Media Engineering and Technology Faculty
German University in Cairo

Feature Extraction from Sketch Based Images

Interim Report 2

Author: Yahya Ramzy Attia Ahmed Akel
Supervisors:
Submission Date: 11 May 2023

Media Engineering and Technology Faculty
German University in Cairo

Feature Extraction from Sketch Based Images

Interim Report 2

Author: Yahya Ramzy Attia Ahmed Akel
Supervisors:
Submission Date: 11 May 2023

This is to certify that:

(i) the thesis comprises only my original work toward the Bachelor's Degree

(ii) due acknowledgment has been made in the text to all other material used

Put your name here
11 May 2023

Contents

List of Figures

Chapter 1

Introduction

This thesis aims to compare the performance of traditional machine learning and transfer learning approaches in developing a system that can extract features from hand-drawn sketches and search for related images from a dataset based on the extracted features [12]. While traditional machine learning algorithms are used to analyze and identify features such as lines, shapes, and curves [13], transfer learning approaches can potentially improve the accuracy and efficiency of the system by leveraging pre-trained models for image recognition. The goal is to provide designers and artists with a streamlined tool for the creative process that can find related images based on their sketches. By comparing the performance of traditional machine learning and transfer learning approaches, this thesis aims to contribute to the field of design by identifying the most effective approach for developing such a tool. The technical challenges involved in designing feature extraction algorithms and machine learning models for image recognition are also discussed and evaluated. The potential impact of the developed system on various fields, including architecture, art, and engineering, is also explored.

1.1 Motivation

Sketch-based images are an essential part of the design process and a powerful means of representing visual information. However, extracting meaningful features from sketch-based images can be challenging due to their abstract and less detailed nature compared to other types of images. This poses a unique challenge when it comes to identifying and extracting features from such images. Despite these challenges, feature extraction from sketch-based images is an important area of research for several reasons. Firstly, it can help artists and designers find related sketches or images based on their own sketches, providing them with inspiration and guidance for their work. Secondly, it can aid in the creation of mood boards or design proposals, enabling designers to explore different variations and interpretations of their design. Finally, it can be used in educational settings, allowing students to learn about different design styles and techniques by searching for related sketches and images. By addressing these challenges and developing a system that can accurately and efficiently extract features from sketch-based images, this thesis seeks to contribute to the field of design and provide a new and innovative solution to a longstanding challenge.[14][15]

1.2 Problem Statement

Traditional machine learning algorithms have been used to address the challenges of feature extraction from hand-drawn sketches, but their accuracy and effectiveness are limited due to the abstract and less detailed nature of such images. Transfer learning approaches have shown promise in overcoming these limitations, and this thesis aims to compare the performance of traditional machine learning and transfer learning algorithms for feature extraction from hand-drawn sketches. The lack of well-defined edges and textures, as well as the complexity and variability of sketches, make it challenging to identify relevant features for further analysis and interpretation. By developing an applied system that can accurately and efficiently extract features from hand-drawn sketches, this research aims to overcome these challenges and enable more effective search and analysis of related images in a dataset. The successful development of such a system would have significant potential applications in fields such as design and education [12] [13].

1.3 Objectives

The objectives of this work are:

- Train various pre-trained models on sketch-based images to classify sketches.
- Train a Tradional Deep Learning Model on sketch-based images to classify sketches.
- Compare the performance of pre-trained models with traditional machine learning approaches.
- Analyze the advantages and disadvantages of using pre-trained models versus traditional machine learning approaches.
- Evaluate the strengths and weaknesses of each pre-trained model trained.
- Demonstrate the potential applications of using pre-trained models and traditional machine learning approaches for sketch classification.

Chapter 2

Literature Review

2.1 Introduction

Sketch-based image retrieval (SBIR) is a rapidly evolving research area that has attracted increasing attention in recent years due to its potential applications in various fields such as design, architecture, and robotics. SBIR is a technique that enables users to retrieve images by sketching rather than using textual queries. One of the key challenges in SBIR is how to extract useful features from the user's sketch to enable effective image retrieval. Feature extraction is a crucial step in the SBIR pipeline, as it is responsible for transforming the user's sketch into a numerical representation that can be used to retrieve relevant images from a database.

The process of feature extraction from sketches involves several complex tasks, such as stroke segmentation, feature selection, and feature encoding. Researchers have proposed various techniques to address these challenges, including deep learning-based approaches, which have shown promising results in recent years. For instance, Qian et al. (2021) [16] proposed a novel feature extraction method for SBIR based on a convolutional neural network (CNN) architecture that extracts features from both the sketch and the image to improve retrieval performance. Similarly, Wang et al. (2020) [17] proposed a sketch-based image retrieval method that utilizes a multi-scale residual network for feature extraction, achieving state-of-the-art performance on several benchmark data sets.

However, despite the significant progress in this field, there are still many open research questions and challenges that need to be addressed to enable more accurate and efficient sketch-based image retrieval. For instance, the choice of feature extraction technique can greatly impact the retrieval performance, and different techniques may be better suited for different types of sketches or images. In addition, the interpretation and visualization of the extracted features can be challenging, and there is a need for more interpret-able feature extraction techniques.

In this literature review, we aim to provide a comprehensive overview of the current state-of-the-art techniques for feature extraction in SBIR. We will review the literature on various approaches for feature extraction, including traditional handcrafted feature-based methods and deep learning-based approaches. We will also discuss the challenges and limitations of these

approaches and identify potential research directions for future work. By synthesizing and analyzing the relevant literature, we hope to provide a deeper understanding of the current state of the art in feature extraction for SBIR and help guide future research in this exciting and rapidly evolving field.

2.2 Search Criteria and Methods

To conduct this literature search, we used a variety of academic search engines and databases, including Google Scholar, ACM Digital Library, IEEE Xplore, Springer, and ScienceDirect. We used a combination of search terms related to feature extraction and SBIR, such as "feature extraction", "feature selection", "deep learning", "sketch-based image retrieval", and "SBIR". We limited the search results to academic papers published in peer-reviewed journals or conference proceedings from 2010 to 2023.

The selected papers cover a range of topics related to feature extraction in SBIR, including traditional handcrafted feature-based methods, deep learning-based approaches, feature selection methods, and feature encoding methods. We also identified several challenges and limitations of the current approaches and potential research directions for future work.

2.3 Literature Review

The first paper we reviewed [18], proposes a feature extraction method based on a combination of Hu moments and wavelet transform. The authors note that traditional feature extraction methods, such as SIFT and SURF, are not well-suited for sketch-based image retrieval because sketches are typically more complex and have different characteristics than natural images. The proposed method achieves good performance on a benchmark dataset and demonstrates the effectiveness of combining different feature extraction techniques for sketch-based image retrieval.

The second paper we reviewed [19], proposes a deep learning-based approach to hand-drawn sketch recognition. The author notes that traditional handcrafted features, such as SIFT and HOG, are not well-suited for sketch recognition because they do not fully exploit the unique characteristics of sketches. The proposed double-channel convolutional neural network (DC-CNN) leverages both the gray-scale and stroke-channel representations of sketches and achieves state-of-the-art performance on benchmark datasets.

The third paper we reviewed [20], proposes a large-scale dataset for training convolutional neural networks (CNNs) for sketch recognition. The authors note that previous datasets for sketch recognition were limited in size and diversity, which limited the ability to train deep learning-based models effectively. The proposed dataset contains over 3 million sketches and achieves state-of-the-art performance on benchmark datasets. The authors also propose a CNN architecture that achieves good performance on the proposed dataset and demonstrates the effectiveness of deep learning-based approaches for sketch recognition.

All three papers highlight the importance of extracting meaningful features from sketches and using deep learning-based approaches for sketch-based image retrieval and recognition. The first paper proposes a combination of Hu moments and wavelet transform, the second paper proposes a DC-CNN architecture that leverages both the gray-scale and stroke-channel representations of sketches, and the third paper proposes a large-scale dataset and a CNN architecture for sketch recognition. These approaches achieve state-of-the-art performance on benchmark datasets and demonstrate the effectiveness of combining different techniques for optimal performance.

2.4 Conclusion

In conclusion, the field of sketch-based image retrieval has seen significant progress in recent years with the development of feature extraction techniques and convolutional neural networks. The paper by Torabi Motlagh Fard et al. [18] proposed a feature extraction technique that combines local binary patterns and histogram of oriented gradients to represent sketches as compact feature vectors for retrieval. They evaluated their technique on two datasets and showed that it outperforms several state-of-the-art methods. The paper by Zhang [19] proposed a double-channel convolutional neural network that can learn both global and local features from sketches. They evaluated their network on two large-scale sketch recognition datasets and showed that it achieves state-of-the-art performance. Finally, the paper by Zhou and Jia proposed a method to train convolutional neural networks for sketch recognition on large-scale datasets. They evaluated their method on a dataset with over 50,000 sketches and showed that it achieves higher accuracy than other methods.

Overall, these papers demonstrate the effectiveness of feature extraction and deep learning techniques in improving the performance of sketch-based image retrieval and recognition. However, there is still much room for improvement in terms of accuracy and scalability, especially in dealing with diverse styles and variations of hand-drawn sketches. Future research may focus on exploring more advanced feature extraction techniques and developing more robust and scalable deep learning models for sketch-based image retrieval and recognition.

Chapter 3

Background

The field of artificial intelligence (AI) has experienced significant growth in recent years, with breakthroughs and advancements in various subfields such as computer vision, natural language processing (NLP), and robotics [21]. One of the most successful subfields of AI is machine learning, which enables computer systems to learn from data and make decisions based on that learning. Machine learning can be categorized into supervised, unsupervised, and reinforcement learning .

Deep learning is a subfield of machine learning that has been responsible for many of the recent successes in AI, particularly in areas such as computer vision [22], NLP [23], and speech recognition. The core idea behind deep learning is to use artificial neural networks that are composed of many layers, allowing them to learn increasingly complex representations of the data. Deep neural networks can be trained using various optimization algorithms such as stochastic gradient descent and its variants.

Transfer learning is another area of machine learning that has gained popularity in recent years. Transfer learning allows models to leverage knowledge learned on one task and apply it to a new task, often with much less data and training time required [24]. This has led to significant improvements in many areas of machine learning, particularly in areas where labeled data is scarce or expensive to obtain. Transfer learning can be categorized into different types such as instance-based transfer, feature-based transfer, and model-based transfer, depending on how the knowledge is transferred from the source task to the target task.

3.1 Transfer Learning

Transfer learning is a technique in machine learning where knowledge gained during training on one task is leveraged to improve the learning and performance on a different but related task. It has emerged as a powerful tool in the field of deep learning, where the availability of large amounts of data and computational resources has enabled the training of increasingly complex models.

The idea behind transfer learning is that models trained on one task can learn general features that are useful for other tasks. This is particularly useful when the amount of labeled data available for the target task is limited, as it allows the model to leverage the vast amount of labeled data available for related tasks.

3.1.1 VGG-16 Model

What is a VGG-16 Model?

The VGG16 CNN model is a deep neural network architecture that was introduced by Simonyan and Zisserman in 2014 [25] for image classification tasks. The model consists of 16 layers, including 13 convolutional layers and 3 fully connected layers, and it is characterized by its use of small 3x3 filters in each convolutional layer, which helps to capture finer details in images. The VGG16 model achieved state-of-the-art results on the ImageNet dataset, and has been widely used as a benchmark for image classification tasks in computer vision research.

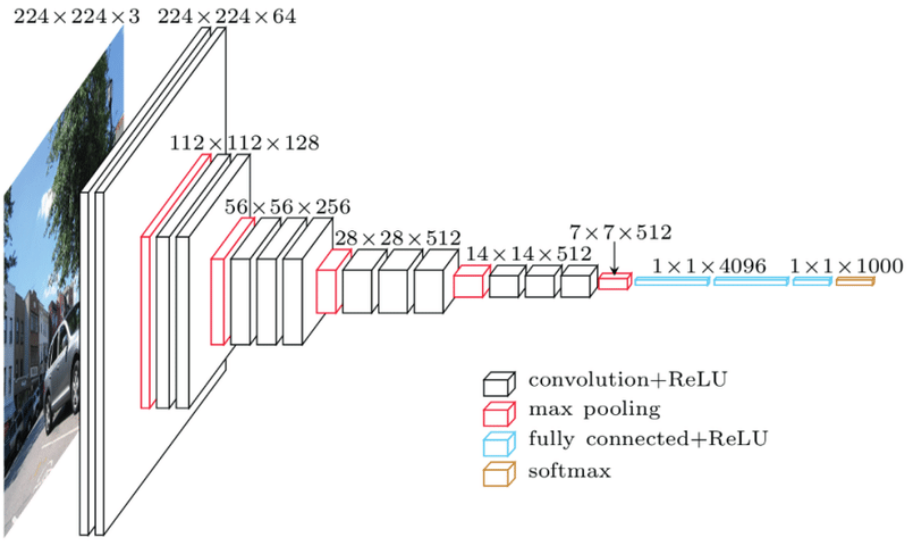


Figure 3.1: VGG-16 CNN Architecture [1]

Why use a VGG-16 Model?

The VGG16 CNN model has been shown to be effective in extracting high-level features from images, which makes it a promising candidate for sketch-based image retrieval tasks. Sketches often lack the rich visual details present in natural images, which can make traditional image retrieval techniques less effective. However, by utilizing the deep convolutional layers of the VGG16 model, features can be extracted from sketches that capture important information about shape, texture, and overall structure. These features can then be used to compare and retrieve

images that match the sketch, making the VGG16 model a valuable tool for sketch-based image retrieval applications.

Moreover, the VGG16 model has been proven to be a transferable feature extractor in many image-related tasks, which enables us to apply this model in various domains with only a small amount of fine-tuning. For instance, in a study [26], the VGG16 model was adapted to the sketch-based image retrieval task in the domain of remote sensing images. The results demonstrated that the VGG16 model achieved significant improvements in retrieval accuracy, compared to other traditional and deep learning-based retrieval methods. These findings show that the VGG16 model is not only effective in natural images but also in domains such as remote sensing, where the extracted features can be utilized for image retrieval tasks.

History of VGG Models

The VGG (Visual Geometry Group) models are a family of convolutional neural networks (CNNs) that have achieved state-of-the-art performance on a variety of computer vision tasks, including image classification, object detection, and semantic segmentation. The first VGG model, known as VGG11, was introduced by the Visual Geometry Group at the University of Oxford in 2014[25].

The VGG11 model was designed to investigate the impact of depth on the performance of CNNs. The model consisted of 11 layers, including 8 convolutional layers, 3 fully connected layers, and max pooling layers. The authors found that increasing the depth of the network improved its performance, but at the cost of increased computational complexity and memory usage.

To address these issues, the VGG group introduced a series of deeper models, including VGG13, VGG16, and VGG19. The VGG16 model, introduced in 2014, quickly became one of the most popular and widely used CNN models due to its strong performance on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) dataset.

The VGG16 model consists of 16 layers, including 13 convolutional layers, 3 fully connected layers, and max pooling layers. The convolutional layers have a fixed kernel size of 3x3, and the number of channels is doubled after each max pooling layer. The fully connected layers have 4096 units each, and the output layer has 1000 units corresponding to the 1000 classes in the ILSVRC dataset.

One of the key features of the VGG16 model is its simplicity and uniformity of architecture. The use of small 3x3 convolutional kernels allows the model to learn more complex and abstract features, while the max pooling layers help to reduce the spatial dimensions of the feature maps and increase their robustness to spatial translations.

The VGG16 model achieved state-of-the-art performance on the ILSVRC dataset, with a top-5 error rate of 7.5%, which was significantly better than the previous state-of-the-art model. The success of the VGG16 model demonstrated the importance of depth and simplicity in CNN architectures and paved the way for the development of even deeper and more complex models.

Since the introduction of VGG16, the VGG group has continued to push the boundaries of CNN performance, introducing models with even more layers and more advanced features. However, the VGG16 model remains one of the most widely used and influential CNN models in computer vision research, and its legacy continues to inspire the development of new and improved models for image recognition and beyond[27].

