layers learn and extract the low-level features (with higher abstraction). Figure 4.1 displays an example of a CNN architecture.

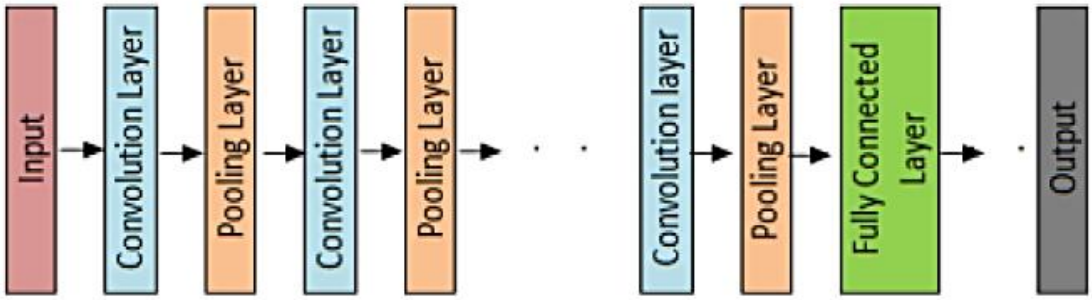

Figure 4. 1 Example of CNN model

4.2.2 Convolutional Layer

Convolutional layer is the most important component of any CNN architecture. It contains a set of convolutional kernels (also called filters), which gets convolved with the input image (N-dimensional metrics) to generate an output feature map.

4.2.2.1 What is a kernel?

A kernel can be described as a grid of discrete values or numbers, where each value is known as the weight of this kernel. An example of a kernel window is presented in Figure 4.2.

0	1
-1	2

Figure 4. 2 Example of kernel window

At the beginning of the training process of a CNN model, all the kernel's weights are assigned random numbers (different approaches are also available there for initializing the weights). Then, with each training epoch, the weights are tuned, and the kernel learned to extract meaningful features.

4.2.2.2 What is Convolution Operation?

In convolution operation, a 2 × 2 kernel slides over the complete 4×4 grayscale image horizontally as well as vertically, and along the way the dot product between the kernel and the input image is computed by multiplying the corresponding values of them and summing up all values to generate one scaler value in the output feature map. This process continues until the kernel can no longer slide further.

The entire process is shown in Figure 4.3. The final output feature map is presented in Figure 4.4.

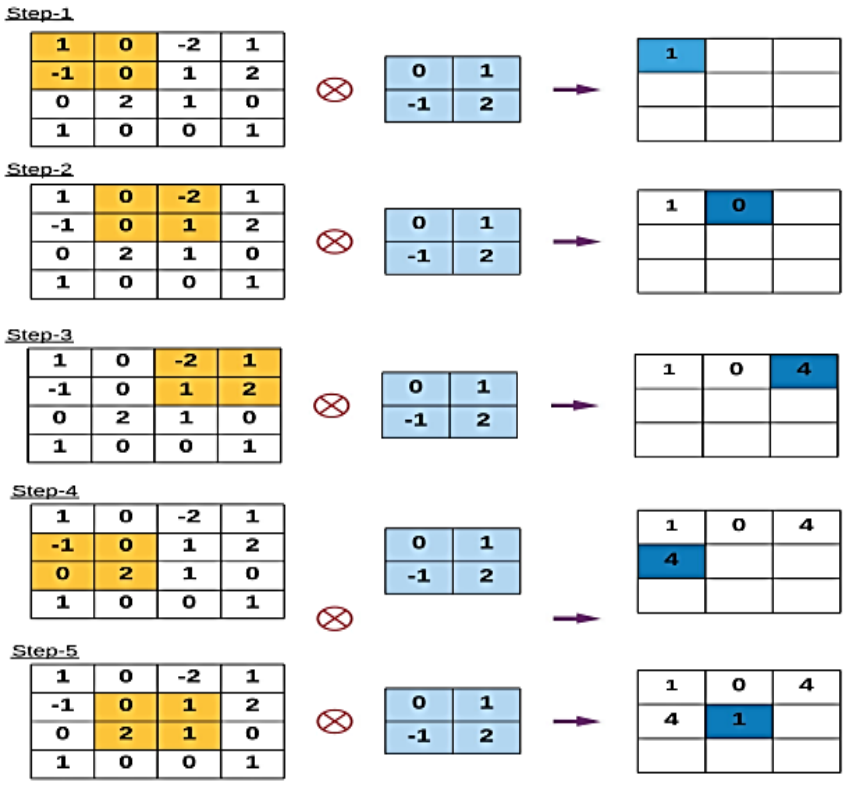

Figure 4. 3 Convolution Operation

4.2.2.3 Padding

Sometimes filter does not perfectly fit the input image. There are two options:

- Pad the picture with zeros (zero-padding) so that it fits.
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.

The computations performed at each step, where the 3 × 3 kernel (shown in light blue color) is multiplied with the same sized region (shown in yellow color) within the 6 × 6 input image (where we applied zero-padding to the original input image of 4 × 4 dimension and it becomes of 6 × 6 dimensional), and values are summed up to obtain a corresponding entry (shown in deep green) in the output feature map at each convolution step as illustrated in Figure 4.5.

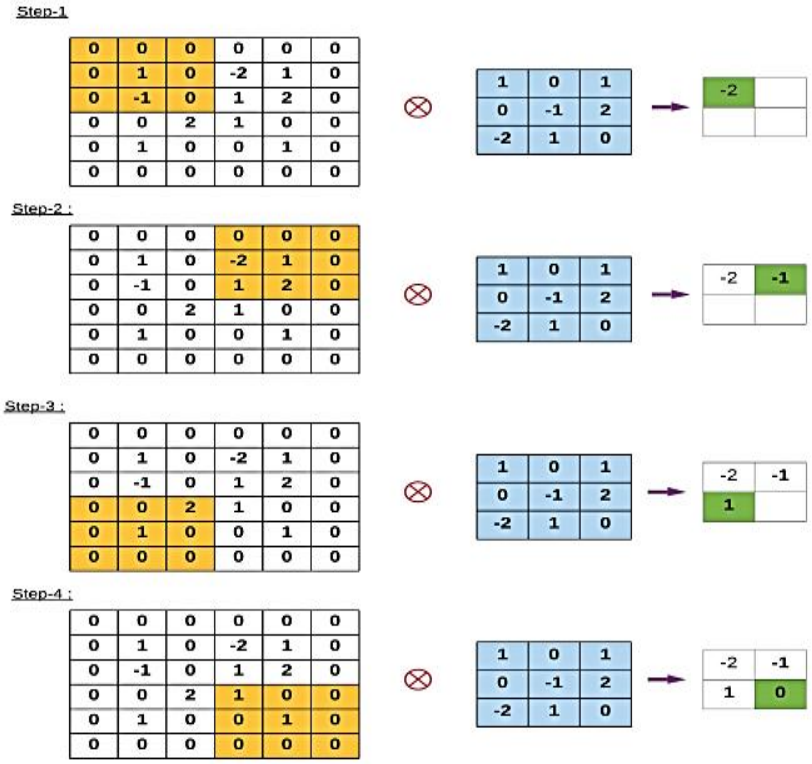

Figure 4. 4 Convolution Operation after padding

4.2.3 Pooling Layer

The pooling layers are used to sub-sample the feature maps (produced after convolution operations). It takes the larger size feature maps and shrinks them to lower sized feature maps. While shrinking the feature maps, it always preserves the most dominant features (or information) in each pool steps. The pooling operation is performed by specifying the pooled region size and the operation stride, like convolution operation. There are different types of pooling techniques that are used in different pooling layers, such as max pooling, min pooling, average pooling, gated pooling, tree pooling, etc. Max Pooling is the most popular and widely used pooling technique. Figure 4.6 demonstrates the Max-Pooling process.

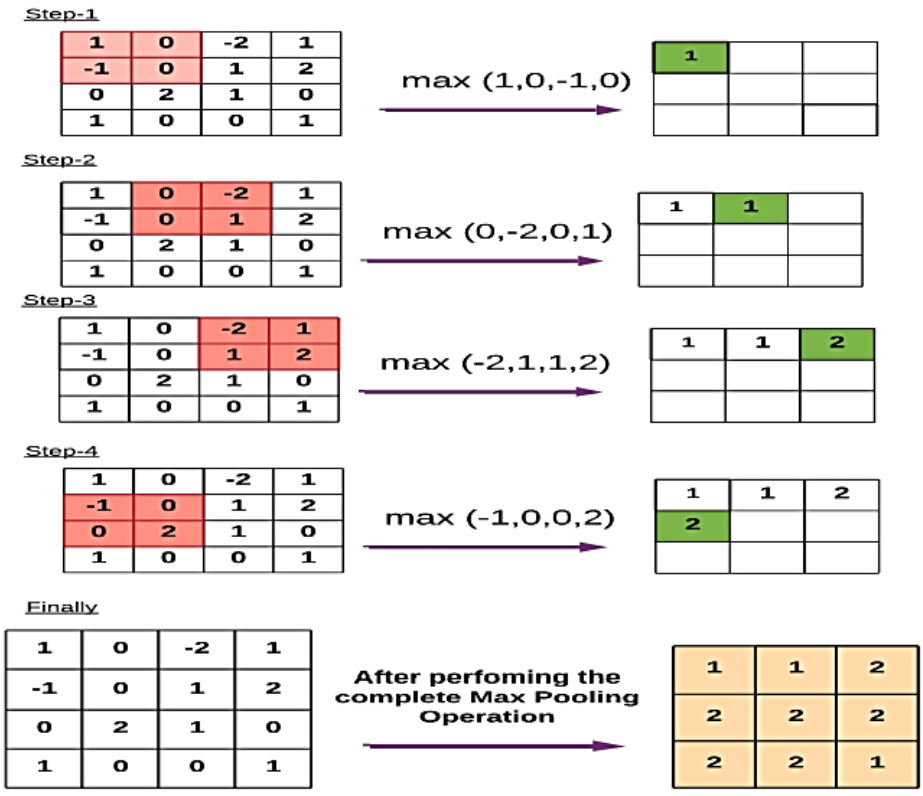

Figure 4. 5 Max-Pooling Operation

4.2.4 Activation Functions

ReLU's purpose is to introduce non-linearity in our ConvNet. Since the real-world data would want our ConvNet to learn, it would be non-negative linear values.

ReLU stands for Rectified Linear Unit for a non-linear operation [10]. The output is $f(x) = \max(0, x)$.

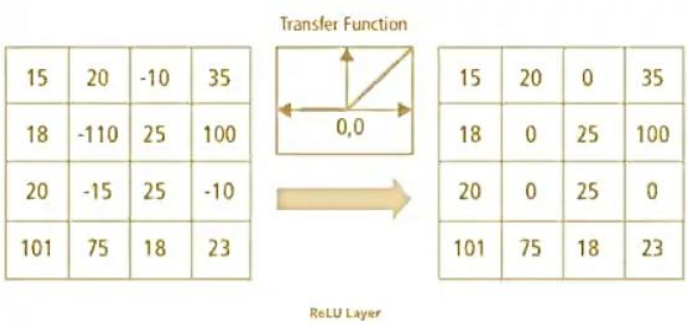

Figure 4. 6 ReLU Activation function

There are other nonlinear functions, such as tanh or sigmoid, that can be used instead of ReLU. Most of the data scientists use ReLU, since performance-wise, ReLU surpasses the other two.

4.2.5 Fully Connected Layer

Usually, the last part (or layers) of every CNN architecture (used for classification) is consist of fully connected layers (FC Layers), where each neuron inside a layer relates to each neuron from its previous layer. The last layer of the FC layers is used as the output layer (classifier) of the CNN architecture (Team S. , 2018). The FC layers take input from the final convolutional or pooling layer, which is in the form of a set of metrics (feature maps), and those metrics are flattened to create a vector. Then, this vector is fed into the FC layer to generate the final output of CNN, as shown in Figure 4.7.

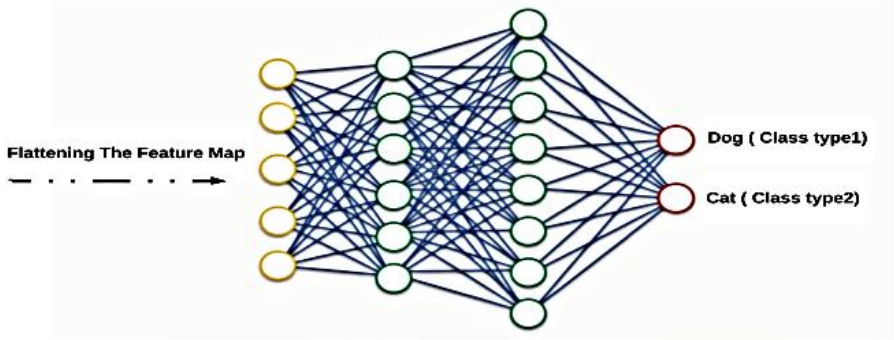

Figure 4. 7 Fully connected layer

4.2.5.1 Loss Functions

In this output layer, we calculate the prediction error generated by the CNN model over the training samples using some Loss Function. This prediction error tells the network how off their prediction from the actual output, and then this error will be optimized during the learning process of the CNN model. The loss function uses two parameters to calculate the error, the first parameter is the estimated output of the CNN model (also called the prediction), and the second one is the actual output (also known as the label). There are multiple types of loss functions that may be used in various problems. Some of the most used loss functions are described briefly in the following subsections.

- Soft-Max Loss Function

  Cross-entropy loss also called the log loss function, is widely used to measure the performance of the CNN model, whose output is the probability $p \in \{0, 1\}$. It is widely used as an alternative to the squared error loss function in multi-class classification problems. It uses soft-max activations in the output layer to generate the output within a probability distribution, i.e., $p, y \in R^N$, where p is the probability for each output category and y denotes the desired output. The probability of each output class can be obtained as shown in Figure 4.8.

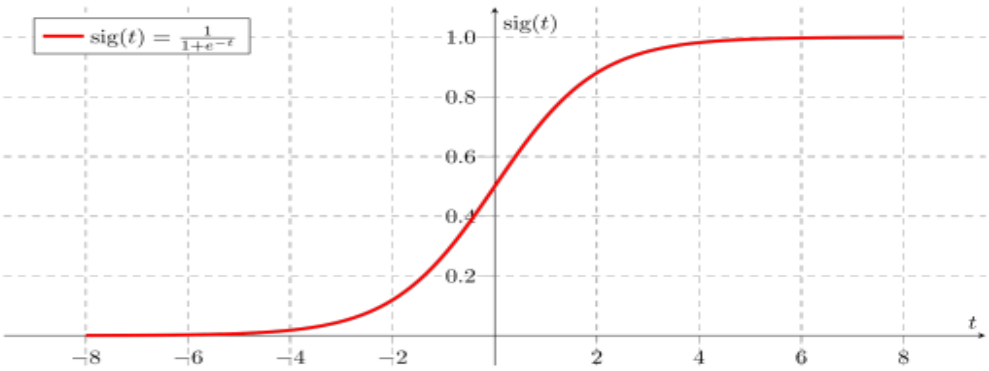

Figure 4. 8 Soft-Max

4.2.6 Training of CNN

A training dataset comprising a set of images and labels (classes, bounding boxes, masks) is used to train a CNN model. The algorithm used to train a CNN is called backpropagation, that uses the output value of the last layer to measure an error value. This error value is used to update the weights of each neuron in that layer. The new weights are used to measure an error value and update the weights of the previous (Team S. , 2018). The algorithm repeats the process until it reaches the first layer, see Figure 4.9.

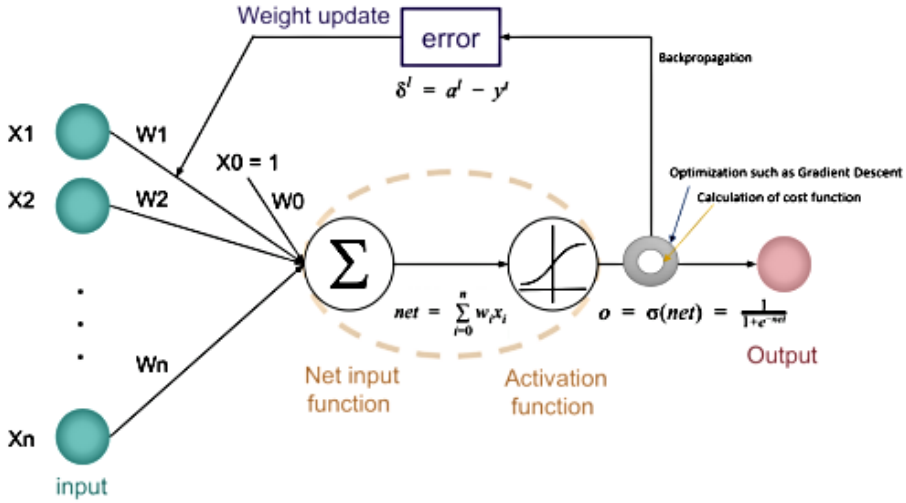

Figure 4. 9 Back-Propagation process

4.2.7 Applications of CNN

4.2.7.1 Image classification

CNNs are often used for image classification. CNN can be also used in agriculture. The networks receive images from satellites like LSAT and can use this information to classify lands based on their level of cultivation. Consequently, this data can be used for making predictions about the fertility level of the grounds or developing a strategy for the optimal use of farmland. Hand-written digit recognition is also one of the earliest uses of CNN for computer vision.

4.2.7.2 Object detection

Self-driving cars, AI-powered surveillance systems, and smart homes often use CNN to be able to identify and mark objects. In real-time, CNN can identify and classify objects depicted in photos, providing accurate labeling. Through this, an automated vehicle finds its way around other cars and detects pedestrians, and smart homes can realize the face of their owners among all others.

4.2.7.3 Audio visual matching

YouTube, Netflix, and other video streaming services use audio visual matching to improve their platforms. Sometimes the user's requests can be specific thoroughly, for example, Movies about zombies in space, and the search engine must fulfill even such exotic requests.

4.2.7.4 Object reconstruction

You can use CNN for 3D modelling of real objects in the digital space. Today there are CNN models that create 3D face models based on just one image. Similar technologies can be used for creating digital twins, which are useful in architecture, biotech, and manufacturing.

4.2.7.5 Speech recognition

Even though CNNs are often used to work with images, it is not the only possible use for them. ConvNet can help with speech recognition and NLP. For example, Facebook's speech recognition technology is based on CNNs.

4.3 Transfer Learning (TL)

4.3.1 Definition of TL

The reuse of a pre-trained model on a new problem is known as TL in ML. A machine uses the knowledge learned from a prior assignment to increase prediction about a new task in TL (Huilgol, 2020).

The knowledge of an already trained ML model is transferred to a different but closely linked problem throughout TL. For example, if you trained a simple classifier to predict whether an image contains a backpack, you could use the model's training knowledge to identify other objects such as sunglasses. Because of the massive amount of CPU power required, TL is typically applied in computer vision and NLP tasks like sentiment analysis.

4.3.2 TL Method

In computer vision, neural networks typically aim to detect edges in the first layer, forms in the middle layer, and task-specific features in the latter layers. The early and central layers are employed in TL, and the latter layers are only retrained. It makes use of the labelled data from the task it was trained on.