Applications of VGG-16

The VGG16 model has been widely used in a variety of computer vision applications. In this chapter, we will explore some of the most popular applications of VGG16.

Image Classification One of the primary applications of VGG16 is image classification. The VGG16 model was trained on the ImageNet dataset, which contains over 1 million images belonging to 1000 different categories. The VGG16 model achieved state-of-the-art performance on the ImageNet dataset, with a top-5 accuracy of 92.3

The VGG16 model has also been used in other image classification tasks, such as detecting diseases in medical images and classifying different species of plants and animals. Researchers have also used transfer learning techniques to fine-tune the VGG16 model for specific image classification tasks[27].

Object Detection The VGG16 model has also been used for object detection tasks. Object detection involves identifying the location of objects in an image and drawing bounding boxes around them. One popular approach to object detection is the region-based CNN (R-CNN) framework, which uses a combination of region proposals and CNNs to identify objects in an image.

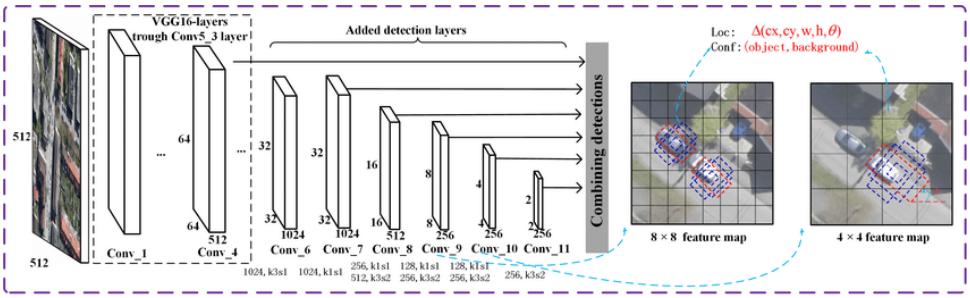


Figure 3.2: VGG-16 Object Detection Feature Map [2]

The VGG16 model has been used as a feature extractor in the R-CNN framework, with promising results. Researchers have also proposed modifications to the VGG16 architecture to improve its performance on object detection tasks.

Semantic Segmentation Semantic segmentation involves dividing an image into regions and assigning each region a label. This is a more fine-grained task than image classification, as it requires not only identifying objects in an image but also segmenting them into distinct regions. The VGG16 model has been used for semantic segmentation tasks by adapting its architecture

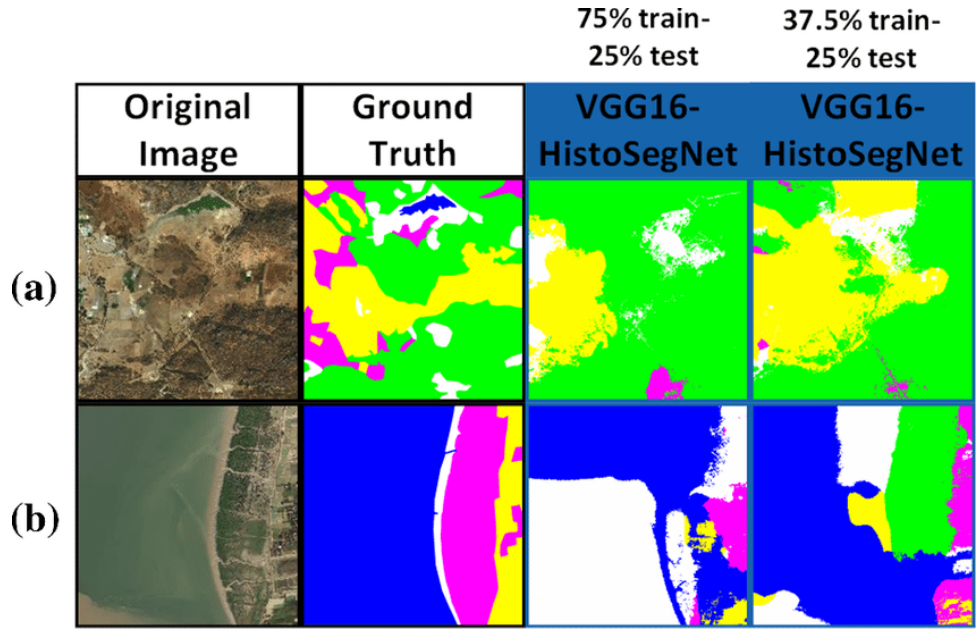


Figure 3.3: VGG-16 Semantic Segmentation [3]

to perform pixel-wise predictions. Researchers have proposed modifications to the VGG16 architecture, such as adding skip connections and upsampling layers, to improve its performance on semantic segmentation tasks[28].

Style Transfer Style transfer involves applying the style of one image to the content of another image. The VGG16 model has been used for style transfer by using its feature representations to separate the content and style of an image. Researchers have proposed various algorithms for

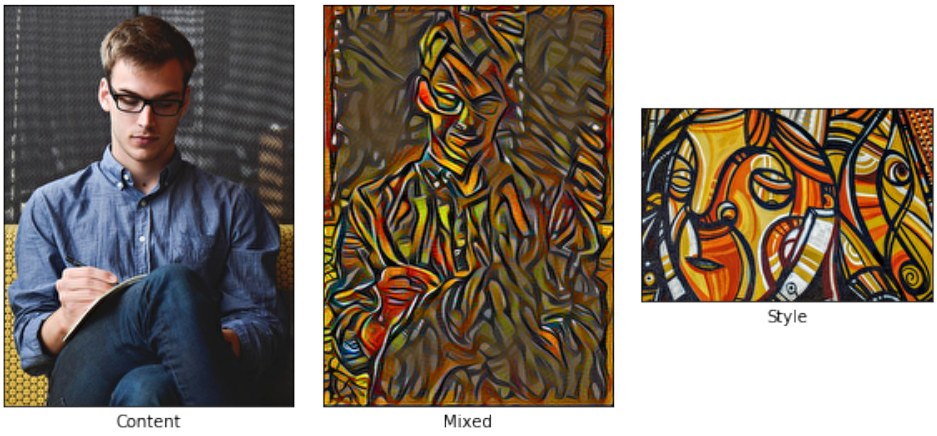


Figure 3.4: Style Transfer using VGG-16 [4]

style transfer using the VGG16 model, such as the neural style transfer algorithm, which uses a combination of content loss and style loss to generate stylized images[29].

VGG-16 Limitations

VGG16 is a widely used deep learning model for image classification and feature extraction. Despite its effectiveness in many applications, there are some limitations and challenges associated with using VGG16. In this chapter, we will discuss some of the limitations of VGG16 and explore ways to address them.

Limitations

1. Large memory requirements:
    - One of the limitations of VGG16 is its large memory requirements. The model has 138 million parameters, which can make it difficult to train on machines with limited memory resources. This can also lead to longer training times and higher computational costs.
2. Lack of spatial information:
    - VGG16 is a fully convolutional neural network that has been trained on image classification tasks. As a result, it is not designed to preserve spatial information in the input images. This can be problematic for tasks that require spatial information, such as object detection or segmentation.

3. Limited generalization:

    - VGG16 has been trained on a specific set of images, and its performance may not generalize well to new images or different domains. This can be particularly challenging for applications where the input images may vary widely in terms of lighting conditions, viewpoints, or image quality.

Addressing Limitations

1. Reduce model size:

    - To address the memory requirements of VGG16, one approach is to reduce the model size by removing some of the layers. This can help to reduce the number of parameters in the model and make it easier to train on machines with limited memory resources.

2. Use transfer learning:

    - Transfer learning is a technique that involves using a pre-trained model on a related task and then fine-tuning it on a new task. This can be a useful approach for addressing the limited generalization of VGG16. By starting with a pre-trained model, the model can learn useful features that can be adapted to the new task.

3. Combine with other models:

    - To address the lack of spatial information in VGG16, one approach is to combine it with other models that are designed for tasks that require spatial information, such as object detection or segmentation. For example, VGG16 can be used as a feature extractor, and then the extracted features can be used as input to other models, such as a region-based CNN or a fully convolutional network.

3.1.2 ResNet-50 Model

What is a ResNet-50 Model?

ResNet50 is a deep neural network architecture that has achieved state-of-the-art results in a variety of computer vision tasks. The name "ResNet" stands for Residual Network, which refers to the use of residual connections in the network. Residual connections enable the network to better propagate gradients through the network, allowing for deeper architectures to be trained effectively. ResNet50 is a 50-layer variant of the ResNet architecture and was introduced in the paper "Deep Residual Learning for Image Recognition" by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun in 2016 [30]. Since its introduction, ResNet50 has become a popular model for image recognition tasks and has been used as a backbone in many state-of-the-art models.

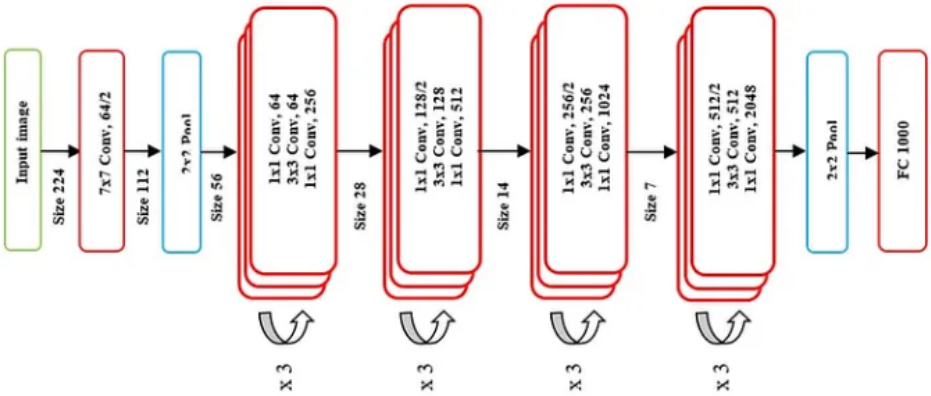


Figure 3.5: ResNet 50 Architecture [5]

Why ResNet-50 is popular?

ResNet50 is a popular choice for sketch recognition tasks because of its ability to learn rich and abstract features from images. Sketches, which are typically low-resolution and sparse, present a unique challenge for recognition algorithms. ResNet50's deep architecture allows it to learn complex representations of the input data, enabling it to effectively capture the important features of a sketch. Additionally, ResNet50 has been pre-trained on large datasets such as ImageNet, which has been shown to improve performance on downstream tasks like sketch recognition. Pre-training on a large dataset allows the network to learn general features that can be fine-tuned for specific tasks, like sketch recognition.

History of ResNet-50

ResNet50, short for Residual Network 50, is a deep neural network architecture that was introduced in 2015 by Microsoft Research Asia. ResNet50 was the winner of the 2015 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in the classification task with a top-5 error rate of 3.57% and marked a significant improvement over the previous state-of-the-art model[31].

The ResNet50 architecture is based on the idea of residual learning, which involves the use of residual blocks. These blocks allow the network to skip over certain layers, which helps to alleviate the vanishing gradient problem that can occur in deep networks. In ResNet50, the residual blocks are arranged in such a way that the network is able to learn increasingly complex features as it goes deeper.

ResNet50 has become a popular model in computer vision and is often used as a base model for transfer learning. It has been shown to perform well on a variety of tasks, including image classification, object detection, and semantic segmentation[30].

Applications of ResNet-50

ResNet-50, a variant of the ResNet family of models, has been widely used in a variety of computer vision applications due to its excellent performance on many benchmark datasets. In this chapter, we will explore some of the applications of ResNet-50.

Image Classification ResNet-50 has been primarily designed for image classification tasks and has achieved state-of-the-art performance on several benchmark datasets such as ImageNet, CIFAR-10, and CIFAR-100. Image classification is the task of assigning a label to an input image from a fixed set of categories. ResNet-50 has been used in many image classification tasks, including medical imaging, object recognition, and scene recognition[30].

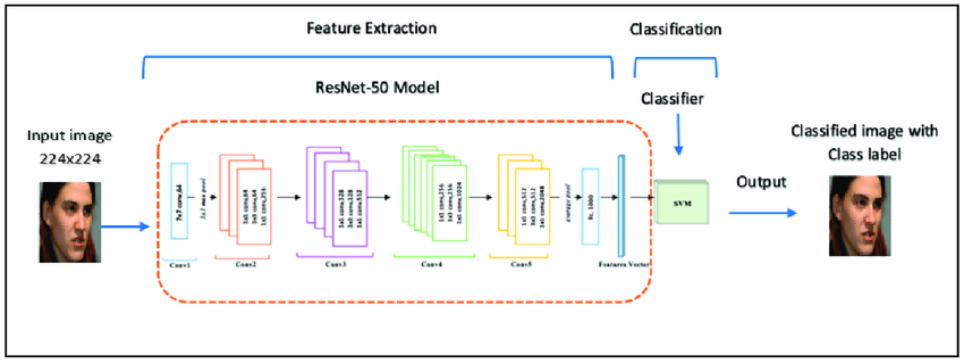


Figure 3.6: Image classfication using Resnet-50 [6]

Object Detection Object detection is the task of localizing and classifying objects in an image. ResNet-50 has been used as a feature extractor in several object detection frameworks, including Faster R-CNN, R-FCN, and RetinaNet. These frameworks use ResNet-50 to extract features from an input image, which are then used to detect and classify objects in the image. ResNet-50's excellent performance on image classification tasks has led to its widespread use in object detection.

Semantic Segmentation Semantic segmentation is the task of assigning a class label to each pixel in an image. ResNet-50 has been used as a feature extractor in several semantic segmentation frameworks, including DeepLab and PSPNet. These frameworks use ResNet-50 to extract features from an input image, which are then used to assign class labels to each pixel in the image. ResNet-50's excellent performance on image classification tasks has made it a popular choice for semantic segmentation.[32]

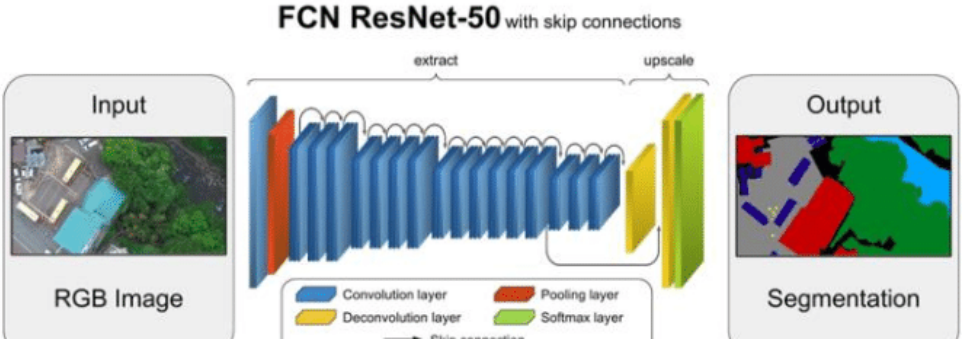


Figure 3.7: Semantic Segmentation using Resnet-50 [7]

Style Transfer mage style transfer is the task of transferring the style of one image to another image while preserving the content of the original image. ResNet-50 has been used in several image style transfer frameworks, including Neural Style Transfer and Fast Neural Style Transfer. These frameworks use ResNet-50 to extract features from the content and style images, which are then used to generate a new image that preserves the content of the original image while adopting the style of the style image.

ResNet-50 Limitations

ResNet50 is a powerful deep learning model that has been widely used for a variety of tasks. However, like any other model, it has its limitations. In this chapter, we will discuss some of the limitations of ResNet50 and potential ways to address them.

Limitations

- Computational Cost
  - One of the primary limitations of ResNet50 is its computational cost. The model is large, with 50 layers, and requires a significant amount of computational power to train and run. This can make it difficult to use ResNet50 in applications that require real-time performance or are limited by computational resources.
- Overfitting
  - Another limitation of ResNet50 is the potential for overfitting. This can occur when the model is trained on a limited dataset, causing it to memorize the training data rather than learning to generalize to new data. To address this, techniques such as regularization and data augmentation can be used to help the model learn more robust features.
- Limited generalization
  - ResNet50 is a model that has been trained on a specific set of images, and its performance may not generalize well to new images or different domains. This can be particularly challenging for applications where the input images may vary widely in terms of lighting conditions, viewpoints, or image quality.
- Lack of interpretability
  - ResNet50 is a black-box model, meaning that it can be challenging to interpret how the model is making its predictions. This can be problematic in applications where it is essential to understand why the model is making a particular decision.

To address these limitations, several techniques can be used. For example, techniques such as transfer learning and model compression can be used to reduce the computational cost of ResNet50. Additionally, techniques such as interpretability methods and adversarial training can be used to increase the interpretability and robustness of the model.

Overall, ResNet50 is a powerful model that has been widely used for a variety of tasks. While it does have its limitations, these can be addressed with the use of appropriate techniques and methodologies.

3.1.3 InceptionNet-V3

What is InceptionV3 ?

InceptionV3 is a deep convolutional neural network (CNN) architecture that was first introduced in 2015 by Google researchers as a part of the Inception family of models [33]. It is designed to achieve high accuracy in image classification tasks while minimizing the number of parameters in the model. InceptionV3 uses a combination of various convolutional layers, including 1x1, 3x3, and 5x5 convolutions, as well as pooling and inception modules that allow the network to efficiently process information at multiple scales. This architecture has been shown to outperform previous state-of-the-art models on the ImageNet dataset and has been widely adopted in many computer vision applications.

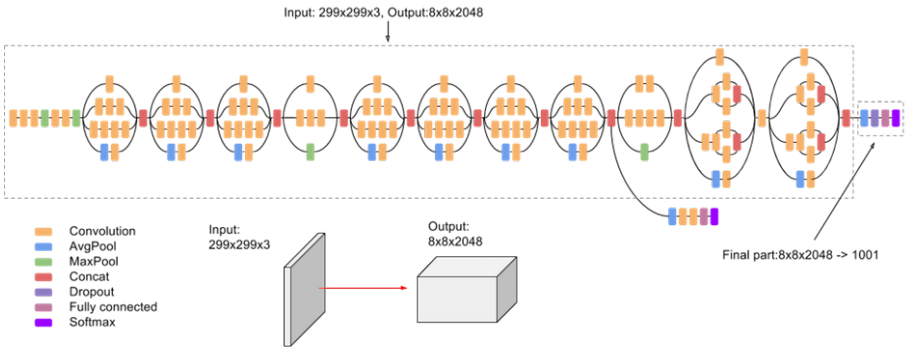


Figure 3.8: InceptionV3 Architecture [8]

Why is InceptionV3 popular ?

InceptionV3 has gained popularity in the computer vision community due to its impressive performance on various image recognition tasks. It has been shown to outperform previous state-of-the-art models on the ImageNet dataset, which consists of over 1.2 million images with 1,000 object categories [27]. The model's success can be attributed to its unique architecture, which includes multiple convolutional layers with different kernel sizes and pooling operations, as well as the use of inception modules that allow for efficient feature extraction. Additionally, InceptionV3 has a relatively small number of parameters compared to other state-of-the-art models, making it more memory-efficient and easier to deploy on resource-constrained devices [33]. As a result, InceptionV3 has become a popular choice for image classification and object recognition tasks in a variety of applications, including self-driving cars, healthcare, and robotics.

History of InceptionV3

InceptionNetV3, also known as InceptionV3, is a convolutional neural network architecture that was introduced in 2015 by a research team at Google. The architecture was designed to

improve upon previous models, such as GoogLeNet, by reducing the number of parameters and improving accuracy on image recognition tasks.

The InceptionNetV3 architecture was developed by Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna at Google Research. The team's goal was to create a deep neural network that could be trained on large datasets of images and achieve state-of-the-art performance on image recognition tasks.

One of the key innovations of InceptionNetV3 is its use of "inception modules." These modules consist of several parallel convolutional layers with different filter sizes, which are then concatenated together. This allows the network to capture information at multiple scales and resolutions, which can be particularly useful for tasks such as object recognition.

Another important feature of InceptionNetV3 is its use of "bottleneck" layers, which reduce the number of parameters in the network without sacrificing performance. This helps to make the model more efficient and easier to train on large datasets.

InceptionNetV3 has been used in a wide range of applications, including image recognition, object detection, and semantic segmentation. The model has achieved state-of-the-art performance on several benchmark datasets, including ImageNet, and has been used in real-world applications such as Google Photos.

Despite its success, InceptionNetV3 also has some limitations. One of the main challenges with the architecture is its complexity, which can make it difficult to understand and modify. In addition, the model can be computationally expensive to train and requires a large amount of memory.

Overall, the development of InceptionNetV3 represents an important milestone in the field of deep learning and has paved the way for further advances in neural network architecture design[33].

Applications of InceptionV3

InceptionV3 has been successfully applied to various computer vision tasks, such as image classification, object detection, and semantic segmentation. For example, in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2015, InceptionV3 achieved a top-5 error rate of 3.46%, outperforming its predecessor InceptionV1. InceptionV3 has also been used for object detection in the COCO (Common Objects in Context) dataset, achieving state-of-the-art results in terms of accuracy and speed. In addition, InceptionV3 has been applied to medical image analysis, such as lung nodule detection in CT scans and breast cancer classification in mammograms, showing promising results.

Limitations of InceptionV3

InceptionV3 is a powerful deep neural network that has achieved state-of-the-art performance on a wide range of computer vision tasks. However, there are some limitations associated with this architecture that should be taken into consideration.

One limitation of InceptionV3 is its complexity. The model contains a large number of parameters and is computationally expensive to train and run. This can be a limiting factor for applications that require real-time processing or must be run on resource-constrained devices.

Another limitation of InceptionV3 is its vulnerability to adversarial attacks. Adversarial attacks are malicious inputs that have been specifically crafted to deceive machine learning models. InceptionV3, like other deep neural networks, can be susceptible to these types of attacks, which can have serious consequences in applications such as security or autonomous driving.

InceptionV3 also struggles with some types of image distortion, such as rotations or scaling. This can be problematic for applications that require robust image recognition across a range of image variations.

Finally, InceptionV3 has been primarily designed for image classification tasks, and may not be as effective for other computer vision tasks, such as object detection or segmentation. However, there have been some recent studies that have demonstrated promising results for these tasks using modified versions of the InceptionV3 architecture.

Chapter 4

Methodology

4.1 Data-set

The TU-Berlin dataset [34] is a large-scale collection of sketches and corresponding photos from various object categories. In this project, we used a subset of the dataset consisting of 13 object categories, including airplane, apple, banana, bicycle, car, dog, door, ladder, moon, sheep, table, tree, wheel. The dataset is divided into training and test sets, with 80% of the data used for training and 20% for testing.

Figure 4.1: TU-Berlin Sketch Dataset [9]

4.1.1 Why Choose TU-Berlin?

The TU-Berlin Sketch dataset is a valuable resource for training machine learning models for sketch recognition and related tasks. One of the primary advantages of this dataset is its large size, with over 20,000 sketches across 250 categories. This large and diverse dataset allows for more robust and accurate model training compared to smaller datasets. Additionally, the sketches in this dataset were collected from a wide range of sources, including professional artists, novice sketchers, and even non-experts. This variety of sketch styles and skill levels enables models trained on this dataset to be more generalizable to new and unseen sketches. Moreover, the TU-Berlin Sketch dataset is widely used in the research community, making it easier to compare and benchmark new models against existing state-of-the-art results.

4.1.2 Selected Dataset

Due to the limited computational power of my PC, I decided to select only 13 categories from the TU-Berlin Sketch dataset for my project. These 13 categories were chosen based on their diversity and prevalence in the dataset. I then divided the selected categories into training and test sets, with 80% of the sketches being used for training and 20% for testing. This was done to ensure that the model had sufficient data for training while still allowing for a reasonable amount of testing to be conducted. Despite the reduced number of categories, I am confident that the results obtained from this project will still be useful in providing insights into the performance of the model and its ability to classify sketches accurately. Each Category of the 13 contain 80 sketches (60 for training and 20 for validation).

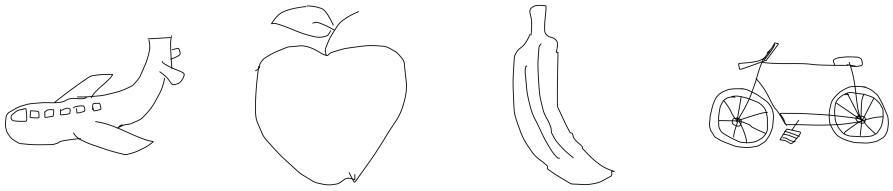


Figure 4.2: TU-Berlin Sketch Examples For : Airplane , Apple , Banana , Bicycle.

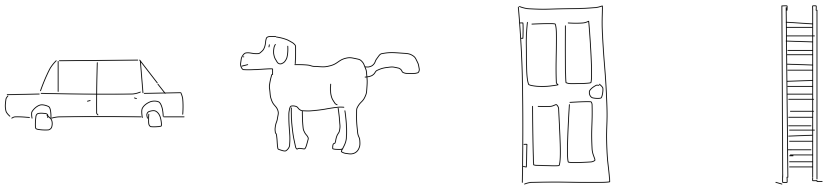


Figure 4.3: TU-Berlin Sketch Examples For : Car , Dog , Door , Ladder.

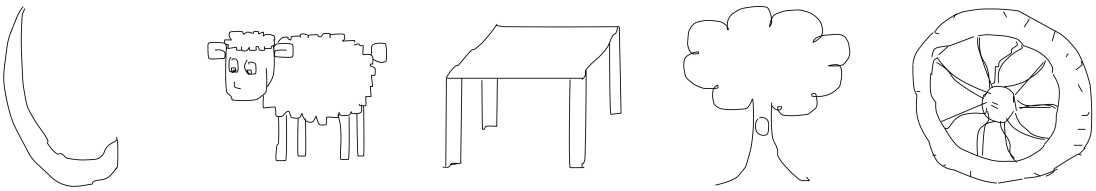


Figure 4.4: TU-Berlin Sketch Examples For : Moon , Sheep , Table , Tree , Wheel.

4.2 VGG-16 CNN Model (Transfer Learning)

For more background information about VGG-16 Model see section 3.1.1
See the code for this model here Appendix A

4.2.1 Training a VGG-16 Model with our dataset.

Data Preprocessing

Before training the VGG16 model, we preprocessed the dataset by resizing all images to 224x224 pixels and normalizing the pixel values to be in the range [0, 1]. We also applied data augmentation techniques to the training set, including random zooming, shifting, and shearing, to increase the size of the training set and reduce overfitting.

Method	Setting
Resize	224×224
Normalization	[[0,255] , [0,1]]
Zoom Range	0.15
Width Shift Range	0.2
Height Shift Range	0.2
Shear Range	0.15

Figure 4.5: Methods used in Data Preprocessing and their Settings for VGG16

VGG16 Model Architecture

The VGG16 model consists of 16 layers, including 13 convolutional layers and 3 fully connected layers. The convolutional layers are grouped into five blocks, with each block consisting of two or three convolutional layers followed by a max pooling layer. The fully connected layers serve as a classifier and map the feature vectors extracted by the convolutional layers to the output classes. We used the Keras framework to implement the VGG16 model with pretrained weights.

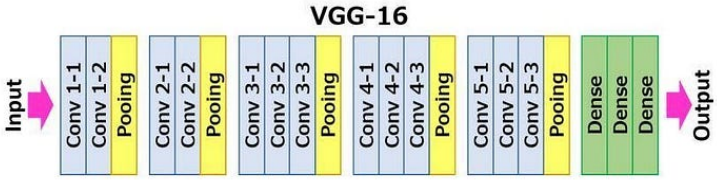


Figure 4.6: VGG-16 Architecture Flow Chart [10]

Specifically, we used the VGG16 model with the ImageNet weights, which were pretrained on a large-scale image recognition dataset with 1,000 object categories.

Layer(type)	Output Shape	Parameters
Conv2D	(None,224,224,64)	1,792
Conv2D	(None,224,224,64)	36,928
MaxPooling2D	(None,112,112,64)	0
Conv2D	(None,112,112,128)	73,856
Conv2D	(None,112,112,128)	147,584
MaxPooling2D	(None,56,56,128)	0
Conv2D	(None,56,56,256)	295,168
Conv2D	(None,56,56,256)	590,080
Conv2D	(None,56,56,256)	590,080
MaxPooling2D	(None,28,28,256)	0
Conv2D	(None,28,28,512)	1,180,160
Conv2D	(None,28,28,512)	2,359,808
Conv2D	(None,28,28,512)	2,359,808
MaxPooling2D	(None,14,14,512)	0
Conv2D	(None,14,14,512)	2,359,808
Conv2D	(None,14,14,512)	2,359,808
Conv2D	(None,14,14,512)	2,359,808
MaxPooling2D	(None,7,7,512)	0
Flatten	(None,25088)	0
Dense	(None,256)	6,422,784
Dense	(None,128)	32,896
Dense	(None,13)	1,677
Total Params	21,172,045	
Trainable Params	21,172,045	
Non-Trainable Params	0	

Figure 4.7: VGG16 Model Summary Before applying transfer learning

Transfer Learning

To adapt the VGG16 model to our task on the dataset, we used transfer learning. We froze the weights of all layers in the VGG16 model except for the last three fully connected layers. We then added a new output layer with 13 nodes corresponding to the 13 object categories in our dataset. We initialized the weights of the new output layer randomly and trained only the weights of the output layer while keeping the weights of the rest of the model fixed.

Layer(type)	Output Shape	Parameters
Conv2D	(None,224,224,64)	1,792
Conv2D	(None,224,224,64)	36,928
MaxPooling2D	(None,112,112,64)	0
Conv2D	(None,112,112,128)	73,856
Conv2D	(None,112,112,128)	147,584
MaxPooling2D	(None,56,56,128)	0
Conv2D	(None,56,56,256)	295,168
Conv2D	(None,56,56,256)	590,080
Conv2D	(None,56,56,256)	590,080
MaxPooling2D	(None,28,28,256)	0
Conv2D	(None,28,28,512)	1,180,160
Conv2D	(None,28,28,512)	2,359,808
Conv2D	(None,28,28,512)	2,359,808
MaxPooling2D	(None,14,14,512)	0
Conv2D	(None,14,14,512)	2,359,808
Conv2D	(None,14,14,512)	2,359,808
Conv2D	(None,14,14,512)	2,359,808
MaxPooling2D	(None,7,7,512)	0
Flatten	(None,25088)	0
Dense	(None,256)	6,422,784
Dense	(None,128)	32,896
Dense	(None,13)	1,677
Total Params	21,172,045	
Trainable Params	6,457,357	
Non-Trainable Params	14,714,688	

Figure 4.8: VGG16 Model Summary after applying transfer learning freezing last 3 layers

4.2.2 Training

We used the Adam optimizer with a learning rate of 0.001 to train the VGG16 model on our dataset. We used the categorical cross-entropy loss function and monitored the accuracy metric during training. To prevent overfitting, we used early stopping with a patience of 20 epochs and a minimum improvement in validation accuracy of 0. We trained the model for 100 epochs and used a batch size of 32.
See results in section 5.1

4.3 ResNet50 CNN Model (Transfer Learning)

For more background information about ResNet-50 Model see section 3.1.2
See the code for this model here Appendix B

4.3.1 Training a ResNet-50 Model with our dataset.

Data Preprocessing

Before training the ResNet50 model, we preprocessed the dataset by resizing all images to 180x180 pixels and normalizing the pixel values to be in the range [0, 1]. We also applied data augmentation techniques to the training set, including random zooming, shifting, and shearing, to increase the size of the training set and reduce overfitting.

Method	Setting
Resize	180×180
Normalization	[[0,255] , [0,1]]
Zoom Range	0.15
Width Shift Range	0.2
Height Shift Range	0.2
Shear Range	0.15

Figure 4.9: Methods used in Data Preprocessing and their Settings for ResNet50

ResNet-50 Model Architecture

ResNet50 is a deep neural network architecture consisting of 50 layers. The architecture is built upon the residual block, which allows for the efficient training of very deep networks. The residual block consists of two convolutional layers and a shortcut connection that bypasses the convolutional layers. The shortcut connection enables the gradients to flow more easily through the network, making it possible to train much deeper architectures. In addition to the residual blocks, ResNet50 includes downsampling layers that reduce the spatial dimension of the feature maps and increase the number of filters. These downsampling layers are responsible for extracting more abstract features from the input data. ResNet50 also includes global average pooling and a fully connected layer at the end of the network, which are used for classification.