4.3.3 Advantages of TL

TL offers several advantages, the most important of which are reduced training time, improved neural network performance (in most circumstances), and the absence of a large amount of data. To train a neural model from scratch, a lot of data is typically needed, but access to that data isn't always possible – this is when TL comes in handy. Because the model has already been pre-trained, a good ML model can be generated with little training data using TL. This is especially useful in NLP, where huge,

labelled datasets require a lot of expert knowledge. Additionally, training time is decreased because building a deep neural network from the start of a complex task can take days or even weeks. Figure 3.10 illustrate the TL approach.

4.3.4 Use Cases of TL

If you used TensorFlow to train the original model, you might simply restore it and retrain some layers for your job. TL, on the other hand, only works if the features learnt in the first task are general, meaning they can be applied to another activity. Furthermore, the model's input must be the same size as it was when it was first trained. If you don't have it, add a step to resize your input to the required size.

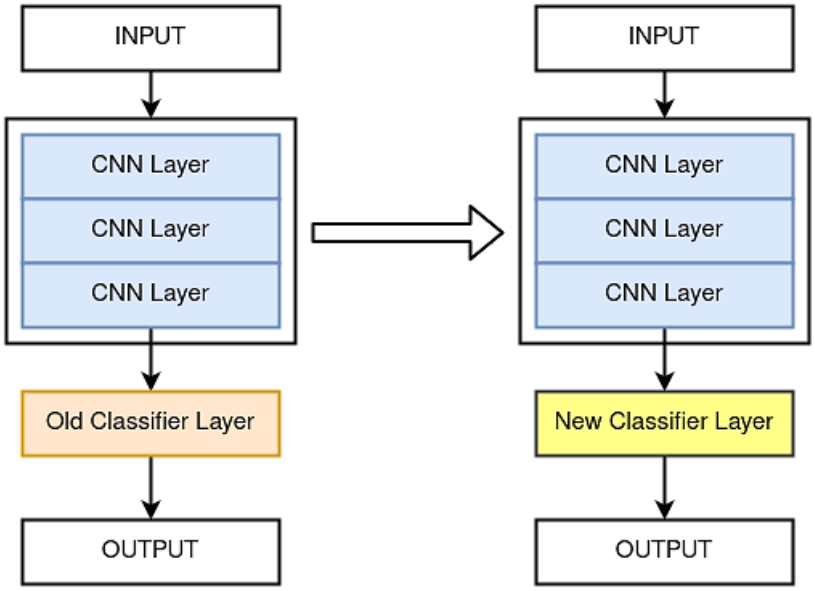

Figure 4. 10 Transfer Learning method

4.3.5 TL Models Using CNN

4.3.5.1 VGG16

The VGG16 is one of the most popular pre-trained models for image classification, introduced in the famous ILSVRC 2014 Conference (Karen Simonyan, Cornell University, n.d.). It was and remains the model to beat even today. It was developed at the Visual Graphics Group at the University of Oxford. VGG16 beat the then standard of AlexNet and was fast adopted by researchers and the industry for their image classification tasks. The architecture of VGG16 is shown in Figure 4.11.

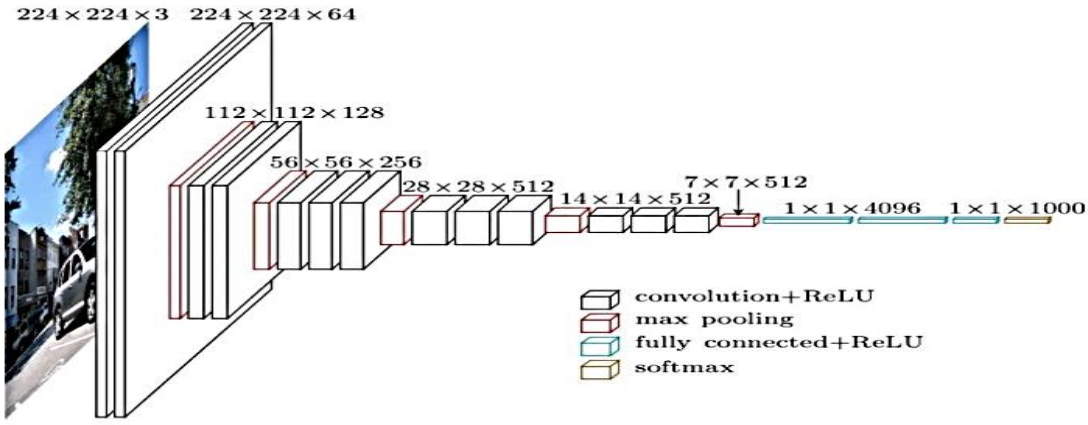

Figure 4. 11 VGG16 architecture

Additionally, there are variations of the VGG16 model, which are basically improvements to it, like VGG19 (19 layers) (Huilgol, 2020).

4.3.5.2 Inception

Although VGG16 managed to secure the second rank in the ILSVRC of 2014, it was beaten by Google's GoogLeNet (now known as Inception) model, which took the top spot. As can be noticed in Figure 4.12, the Inception module merely performs convolutions with different filter sizes on the input, fulfils Max Pooling, and concatenates the result for the next Inception module. The introduction of the 1 * 1 convolution operation reduces the parameters drastically. The Inceptionv2 model was a significant improvement on the Inceptionv1 model, which increased the accuracy and further made the model less complex. Furthermore, in the same research as Inceptionv2 (Karen Simonyan, Cornell University, 2019), the authors introduced the Inceptionv3 model with a few more improvements on v2.

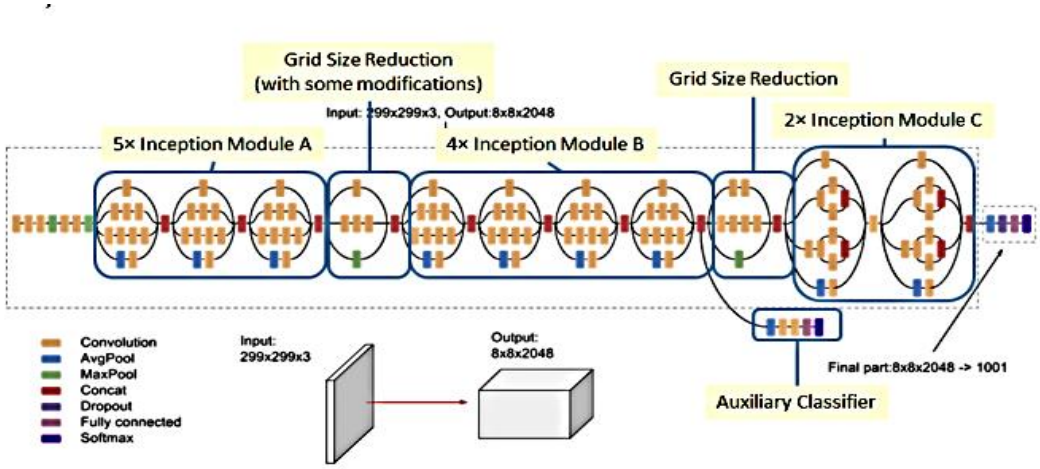

Figure 4. 12 Inception architecture

As you can see that the number of layers is 42, compared to VGG16's paltry 16 layers. Also, Inceptionv3 reduced the error rate to only 4.2%.

4.3.5.3 ResNet50

Just like Inceptionv3, ResNet50 is not the first model coming from the ResNet family. The original model was called the Residual net or ResNet and was another milestone in the domain back in 2015.

The primary motivation behind this model was to avoid poor accuracy as the model become deeper. Additionally, the ResNet model aimed to tackle the Vanishing Gradient issue also. Here is the architecture of the earliest variant: ResNet34 (ResNet50 also follows a similar technique with just more layers). After every 2 convolutions, we are bypassing/skipping the layer in between. This is the main concept behind ResNet models. These skipped connections are called ‘identity shortcut connections” and use what is called residual blocks are illustrated in Figure 4.13. The ResNet model has many variants, of which the latest is ResNet152. Figure 4.14 shows architecture of ResNet. The output of the weight layers receives the residue input 'x' before undergoing activation, which is executed using Relu activation. ResNet50 is composed of a 50-layer Residual network. Furthermore, it has other variations, including ResNet101 and ResNet152.

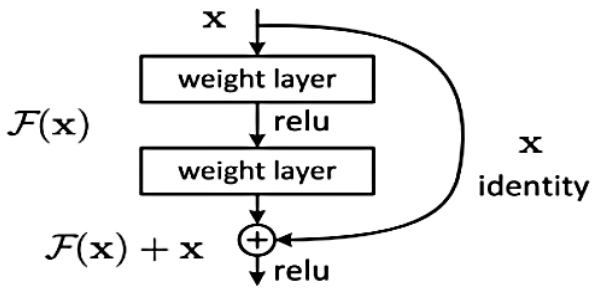

Figure 4. 13 Residual blocks in ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 4. 14 Architecture of the ResNet family

4.3.5.4 MobileNetV2

MobileNetV2 is a lightweight neural network architecture designed for efficient inference on mobile and embedded devices with limited computational resources. It was introduced as an improvement over the original MobileNet architecture. MobileNetV2 uses a new block structure called inverted residuals. The MobileNetV2 model architecture is depicted in Figure 4.15. It includes 2D convolutional, pooling, and bottleneck layers. The bottleneck layer comprised sub-layers such as expansion, normalization, activation, addition, 3x3 depth-wise convolution, and projection. The objective of the expansion operation is to increase the number of channels in the input data, determined by the model hyperparameters. Conversely, the projection operations lessen the input data's channel count and reduce the data space. The normalization, pooling, convolution, and activation layers employ the corresponding operations necessary for the training and inference processes.

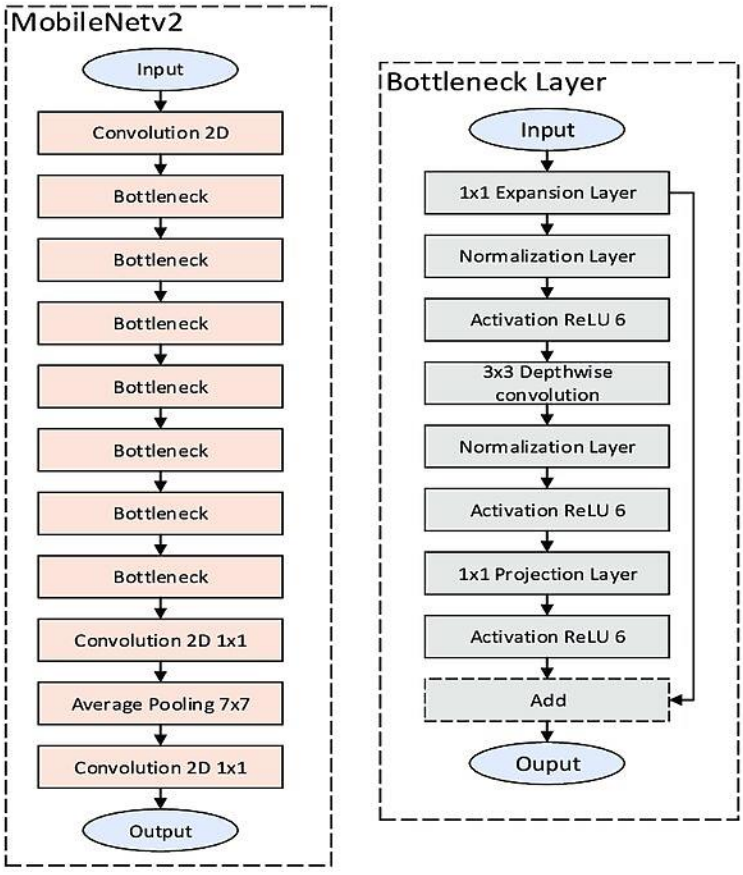

Figure 4. 15 MobileNetV2 Architecture

4.3.5.5 EfficientNet

In EfficientNet, a new Scaling method called Compound Scaling is proposed. The long and short of it is this: The earlier models like ResNet follow the conventional approach of scaling the dimensions arbitrarily and by adding up more and more layers. The scaling coefficients can be in fact decided by the user. Though this scaling technique can be used for any CNN-based model, the authors [12] started off with their own baseline model called EfficientNetB0, shown in Figure 4.16.

EfficientNet is a popular CNN architecture known for its efficiency in computer vision tasks. It is commonly used for transfer learning, where a pre-trained model is repurposed for a new task. The process involves freezing the pre-trained weights and modifying the final layers for task-specific classification. Optionally, earlier layers can be fine-tuned to adapt to the new dataset. The model is then trained on the target dataset, evaluated for performance, and used for making predictions. EfficientNet's TL approach saves time, improves performance, and reduces computational resources compared to training from scratch.

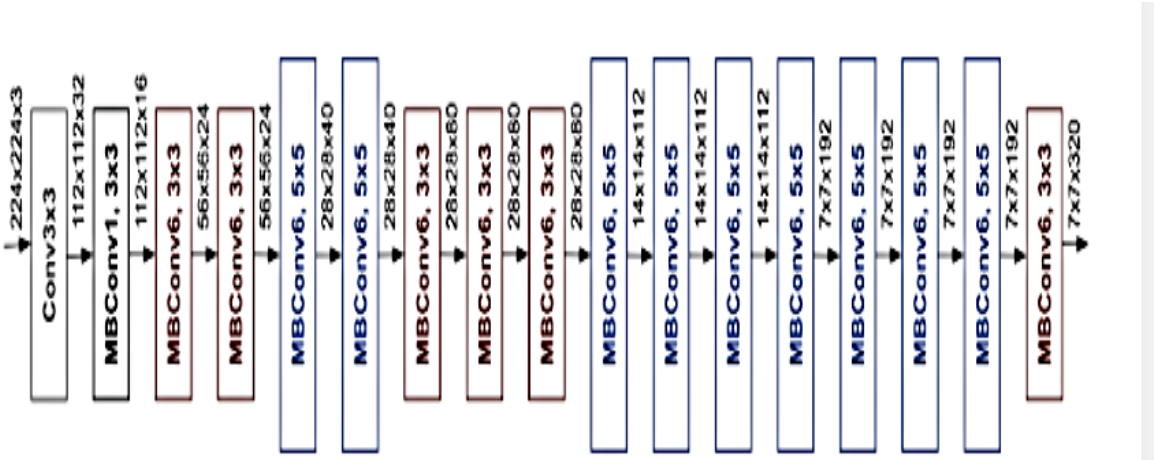

Figure 4. 16 Architecture of EfficientNet

MBConv stands for mobile inverted bottleneck Convolution (similar to MobileNetv2). They also propose the Compound Scaling formula with the following scaling coefficients:

• Depth = 1.20 • Width = 1.10 • Resolution = 1.15

The compound scaling formula is used to again build a family of EfficientNets – EfficientNetB0 to EfficientNetB7 (Huilgol, 2020).

4.3.5.6 YOLO-V7 Detection

The YOLO-V7 (You Only Look Once) object detection technique is a deep learning-based approach to detecting object in images. It is a single-stage object detection system; it performs object detection and classification in a single pass through the network.

The YOLO-V7 detection technique works by dividing the input image into a grid of cells and predicting the bounding boxes and class probabilities for each cell. The network predicts the bounding boxes relative to the cell, which helps to reduce the number of predictions required.

YOLO-V7 also uses anchor boxes, which are pre-defined shapes that help the network predict the size and shape of the objects in the image. The network predicts the offsets for each anchor box, which helps to improve the accuracy of the predicted bounding boxes.

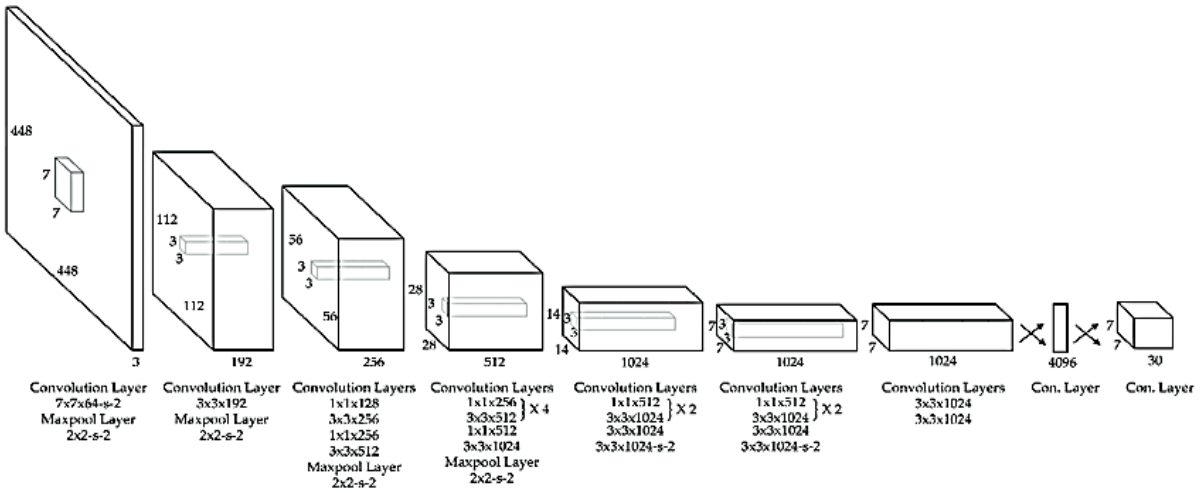

Figure 4. 17 YOLO-V7 detection architecture

The YOLO-V7 framework has three main components; Backbone, Head, and Neck. The Backbone mainly extracts essential features of an image and feeds them to the Head through Neck. The Neck collects feature maps extracted by the Backbone and creates feature pyramids. Finally, the head consists of output layers that have final detections.

The YOLO-V7 model architecture is depicted in Figure 4.17. Since the first release of YOLO-V7 in 2015, it has evolved a lot with different versions, see Figure 4.18.

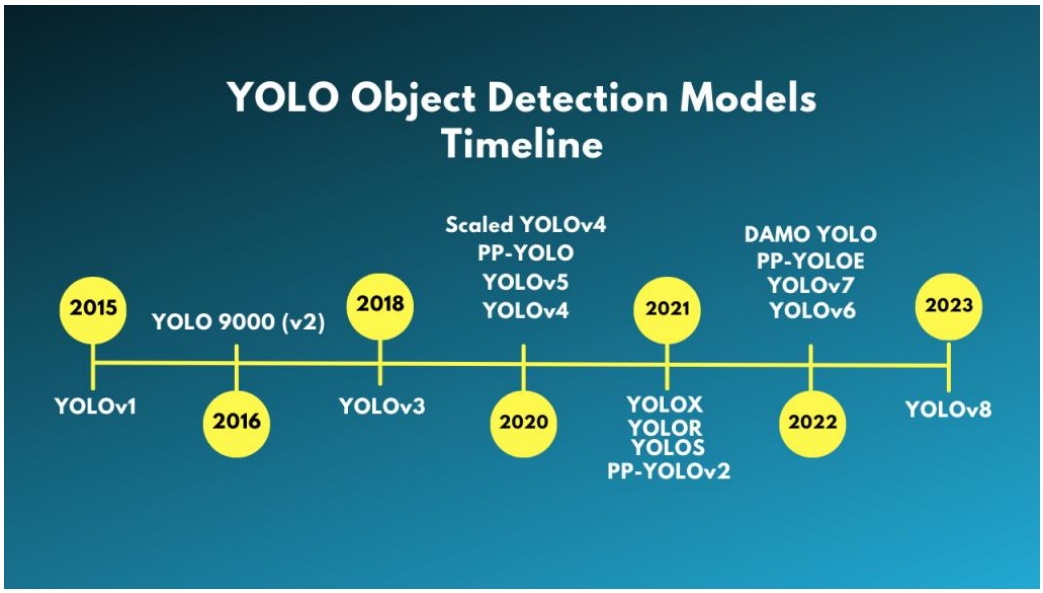

Figure 4. 18 The timeline of YOLO-V7 object detection models

The methodology we adopted in this work will be explained in the subsequent section.