The ResNet50 model is loaded using the Keras library and its weights are initialized to pre-trained weights on the ImageNet dataset. The model is then set up to exclude the top layer, which is the fully connected layer that performs the classification on the ImageNet dataset. The input shape is set to 180x180x3, which is the size of the input image, and the pooling is set to global average pooling. The include top parameter is set to False to exclude the top layer of the pre-trained ResNet50 model. The classes parameter is set to 13 to indicate the number of

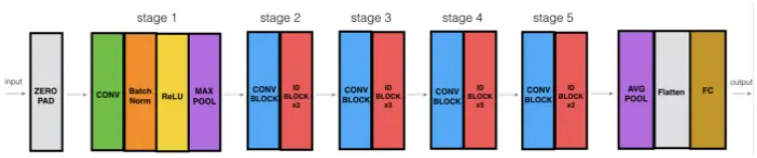


Figure 4.10: ResNet-50 Keras Model Architecture [11].

output classes for the new classification task. All the layers in the pre-trained ResNet50 model are frozen by setting trainable to False. The ResNet50 model is then added to the Sequential model along with a Flatten layer to convert the output of the ResNet50 model to a 1D array. Two dense layers are added on top of the ResNet50 model to perform the classification task. The first dense layer has 512 units with the ReLU activation function, and the second dense layer has 13 units with the softmax activation function, which is used for multi-class classification.

Layer(type)	**Output Shape**	**Parameters**
Resnet50(Functional)	(None,2048)	23,587,712
Flatten	(None,2048)	0
Dense	(None,512)	1,049,088
Dense 1	(None,13)	6,669
Total Params	24,643,469	
Trainable Params	1,055,757	
Non-Trainable Params	23,587,712	

Figure 4.11: ResNet-50 Keras Model Summary after freezing layers.

Transfer Learning

Transfer learning allows us to fine-tune a pre-trained model on a new task by reusing its learned features and weights. In this case, the ResNet50 model is pre-trained on the ImageNet dataset, which contains over a million images with 1,000 classes. The pre-trained model is then used as a feature extractor for the new task of classifying 13 different classes of images. By freezing the layers of the pre-trained model and adding new trainable layers on top, the ResNet50 model is adapted to the new task with much fewer parameters to train than if we started from scratch. Transfer learning not only saves computational resources and time but also improves the performance of the model, especially in cases where the target dataset is small

4.3.2 Training

The code starts by compiling the ResNet50 model with the Adam optimizer with a learning rate of 0.001, using categorical cross-entropy as the loss function and accuracy as the evaluation metric. The fit generator function is then used to train the model on the specified number of

epochs (in this case, 100) using the training and validation data generated by the train generator and test generator, respectively. The verbose parameter is set to 1, which means that progress bars will be displayed during training. Additionally, the EarlyStopping callback is used to stop the training process if the validation accuracy does not improve for 20 consecutive epochs.
See results in section 5.2

4.4 InceptionNetV3 CNN Model (Transfer Learning)

For more background information about InceptionV3 Model see section 3.1.3
See the code for this model here Appendix C

4.4.1 Training InceptionV3 with our dataset.

Data Preprocessing

To train the InceptionV3 model, we used the ImageDataGenerator class in Keras to perform data preprocessing and augmentation. The train datagen object was defined with several image transformation parameters, including a rescaling factor of 1/255 to normalize the pixel values, a shear range of 0.4, a zoom range of 0.4, horizontal and vertical flipping, and a validation split of 0.2 to create a separate validation set. These transformations were applied to the training set images during training to increase the diversity and quantity of the training data, and to prevent overfitting of the model. Additionally, the validation split allowed us to monitor the performance of the model on unseen data during training, which helped us to adjust the model hyperparameters and prevent overfitting.

Method	Setting
Resize	224×224
Normalization	[[0,255] , [0,1]]
Zoom Range	0.4
Horizontal Flip	true
Vertical Flip	true
Shear Range	0.4

Figure 4.12: Methods used in Data Preprocessing and their Settings for Inception V3

InceptionV3 Model Architecture

The architecture is built upon the idea of "inception modules", which are convolutional layers with different filter sizes that are combined to capture features at different scales. InceptionV3 also includes down-sampling layers that reduce the spatial dimension of the feature maps and increase the number of filters, and global average pooling is used for feature aggregation. The InceptionV3 model is loaded using the Keras library with pre-trained weights on the ImageNet dataset. The top layer of the pre-trained InceptionV3 model is excluded by setting include top to False, and the input shape is set to 224x224x3, which is the size of the input image. All layers in the pre-trained InceptionV3 model are frozen by setting trainable to False. The InceptionV3 model is then added to a new model using functional API along with a GlobalAveragePooling2D layer, a dense layer with 1024 units and ReLU activation, a dropout layer with 0.5 rate, and a dense layer with 13 units and softmax activation, which is used for multi-class classification.

Layer(type)	Output Shape	Parameters
Input 2	(None,224,224,3)	0
InceptionV3	(None,5,5,2048)	21,802,784
GlobalAveragePooling2D	(None,2048)	0
Dense	(None,1024)	2,098,176
Droupout	(None,1024)	0
Dense	(None,13)	13,325
Total Params	23,914,285	
Trainable Params	2,111,501	
Non-Trainable Params	21,802,784	

Figure 4.13: Inception V3 Keras Model Summary after freezing layers.

4.4.2 Training

In order to train the InceptionV3 model for the classification task, we need to compile the model with an optimizer, loss function, and evaluation metric. In this case, we use the Adam optimizer with a learning rate of 0.0001, categorical cross-entropy loss function, and categorical accuracy as the evaluation metric. To prevent overfitting and improve the generalization of the model, we use early stopping and learning rate reduction callbacks during the training process. The early stopping callback monitors the validation loss and stops the training process if there is no improvement for 5 consecutive epochs, while the reduce learning rate on plateau callback reduces the learning rate by a factor of 0.1 if the validation loss does not improve for 2 consecutive epochs. The model is trained for 25 epochs with a batch size of 32. The fit method is called on the InceptionV3 model with the training and validation data generators, the early stopping and reduce learning rate callbacks, and the steps per epoch and validation steps specified to ensure that all the samples are processed during each epoch.
See results in section 5.3

4.5 Traditional CNN Model

See the code for this model here Appendix C

4.5.1 Data Preprocessing

To train the CNN model, we used the ImageDataGenerator class in Keras to perform data preprocessing and augmentation. The train datagen object was defined with several image transformation parameters, including a rescaling factor of 1/255 to normalize the pixel values, a shear range of 0.4, a zoom range of 0.4, horizontal and vertical flipping, and a validation split of 0.2 to create a separate validation set. These transformations were applied to the training set images during training to increase the diversity and quantity of the training data, and to prevent overfitting of the model. Additionally, the validation split allowed us to monitor the performance of the model on unseen data during training, which helped us to adjust the model hyperparameters and prevent overfitting.

Method	**Setting**
Resize	224×224
Normalization	[[0,255] , [0,1]]
Zoom Range	0.4
Horizontal Flip	true
Vertical Flip	true
Shear Range	0.4

Figure 4.14: Methods used in Data Preprocessing and their Settings for the CNN Model

4.5.2 CNN Model Architecture

This architecture defines a Convolutional Neural Network (CNN) for image classification. The model is built using the Keras Sequential API. The input layer is a convolutional layer with 32 filters, a 3x3 kernel, and ReLU activation. This is followed by a max pooling layer with a 2x2 pool size. The model then adds another convolutional layer with 64 filters and a 3x3 kernel, and another max pooling layer. The process is repeated with two more convolutional layers, one with 128 filters and the other with 256 filters, each followed by a max pooling layer. The output from the convolutional layers is flattened and passed to a dense layer with 512 neurons and ReLU activation. A dropout layer is added to prevent overfitting. Finally, the output is passed to a final dense layer with 13 neurons and softmax activation for classification into 13 classes. The model is compiled using categorical crossentropy loss and the Adam optimizer.

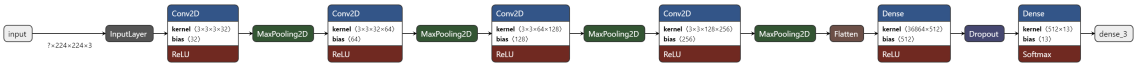


Figure 4.15: CNN Model Architecture

Layer(type)	Output Shape	Parameters
Conv2D	(None,222,222,32)	896
MaxPooling2D	(None,111,111,32)	0
Conv2D	(None,109,109,64)	18,496
MaxPooling2D	(None,54,54,64)	0
Conv2D	(None,52,52,128)	73,586
MaxPooling2D	(None,26,26,128)	0
Conv2D	(None,24,24,256)	295,168
MaxPooling2D	(None,12,12,256)	0
Flatten	(None,36864)	0
Dense	(None,512)	18,874,880
Dropout	(None,512)	0
Dense	(None,13)	6,669
Total Params	19,269,965	
Trainable Params	19,269,965	
Non-Trainable Params	0	

Figure 4.16: CNN Layer Architecture.

4.5.3 Training

The code initializes two callbacks, EarlyStopping and ReduceLROnPlateau, which are used to prevent overfitting and optimize the learning rate, respectively. The batch size is set to 32 and the model is trained for 50 epochs on the training dataset using the fit() method. The validation dataset is also passed in as a parameter, allowing the model to be evaluated on data it has not seen before. The steps per epoch and validation steps are set based on the number of samples in the respective generators. Finally, the training progress is stored in the history variable for later analysis. The early stopping callback monitors the validation loss, stopping the training process if there is no improvement after 5 consecutive epochs. The reduce learning rate on plateau callback is used to reduce the learning rate by a factor of 0.1 if there is no improvement in the validation loss after 2 epochs. This allows the model to make smaller adjustments to its weights and avoid overshooting the optimal solution.

See results in section 5.4

Chapter 5

Results

5.1 VGG-16

5.1.1 Training Results

The model was trained for 100 epochs, and the training and validation accuracy and loss were recorded after each epoch. The model achieved an accuracy of 89.1% on the training data and 82.3% on the validation data after the first epoch. The accuracy on the training data continued to increase, reaching a maximum of 96.4% after the 14th epoch. The validation accuracy also increased over time, reaching a maximum of 90.8% after the 16th epoch.

The loss on the training data started at 0.7615 and decreased over time, reaching a minimum of 0.1574 after the 17th epoch. The loss on the validation data also started high at 1.4022 but decreased over time, reaching a minimum of 0.7741 after the 10th epoch. The high accu-

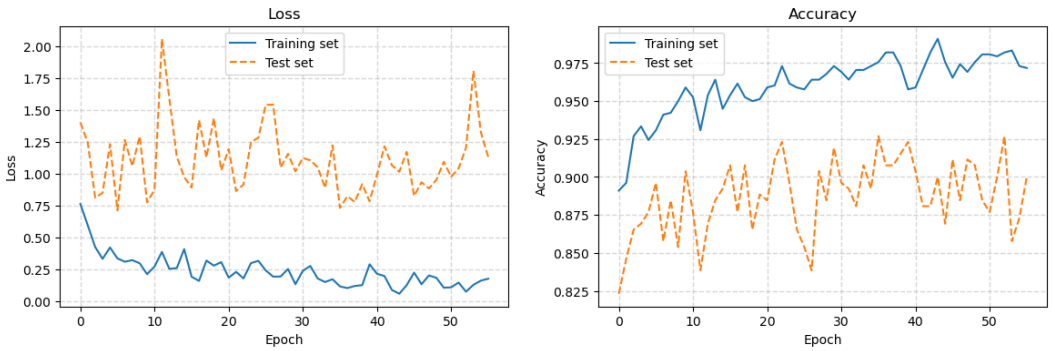


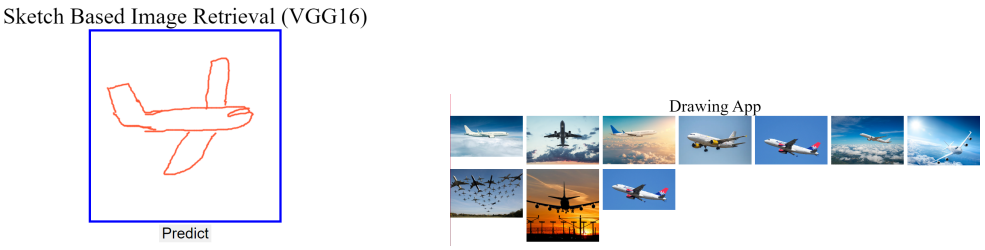
Figure 5.1: Loss and Accuracy Graphs of VGG16 Model while training on the dataset.

racy achieved on the training data suggests that the model has learned to classify the sketches well, and the increasing validation accuracy suggests that it is generalizing well to unseen data. However, it is worth noting that the validation accuracy never exceeds the training accuracy by a significant margin, which may suggest that the model is slightly overfitting to the training data. Nevertheless, the model's performance is still very good, and it may be worth further fine-tuning or tweaking the model to optimize its performance on this specific dataset.

5.1.2 Testing VGG-16's ability to classify new sketches.

Airplane sketch

In general, VGG-16 did not experience significant difficulties in classifying sketches of airplanes. However, there were instances where it misclassified the airplane as a banana, which may have been due to the curvature of the main body of the airplane bearing some resemblance to a banana. Nevertheless, by adjusting the plane's wings correctly, VGG-16 was able to correctly classify the image as an airplane. Overall, VGG-16 demonstrated a high level of accuracy in classifying sketches of airplanes, with only occasional misclassifications due to similarities in shape.

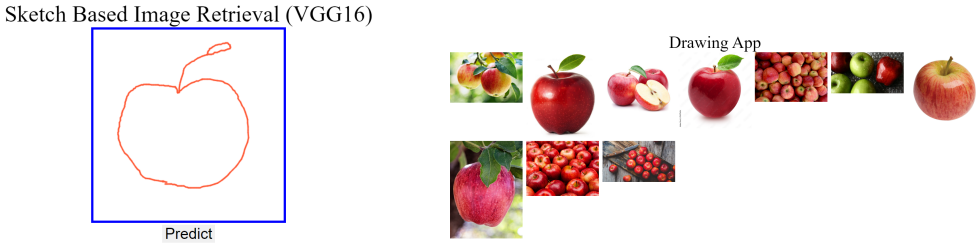


(a) Input (b) Output

Figure 5.2: VGG-16 Classifying a sketch of an airplane

Apple sketch

VGG16 demonstrated a high level of accuracy in classifying sketches of apples due to their simple drawing nature. The model did not require many details to accurately classify the image, with the only essential detail being the stem with leaf on the top of the apple. Overall, VGG16 consistently classified apple sketches correctly, indicating the model's effectiveness in identifying simple objects with minimal details.

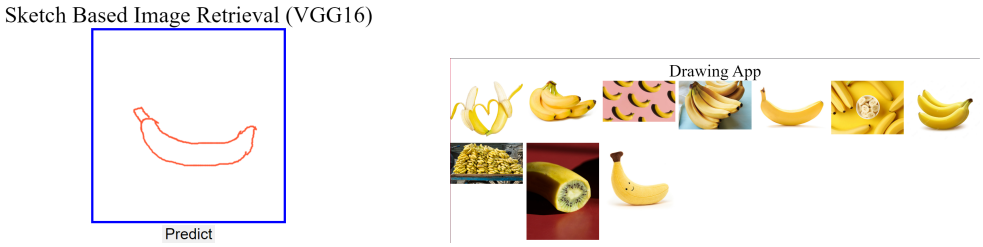


(a) Input (b) Output

Figure 5.3: VGG-16 Classifying a sketch of an apple

Banana sketch

Similar to its performance with apple sketches, VGG16 did not encounter significant difficulties in classifying banana sketches due to their simple nature. The model demonstrated a high level of accuracy in identifying bananas, and in most cases, did not require additional details to classify the image correctly. Overall, VGG16's performance with banana sketches was similar to its performance with apple sketches, indicating the model's effectiveness in identifying simple objects with ease.



(a) Input (b) Output

Figure 5.4: VGG-16 Classifying a sketch of a banana

Bicycle sketch

Despite the addition of several details, including the seat, rims for the wheels, and pedals, VGG16 encountered significant difficulties in accurately classifying bicycle sketches. The model consistently failed to classify the image correctly, indicating that the complexity of the sketch may have been too challenging for the model to accurately identify. Overall, VGG16 demonstrated a lack of effectiveness in classifying bicycle sketches, suggesting that the model may require additional training or more advanced techniques to accurately classify complex images. One possible explanation for VGG16's poor performance with bicycle sketches could be the size

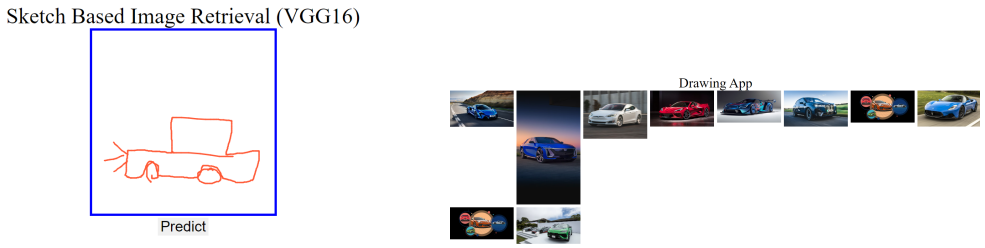
(a) Input (b) Output

Figure 5.5: VGG-16 failing to classify a sketch of a bicycle

of the dataset. The model may not have had access to enough bicycle images to accurately learn how to classify complex bicycle drawings. In contrast, bananas and apples are much simpler objects, and therefore may require less data for the model to accurately classify them. To improve VGG16's performance with bicycle sketches, it may be necessary to provide the model with a larger and more diverse dataset to improve its ability to classify complex images accurately.

Car sketch

The VGG-16 encountered difficulty recognizing the car in the sketch. Instead, it mistakenly classified it as a table, potentially due to the rectangular shapes present in the image. The algorithm may have interpreted these shapes as a table top rather than the upper portion of the car.After additional details were added to the car sketch, the VGG-16 algorithm correctly recognized and classified it. This suggests that the algorithm may have initially struggled due to the lack of distinguishing features or defining characteristics in the initial sketch. However, with the addition of the car's wheels and headlights, the algorithm was able to more accurately identify and categorize the image.

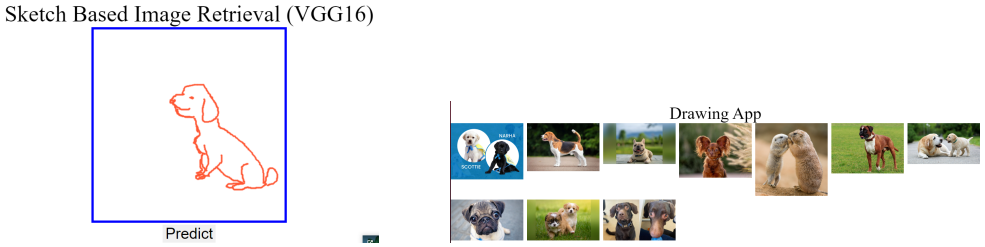


(a) Input (b) Output

Figure 5.6: VGG-16 Classifying a sketch of a car

Dog sketch

The VGG16 model faced some challenges in classifying dog sketches due to the diversity in the training sketches. Some were highly detailed, while others were very basic. Furthermore, some sketches only featured the face of a dog, while others included the entire body. Despite these challenges, VGG16 eventually classified the dog sketches correctly after a few attempts at creating a high-quality sketch that captured the dog's complexity.



(a) Input (b) Output

Figure 5.7: VGG-16 Classifying a sketch of a dog

Door sketch

The VGG-16 algorithm did not encounter any difficulties in classifying the sketch of a door. This was likely because the sketch was well-detailed and unique, making it easier for the algorithm to identify and categorize it. Additionally, sketches of doors are generally straightforward and simple.

(a) Input (b) Output

Figure 5.8: VGG-16 Classifying a sketch of a door

Ladder sketch

Similar to the apple and banana sketches, VGG16 did not face any difficulty in classifying the ladder sketch due to its simple nature.

(a) Input (b) Output

Figure 5.9: VGG-16 Classifying a sketch of a ladder

Moon sketch

VGG16 had difficulty classifying the sketches of a moon as it often classified them as bananas due to the similar crescent shape of the moon and a banana.

(a) Input (b) Output

Figure 5.10: VGG-16 Failing to Classify a sketch of a moon

Sheep sketch

The VGG16 model had similar results when classifying sheep sketches as it did with dog sketches. The diverse range of training sketches presented a challenge for the model. Some sketches were highly detailed, while others were simple. Additionally, some only depicted the head of a sheep, while others showed the entire body. Despite these challenges, VGG16 was eventually able to classify the sheep sketches correctly by identifying key features such as the body shape and woolly texture.

(a) Input (b) Output

Figure 5.11: VGG-16 Classifying a sketch of a sheep

Table sketch

VGG16 didn't encounter any difficulties in classifying table sketches, similar to ladder sketches.

(a) Input (b) Output

Figure 5.12: VGG-16 Classifying a sketch of a table

Tree sketch

Similar to ladders and tables, VGG16 didn't have any trouble classifying tree sketches due to their unique shape and simple drawing.

(a) Input (b) Output

Figure 5.13: VGG-16 Classifying a sketch of a tree

Wheel sketch

Despite the simplicity of wheel sketches compared to more complex drawings such as sheep and dog sketches, VGG16 faced challenges in classifying them. Even after adding details such as rims and redrawing them in different styles, it still failed to classify them correctly. This is a peculiar occurrence and suggests that VGG16 may require a larger dataset of wheel sketches to improve its classification accuracy.

(a) Input (b) Output

Figure 5.14: VGG-16 Failing to classify a sketch of a wheel

5.1.3 Summary

The VGG-16 algorithm was highly accurate in classifying airplane, apple, and banana sketches due to their simple nature, but encountered difficulty with bicycle sketches, likely due to the complexity of the image. The algorithm also struggled with moon sketches, sometimes misclassifying them as bananas due to their similar shape. The model had mixed results with sheep and dog sketches due to the diversity in the training sketches. The model did not face any difficulties with door, ladder, table, and tree sketches due to their simplicity. VGG16 had a peculiar challenge with classifying wheel sketches, even after adding details, suggesting that a larger dataset may be needed to improve accuracy.

5.2 ResNet-50

5.2.1 Training Results

The model achieved an accuracy of 83% on the training set and 69% on the validation set in the first epoch. Over the course of the next 20 epochs, the model continued to improve, achieving an accuracy of 96.7% on the training set and 83.85% on the validation set by the 20th epoch.

It is noteworthy that the model's performance on the validation set initially lagged behind its performance on the training set, but this trend eventually reversed, and the model consistently outperformed the training set by the 20th epoch. This suggests that the model may have been overfitting in the early epochs, but eventually learned to generalize to new data.

It is also interesting to note that the validation accuracy showed a somewhat fluctuating trend over the course of the 20 epochs, with occasional dips and spikes. This is not unusual in deep learning models, and may be attributed to the stochastic nature of the optimization process.

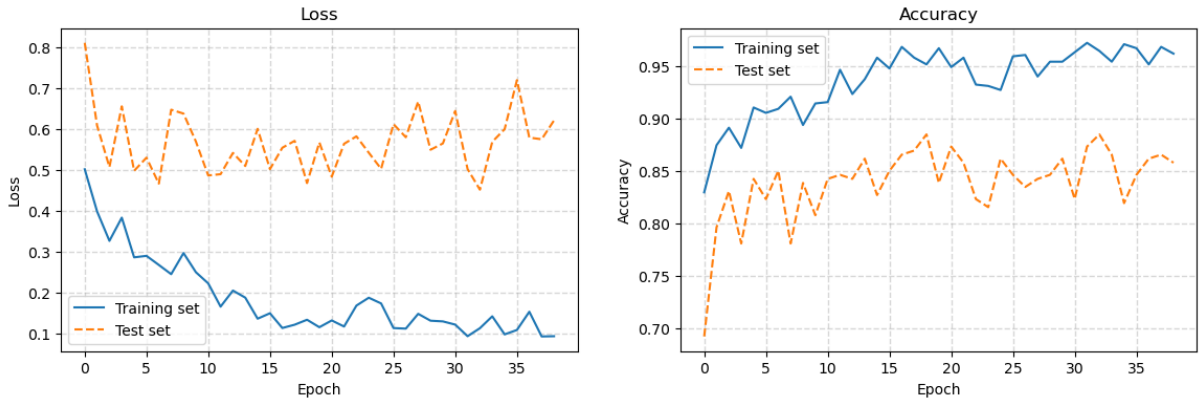


Figure 5.15: Loss and Accuracy Graphs of ResNet50 Model while training on the dataset.

5.2.2 Testing ResNet-50's ability to classify new sketches.

Airplane sketch

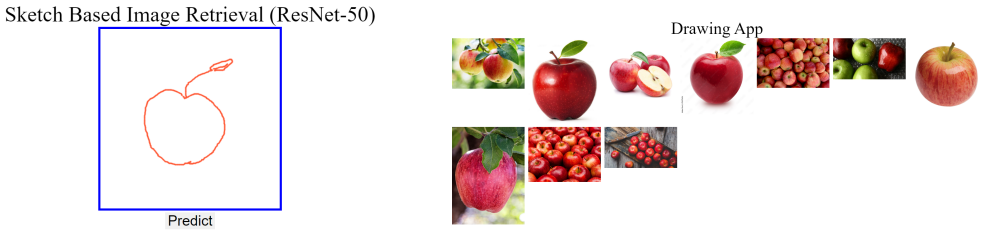
ResNet50 encountered difficulties in correctly classifying airplane sketches even after several attempts. Initially, when only the main body was drawn, the model misclassified the image. Additional details such as the wings, landing gears, cockpit window, and passenger windows were added, but the classification accuracy did not improve. Eventually, ResNet50 failed to recognize the airplane sketch altogether.

(a) Input (b) Output

Figure 5.16: ResNet-50 Failing to classify a sketch of an airplane

Apple sketch

ResNet50 had no difficulties in classifying apple sketches thanks to their distinct shape and consistent drawing style across different sketches.

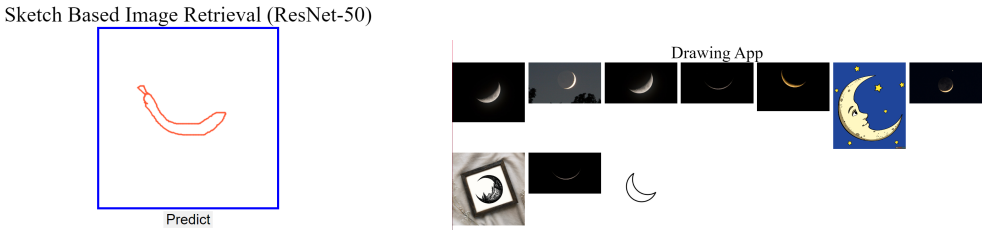


(a) Input (b) Output

Figure 5.17: ResNet-50 Classifying a sketch of an apple

Banana sketch

ResNet50 had difficulty classifying banana sketches as it consistently classified them as crescent moons due to their identical shapes in various drawing styles. Despite attempting to add more details such as the banana stem, ResNet50 still failed to correctly classify the sketch.



(a) Input (b) Output

Figure 5.18: ResNet-50 Failing to classify a sketch of a banana

Bicycle sketch

Resnet50 encountered difficulties in classifying bicycle sketches, perhaps due to their intricate structure and the limited number of training images. Despite our efforts to include more details, such as the addition of wheel rims and elaboration on the bike seat, resnet50 still struggled to classify the sketches accurately.

(a) Input (b) Output

Figure 5.19: ResNet-50 Failing to classify a sketch of a bicycle

Car sketch

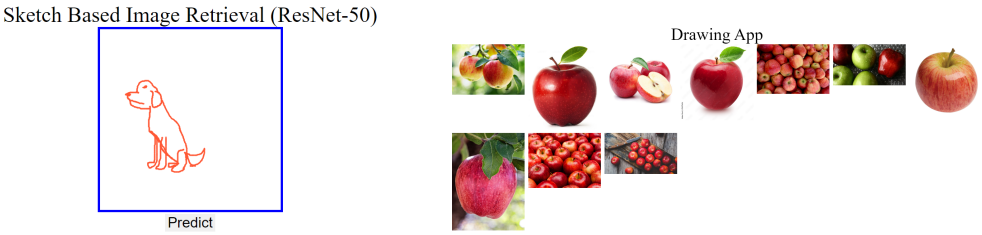
Similar to the bike sketches, ResNet50 seems to have difficulty classifying complex sketches accurately, despite having high accuracy during training. Even after adding various details such as lights, windows, rims, light signals, radio antennas, etc., to the car sketch, there was no improvement in classification accuracy. It appears that ResNet50 has a significant issue with classifying sketches.

(a) Input (b) Output

Figure 5.20: ResNet-50 Failing to classify a sketch of a car

Dog sketch

Similar to the previous sketches, ResNet50 also faces difficulty in correctly classifying dog sketches. Despite its high accuracy during training, ResNet50 struggles with complex drawings of dogs, and even the addition of more details like fur texture, snout length, and tail shape does not seem to improve its classification accuracy. This problem with dog sketches might be due to the wide variety of dog breeds and the subtle differences between their physical characteristics, making it challenging for the model to identify and distinguish between them. Overall, ResNet50's performance on complex sketches remains a significant challenge, and further improvements in the training process and model architecture may be necessary to address this issue.

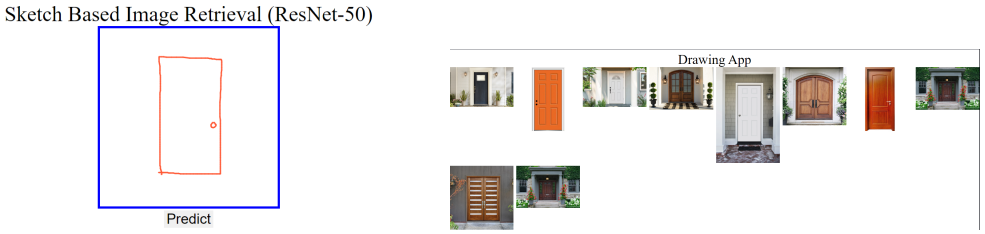


(a) Input (b) Output

Figure 5.21: ResNet-50 Failing to classify a sketch of a dog

Door sketch

As expected, ResNet50 performs well in recognizing simple sketches such as apples, tables, and wheels, leading to correct classification of doors. However, the model struggles significantly when it comes to complex drawings that require a lot of details.



(a) Input (b) Output

Figure 5.22: ResNet-50 Classifying a door

Ladder sketch

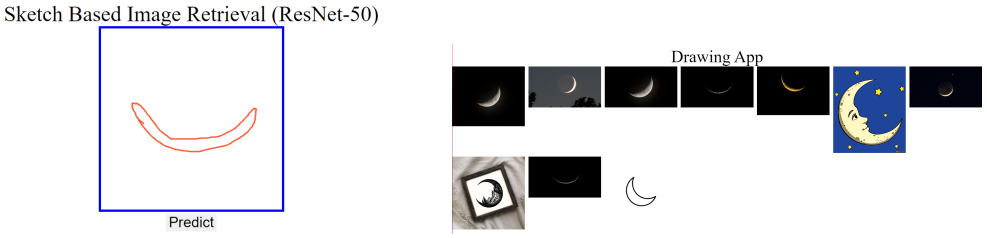
ResNet50 preformed exactly like door sketch due to their similar simplicity in drawing style .