Chapter 5

METHODOLOGY

This chapter outlines the proposed methodology to address the problem of CMFD, which will be conducted in three phases, gradually improving as we move from one phase to another. Initially, in the first phase, the development of deep learning classifiers established on exploiting the content-based features of the entire image takes place. The second phase relied on leveraging features from image patches. The third and final phase involves the implementation of an object detection technique. In the first and second phases, multiple models applying TL, including VGG16, MobileNetV2, and ResNet50, are tested and evaluated using the selected dataset. Additionally, a customized structural CNN architecture is created and assessed. While in the third phase, YOLO-V7 is adopted. The team considered multiple aspects of the project, including selecting an appropriate dataset, image preprocessing, and deep learning model application.

5.1 Phase 1

In this phase, we developed four CNN classifiers to detect copy-move forgery. These classifiers incorporate customized architectural CNN and TL-based models, including VGG16, MobileNetV2, and ResNet50.

5.1.1 Exploratory Data Analysis:

1. The DEFACTO and Flickr datasets are combined, resulting in a dataset comprising 31,783 genuine images and 18,194 fake images.
2. The fake image dimensions differ in height and width, with a minimum and maximum height range of 135 and 640 pixels, respectively. Whereas the minimum and maximum widths of 125 and 640 pixels, respectively.

3. The original images display varying heights and widths, with a minimum and maximum height span of 112 and 500 pixels, respectively. Conversely, the minimum and maximum width values are 164 and 500 pixels, respectively.
4. Both the original and forged images are RGB images.

5.1.2 Image Preprocessing

1. Image Scaling: Image scaling is done to reduce the computational effort needed for image processing. Every image will be resized to 128x128 pixels for further processing.
2. The images, both authentic and fake, are split into 80% for training, 10% for testing, and 10% for validation purposes.
3. The final dataset incorporates separate lists of images, training, validation, and test sets. Each image within these sets is associated with a corresponding label indicating the image is real (0) or fake (1).
4. The image data generator function is applied to augment the number of images during the training phase, thereby enhancing the performance of the training stage.

5.1.3 Image Data Generator

The Image Data Generator is an object in Keras that can be used for real-time data augmentation and preprocessing during model training. It takes as input the original images and their corresponding labels and generates augmented versions of the images on-the-fly during training to increase the size of the training set. Some common augmentations provided by the Image Data Generator include rotation, shifting, shearing, zooming, and flipping. The Image Data Generator also includes options for rescaling and normalization of the images. The batch size for training is defined, and a generator is created using the `flow` function of the Image Data Generator. The generator yields batches of augmented images and their corresponding labels during model training, allowing the model to see a larger and more diverse set of images without having to store all of the augmented images in memory.

5.1.4 Custom Model Architecture

This architecture, as displayed in Figure 5.1, is a CNN consisting of 4 Conv2D layers with increasing numbers of filters, followed by MaxPooling2D layers to reduce the spatial dimensions of the feature maps. The output is then flattened and fed into two Dense layers with 512 and 1 units, respectively. The first Dense layer uses the ReLU activation function, while the second Dense layer uses a linear activation function. Two dropout layers are applied to prevent overfitting during training. This architecture is designed for image classification tasks and has 2,601,153 trainable parameters.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 61, 61, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 4608)              0
_________________________________________________________________
dense (Dense)                (None, 512)               2359808
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 2,601,153
Trainable params: 2,601,153
Non-trainable params: 0
```

Figure 5. 1 Architecture of Custom Model

5.1.5 VGG16

In this context, the VGG16 model from Keras will be employed for TL to accurately identify whether input images are genuine or forged. The Keras VGG16 architecture is illustrated in Figure 5.2.

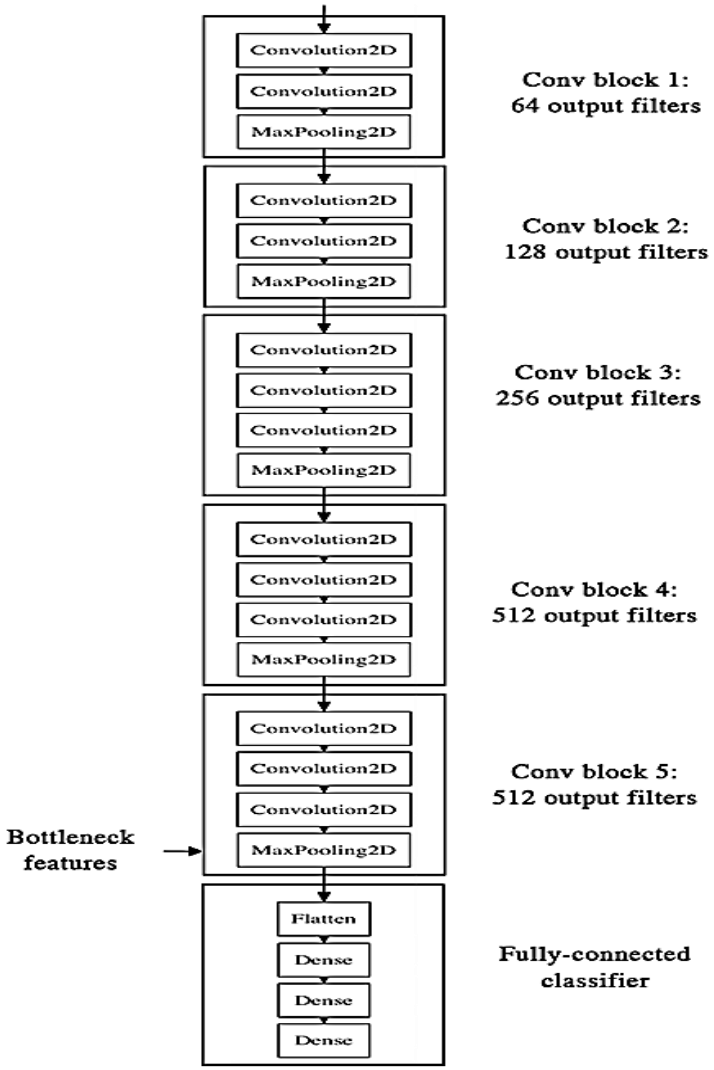

Figure 5. 2 Structure of VGG16 Model

To construct the model, we will begin by creating a sequential model and incorporating VGG16 as the initial layer. We will specify "include_top" as False to eliminate the fully connected classifier from the end of the model. As a result, we'll need to add a Flatten layer and multiple Dense layers to classify the input images. We included three dense layers, with the first two using the ReLU activation function and the third using a linear activation function.

5.1.6 ResNet 50

In this model architecture, we adopted a CNN model by applying the TL method ResNet50, which has been fine-tuned with the addition of several layers in the top layer of ResNet50, which does not use the default top layer. The ResNet50 architecture composed of 48 Convolution layers along with 1 MaxPool and 1 Average Pool layer., as shown in Figure 5.3.

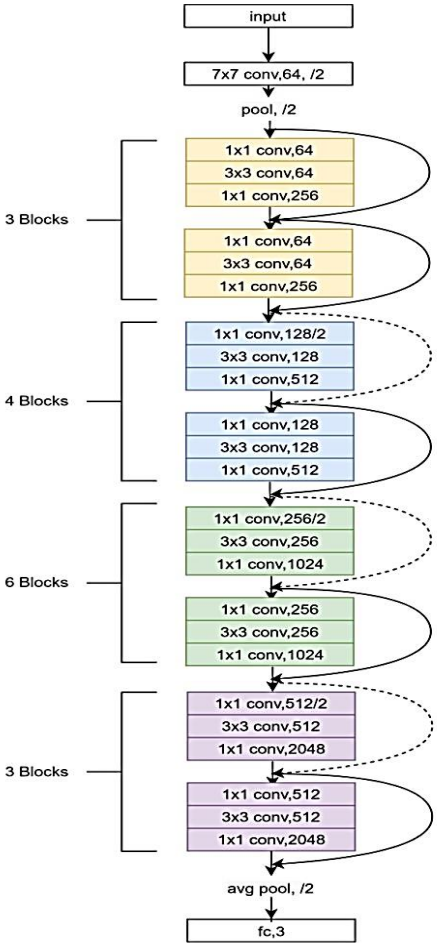

Figure 5. 3 Structure of ResNet50 Model

We will kick-start the development of our model by composing a sequential model with ResNet50 as the initial layer. Since we do not want the fully connected classifier to be included at the end of the model, we will set "include_top" to False. Consequently, we will have to attach a Flatten layer and two Dense layers to categorize the input images. The first Dense layer will employ the ReLU activation function, whereas the second layer will use a linear activation function. Furthermore, we freeze the first stage of the backbone (47 convolution layers), whereas the last 13 layers were pre-trained.

5.1.7 MobileNetV2

The process implemented in ResNet50 will be replicated in MobileNetV2. With the exception that, as MobileNetV2 consists of 154 layers, we will only freeze the first layers and fine-tune the last 34 layers.

5.2 Phase 2

During this phase, we utilized a sample-based technique to divide the input images into multiple samples. These samples are then utilized for training three CNN architectures schemed to detect fraudulent images. As part of this phase, we'll combine the sample-based technique with the CNN architectures discussed in the previous section (Customized CNN, VGG16, MobileNetV2, and ResNet50). The process is composed of three key stages: feature learning which involves mask extraction and patch sampling, feature extraction, and classification. The initial step involves training the CNN model by utilizing labeled patch samples from training images. The cloned patches are utilized as the boundaries for positive patch samples in forged images, while negative patch samples' boundaries are randomly selected from original images. During the second phase, the entire image is scanned with a patch-sized sliding window, and subsequently, patch-based features are extracted using the recommended deep learning model.

5.2.1 Feature Learning

In this stage, a sampling fragments the input images into patches for easier processing and analysis. For every fake image, we have a corresponding mask. Each mask is used to sample the corresponding fake image along the spliced region's boundary, ensuring that at least a 20% contribution from both the forged and unforged patches. The authentic images, on the other hand, are randomly sampled. The samples collected exhibit unique boundaries found only in fake images, which the CNNs we develop will learn.

5.2.1.1 Data Preprocessing steps

The sampling process is a crucial step in CMFD. To ensure our analysis is performed on a diverse and representative set of images, we will select a smaller number of images and make sure that all dimensions are one channel. Masks will then be used to sample image patches for the creation of train and cross-validation data, with the label set to "fake". This approach ensures that the CNN is trained to learn the boundaries created by the forgery operations. This is due to the change in statistics of the underlying image at these boundaries, as shown in Figure 5.4.

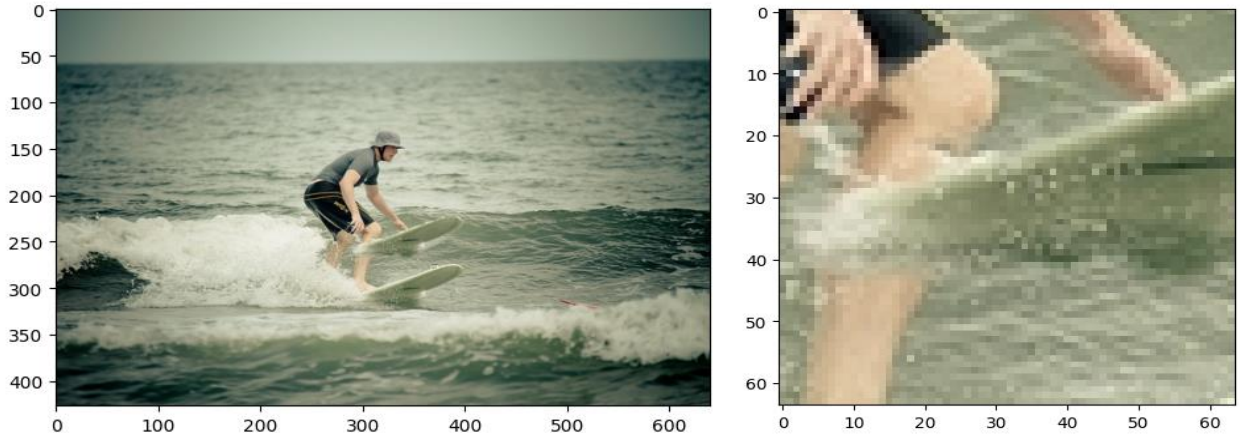

Figure 5. 4 The variation at the boundary of the copy-move forgery patch

To capture information from every image, we use a patch size of 64 x 64 and ensure a minimum of 20% contribution from the forged and authentic areas. This approach ensures that Before sampling, we convert grayscale masks to binary (0 or 255) and determine the number of pixels in the mask representing the forgery region (255). We will not resize the images before sampling to ensure that we capture information from the whole image.

To assess our model precisely, we apply the same sampling procedures used for the training set to the test set. This approach ensures that our model is evaluated on a representative and diverse set of images, providing an exhaustive assessment of its performance.

5.2.2 Feature Extraction and Classification

The fake and real patches are fed to the customized CNN, VGG16, MobileNetV2, and ResNet50 models. Using these deep learning models, the feature extraction process is conducted on patch-based characteristics of the input images. Subsequently, the extracted features are utilized to train the implemented CNN models obtaining an efficient deep-learning model to discriminate between genuine and forged images. In the testing phase, the input image is divided into non-overlapping patches. These features are then supplied to the pre-trained models capable of determining the class of the input image.

5.3 Phase 3

As YOLO-V7 surpasses all previous object detectors in terms of both speed and accuracy (Chien-Yao Wang, 2022) we adopted it in our submitted work.

YOLO-V7 is utilized to detect objects and examine their characteristics in order to identify similarities that indicate the presence of copy-move entities. Subsequently, the counterfeit images are recognized, and the altered region within the image is located.

The YOLO-V7 algorithm utilizes image classification to first identify whether the input image contains an object or not and subsequently locate the object's coordinates within the image. The input image will be in the format of a grid with dimensions 640*640 During the prediction process, every bounding box will consist of five elements - (x, y, w, h, prediction), where (x, y) represent the location of the center of the bounding box, and (w, h) correspond to its width and height.

In order to accomplish this task, some image preprocessing steps are performed. This will be our subject in the subsequent section.

5.3.1 Data Preprocessing Steps

1. Data selection

The dataset is chosen based on the specific needs of the YOLO-V7 algorithm, which can detect object regions without requiring both original and fake data. As a result, only fake images are required for this application. This approach simplifies the data selection process and allows for a more focused dataset, making it easier to train the YOLO-V7 model to identify accurately forged regions within the images.

To train the YOLO-V7 model with image data, annotations are required. In this case, a Python script is used to create annotations using the OpenCV library. These annotations provide information on the location and size of the forged regions within each image. The annotations are then used to train the YOLO-V7 model to detect accurately and classify forged regions within the input images. This step is critical for ensuring that the YOLO-V7 model can identify precisely and locate forged regions within the input images.

2. Check label

Once the annotations have been generated for the training data, the labels are checked to ensure their correctness and accuracy. This is a crucial step in the preprocessing of the YOLO-V7 model, as inaccurate or erroneous labels can lead to the model being trained on low-quality data, which can result in deficient performance. Any labels that are found to contain errors or inaccuracies are removed, and the remaining high-quality labels are used to train the YOLO-V7 model. By ensuring that the labels are accurate and free of errors, the YOLO-V7 model is better able to learn to accurately detect and classify forged regions within the input images.

3. Split data

To ensure that the YOLO-V7 model is able to generalize well to new data, the dataset is divided into separate training, validation, and testing sets. This is a frequent practice in ML and deep learning applications, as it allows the model to be trained on a subset of the data and tested on another subset of data to evaluate its performance. Although the specific method of splitting the dataset is not mentioned, it is typically done randomly, with a certain percentage of the data allocated for training, validation, and testing.

This process helps to ensure that the YOLO-V7 model can accurately detect and classify forged regions in new, unseen images.

These preprocessing steps are designed to ensure that the YOLO-V7 model is trained on high-quality data and can accurately detect object regions within the input images. By selecting the appropriate dataset, generating accurate annotations, checking the labels for errors, and splitting the dataset into training, validation, and testing sets, the YOLO-V7 model can learn to generalize well to new data and achieve high accuracy.

Chapter 6

EXPERIMENTAL RESULTS

In this section, we present the experimental outcomes obtained from our deep learning-based CMFD approaches. Specifically, we describe the employed performance metrics and the detailed analysis of the achieved results. Finally, we provide some insights and observations obtained from the experimental results that can guide future research in the domain of deep learning-based CMFD.

6.1 Evaluation Metrics

When evaluating a ML model, it's essential to consider multiple evaluation metrics, as each metric provides a different perspective on the model's performance. The choice of evaluation metrics depends on the specific problem, the nature of the data, and the objectives of the task at hand. Here are the ones we used at each phase.

6.1.1 Classification

6.1.1.1 Accuracy curve

The accuracy curve, also known as the training accuracy curve, depicts the performance of a ML model over the course of training iterations or epochs. It shows how the accuracy of the model changes as the training progresses.

6.1.1.2 Loss curve

The loss curve, also known as the training loss curve or the cost curve, illustrates the model's training loss over time. The training loss represents the discrepancy between the predicted outputs of the model and the true labels in the training data.

6.1.1.3 Confusion matrix

A confusion matrix is a table that summarizes the performance of a classification model by comparing the predicted class labels with the true ones

from a test dataset. It provides a detailed breakdown of the model's predictions, allowing for a more comprehensive evaluation.

6.1.1.4 Classification report

A summary of evaluation metrics used to assess the performance of a classification model. It provides a detailed breakdown of various metrics for each class in the classification problem. The report typically includes metrics such as precision, recall, F1 score, and support.

- Precision

Precision measures the proportion of correctly predicted positive instances (true positives) out of the total instances predicted as positive (true positives + false positives).

Precision = True Positives / (True Positives + False Positives)

- Recall (also known as Sensitivity or True Positive Rate)

Recall measures the proportion of correctly predicted positive instances (true positives) out of the total actual positive instances (true positives + false negatives).

Recall = True Positives / (True Positives + False Negatives)

- F1 Score

The F1 score is the harmonic mean of precision and recall. It provides a single metric that balances both precision and recall. It is useful when you want to consider both false positives and false negatives equally.

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

- Support

Support represents the number of occurrences of each class in the true dataset. It indicates the number of instances that belong to each class.

These metrics are typically computed for each class in a classification problem and summarized in a classification report. The classification report provides a breakdown of these metrics for each class, allowing you to assess the model's performance on individual classes as well as overall performance.

6.1.2 Object detection

6.1.2.1 Mean avg precision

Mean Average Precision (mAP) is an evaluation metric commonly used in information retrieval tasks, such as search engines, recommender systems, or object detection it combines precision and recall to provide an aggregated measure of the quality of a model's predictions across multiple classes or queries. The computation of mAP involves steps such as constructing precision-recall curves, calculating precision and recall at varying confidence thresholds, interpolating precision values for smoothness, and computing Average Precision (AP) by calculate the area under the precision-recall curve (using the interpolated values) for each class. The final mAP is obtained by averaging the AP values across all classes.

6.2 Results

6.2.1 Phase 1 Results (classification phase)

6.2.1.1 Version 1

In this version, we utilized a preliminary custom structural model, along with Vgg16, ResNet50, and MobileNetV2, without involving TL.

6.2.1.1.1 Preliminary custom model

The preliminary custom structural model architecture is displayed in Figure 6.1

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 126, 126, 64)      1792
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 63, 63, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 61, 61, 32)        18464
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 30, 30, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 28, 28, 16)        4624
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 14, 14, 16)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_5 (Dense)              (None, 128)               401536
_________________________________________________________________
activation_1 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 129
=================================================================
Total params: 426,545
Trainable params: 426,545
Non-trainable params: 0
_________________________________________________________________
```

Figure 6. 1 Structure of Preliminary Custom Model

Figure 6.2 illustrates the progress of accuracy as the number of epochs increases. While Figure 6.3 depicts the progression of loss with the increasing number of epochs. The confusion matrix is shown in Figure 6.4.