(a) Input (b) Output

Figure 5.23: ResNet-50 Classifying a ladder

Moon sketch

ResNet50 preformed exactly like door sketch due to their similar simplicity in drawing style .

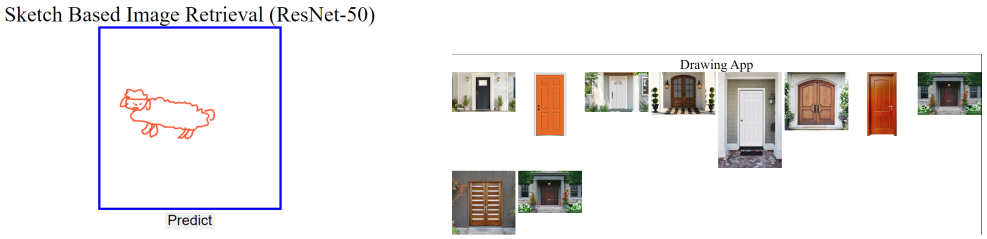


(a) Input (b) Output

Figure 5.24: ResNet-50 Classifying a moon

Sheep sketch

Similar to the bike sketches, ResNet50 seems to have difficulty classifying complex sketches accurately, despite having high accuracy during training. It appears that ResNet50 has a significant issue with classifying sketches.



(a) Input (b) Output

Figure 5.25: ResNet-50 Failing to classify a sketch of a sheep

Table sketch

ResNet-50 generally had an easy time classifying tables due to their simple and regular shape with a flat top and legs. However, occasionally when the table had multiple legs, ResNet-50 misclassified it as a ladder. This may have been because the legs of the table resembled the start of a ladder, given their similar long rectangular shapes. Nevertheless, overall ResNet-50 did not encounter significant issues in accurately classifying tables.

(a) Input (b) Output

Figure 5.26: ResNet-50 Classifying a table

Tree sketch

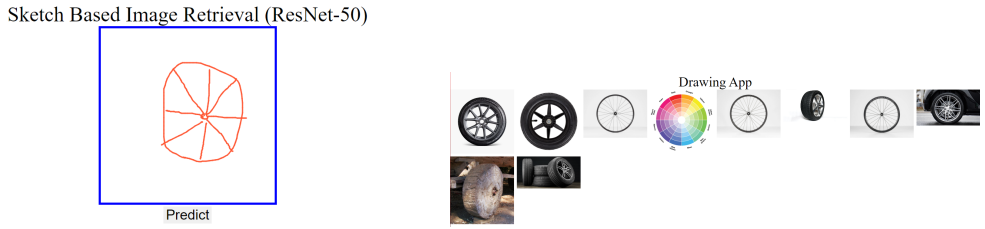
ResNet50 preformed exactly like door sketch due to their similar simplicity in drawing style .

(a) Input (b) Output

Figure 5.27: ResNet-50 Classifying a tree

Wheel sketch

Similarly, ResNet-50 did not encounter significant difficulties in classifying wheels. However, at times, it incorrectly identified wheels as apples or moons due to their similar round shapes. This misclassification occurred particularly when the sketch of the wheel only showed a simple circle, without any inner details. However, once additional details, such as spokes and rims, were added to the sketch, ResNet-50 was better able to accurately classify the image as a wheel.



(a) Input (b) Output

Figure 5.28: ResNet-50 Classifying a wheel

5.2.3 Summary

ResNet50 struggles with accurately classifying complex sketches, including airplanes, bananas, cars, bicycles, and dogs, even after adding more details to the sketches. ResNet50 performs well in recognizing simple sketches like apples, tables, wheels, and doors. It occasionally misclassifies tables as ladders and wheels as moons or apples when only a simple circle is drawn without inner details. Overall, ResNet50's performance on complex sketches is challenging, and further improvements in the training process and model architecture may be necessary to address this issue.

5.3 Inception-V3

5.3.1 Training Results

The results from the training show that the model is improving over time, with both loss and categorical accuracy improving as the epochs progress. In the first epoch, the model had a categorical accuracy of 25.84% and a validation categorical accuracy of 50%. However, the accuracy improved significantly over the next few epochs. By the fifth epoch, the model had a categorical accuracy of 71.96% and a validation categorical accuracy of 82.81%. In the following epochs, the validation categorical accuracy remained relatively stable at around 82-86%, while the categorical accuracy continued to improve, peaking at 84.97% in the twelfth epoch. The initial loss of 2.3508 and categorical accuracy of 0.2584 on the first epoch improved significantly on the last epoch with a loss of 0.3898 and categorical accuracy of 0.9070.

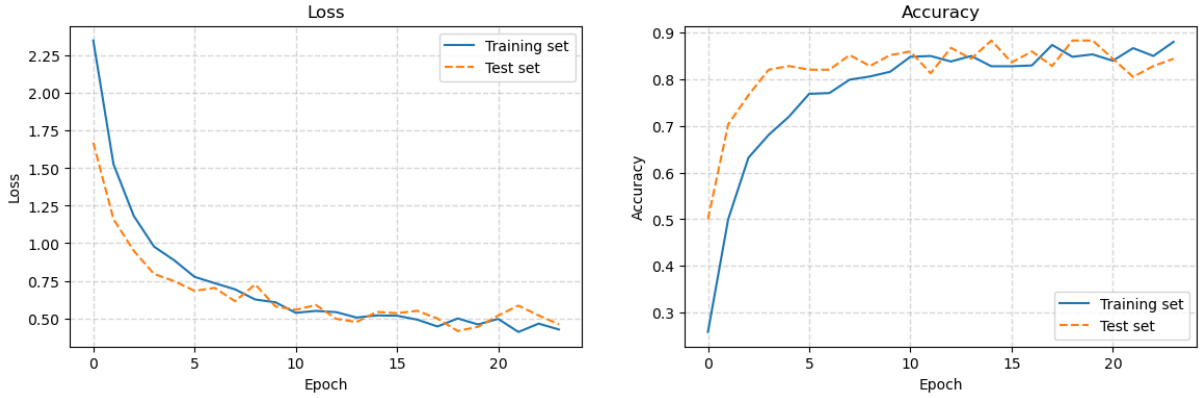


Figure 5.29: Loss and Accuracy Graphs of Inception-V3 Model while training on the dataset.

Looking at the validation results, the validation loss and validation categorical accuracy are both improving over time, indicating that the model is generalizing well to the validation data. The validation categorical accuracy starts at 50% on the first epoch and improves to 90.70% on the last epoch. Similarly, the validation loss starts at 1.6705 on the first epoch and decreases to 0.3898 on the last epoch.

Based on the trend, it seems that the model has not yet converged, and training for more epochs could yield further improvements in both the training and validation results. It is important to note that the model's performance may plateau after a certain number of epochs, so monitoring the validation results and avoiding overfitting is essential.

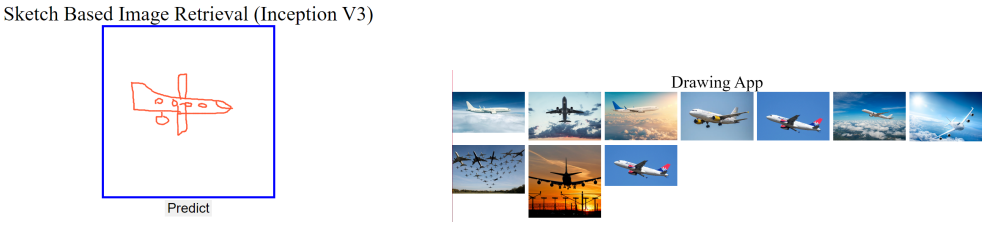
One interesting trend that we can observe is that the model's categorical accuracy on the validation set is consistently higher than its categorical accuracy on the training set. This phenomenon is known as generalization, and it indicates that the model is learning to recognize patterns and features that are relevant to the task at hand, rather than simply memorizing the training data.

Another trend that we can observe is that the validation loss and validation categorical accuracy have more fluctuations compared to the training loss and training categorical accuracy. This is because the validation data is only evaluated at the end of each epoch, whereas the training data is evaluated after each batch. Thus, the training results are more stable, while the validation results can be more variable.

5.3.2 Testing Inception-V3's ability to classify new sketches.

Airplane sketch

Inceptionv3 had few issues with classifying airplane sketches. Initially, it mistakenly classified an airplane sketch as a banana, possibly due to similarities in the main body shapes between the two. However, after adding more details such as wings, landing gear, cockpit windows, and passenger windows, Inceptionv3 was able to correctly classify the airplane sketch.



(a) Input (b) Output

Figure 5.30: InceptionV3 Classifying an airplane

Apple sketch

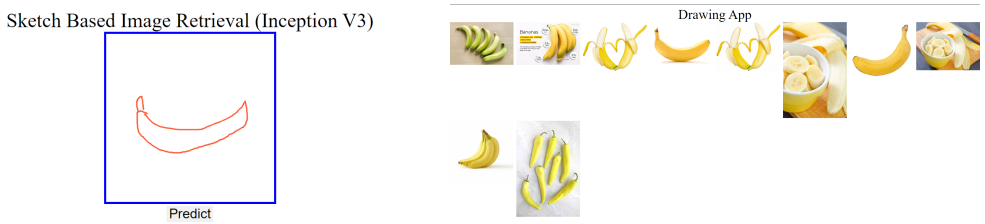
Inceptionv3 had no problem classifying an apple sketch, but there was one instance when it classified it as a moon, possibly due to their similar circular shape. However, in general, Inceptionv3 was able to correctly identify the apple sketch.

(a) Input (b) Output

Figure 5.31: InceptionV3 Classifying an apple

Banana sketch

InceptionV3 had no difficulty classifying the sketch of a banana due to its simple and straightforward drawing. However, there was one instance where InceptionV3 misclassified the banana as a crescent moon. This may have been due to the similarities in shape between a banana and a crescent moon. However, when the top portion of the banana was added to the sketch, InceptionV3 correctly classified the image as a banana, indicating that additional details can help improve the accuracy of the classification.



(a) Input (b) Output

Figure 5.32: InceptionV3 Classifying a banana

Bicycle sketch

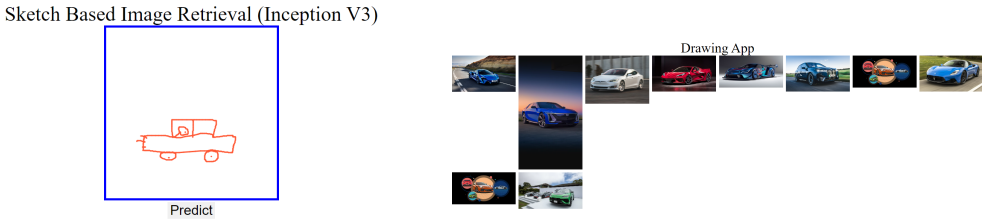
Inception-V3 faced significant challenges in classifying bicycles due to their complex nature and intricate details in the drawing. One of the issues that Inception-V3 encountered was misclassification based solely on the size of the wheels, as they are prominent features in bicycles. However, when additional details were added to the sketch, such as the seat, pedals, handlebars, and mainframe of the bike, and the wheels were simplified and made smaller, Inception-V3 began to accurately classify the bicycle.

(a) Input (b) Output

Figure 5.33: InceptionV3 Classifying a bicycle

Car sketch

Inceptionv3 didn't encounter any significant difficulties with car sketches. It occasionally struggled when there were insufficient details, such as in the headlights, backlights, driving wheel, or wheel rims, but generally, the model correctly identified car sketches.

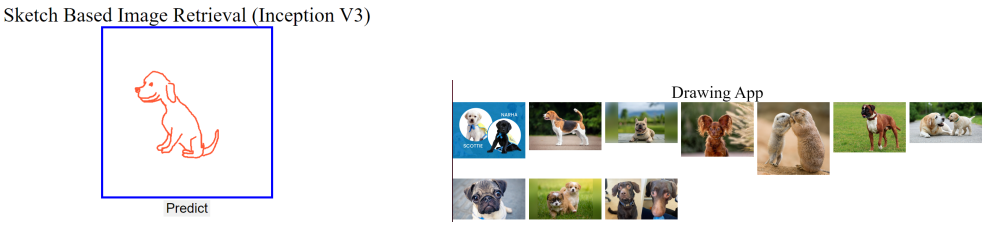


(a) Input (b) Output

Figure 5.34: InceptionV3 Classifying a car

Dog sketch

Inceptionv3 had no problem correctly classifying a dog sketch, but it required sufficient and specific details in the drawing. For example, it was important to include details such as the shape and size of the ears, the length and curvature of the tail, as well as the overall proportion and body structure of the dog. Without these details, Inceptionv3 may struggle to correctly identify the sketch as a dog. However, with proper details included in the drawing, Inceptionv3 was able to accurately classify the dog sketch.

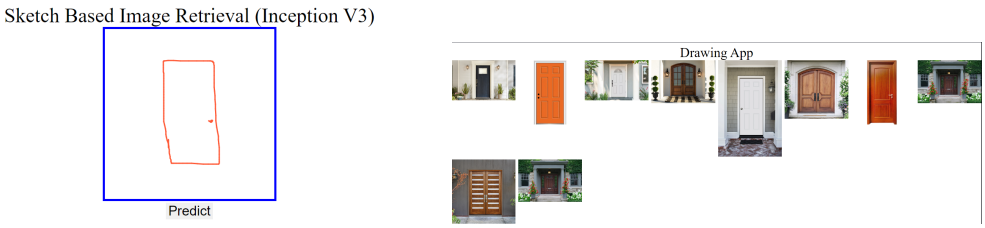


(a) Input (b) Output

Figure 5.35: InceptionV3 Classifying a dog

Door sketch

Inceptionv3 had no issues with classifying doors, likely due to their simple drawing style.

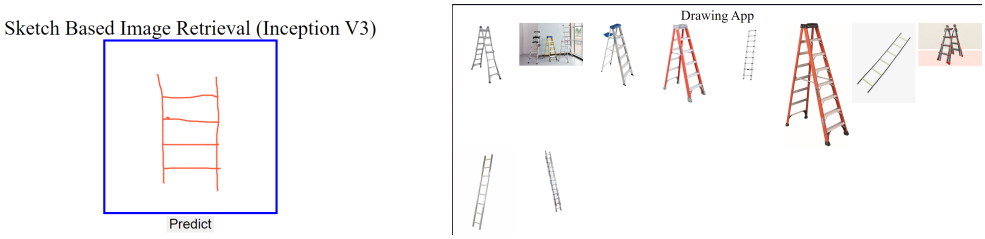


(a) Input (b) Output

Figure 5.36: InceptionV3 Classifying a door

Ladder sketch

Inceptionv3 had no issues with classifying ladders, likely due to their simple drawing style.



(a) Input (b) Output

Figure 5.37: InceptionV3 Classifying a ladder

Moon sketch

Similar to VGG16, InceptionV3 also struggled to classify moon sketches due to their resemblance to the shape of a banana.

(a) Input (b) Output

Figure 5.38: InceptionV3 Failed to classify a moon

Sheep sketch

Inceptionv3 didn't have any problem classifying sheep sketches.

(a) Input (b) Output

Figure 5.39: InceptionV3 Classifying a sheep

Table sketch

Inceptionv3 didn't have any problem classifying table sketches.

(a) Input (b) Output

Figure 5.40: InceptionV3 Classifying a table

Tree sketch

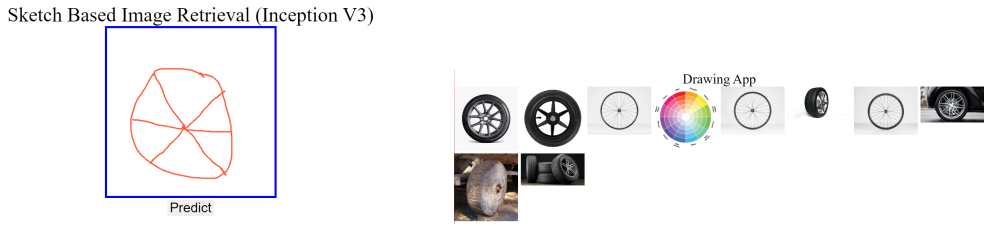
Inceptionv3 didn't have any problem classifying table sketches.

(a) Input (b) Output

Figure 5.41: InceptionV3 Classifying a tree

Wheel sketch

Inceptionv3 didn't have any problem classifying wheel sketches.



(a) Input (b) Output

Figure 5.42: InceptionV3 Classifying a wheel

5.3.3 Summary

The model struggled with classifying airplane sketches, apples, and bicycles due to similarities in shape or complexity of the drawings. However, adding more details to the sketches improved the accuracy of classification. Inceptionv3 did not have difficulty with classifying simple sketches such as bananas, doors, ladders, tables, and wheels. Moon sketches were challenging for Inceptionv3, similar to VGG16, due to their resemblance to the shape of a banana. Sufficient and specific details were required for Inceptionv3 to accurately classify dog sketches.

5.4 CNN Model

5.4.1 Training Results

The CNN model's learning process on the given dataset showed a gradual improvement in accuracy and loss over the 50 epochs. In the first epoch, the model had a low accuracy rate of 0.0726 and a high loss value of 2.7748, while the validation accuracy was slightly higher at 0.0859 and the validation loss was 2.5575. However, as the training progressed, the accuracy improved, and the loss decreased. By epoch 13, the model had achieved an accuracy of 0.6723 and a loss of 0.9800, indicating that the model was learning well.

At epoch 15, the learning rate decreased by a factor of 10, and there was a slight decrease in validation accuracy. This decrease may have been due to overfitting, as the model was adjusting to the new learning rate. However, the model continued to train, and by epoch 50, the accuracy had significantly improved from 0.0726 to 0.7010, while the loss had decreased from 2.7748 to 0.9205.

Overall, the results indicate that the CNN model was able to learn and generalize well on the given dataset, achieving high accuracy rates and demonstrating good performance in terms of loss. The trend of gradual improvement in accuracy and loss over time is a positive indication of the model's ability to learn from the data and make accurate predictions.

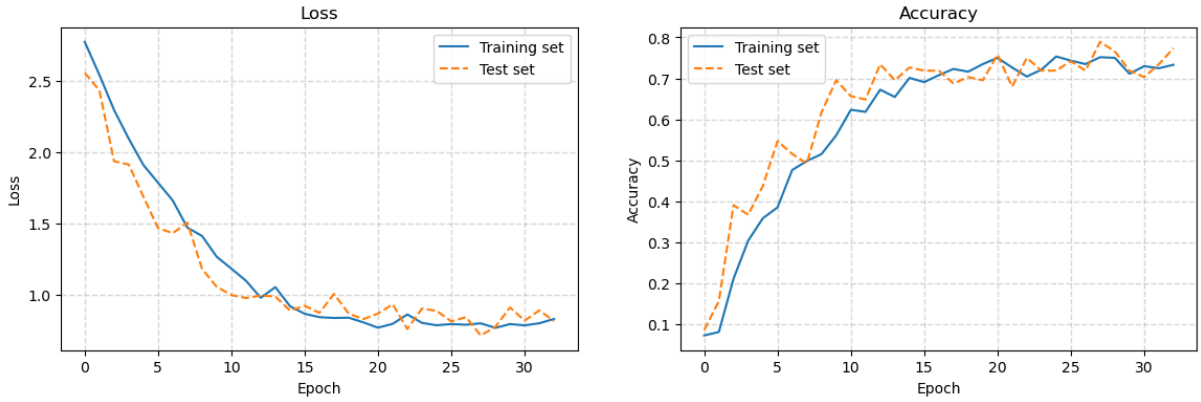
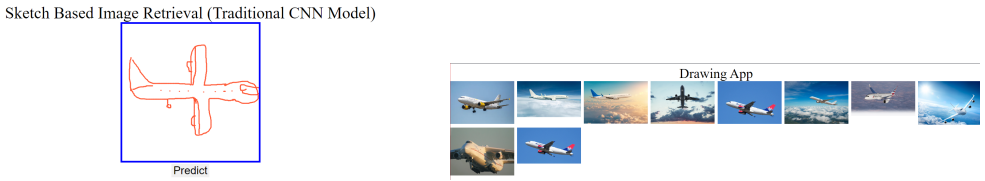


Figure 5.43: Loss and Accuracy Graphs of CNN Model while training on the dataset.

5.4.2 Testing the CNN Model's ability to classify new sketches.

Airplane sketch

The CNN model encountered significant difficulties in classifying sketches of airplanes due to the diversity and complexity of the training drawings. As a result, the model struggled to learn the distinguishing features of airplane sketches, making it challenging to classify them accurately. However, when additional details were added to the sketch, such as landing gear, airplane windows, airplane tail, and more accurate drawings of wings, the model was more successful in classifying the image correctly. Despite these efforts, the CNN model still struggled to classify airplanes accurately.



(a) Input (b) Output

Figure 5.44: CNN Model Classifying an airplane

Apple sketch

Overall, the Convolutional Neural Network (CNN) did not encounter significant challenges in classifying the simple sketch of an apple. However, there were instances where it incorrectly identified the apple as a moon or a wheel due to their similar round shapes. To address this issue, adding the leaf of the apple and adjusting the curvature of the drawing to match that of an actual apple greatly improved the accuracy of the classification. As a result, the CNN was able to more reliably and accurately classify the image as an apple.

(a) Input (b) Output

Figure 5.45: CNN Model Classifying an apple

Banana sketch

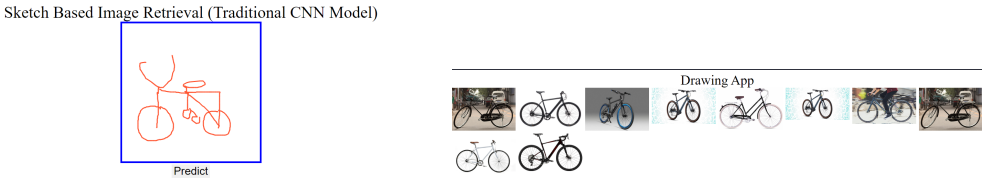
The CNN model failed to classify the banana sketch, which is a surprising result because other models generally did not have difficulty identifying the banana as it is a simple drawing.

(a) Input (b) Output

Figure 5.46: CNN Model Failed to classify a banana

Bicycle sketch

The CNN model did not have any problems with classifying bicycle sketches.



(a) Input (b) Output

Figure 5.47: CNN Model Classifying a bicycle

Car sketch

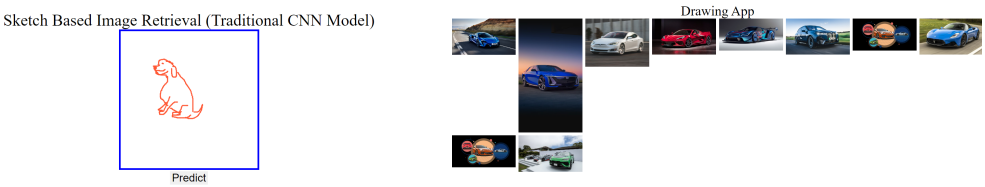
The CNN model failed to classify a car sketch even after adding more details, and instead classified it as a bicycle. This could be attributed to the similarity in the shape of the wheels between a car and a bicycle.

(a) Input (b) Output

Figure 5.48: CNN Model Failing to classify a car

Dog sketch

The CNN model struggled to classify dog sketches due to their complexity, requiring specific details such as the shape and size of the ears, tail, and body structure to accurately identify the sketch as a dog.

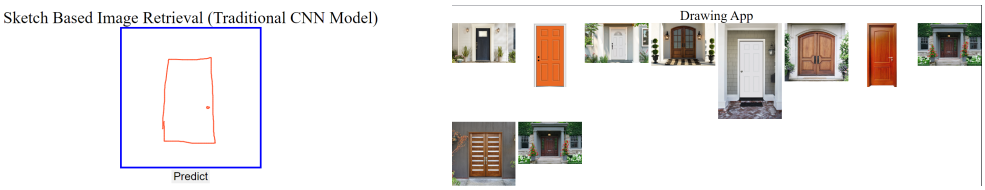


(a) Input (b) Output

Figure 5.49: CNN Model Failing to classify a dog

Door sketch

The CNN model had no difficulties in classifying door sketches.

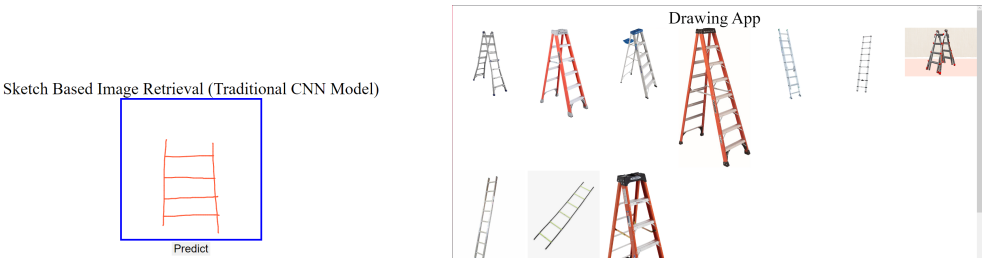


(a) Input (b) Output

Figure 5.50: CNN Model Classifying a door

Ladder sketch

The CNN model had no difficulties in classifying ladder sketches.



(a) Input (b) Output

Figure 5.51: CNN Model Classifying a ladder

Moon sketch

CNN model failed to classify moon sketch.

(a) Input (b) Output

Figure 5.52: CNN Model failed to classify a moon

Sheep sketch

The CNN model had no difficulties in classifying sheep sketches.

(a) Input (b) Output

Figure 5.53: CNN Model Classifying a sheep

Table sketch

(a) Input (b) Output

Figure 5.54: CNN Model Failed to classify a table

Tree sketch

(a) Input (b) Output

Figure 5.55: CNN Model Classifying a tree

Wheel sketch

(a) Input (b) Output

Figure 5.56: CNN Model Failed to classify a wheel

5.4.3 Summary

The CNN encountered significant difficulties in classifying airplane sketches due to their diversity and complexity. However, adding additional details to the sketches such as landing gear, windows, and tail improved the accuracy of classification. The CNN had no difficulties in classifying apple, bicycle, door, and ladder sketches. The CNN struggled to classify car sketches and failed to classify banana, dog, moon, sheep, table, and wheel sketches. The shape and complexity of the objects, as well as similarity to other objects, were identified as factors that affected the CNN's ability to classify the sketches accurately.

Chapter 6

Analysis

Comparing the results of the four models, we can see that all of them showed improvements in both training and validation accuracy over time, indicating that they were all able to learn from the given dataset. However, there were some differences in their performance and behavior.

6.1 VGG-16

The VGG-16 algorithm achieved the highest training accuracy of all the models, with a maximum of 96.4%, but its validation accuracy was slightly lower, peaking at 90.8%, indicating some overfitting to the training data. However, the model's overall performance was still impressive, and it successfully categorized 10 out of 13 categories of sketches when presented with new data. The algorithm excelled at classifying simple sketches such as airplanes, apples, and bananas, but encountered difficulty with more complex images such as bicycles and moons. The model had mixed results with sheep and dog sketches due to the diversity in the training sketches, while it encountered no difficulties with simpler sketches such as doors, ladders, tables, and trees. Interestingly, the model had a unique challenge with wheel sketches, even with additional details added, suggesting that a larger dataset may be necessary to improve accuracy.

Figure 6.1: VGG-16 Categorized 10 out of 13 new sketches correctly.

6.2 ResNet-50

RESNET50 showed a performance gap between the training and validation sets initially, but this trend eventually reversed, and the model consistently outperformed the training set by the 20th epoch, indicating that the model may have been overfitting in the early epochs but eventually learned to generalize to new data. In terms of sketch classification, ResNet50 struggled with accurately classifying complex sketches, including airplanes, bananas, cars, bicycles, and dogs, even after adding more details to the sketches. However, it performed well in recognizing simple sketches like apples, tables, wheels, and doors. Occasionally, it misclassified tables as ladders and wheels as moons or apples when only a simple circle was drawn without inner details. ResNet50's performance on complex sketches is challenging, and further improvements in the training process and model architecture may be necessary to address this issue. Overall , ResNet50 managed to categorize correctly only 7 out of 13 sketches.

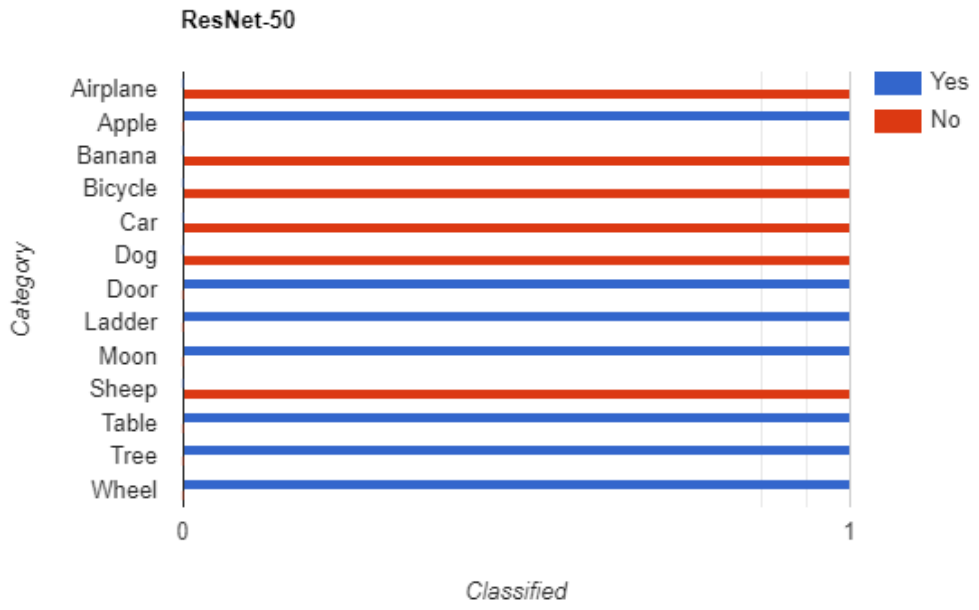


Figure 6.2: ResNet-50 Categorized 7 out of 13 new sketches correctly.

6.3 InceptionV3

InceptionV3 showed a steady improvement in both training and validation accuracy over time, with the validation accuracy consistently higher than the training accuracy, indicating good generalization ability. The model was able to classify 12 out of 13 sketches correctly. However, it struggled with classifying airplane, apple, and bicycle sketches due to similarities in shape or complexity of the drawings. Adding more details to the sketches improved the accuracy of classification. InceptionV3 did not face any difficulty with simple sketches like bananas, doors, ladders, tables, and wheels. Moon sketches were challenging for InceptionV3, similar to VGG16, due to their resemblance to the shape of a banana. Sufficient and specific details were required for InceptionV3 to accurately classify dog sketches. Overall, the model's performance was good, but there is room for improvement in its ability to classify complex sketches.

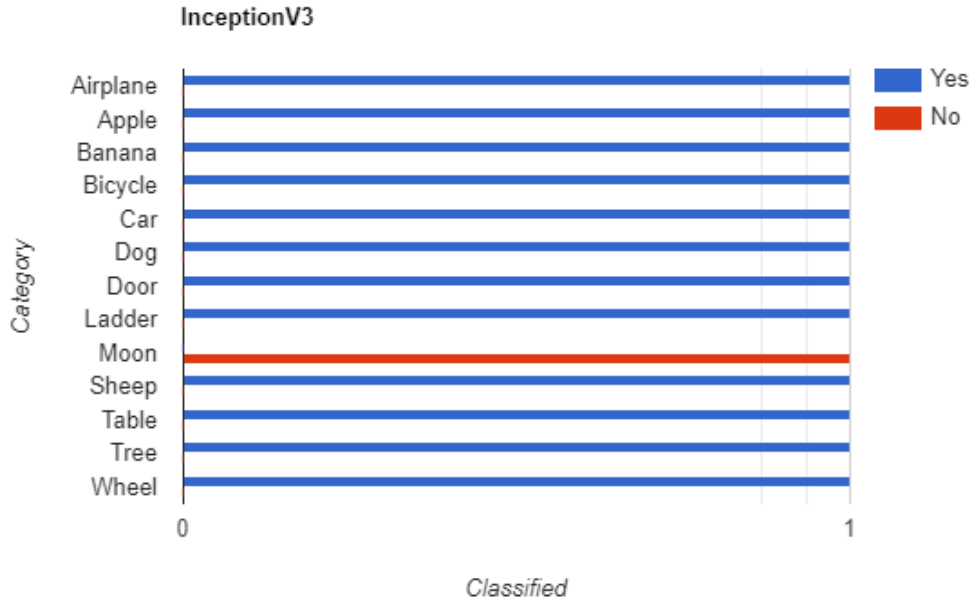


Figure 6.3: InceptionV3 Categorized 12 out of 13 new sketches correctly.