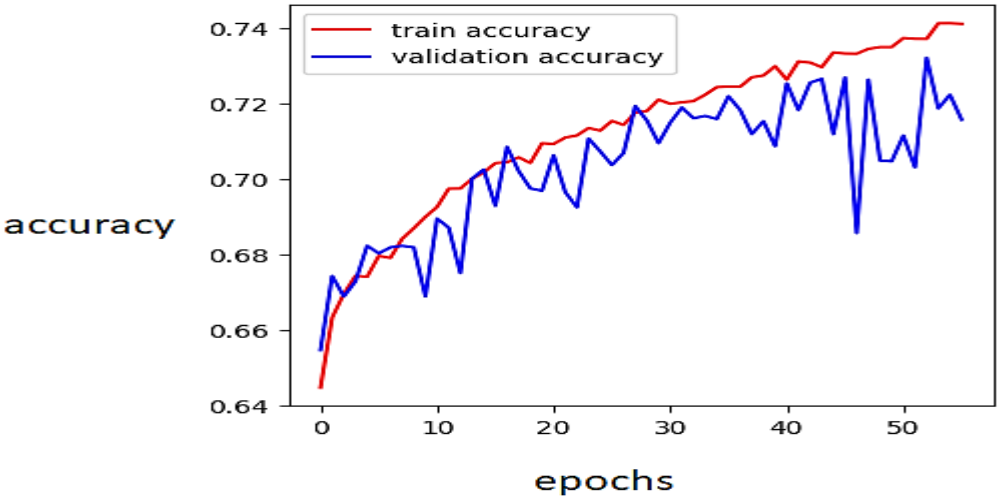

Figure 6. 2 Train accuracy classification of Preliminary custom model (V1)

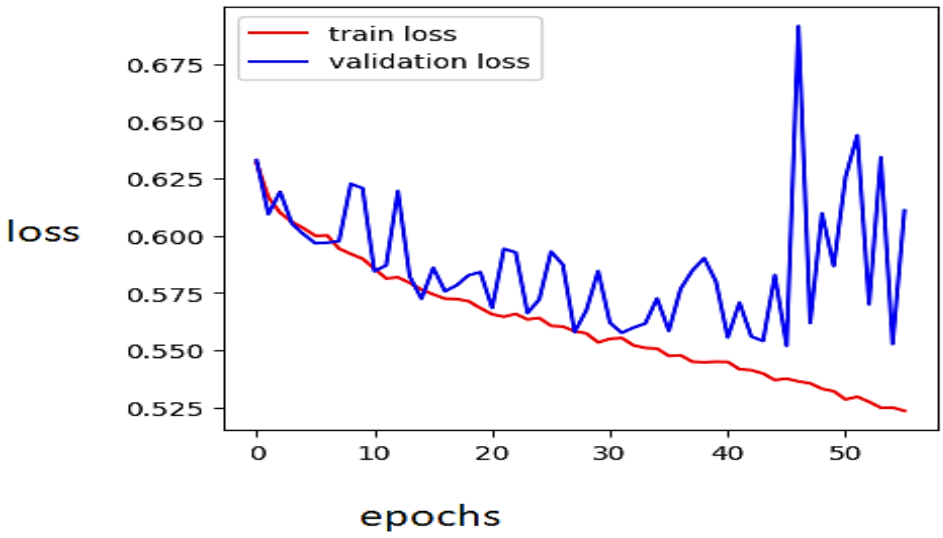

Figure 6. 3 Train loss classification of Preliminary custom model (V1)

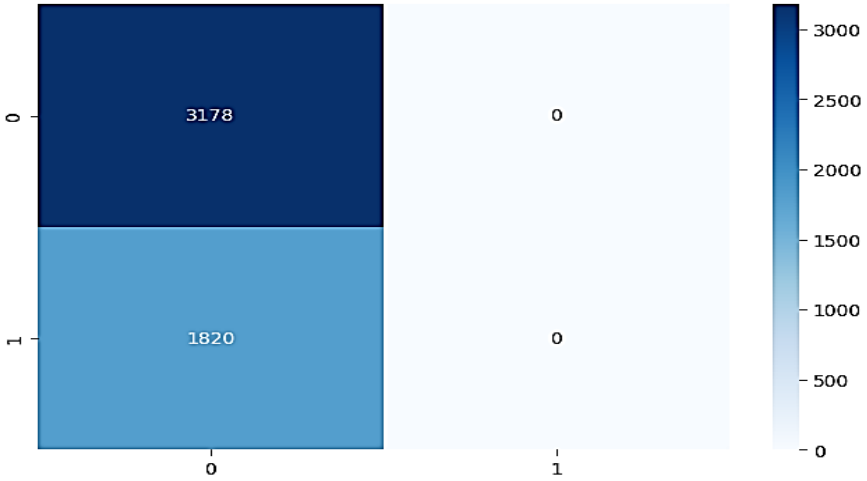

Figure 6. 4 Confusion matrix classification of Preliminary custom model (V1)

The results demonstrate that the proposed preliminary model has failed to detect copy-move forgery. Thus, subsequently, we improved this model structure. The improved custom deep learning model presented in Figure 5.1 is then applied. The progression of training accuracy and loss as the number of epochs increases is depicted in Figures 6.5 and 6.6, respectively. Additionally, Figure 6.7 displays the obtained testing confusion matrix.

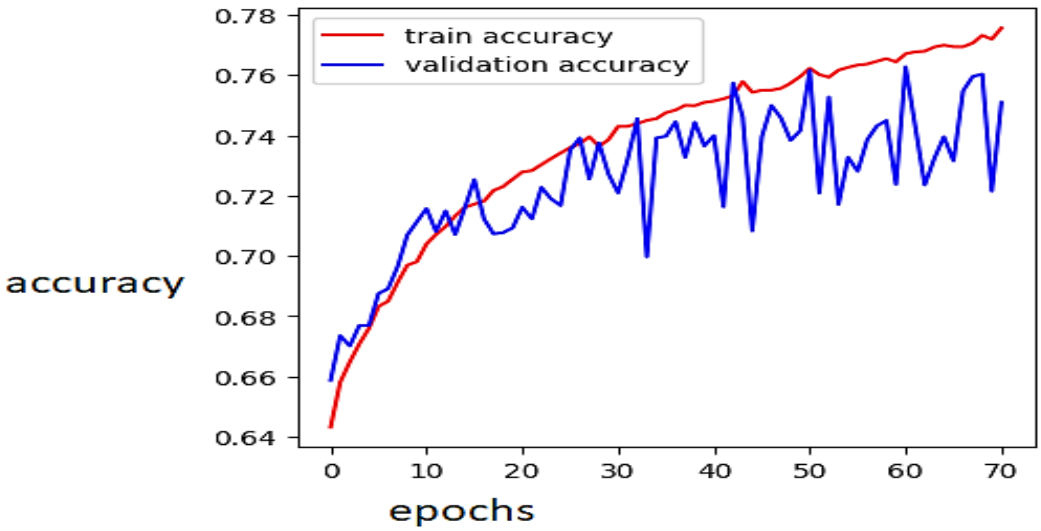

Figure 6. 5 Train accuracy classification of improved custom model (V2)

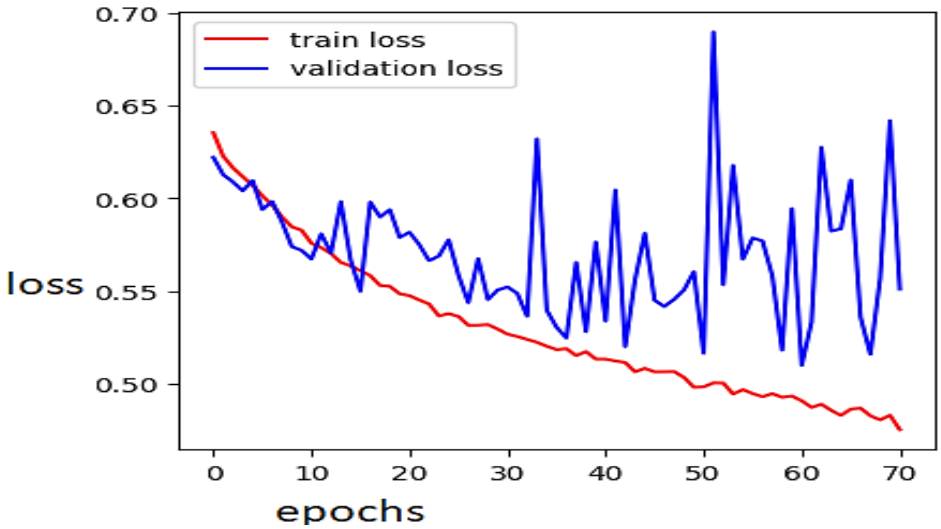

Figure 6. 6 Train loss classification of improved custom model (V2)

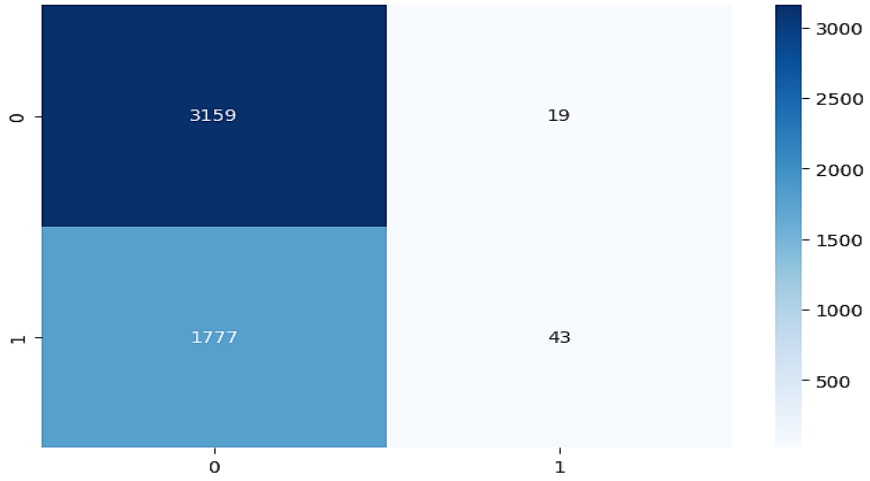

Figure 6. 7 Confusion matrix classification of improved custom model (V2)

6.2.1.1.2 Vgg16 model

The advancement of training accuracy and loss, as the number of epochs augments, are illustrated in Figures 6.8 and 6.9, respectively. The testing confusion matrix, on the other hand, is displayed in Figure 6.10.

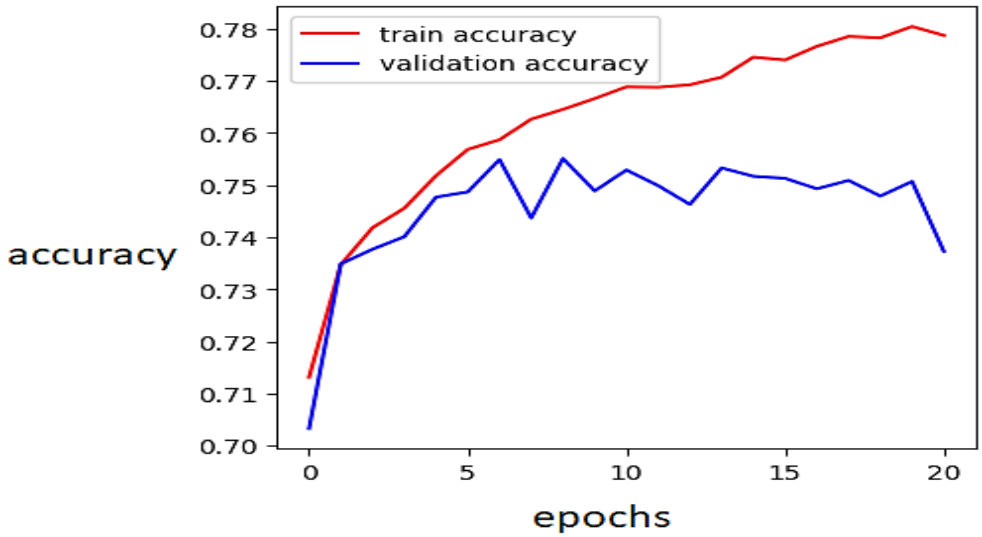

Figure 6. 8 Train accuracy classification of Vgg16 model (V1)

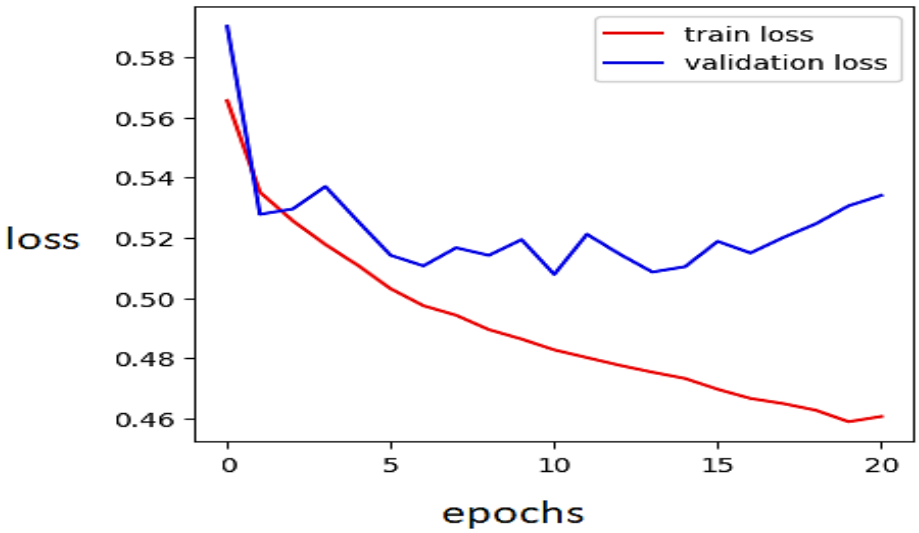

Figure 6. 9 Train loss classification of Vgg16 model (V1)

.

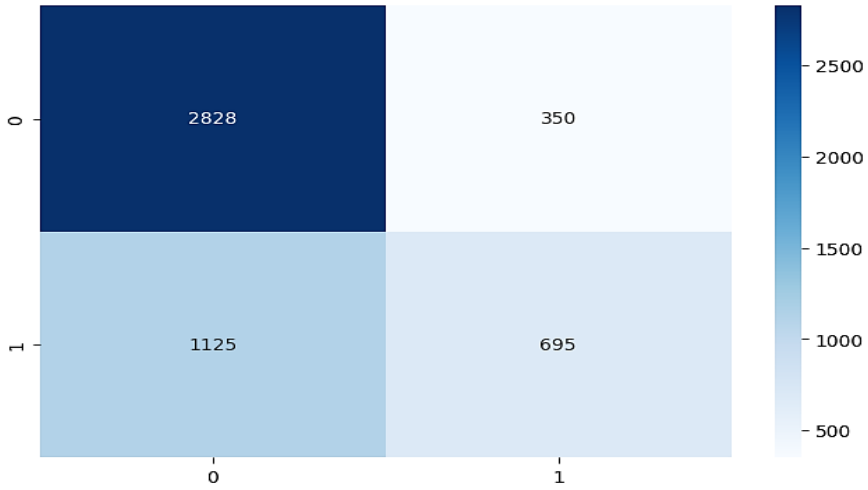

Figure 6. 10 Confusion matrix classification of Vgg16 model (V1)

6.2.1.1.3 Resnet50

When applying Resnet50, the accuracy and loss during the training stage are displayed in Figures 6.11 and 6.12, respectively. Furthermore, the testing confusion matrix is presented in Figure 6.13.

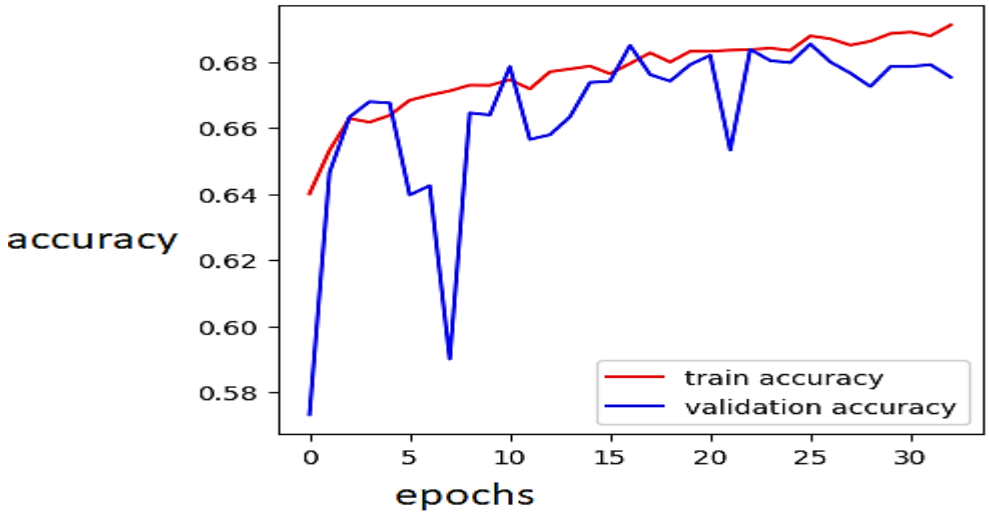

Figure 6. 11 Train accuracy classification of Resnet50 model (V1)

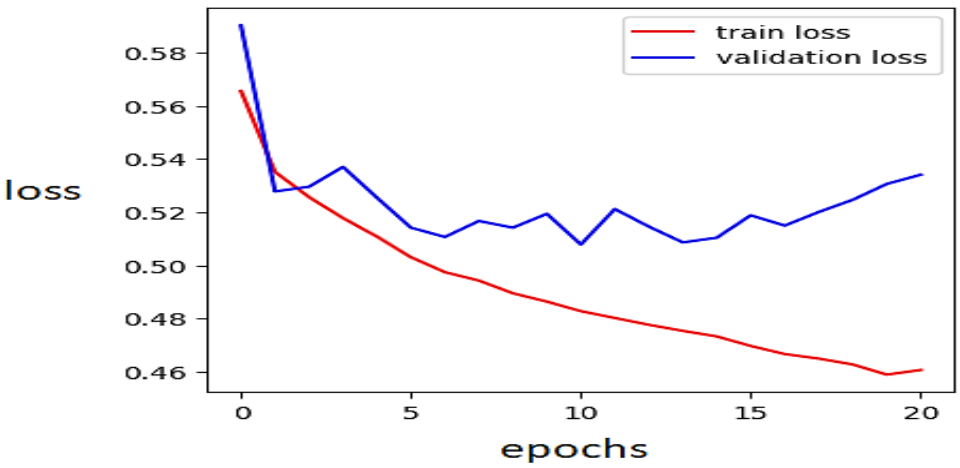

Figure 6. 12 Train loss classification of Resnet50 model (V1)

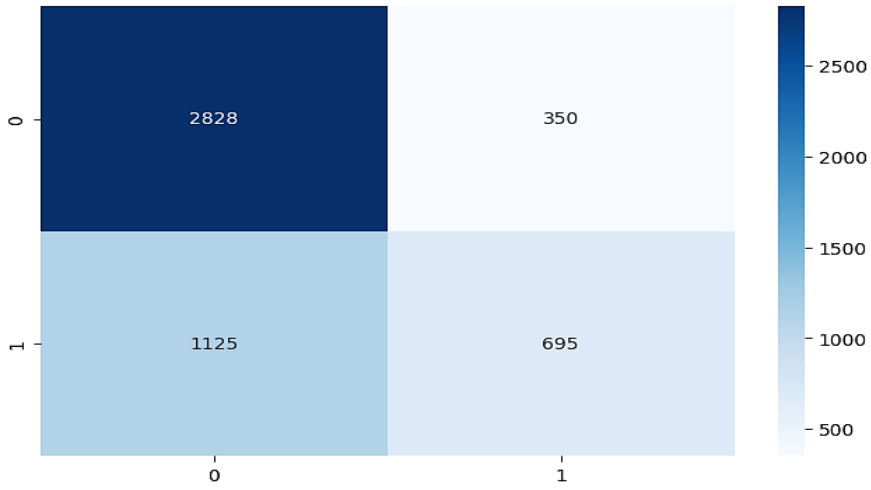

Figure 6. 13 Confusion matrix classification of Resnet50 model (V1)

6.2.1.2 Version 2

In this stage, a fine-tuning procedure is applied to Vgg16, Resnet50, and MobileNetv2 models.

6.2.1.2.1 Vgg16

6.2.1.2.1.1 Train accuracy

The train accuracy and loss are portrayed in Figures 6.14 and 6.15, respectively. Moreover, Figure 6.16 shows the testing confusion matrix of applying the fine-tuned Vgg16.

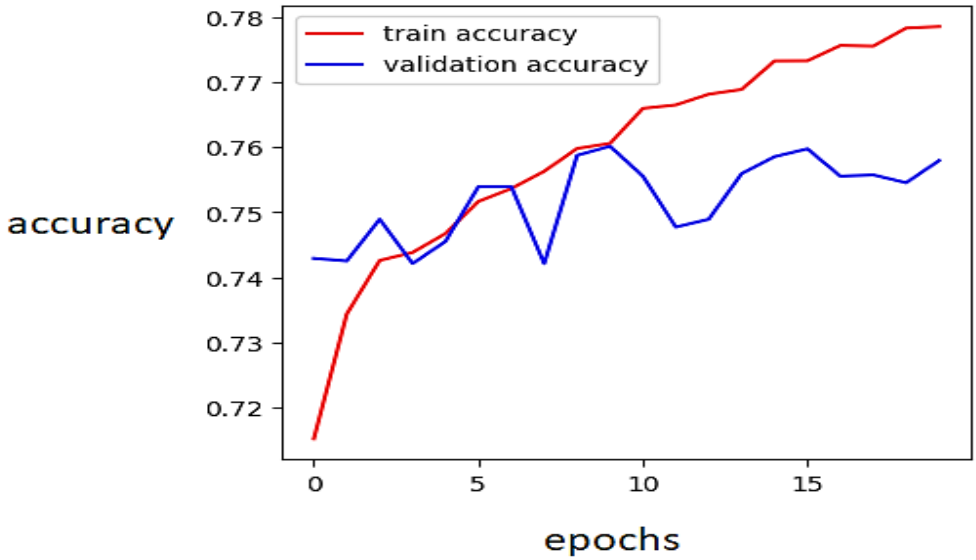

Figure 6. 14 Train accuracy classification of fine-tuned Vgg16 (V2)

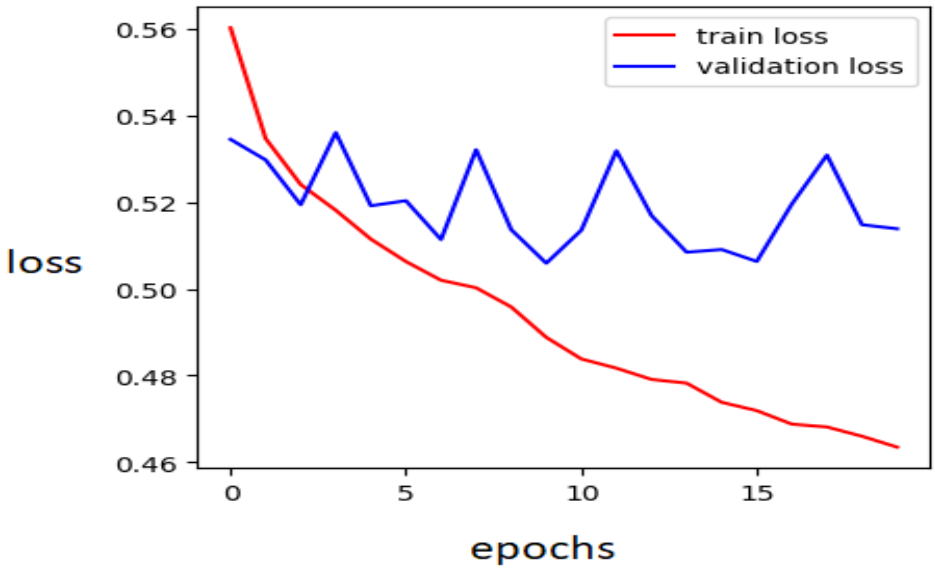

Figure 6. 15 Train loss classification of fine-tuned Vgg16 (V2)

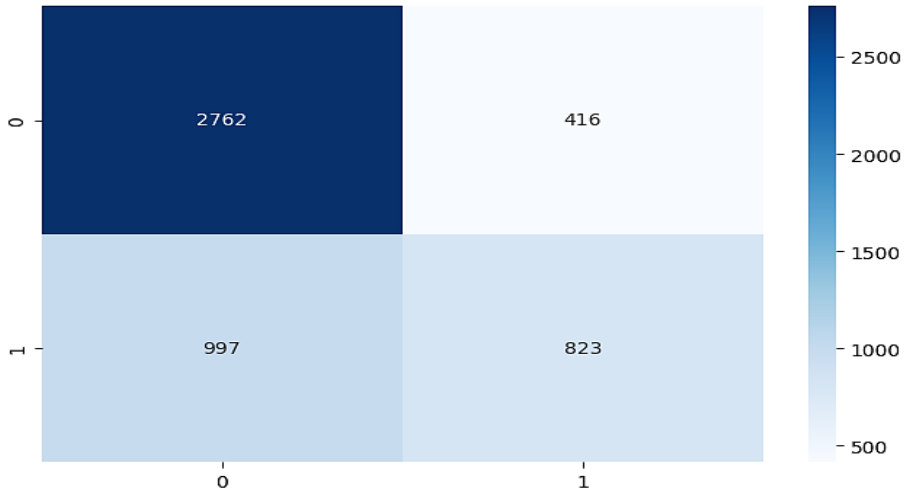

Figure 6. 16 Confusion matrix classification of fine-tuned Vgg16 (V2)

6.2.1.2.2 Resnet 50

In figures 6.17 and 6.18, the graphs illustrate the progression of training accuracy and loss as the number of epochs rises, respectively. Whereas, Figure 6.19 presents the obtained testing confusion matrix.

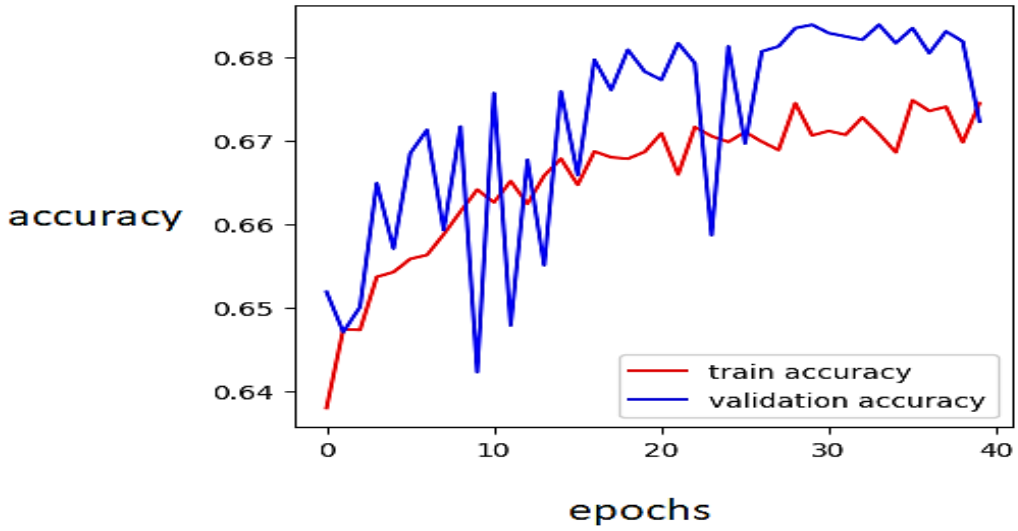

Figure 6. 17 Train accuracy classification of fine-tuned Resnet50 (V2)

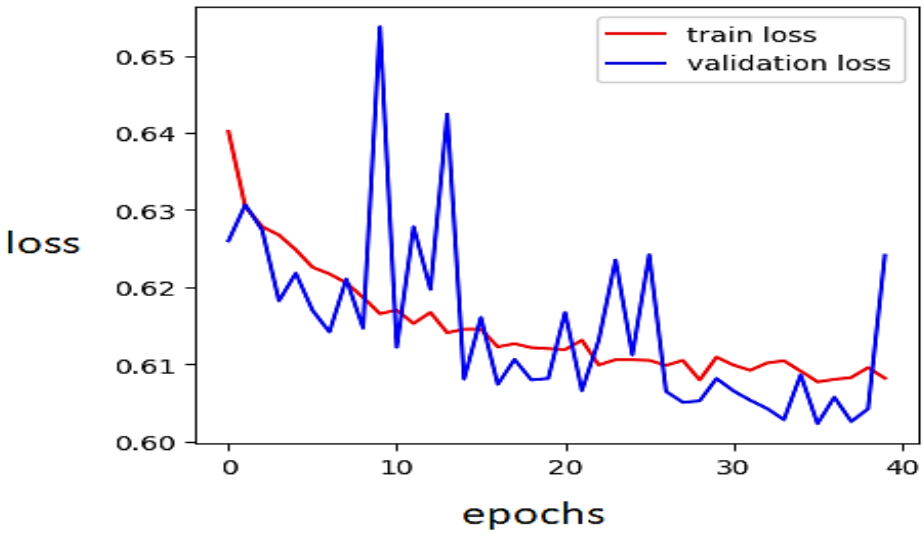

Figure 6. 18 Train loss classification of fine-tuned Resnet50 (V2)

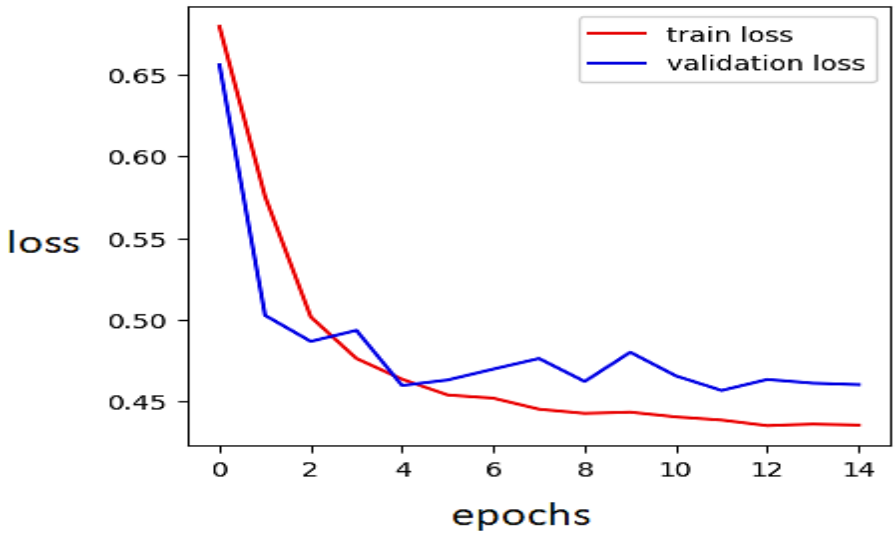

Figure 6. 19 Confusion matrix classification of fine-tuned Resnet50 (V2)

6.2.1.2.3 MobileNetv2

The training accuracy and loss are depicted in Figures 6.20 and 21, respectively. Figure 22, on the other hand, describes the testing confusion matrix.

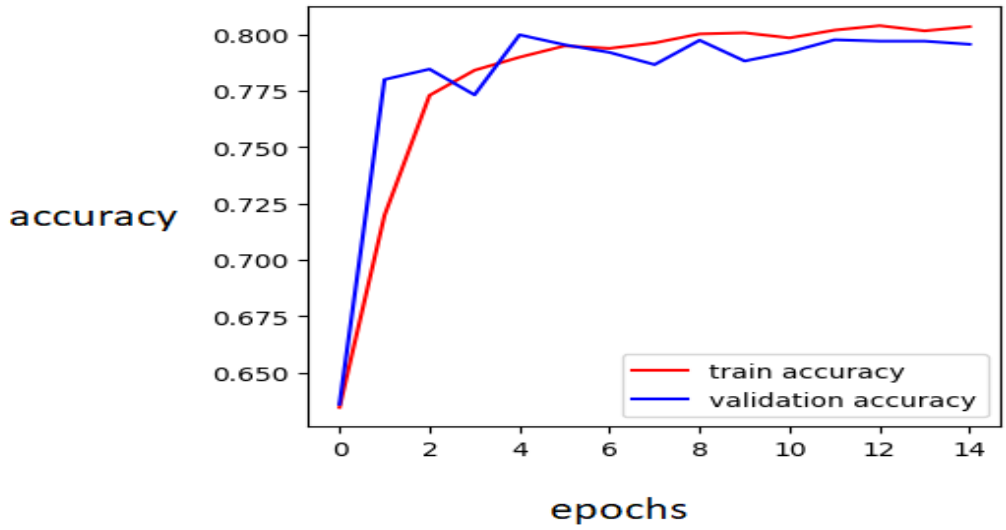

Figure 6. 20 Train accuracy classification of fine-tuned MobileNetv2 (V2)

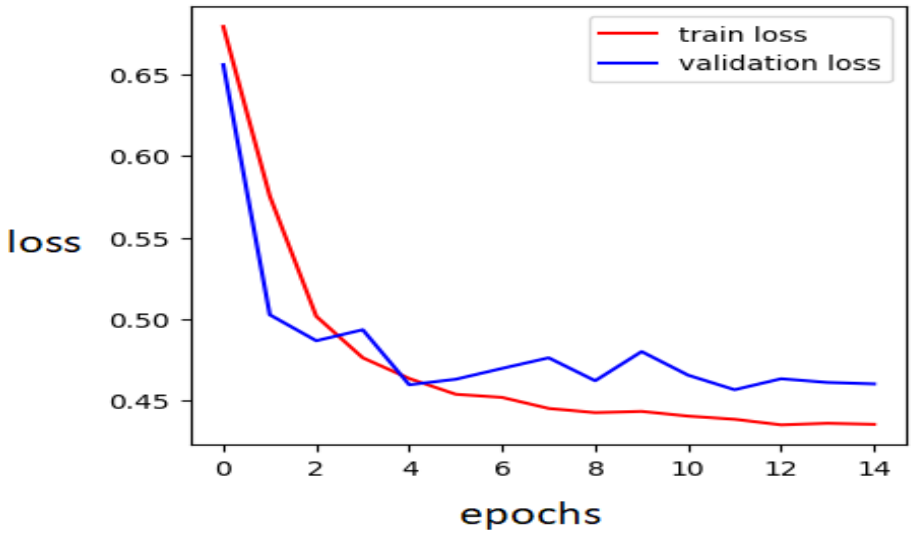

Figure 6. 21 Train loss classification of fine-tuned MobileNetv2 (V2)

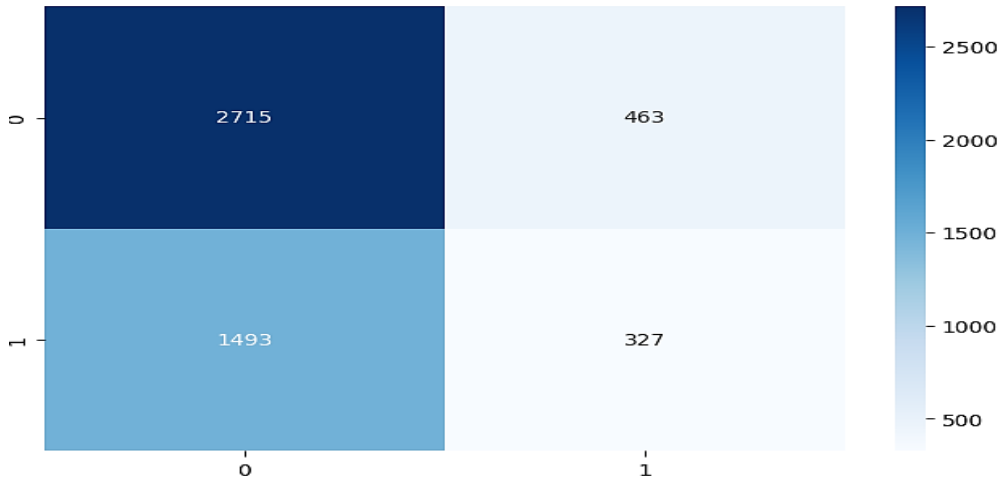

Figure 6. 22 Confusion matrix classification of fine-tuned MobileNetv2 (V2)

6.2.2 Phase 2 Results (Sampling)

In this phase, we use 43985 samples: 20918 fake samples and 23067 authentic samples, each of shape 64 X 64.

6.2.2.1 Improved custom model

The obtained progress of accuracy and loss for training the improved custom model on image patches are exhibited in Figures 6.23 and 6.24, respectively. Furthermore, the testing confusion matrix is illustrated in Figure 6.25.

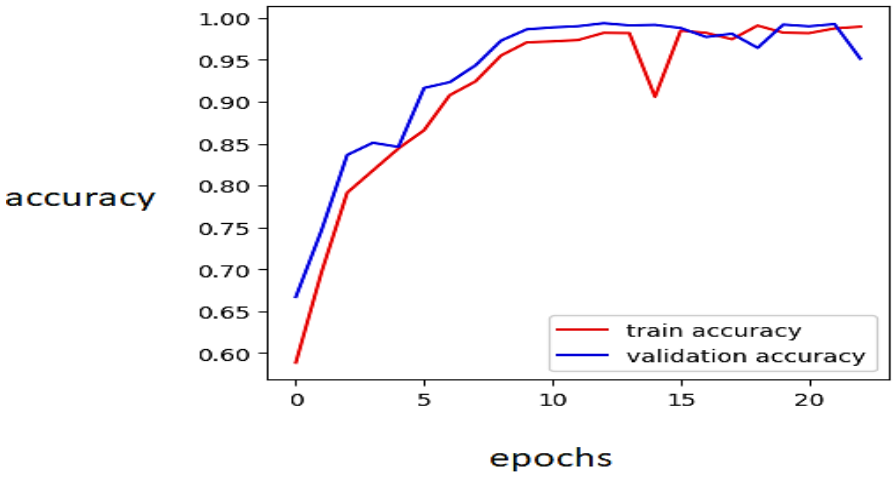

Figure 6. 23 Patch-train accuracy of the improved custom model

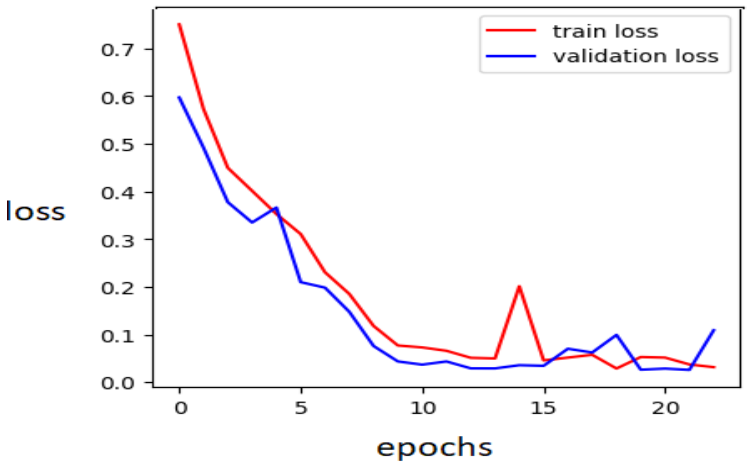

Figure 6. 24 Patch-train loss of the improved custom model

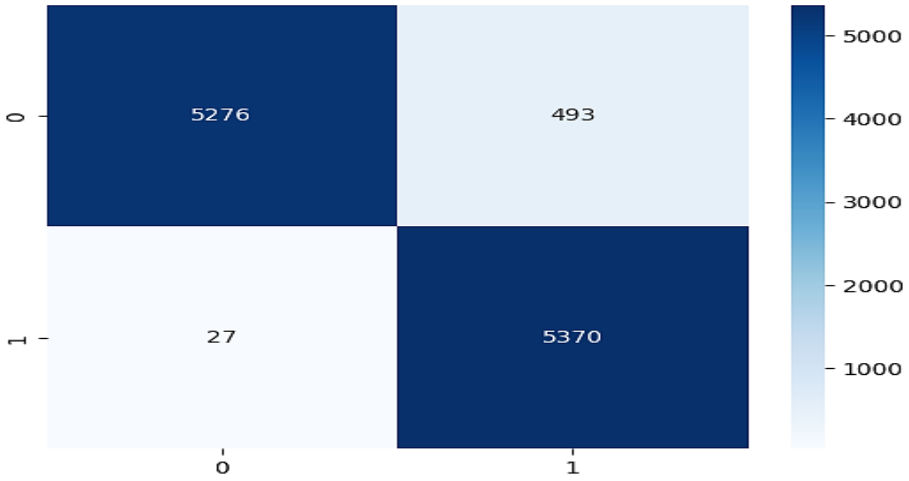

Figure 6. 25 Patch-test confusion matrix of the improved custom model

6.2.2.2 VGG16

Applying the Vgg16, we obtained the training accuracy and loss as portrayed in Figures 6.26 and 6.27, respectively. Besides, Figure 6.28 shows the testing confusion matrix.

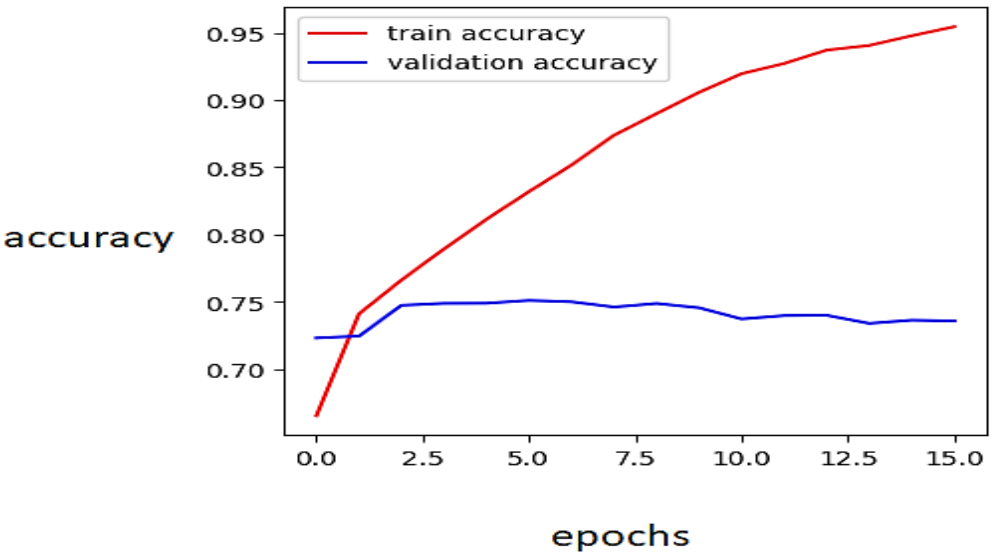

Figure 6. 26 Patch-train accuracy of the fine-tuned Vgg16 model

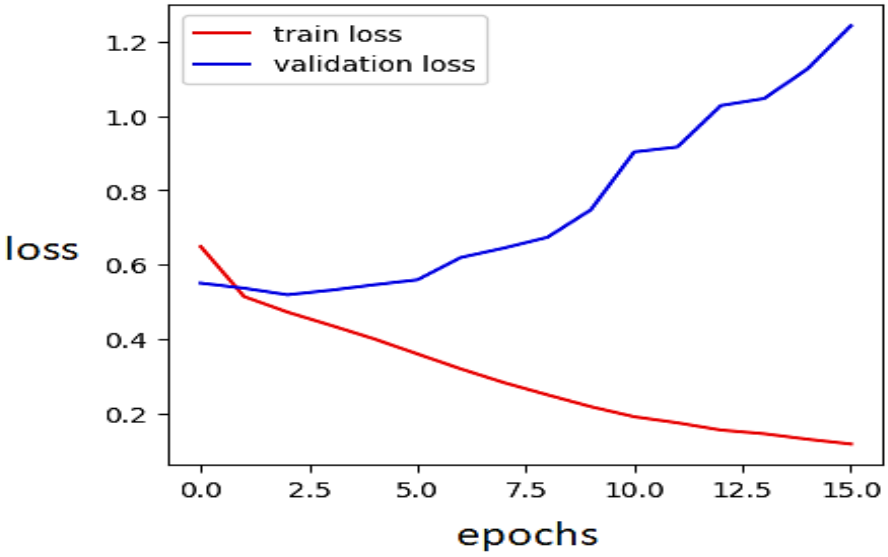

Figure 6. 27 Patch-train loss of the fine-tuned Vgg16 model

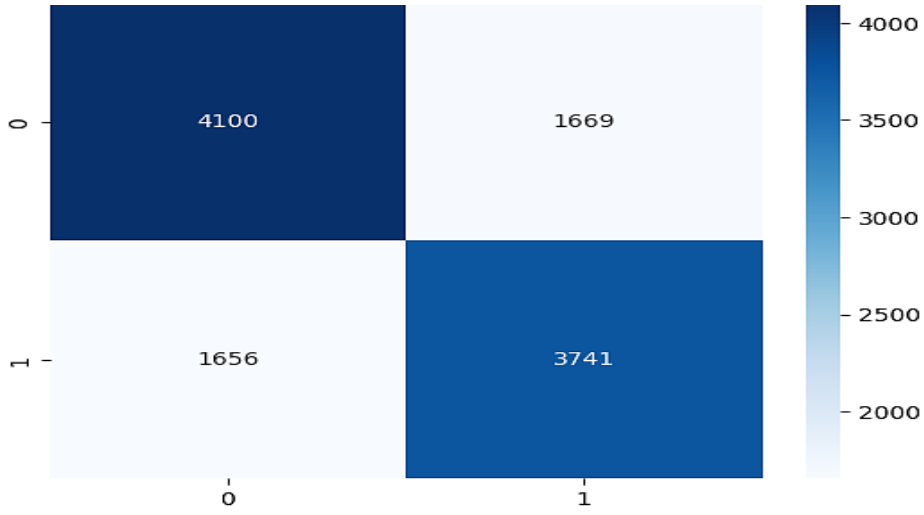

Figure 6. 28 Patch-train confusion matrix of the fine-tuned Vgg16 model

6.2.2.3 Resnet50

Figures 6.29 and 6.30 display the training accuracy and loss obtained by utilizing the Resnet50. Furthermore, the testing confusion matrix is represented in Figure 6.31.

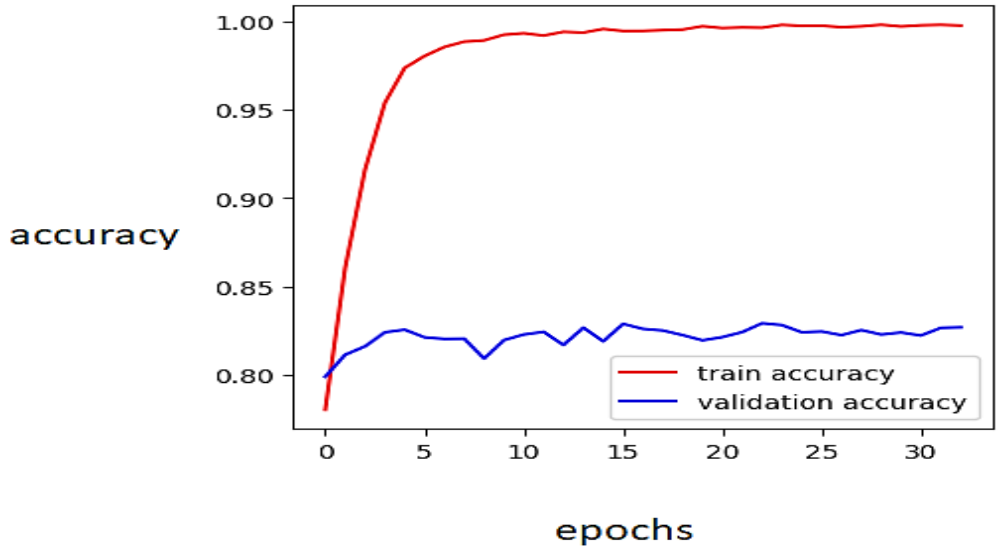

Figure 6. 29 Patch-train accuracy of the fine-tuned resnet50 model

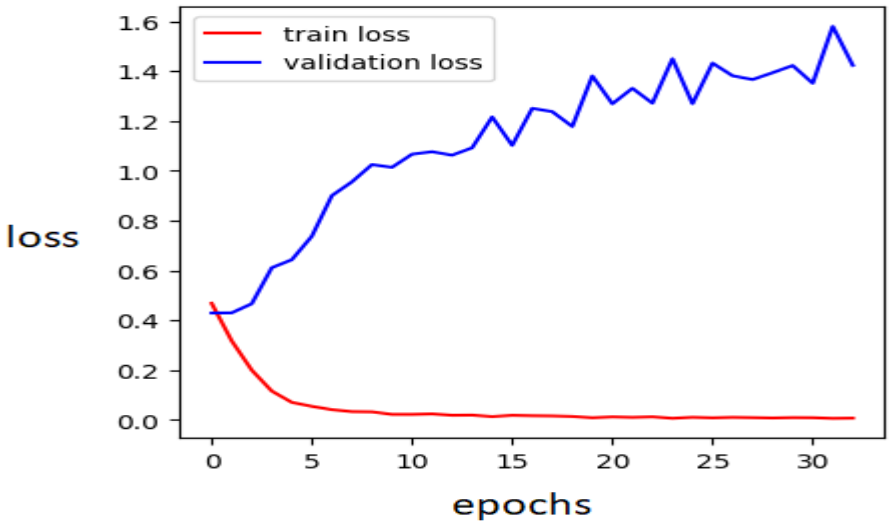

Figure 6. 30 Patch-train loss of the fine-tuned resnet50 model

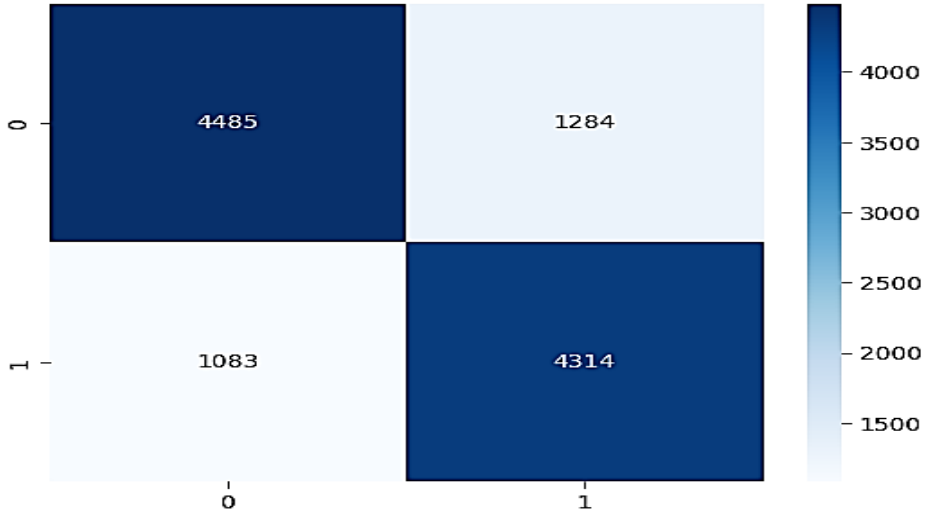

Figure 6. 31 Patch-test confusion matrix of the fine-tuned resnet50 model

6.2.3 Phase 3 Results (object detection)

In most research articles, mAP has extensions like mAP@0.5 and mAP@[0.5:0.95]. Where IoU signifies Intersection Over Union is a metric that finds the difference between ground truth annotations and predicted bounding boxes. A model with mAP@0.5 implements a threshold value of 0.5 to discard unwanted bounding boxes, which is the threshold value employed by many models. Whereas mAP@[0.5:0.95] denotes average mAP over different IoU thresholds, from 0.5 to 0.95, with step 0.05.
Figure 6.32 illustrates the progression of mAP@0.5 with epochs. While the variations of mAP@[0.5:0.95] with epochs is presented in Figure 6.33.

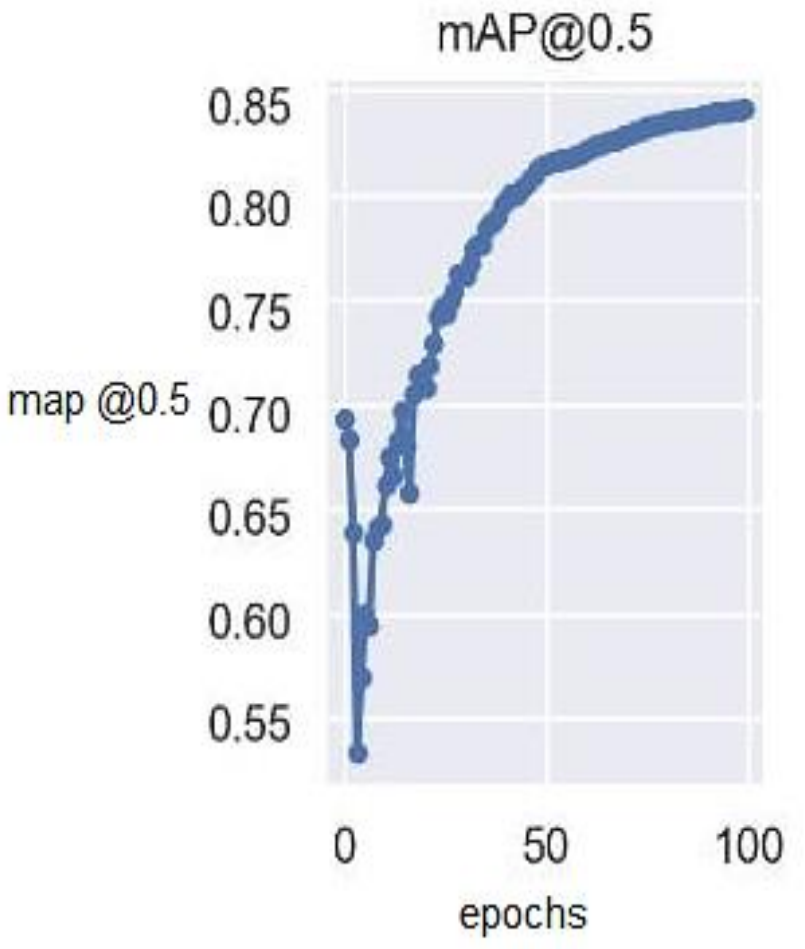

Figure 6. 32 The mAP@0.5 curve in training model along with epoch

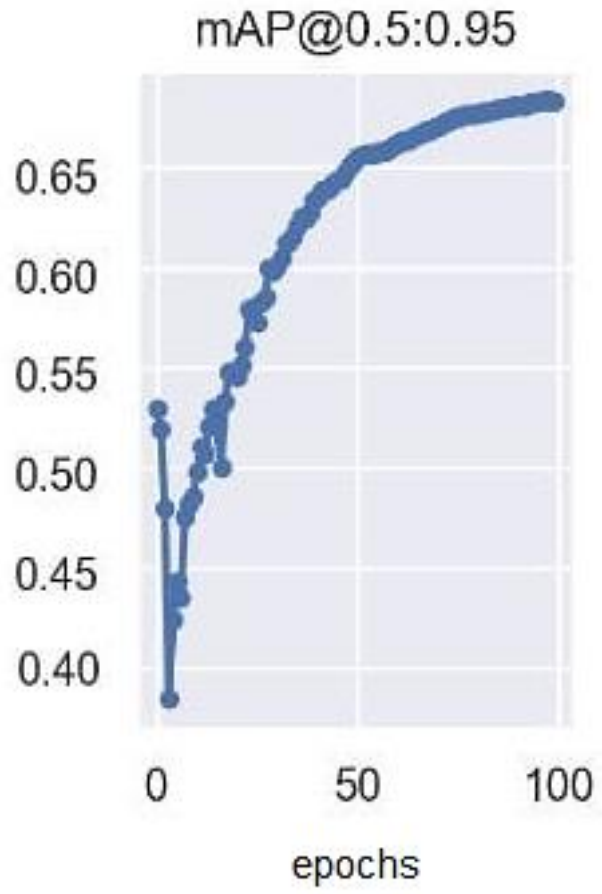

Figure 6. 33 The mAP@[0.5:0.95] curve in training model along with epoch

The object detector predicts bounding boxes, each associated with a confidence score. The confidence score indicates the likelihood of the object class appearing within the bounding box. By applying a threshold, we can classify the confidence probabilities, where detections with score above the threshold are prescribed as true positives (TP), while those below the threshold are classified as false positives (FP). Figures 6.34, 6.35, and 6.36 display the progress of recall, precision and f1 score with epochs, respectively. Since we ought to find the best confidence to achieve the best values for the three metrics, as seen from these figures, we selected the threshold 0. 4, at which precision = 79.9%, recall = 78.3%, and f1 = 78%.

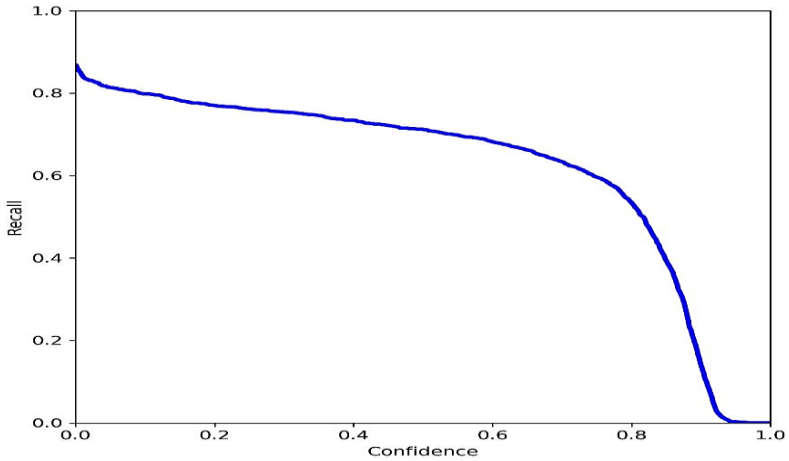

Figure 6. 34 Recall detection confidence threshold curve for YOLO-V7

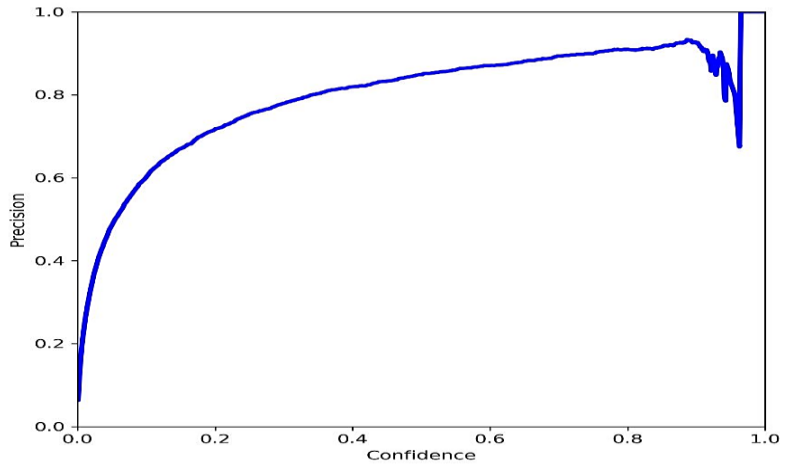

Figure 6. 35 Precision detection confidence threshold curve for YOLO-V7

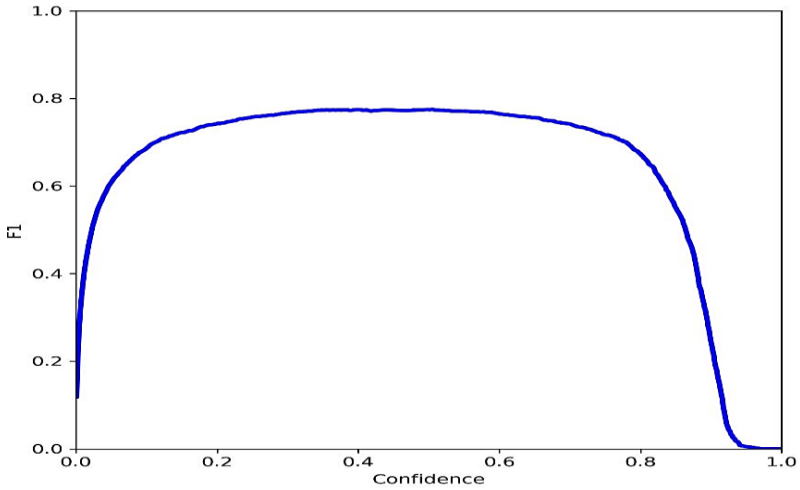

Figure 6. 36 F1 detection confidence threshold curve for YOLO-V7

Chapter 7

CMFD APP

In this chapter, we will demonstrate the development and implementation of a web application for CMFD (CMFD App) using deep learning techniques, with an explicit focus on utilizing object detection. This will enhance the accessibility and user-friendliness of the forgery detection process. The web app is an interface that allows users to satisfactorily upload an image, besides obtaining immediate forgery detection outcomes. By leveraging the power of deep learning and object detection algorithms, the web app provides users with a seamless and efficient solution for identifying instances of copy-move forgery in digital images.

The development of CMFD App aims to simplify the process of CMFD, making it more convenient for users who may not possess advanced technical knowledge in the field of image forensics. Furthermore, the intuitive user interface and seamless operation of the web app enable individuals to quickly assess the authenticity and integrity of their digital images and enhance their ability to detect and address instances of forgery.

By developing CMFD App, we aim to provide a practical and user-friendly tool that can assist individuals in identifying copy-move forgery, thereby contributing to the field of image forensics and enhancing digital image authenticity verification.

7.1 Technologies Used

The development of the CMFD App implicated the utilization of various technologies and frameworks to ensure its efficient and effective implementation. The utilized technologies are described in the following subsections.

7.1.1 Flask

Flask, a lightweight web framework written in Python, was chosen as the foundation for building the web app. Flask provides a simple and flexible framework for handling HTTP requests and responses, making it ideal for developing small to medium-sized web applications. Its extensive ecosystem and ease of use allowed for rapid development and seamless integration with other components.

7.1.2 HyperText Markup Language (HTML)

HyperText Markup Language (HTML) is used for structuring the web app's pages and defining the layout and content. HTML is the standard markup language for creating web pages and providing the necessary structure and elements for building a user interface.

7.1.3 Cascading Style Sheets (CSS)

Cascading Style Sheets (CSS) are employed to enhance the visual appearance and styling of the web app. CSS allows for the customization of fonts, colors, layouts, and other visual elements, resulting in a visually appealing and cohesive user interface.

7.2 CMFD App Architecture

The architecture consists of several key components working together to enable the functionalities of the web app. The following provides an overview of the web app's architecture.

7.2.1 Front-End

The front-end component of the web app is responsible for presenting the user interface and facilitating user interactions. It comprises HTML templates, CSS stylesheets, and client-side scripting languages such as JavaScript. The front-end communicates with the back-end through HTTP requests and receives responses for updating the user interface based on the user's actions.

7.2.2 . Back-End

The back-end component handles the logic and processing of the web app. It receives requests from the front-end, processes them, and returns the necessary

responses. In this case, the back-end is implemented using Flask, a lightweight Python web framework. It utilizes routes and views to handle incoming requests, interacts with the deep learning model for forgery detection, and sends the results back to the front-end.

7.2.3 Deep Learning Model

The core component of the web app is the deep learning model used for CMFD. This model, trained using techniques such as YOLO-V7 object detection, is responsible for processing the uploaded images and identifying instances of forgery. This model, trained using techniques such as YOLO-V7 object detection, is responsible for processing the uploaded images and identifying instances of forgery.

The model takes in an image as input, applies the necessary image processing techniques, performs object detection, and highlights any detected instances of copy-move forgery.

7.3 Pages

7.3.1 Page 1: Image Selection

The model takes in an image as input, applies the necessary image processing techniques, performs object detection, and highlights any detected instances of copy-move forgery. This page provides a user-friendly interface that allows users to conveniently upload an image of their choice. The next subsections describe the purpose and functionality of the Page 1.

7.3.1.1 Purpose

The primary purpose of Page 1 is to enable users to select an image that they want to analyze for copy move forgery. This page acts as a gateway to the forgery detection process, allowing users to provide the input image to the web app.

7.3.1.2 Functionality

Page 1 offers several key functionalities to ensure a smooth user experience:

- Image Upload: Users can upload an image file from their local device by clicking the "Upload" button to browse their device's file system. The web app supports common image file formats such as JPEG, PNG, and GIF.
- Preview: To provide users with immediate feedback, a preview of the uploaded image may be displayed on the page. This allows users to verify that the correct image has been selected before proceeding to the forgery detection process.
- User Experience: The user interface of Page 1 is designed to be intuitive and user-friendly, ensuring that users can easily navigate and interact with the web app. The layout and design elements of the page should be visually appealing and consistent with the overall theme of the web app.

By providing a streamlined and user-friendly image selection process, Page 1 allows users to input the image they want to analyze for copy move forgery. Once an image is successfully uploaded, users can proceed to Page 2, where the forgery detection model processes the image and displays the results.

7.3.2 Page 2: Forgery Detection Results

Page 2 of the web app plays a crucial role in presenting the forgery detection results. After the user selects an image on Page 1, the web app processes the image using the forgery detection model and generates the output. Page 2 then displays the processed image with the detected instances of copy-move forgery. The succeeding depicts the purpose and functionality of Page 2.

7.3.2.1 Purpose

The primary purpose of Page 2 is to provide users with a comprehensive view of the forgery detection results. It allows users to visually identify any instances of copy-move forgery in the uploaded image and gain insights into potential tampering.

7.3.2.2 Functionality

Page 2 proffers several pivotal functionalities to effectively present the forgery detection results.

- Processed Image Display: The processed image, with the detected instances of copy-move forgery highlighted or marked, is displayed prominently on Page 2. Users can see the regions where fraud has been detected, facilitating their understanding of the analysis results.
- Visualization of Detected Forgery: To aid users in identifying the forged regions, visual cues such as bounding boxes or overlays may be employed. These visual indicators delineate the areas identified as a copy-move forgery, enabling users to distinguish them from the rest of the image.
- Navigation Option: Page 2 includes an option that allows users to return to Page 1 to select another image. This stimulates users to smoothly commence a new forgery detection process without having to navigate back manually.
- User Experience: The user interface of Page 2 prioritizes clarity and ease of understanding. It presents the forgery detection results in a visually appealing manner, making it intuitive for users to identify the forged regions. The design elements, such as color schemes and layout, are consistent with the overall aesthetics of the web app.

7.4 CMFD App Screenshots

In this subsection, we present screenshots of the CMFD App's user interface, highlighting the key features and functionality of Page 1 (Image Selection) and Page 2 (Forgery Detection Results). The screenshots deliver a visual representation of the CMFD App and its user experience.

7.4.1 Page 1: Image Selection

Figure 7. 1 Image Selection Page

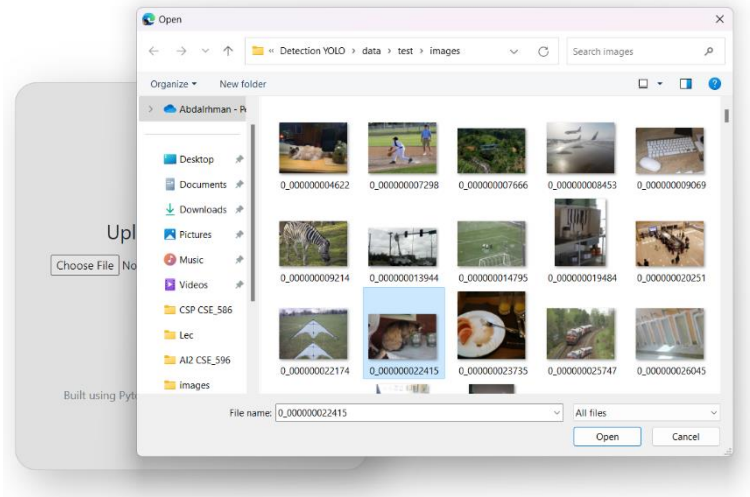

Figure 7. 2 Select The Image

Figure 7. 3 Confirm The Image

7.4.2 Page 2: Forgery Detection Results

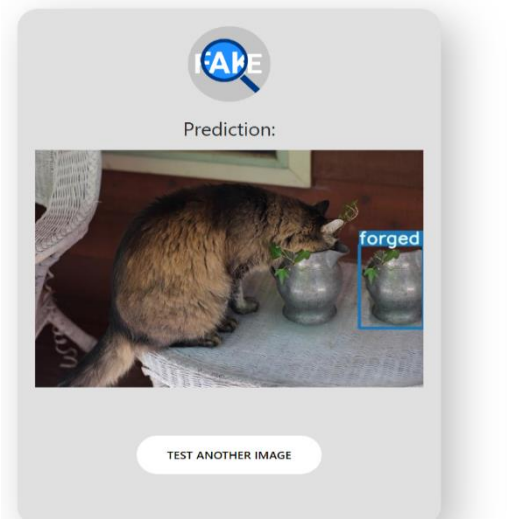

Figure 7. 4 Forgery Detection Result

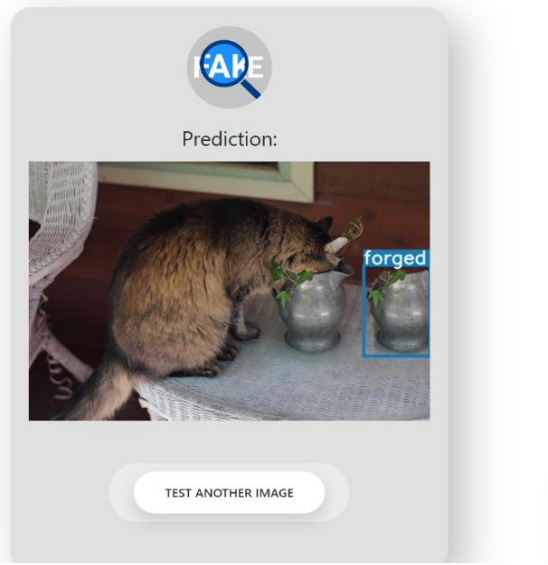

Figure 7. 5 Anouther Test Button

Chapter 8

DISCUSSION

8.1 Classification

The classification method, utilizing custom model, VGG16, ResNet50, and MobileNetV2 architectures, was the initial approach we employed for CMFD. However, the obtained results fell below our expectations, with an overall accuracy of approximately 70%. The obtained results are presented in Table 8.1. Factors such as the complexity of the forgery detection task, and potential suboptimal performance of the chosen pre-trained models might have contributed to this outcome. Recognizing these limitations, we proceeded to explore alternative approaches, including sampling and object detection, which will be discussed in subsequent sections.

Table 8. 1 Classification Algorithms comparison

	Custom model	**VGG-16**	**RaseNet-50**	**MobileNet-V2**
Avg. Accuracy	64%	72%	65%	63%
Avg. Precision	65%	69%	63%	52%
Avg. Recall	49%	65%	50%	48%
Avg. F1_score	39%	67%	38%	34%

8.2 Sampling

We then explored the sampling method as an alternative approach. This involved generating smaller image samples of size 64x64 pixels for training and evaluation. The samples consisted of forgery samples (with at least 20% manipulated and 20% original regions) and randomly selected original samples. The classification models were trained on these samples.

Notably, the sampling method yielded improved results compared to the classification method. The custom model achieved an impressive accuracy of 95% on the generated samples. However, the TL models (VGG16, ResNet50) achieved a relatively lower accuracy of approximately 75%. The obtained out comes are displayed in Table 8.2.

The higher accuracy of the custom model can be attributed to its ability to capture specific characteristics of forgery and original regions through the training process. The limited adaptability of TL models to forgery detection may have impacted their performance. We can try to fine tune the models and it may have better performance.

By employing the sampling method, we demonstrated the potential for improved accuracy in CMFD. However, further research is needed to validate these findings on larger and diverse datasets. Additionally, optimizing TL models for forgery detection is a promising area for future exploration. But for now, we decided to move on and go to more powerful technique that may be the solution for our problem, object detection.

Table 8. 2 Sampling Algorithms Comparison

	Custom model	**VGG-16**	**RaseNet-50**
Avg. Accuracy	95%	70%	79%
Avg. Precision	96%	71%	79%
Avg. Recall	94%	69%	78%
Avg. F1_score	95%	70%	79%

8.3 YOLO-V7 Object Detection

We also explored the object detection method using the YOLO-V7 algorithm. Training the model solely on the forgery images and their corresponding masks, we achieved an impressive mean Average Precision (mAP) of 83.9% at a threshold of 0.5. The object detection approach outperformed the classification and sampling methods by accurately identifying and localizing forgery regions. Its ability to handle varying forgery sizes and locations contributed to its superior performance. This method holds great potential for applications in digital forensics, image authenticity verification, and combatting image manipulation. Further research can focus on fine-tuning the model, exploring advanced architectures, and expanding the dataset for improved detection capabilities.

8.4 Comparative Analysis

In this section, we compare the effectiveness of the different methods used for CMFD: classification, sampling, and object detection. This comparison is exhibited in table 8.3.

Table 8. 3 Comparison of CMFD methods

Method	Strengths	Limitations	Performance
Classification	Intuitive approach	Lower overall accuracy because of complexity of the task	Around 70%
Sampling	Improved performance compared to classification	Accuracy varies based on dataset characteristics; prediction takes long time	Custom model (95%), TL models (70%)
Object Detection	Superior accuracy and precise localization	Reliance on accurate masks	Mean Average Precision (mAP) of 83.9% at threshold 0.5

The table provides a concise overview of the strengths, limitations, and performance of each method. The classification method is intuitive but has lower accuracy. The sampling method improves performance, but results may vary and take a long time. The object detection method excels in accuracy and localization but requires accurate masks. Further details and insights can be obtained by looking at the "Experimental Results" chapter.

8.5 Implications and Limitations

8.5.1 Implications:

The object detection method using YOLO-V7 showed superior accuracy in detecting copy-move forgeries, with implications for digital forensics and image authenticity verification. Integration into forensic tools and image manipulation detection systems can enhance their effectiveness in combating image fraud.

8.5.2 Limitations

The classification models had lower accuracy compared to the object detection method. The sampling method would benefit from a larger and more diverse dataset. The object detection method relies on accurate masks for forgery images. Other types of image manipulations were not explicitly considered. Addressing these limitations can improve future forgery detection research.

Chapter 9

CONCLUSION

9.1 Recapitulation of Objectives

Throughout this project, our main objective was to develop an effective CMFD system using deep learning techniques, with a specific focus on the YOLO-V7 object detection algorithm. We aimed to address the challenges of identifying and localizing forged regions in digital images by leveraging the power of deep learning models. By training and evaluating different methods, including classification, sampling, and object detection, we sought to achieve accurate and reliable forgery detection results. Through our research, we aimed to contribute to the field of digital forensics and image authenticity verification by advancing the capabilities of forgery detection systems.

9.2 Summary of Methodologies

In this study, we employed three main methodologies to tackle the problem of copy-move forgery detection. Firstly, we utilized the classification approach, training custom models as well as pre-trained models such as VGG16, ResNet50, and MobileNetV2. Secondly, we implemented the sampling method, generating smaller image samples with manipulated and original regions, and training classification models on these samples. Lastly, we utilized the object detection method, training the YOLO-V7 algorithm specifically on forgery images and their corresponding masks. These methodologies provided different perspectives and techniques for detecting and localizing copy-move forgeries, allowing us to explore the strengths and limitations of each approach.

9.3 Discussion of Results:

The experimental results obtained from our study provide valuable insights into the performance and effectiveness of the different methodologies for copy-move forgery detection. The classification method, while intuitive, yielded moderate accuracy, with the highest achieved accuracy being around 70%. This indicates the complexity of the forgery detection task and the potential limitations of the chosen pre-trained models.

In contrast, the sampling method showed improved performance compared to classification. By generating smaller image samples and training classification models on these samples, we achieved a remarkable accuracy of 95% with our custom model. However, it is important to note that the effectiveness of the sampling method may vary based on dataset characteristics, such as size, diversity, and complexity of forgery instances.

The object detection method using the YOLO-V7 algorithm emerged as the most successful approach in our study. By training the model exclusively on forgery images and their corresponding masks, we achieved a mean Average Precision (mAP) of 83.9% at a threshold of 0.5. The object detection method excelled in accurately identifying and localizing forgery regions, regardless of their size or location. The spatial relationships captured by the model and its ability to consider overall context contributed to its superior performance.

These results highlight the importance of considering the unique characteristics of copy-move forgeries when designing detection methodologies. The object detection method showcased the significance of leveraging contextual information and spatial understanding to enhance forgery detection accuracy.

9.4 Implications of Findings:

The findings of this study have significant implications for the field of CMFD and digital forensics. The success of the object detection method using the YOLO-V7 algorithm highlights its potential in enhancing the accuracy and efficiency of forgery detection systems. By accurately localizing forged regions, this method can assist forensic investigators in identifying tampered areas within digital images, aiding in the detection of image fraud and manipulation.

The implications of our research extend beyond CMFD. The object detection method's ability to precisely locate forged regions can be valuable in image authenticity verification tasks. Integration of such techniques into forensic tools and image manipulation detection systems can enhance their effectiveness in combating various forms of image fraud, including copy-move forgeries.

Overall, the findings of this study contribute to advancing the field of digital forensics and image authenticity verification. The successful application of deep learning techniques, particularly the object detection method, demonstrates the potential of leveraging advanced algorithms for accurate and reliable forgery detection. Future research can build upon these findings to further improve forgery detection systems, explore other types of image manipulations, and enhance the robustness of the algorithms in real-world scenarios.

9.5 Final Remark

In conclusion, this project highlights the effectiveness of the object detection method, specifically using the YOLO-V7 algorithm, in detecting copy-move forgeries. The findings contribute to advancements in digital forensics and image authenticity verification. Further research is needed to improve classification models, diversify datasets, and explore other types of image manipulations. Overall, this project underscores the importance of deep learning techniques in enhancing forgery detection systems for a more secure digital environment.

REFERENCES

[1] Bala, S. K. (2020, Dec 24). Dual branch convolutional neural network for copy move forgery detection. Retrieved from https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/ipr2.12051

[2] Biswal, A. (2022, Dec 8). Top 10 Deep Learning Algorithms. Retrieved from simplilearn: https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm

[3] Brush, E. B. (2022, Nov). deep learning. Retrieved from tech target: https://www.techtarget.com/searchenterpriseai/definition/deep-learning-deep-neural-network

[4] Chien-Yao Wang, A. B.-Y. (2022, Jul 6). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. (Cornell University) Retrieved 2023, from https://arxiv.org/abs/2207.02696

[5] dataset, F. I. (2018, April). Kaggle. Retrieved from https://www.kaggle.com/datasets/hsankesara/flickr-image-dataset

[6] defacto-copymove. (2022, May 1). Kaggle. Retrieved from https://www.kaggle.com/datasets/defactodataset/defactocopymove

[7] Digital Forensics 2023. (2022, Nov 16). (Zenodo) Retrieved from https://zenodo.org/record/7326540#.Y7SDq9XP23B

[8] FORGERY IMAGE DATASET. (2022, Jun 17). (IEEE) Retrieved from https://ieee-dataport.org/documents/forgery-image-dataset

[9] Huilgol, P. (2020, Aug 18). Top 4 Pre-Trained Models for Image Classification. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2020/08/top-4-pre-trained-models-for-image-classification-with-python-code/

[10] ITU Statistics. (2018, Jun). Retrieved from https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx

[11] Karen Simonyan, A. Z. (2019, Aug). Cornell University. Retrieved from https://arxiv.org/abs/1409.1556v6

[12] Kumar, S. W. (2021, Jul). Fusion of Handcrafted and Deep Features for Forgery Detection in Digital Images. Retrieved from https://www.researchgate.net/publication/353201054_Fusion_of_Handcrafted_and_Deep_Features_for_Forgery_Detection_in_Digital_Images

[13] NG, A. (2020, oct). What is Deep Learning. Retrieved from Net App : https://www.netapp.com/artificial-intelligence/what-is-deep-learning/

[14] Prabhu. (2018, Mar 4). Understanding of Convolutional Neural Network (CNN) — Deep Learning. Retrieved from medium: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

[15] Sedik, M. A. (2021, Mar). Deep learning based algorithm (ConvLSTM) for Copy Move Forgery Detection. Retrieved from https://www.researchgate.net/publication/349855286_Deep_learning_based_algorithm_ConvLSTM_for_Copy_Move_Forgery_Detection

[16] Team, I. (2020, jul 26). What is deep learning? Retrieved from IBM: https://www.ibm.com/topics/deep-learning

[17] Team, S. (2018, Aug 18). Convolutional Neural Networks (CNN): Step 4 - Full Connection. Retrieved from super data science: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-4-full-connection

[18] Zia, E. U. (2021, Nov 11). Deep Learning-Based Digital Image Forgery Detection System. Retrieved from https://www.mdpi.com/2076-3417/12/6/2851

Appendix A

A.1 Classification

Imports

```
import os
import numpy as np
import cv2
from PIL import Image
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator,
img_to_array, load_img
from tensorflow.keras import layers, models
from tensorflow.keras.models import Sequential , Model, load_model
from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense,
Conv2D, MaxPooling2D
from tensorflow.keras.utils import to_categorical
from keras.callbacks import EarlyStopping
from tensorflow.keras.applications.resnet import ResNet50
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2
from keras import optimizers
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import log_loss, accuracy_score, confusion_matrix,
classification_report
```

Exploratory Data Analysis

Number of Images

```
fake_path = 'data/fake/images/'
original_path = 'data/original/'
print('Number of fake images = {}'.format((len(os.listdir(fake_path)))))
print('Number of pristine images
={}'.format((len(os.listdir(original_path)))))
```

Fake Images Shape

```
fakes = os.listdir(fake_path)
fake_shapes=[]
for fake in fakes:
    img = Image.open(fake_path+fake)
    img_array = np.array(img)
    fake_shapes.append(img_array.shape)
# random 20 sample of fake images shape
for i in range(20):
    ind=np.random.randint(0, len(fake_shapes))
    print(str(ind) +'\t\t'+ str(fake_shapes[ind]) +'\t\t'+fakes[ind])
```

Original Images Shape

```
originals = os.listdir(original_path)
original_shapes=[]
for original in originals:
    img = Image.open(original_path+original)
    img_array = np.array(img)
    original_shapes.append(img_array.shape)
# random 20 sample of original images shape
for i in range(20):
    ind=np.random.randint(0, len(original_shapes))
    print(str(ind) +'\t\t'+ str(original_shapes[ind])
+'\t\t'+originals[ind])
```

Train Test Split

```
labels=[0]*31783+[1]*18194
x_train, x, y_train, y = train_test_split(image_names, labels,
test_size=0.2, stratify=labels)
x_test, x_valid, y_test, y_valid = train_test_split(x, y, test_size=0.5,
stratify=y)
print("X train: ", len(x_train) ,", Y train: ", len(y_train))
print("X validation: ",len(x_valid) ,", Y validation: ", len(y_valid))
print("X test: ",len(x_test) ,", Y test: ", len(y_test))
```

Prepare Train Set

```
# read train images and resize it to be (128,128) so we can train the model
on it

dim=(128,128)

x_train_images=[]

for ind, x in enumerate(x_train):
    if y_train[ind]==0:
        img = Image.open(original_path+x)
        img_array = np.array(img)
    if y_train[ind]==1:
        img = Image.open(fake_path+x)
        img_array = np.array(img)
    img = cv2.resize(img_array, dim, interpolation = cv2.INTER_AREA)

    x_train_images.append(img)
```

Prepare Valid Set

```
x_valid_images=[]

for ind, x in enumerate(x_valid):
    if y_valid[ind]==0:
        img = Image.open(original_path+x)
        img_array = np.array(img)
    if y_valid[ind]==1:
```

```
        img = Image.open(fake_path+x)
        img_array = np.array(img)
    img = cv2.resize(img_array, dim, interpolation = cv2.INTER_AREA)

    x_valid_images.append(img)
```

Prepare Test Set

```
x_test_images=[]

for ind, x in enumerate(x_test):
    if y_test[ind]==0:
        img = Image.open(original_path+x)
        img_array = np.array(img)
    if y_test[ind]==1:
        img = Image.open(fake_path+x)
        img_array = np.array(img)
    img = cv2.resize(img_array, dim, interpolation = cv2.INTER_AREA)

    x_test_images.append(img)
```

Image Data Generator

```
# convert the list of images to array of 4 dimensions

x_train_images = np.stack(x_train_images)
x_valid_images = np.stack(x_valid_images)
x_test_images = np.stack(x_test_images)
print( "\nx_train_images shape is: " , x_train_images.shape)
print( "\nx_valid_images shape is: " , x_valid_images.shape)
print( "\nx_test_images shape is: " , x_test_images.shape)

# Create an instance of ImageDataGenerator for data augmentation and
preprocessing

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    vertical_flip=False,
    fill_mode='reflect'
)

# Define the batch size

batch_size = 32

# Define the generator for the training data

train_generator = train_datagen.flow(
    x_train_images,
    y_train,
    batch_size=batch_size,
    seed=42,
    shuffle=True,
```

```
)

# Define the generator for the validation data

valid_datagen = ImageDataGenerator(rescale=1./255)

valid_generator = valid_datagen.flow(
    x_valid_images,
    y_valid,
    batch_size=batch_size,
    seed=42,
    shuffle=True,
)
```

Training Deep Learning Models

Custom Model

In [46]:

```
image_shape=(128,128,3)
custom_model = Sequential()

custom_model.add(Conv2D(32, (3, 3), activation='relu',
input_shape=image_shape))
custom_model.add(MaxPooling2D(pool_size=(2, 2)))
custom_model.add(Conv2D(64, (3, 3), activation='relu'))
custom_model.add(MaxPooling2D(pool_size=(2, 2)))

custom_model.add(Conv2D(128, (3, 3), activation='relu'))
custom_model.add(MaxPooling2D(pool_size=(2, 2)))

custom_model.add(Conv2D(128, (3, 3), activation='relu'))
custom_model.add(MaxPooling2D(pool_size=(2, 2)))

custom_model.add(Flatten())

custom_model.add(Dense(512, activation='relu'))
custom_model.add(Dropout(0.5))

custom_model.add(Dropout(0.3))

custom_model.add(Dense(1, activation='sigmoid'))

custom_model.compile(loss='binary_crossentropy',
                     optimizer='adam',
                     metrics=['accuracy'])

# CallBack function to stop the training
# if 10 epochs pass without increase val_acc

earlystop = EarlyStopping(monitor='val_accuracy', patience=10, mode='max')


custom_model_history = custom_model.fit_generator(train_generator,
                                    validation_data  = valid_generator,
                                    epochs = 100,
```

```
                                    steps_per_epoch=(len(train_generator)),

                                    validation_steps=(len(valid_generator))
                                    ,callbacks=[earlystop])
```

VGG16 Model

```
## Loading VGG16 model
base_model = VGG16(weights="imagenet", include_top=False,
input_shape=image_shape)
base_model.trainable = False ## Not trainable weights
flatten_layer = layers.Flatten()
dense_layer_1 = layers.Dense(50, activation='relu')
dense_layer_2 = layers.Dense(20, activation='relu')
prediction_layer = layers.Dense(1, activation='sigmoid')

vgg_model = models.Sequential([
    base_model,
    flatten_layer,
    dense_layer_1,
    dense_layer_2,
    prediction_layer
])

vgg_model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

vgg_model_history = vgg_model.fit_generator(train_generator,
                                    validation_data  = valid_generator,
                                    epochs = 100,
                                    steps_per_epoch=(len(train_generator)) ,
                                    validation_steps=(len(valid_generator)),
                                    callbacks=[earlystop])
```

RasNet50 Model

```
# load RaseNet50
base_model=ResNet50(weights='imagenet', include_top=False,
input_shape=(128, 128, 3))
base_model.trainable = False

resnet_model=Sequential()
resnet_model.add(base_model)

top_model=Sequential()
top_model.add(Flatten())

top_model.add(Dense(64, activation='relu'))

top_model.add(Dense(1, activation='sigmoid'))

resnet_model.add(top_model)

resnet_model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```
resnet_model_history = resnet_model.fit_generator(train_generator,
                                        validation_data  = valid_generator,
                                        epochs = 100,
                                       steps_per_epoch=(len(train_generator))
                                  ,validation_steps=(len(valid_generator)),
                                        callbacks=[earlystop])
```

MobileNetV2 Model

```
# load MobileNetV2
base_model=MobileNetV2(weights='imagenet', include_top=False,
input_shape=(128, 128, 3))
base_model.trainable = False


mobilenet_model=Sequential()
mobilenet_model.add(base_model)

top_model=Sequential()
top_model.add(Flatten())

top_model.add(Dense(64, activation='relu'))

top_model.add(Dense(1, activation='sigmoid'))

mobilenet_model.add(top_model)

mobilenet_model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

mobilenet_model_history = mobilenet_model.fit_generator(train_generator,
                                                        validation_data  =
valid_generator,
                                                        epochs = 100,

steps_per_epoch=(len(train_generator)) ,

validation_steps=(len(valid_generator)),

callbacks=[earlystop])
```

A.2. Sampling

Sampling The Fake Images

```
# Function that return:
# Number of pixels that have value 255 (mask pixels)

def count_255(mask):
    i=0
    for row in range(mask.shape[0]):
        for col in range(mask.shape[1]):
            if mask[row,col]==255:
                i+=1
    return i

# Function that return:
# samples from each fake image

def sample_fake(img, mask):
    kernel_size=64
    stride=32

    samples=[]

    for y_start in range(0, img.shape[0]-kernel_size+1, stride):
        for x_start in range(0, img.shape[1]-kernel_size+1, stride):

            c_255=count_255(mask[y_start:y_start+kernel_size,
x_start:x_start+kernel_size])

            if (c_255>th) and (kernel_size*kernel_size-c_255>th):
                samples.append(img[y_start:y_start+kernel_size,
x_start:x_start+kernel_size, :3])

    return samples
```

Sampling The Original Images

```
def sample_random(img, num_samples):
    kernel_size=64
    stride=32

    x_start=0
    y_start=0
    samples=[]

    for y_start in range(0, img.shape[0] - kernel_size + 1, stride):
        for x_start in range(0, img.shape[1] - kernel_size + 1, stride):

            samples.append(img[y_start:y_start + kernel_size,
x_start:x_start + kernel_size, :3])

    indices=np.random.randint(0, len(samples), min(len(samples),
num_samples))

    sampled=[]
```

```
    for i in indices:
        sampled.append(samples[i])

    return sampled
```

A.3. Object Detection

label generator

```
mask_dir = "data/masks"
label_dir = "data/labels"
class_id = 0


if not os.path.exists(label_dir):
    os.makedirs(label_dir)

for i, mask_file in enumerate( os.listdir(mask_dir)):
    mask_path = os.path.join(mask_dir, mask_file)
    mask_img = cv2.imread(mask_path)

    # Convert the mask image into binary format
    gray_img = cv2.cvtColor(mask_img, cv2.COLOR_BGR2GRAY)
    binary_img = cv2.threshold(gray_img, 50, 255, cv2.THRESH_BINARY)[1]

    # Find the contours of the forged region in the binary image
    contours, hierarchy = cv2.findContours(binary_img, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    x, y, w, h = cv2.boundingRect(contours[0])

    # Calculate the normalized bounding box coordinates
    img_height, img_width, _ = mask_img.shape
    x_center = (x + w / 2) / img_width
    y_center = (y + h / 2) / img_height
    width = w / img_width
    height = h / img_height

    # Write the YOLO format label file
    label_path = os.path.join(label_dir, os.path.splitext(mask_file)[0] +
".txt")
    with open(label_path, "w") as f:
        f.write("{} {:.6f} {:.6f} {:.6f} {:.6f}".format(class_id, x_center,
y_center, width, height))

    if i % 1000 == 0 :
        print(f"{i} labels are generated")
```

check labels

```
min_width_height = 0.0099
label_dir = "data/labels"
image_dir = "data/images"
counter1 = 0
counter2 = 0
```

```
for file_path in  glob.glob(os.path.join(label_dir, '*.txt')):
    with open(file_path, 'r') as f:
        content = f.read()
        class_id, x_center, y_center, width, height = map(float,
content.split())

    if width < min_width_height and height < min_width_height:
        try:
            os.remove(file_path)
            counter1 +=1
        except PermissionError:
            print(f'Could not remove file: {file_path}. File is currently
in use.')
            continue
print(f'\n\n{counter1} labels deleted!')



# Get a list of all image files in the image directory
image_files = [os.path.splitext(file_name)[0] for file_name in
os.listdir(image_dir)]

# Get a list of all label files in the label directory
label_files = [os.path.splitext(file_name)[0] for file_name in
os.listdir(label_dir)]


# Loop over all image files and delete those whose names don't exist in the
label directory
for image_file in image_files:
    if image_file not in label_files:
        image_path = os.path.join(image_dir, image_file + ".jpg")
        try:
            os.remove(image_path)
            counter2 +=1
        except PermissionError:
            print(f'Could not remove file: {file_path}. File is currently
in use.')
            continue

print(f'\n\n{counter2} images deleted!')
```