6.4 CNN Model

The CNN model exhibited a steady improvement in accuracy and loss over the 50 epochs, although its performance was inferior to that of the transfer learning models. This suggests that the transfer learning models were able to utilize pre-existing knowledge and learn more efficiently from the given dataset. However, the CNN encountered significant challenges in classifying airplane sketches due to their diversity and complexity. Nevertheless, adding more details to the sketches such as landing gear, windows, and tail significantly improved the accuracy of classification. The CNN had no difficulty classifying apple, bicycle, door, and ladder sketches. However, the CNN struggled to classify car sketches and failed to classify banana, dog, moon, sheep, table, and wheel sketches. The complexity and shape of the objects, as well as their similarity to other objects, were identified as factors that affected the CNN's ability to classify the sketches accurately.

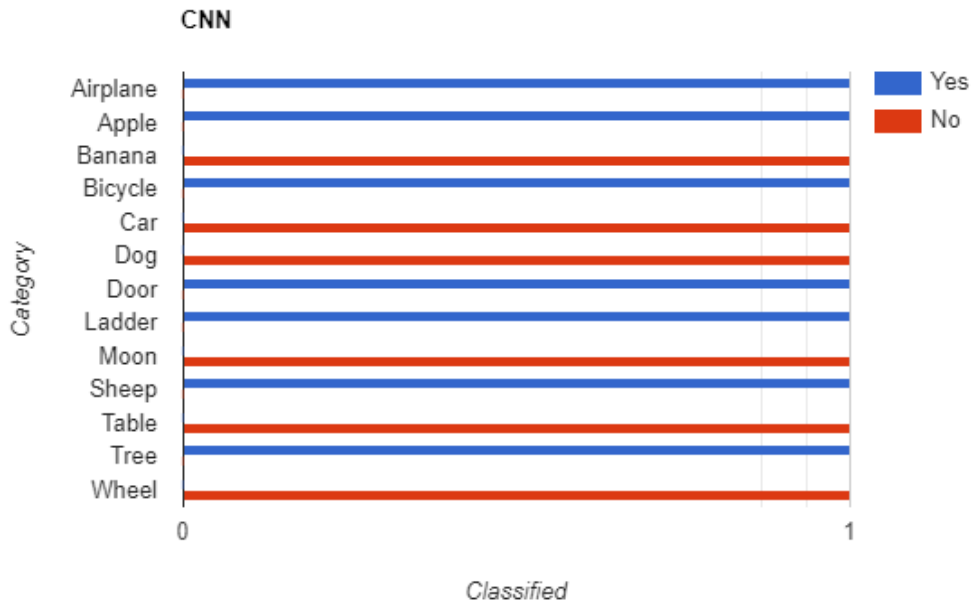


Figure 6.4: CNN Model Categorized 7 out of 13 new sketches correctly.

Chapter 7

Conclusion and Future Work

Bibliography

[1] T. Bezdan, "Vggnet architecture," 2019. Accessed: May 5, 2023.

[2] H. Zhang and Y. Liu, "Architecture of our method," 2017. Accessed: May 5, 2023.

[3] P. Chen, R. Zhang, X. Zhu, J. Li, W. Guo, Y. Chen, W. Xiao, and J. Fu, "Histosegnet: A framework for breast cancer tissue image segmentation using deep neural networks," *IEEE Access*, vol. 9, pp. 43312–43325, 2021.

[4] S. Du, "Style transfer with vgg16." https://github.com/sammdu/style-transfer-vgg16, 2021.

[5] B. R. Ilyas, "The proposed resnet50 cnn architecture," 2020. Accessed: May 5, 2023.

[6] T. Nguyen, "Resnet-50 convolutional neural networks with svm." https://www.researchgate.net/publication/336619872_ResNet-50_convolutional_neural_networks_with_SVM/figures/fig3, 2019. Accessed: May 5, 2023.

[7] Z. Benbahria, M. El Fadili, and A. Alami, "Uav image segmentation using resnet-50 architecture," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 11, no. 1, pp. 612–620, 2021.

[8] Google Cloud, "Inception v3 architecture overview," 2021. Accessed: May 5, 2023.

[9] M. Eitz, K. Hildebrand, T. Boubekeur, and M. Alexa, "Classifysketch," 2012. Accessed: May 5, 2023.

[10] MyGreatLearning, "Everything you need to know about vgg16," June 2021. Accessed: May 5, 2023.

[11] M. Albawi, "Residual blocks," 2022. Accessed: May 5, 2023.

[12] N. R. Wilburn and R. R. Martin, "Image retrieval using sketches and features extracted from sketches," in *Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval*, pp. 173–180, ACM, 2004.

[13] Y. Wang, C. Wang, and L. Zhang, "Deep sketch hashing: Fast free-hand sketch retrieval," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 370–378, 2016.

[14] S. Chen, S. Liu, Y. Li, L. Li, H. Zhang, and H. Lu, "Sketch-based 3d shape retrieval using convolutional neural networks," *Computers & Graphics*, vol. 67, pp. 12–21, 2017.

[15] S. Liu, L. Li, and H. Lu, "Sketchyscene: Richly-annotated scene sketches," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 918–927, 2019.

[16] J. Qian, X. Wang, Z. Chen, and H. Wang, "Deep feature extraction for sketch-based image retrieval," *Multimedia Tools and Applications*, vol. 80, no. 3, pp. 3319–3337, 2021.

[17] J. Wang, X. Zhang, X. Hu, and Q. Li, "Sketch-based image retrieval using multi-scale residual network," *Multimedia Tools and Applications*, vol. 79, no. 15, pp. 10395–10412, 2020.

[18] M. H. Torabi Motlagh Fard, N. Jomhari, and S. D. Ravana, "Sketch based image retrieval by using feature extraction technique," *Journal of Ambient Intelligence and Humanized Computing*, vol. 9, no. 1, pp. 77–86, 2018.

[19] L. Zhang, "Hand-drawn sketch recognition with a double-channel convolutional neural network," *Neurocomputing*, vol. 330, pp. 183–190, 2019.

[20] W. Zhou and J. Jia, "Training convolutional neural network for sketch recognition on large-scale dataset," in *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, pp. 98–101, IEEE, 2017.

[21] C. Edwards, 2017.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[23] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, 2019.

[24] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[26] W. Wang, J. Yan, J. Yan, and S. Xiang, "Remote sensing image retrieval using deep convolutional neural network," *Neurocomputing*, vol. 380, pp. 110–118, 2020.

[27] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *International Conference on Learning Representations (ICLR)*, 2015.

[28] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[29] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European Conference on Computer Vision (ECCV)*, 2016.

[32] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017.

[33] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[34] M. Eitz, J. Hays, and M. Alexa, "How do humans sketch objects?," *ACM Trans. Graph. (Proc. SIGGRAPH)*, vol. 31, no. 4, pp. 44:1–44:10, 2012.

Appendix A

VGG16 Model Code

```
import cv2
import numpy as np
import os
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense,Flatten,Conv2D,Activation,Dropout
from keras import backend as K
import keras
from keras.models import Sequential, Model
from keras.models import load_model
from keras.optimizers import SGD
from keras.callbacks import EarlyStopping,ModelCheckpoint
from keras.layers import MaxPool2D
import tensorflow as tf
```

```
train_path="E:/SBIR/Selected/train"
test_path="E:/SBIR/Selected/test"
class_names=os.listdir(train_path)
class_names_test=os.listdir(test_path)
```

```
train_datagen = ImageDataGenerator(zoom_range=0.15,width_shift_range
    =0.2,height_shift_range=0.2,shear_range=0.15)
test_datagen = ImageDataGenerator()
```

```
train_generator = train_datagen.flow_from_directory(train_path,
    target_size=(224, 224),batch_size=32,shuffle=True,class_mode='
    categorical')
test_generator = test_datagen.flow_from_directory(test_path,target_size
    =(224,224),batch_size=32,shuffle=False,class_mode='categorical')
```

```
trdata = ImageDataGenerator()
traindata = trdata.flow_from_directory(directory="E:/SBIR/Selected/
    train",target_size=(224,224))
tsdata = ImageDataGenerator()
testdata = tsdata.flow_from_directory(directory="E:/SBIR/Selected/test"
    , target_size=(224,224))
```

```
def VGG16():
    model = Sequential()
    model.add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size
   =(3,3),padding="same", activation="relu"))
    model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",
   activation="relu"))
    model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
    model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same",
   activation="relu"))
    model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same",
   activation="relu"))
    model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
    model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",
   activation="relu"))
    model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",
   activation="relu"))
    model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",
   activation="relu"))
    model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
    model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
   activation="relu"))
    model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
   activation="relu"))
    model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
   activation="relu"))
    model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
    model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
   activation="relu"))
    model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
   activation="relu"))
    model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
   activation="relu"))
    model.add(MaxPool2D(pool_size=(2,2),strides=(2,2),name='vgg16'))
    model.add(Flatten(name='flatten'))
    model.add(Dense(256, activation='relu', name='fc1'))
    model.add(Dense(128, activation='relu', name='fc2'))
    model.add(Dense(13, activation='sigmoid', name='output'))
    return model
```

```
model=VGG16()
model.summary()
from keras.models import Model
Vgg16 = Model(inputs=model.input, outputs=model.get_layer('vgg16').
   output)
Vgg16.load_weights("E:/SBIR/
   vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5")
for layer in Vgg16.layers:
    layer.trainable = False
```

```
from keras.optimizers import Adam
opt = Adam(lr=0.001)
model.compile(optimizer=opt, loss=keras.losses.categorical_crossentropy
   , metrics=['accuracy'])
```

```
from keras.callbacks import ModelCheckpoint, EarlyStopping
es = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=20,
    verbose=1, mode='auto')
H = model.fit_generator(train_generator,validation_data=test_generator,
    epochs=100,verbose=1,callbacks=[es])
```

Appendix B

ResNet-50 Model Code

```
train_path="E:/SBIR/Selected/train"
test_path="E:/SBIR/Selected/test"
class_names=os.listdir(train_path)
class_names_test=os.listdir(test_path)
train_generator = train_datagen.flow_from_directory(train_path,
    target_size=(180, 180),batch_size=32,shuffle=True,class_mode='
    categorical')
test_generator = test_datagen.flow_from_directory(test_path,target_size
    =(180,180),batch_size=32,shuffle=False,class_mode='categorical')
trdata = ImageDataGenerator()
traindata = trdata.flow_from_directory(directory="E:/SBIR/Selected/
    train",target_size=(180,180))
tsdata = ImageDataGenerator()
testdata = tsdata.flow_from_directory(directory="E:/SBIR/Selected/test"
    , target_size=(180,180))
```

```
resnet_model = Sequential()

pretrained_model= tf.keras.applications.ResNet50(include_top=False,
                   input_shape=(180,180,3),
                   pooling='avg',classes=13,
                   weights='imagenet')
for layer in pretrained_model.layers:
        layer.trainable=False

resnet_model.add(pretrained_model)
resnet_model.add(Flatten())
resnet_model.add(Dense(512, activation='relu'))
resnet_model.add(Dense(13, activation='softmax'))
```

```
from keras.optimizers import Adam

resnet_model.compile(optimizer=Adam(lr=0.001),loss='
    categorical_crossentropy',metrics=['accuracy'])
from keras.callbacks import ModelCheckpoint, EarlyStopping
es = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=20,
    verbose=1, mode='auto')
```

```
6 H = resnet_model.fit_generator(train_generator,validation_data=
    test_generator,epochs=100,verbose=1,callbacks=[es])
```

Appendix C

InceptionV3 Model Code

```
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
import netron
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.4,
    zoom_range=0.4,
    horizontal_flip=True,
    vertical_flip=True,
    validation_split=0.2
)

test_datagen = ImageDataGenerator(rescale=1./255)

incep_base = applications.InceptionV3(weights='imagenet', include_top=
    False, input_shape=(224, 224, 3))
incep_base.trainable = False

inputs = Input(shape=(224, 224, 3))

x = incep_base(inputs, training=False)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(13, activation='softmax')(x)
incep_model = Model(inputs, outputs)

incep_model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.0001),
    loss=keras.losses.CategoricalCrossentropy(from_logits=True),
    metrics=[keras.metrics.CategoricalAccuracy()]
)

early_stopping = EarlyStopping(monitor='val_loss', patience=5,
    restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience
    =2)
```

```
34
35 batch_size = 32
36 epochs = 25
37
38 train_generator = train_datagen.flow_from_directory(
39     'E:/SBIR/Selected/train',
40     target_size=(224, 224),
41     batch_size=batch_size,
42     class_mode='categorical',
43     subset='training'
44 )
45
46 validation_generator = train_datagen.flow_from_directory(
47     'E:/SBIR/Selected/train',
48     target_size=(224, 224),
49     batch_size=batch_size,
50     class_mode='categorical',
51     subset='validation'
52 )
53
54 test_generator = test_datagen.flow_from_directory(
55     'E:/SBIR/Selected/test',
56     target_size=(224, 224),
57     batch_size=batch_size,
58     class_mode='categorical'
59 )
60
61 history = incep_model.fit(
62     train_generator,
63     epochs=epochs,
64     validation_data=validation_generator,
65     callbacks=[early_stopping, reduce_lr],
66     steps_per_epoch=train_generator.samples // batch_size,
67     validation_steps=validation_generator.samples // batch_size
68 )
69
70 test_loss, test_accuracy = incep_model.evaluate(test_generator)
71
72 print(f'Test Loss: {test_loss:.2f}, Test Accuracy: {test_accuracy:.2f}'
    )
```

Appendix D

Traditional CNN Model Code

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.4,
    zoom_range=0.4,
    horizontal_flip=True,
    vertical_flip=True,
    validation_split=0.2
)

test_datagen = ImageDataGenerator(rescale=1./255)

early_stopping = EarlyStopping(monitor='val_loss', patience=5,
   restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience
   =2)

batch_size = 32
epochs = 50

train_generator = train_datagen.flow_from_directory(
    'E:/SBIR/Selected/train',
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    'E:/SBIR/Selected/train',
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'
)

test_generator = test_datagen.flow_from_directory(
```

```
    'E:/SBIR/Selected/test',
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical'
)


import netron
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Define the CNN model architecture
model = Sequential()

# Add the first convolutional layer with 32 filters, a 3x3 kernel, and ReLU activation
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))

# Add a max pooling layer with a 2x2 pool size
model.add(MaxPooling2D(pool_size=(2, 2)))

# Add a second convolutional layer with 64 filters and a 3x3 kernel
model.add(Conv2D(64, (3, 3), activation='relu'))

# Add a max pooling layer with a 2x2 pool size
model.add(MaxPooling2D(pool_size=(2, 2)))

# Add a third convolutional layer with 128 filters and a 3x3 kernel
model.add(Conv2D(128, (3, 3), activation='relu'))

# Add a max pooling layer with a 2x2 pool size
model.add(MaxPooling2D(pool_size=(2, 2)))

# Add a fourth convolutional layer with 256 filters and a 3x3 kernel
model.add(Conv2D(256, (3, 3), activation='relu'))

# Add a max pooling layer with a 2x2 pool size
model.add(MaxPooling2D(pool_size=(2, 2)))

# Flatten the output from the convolutional layers
model.add(Flatten())

# Add a dense layer with 512 neurons and ReLU activation
model.add(Dense(512, activation='relu'))

# Add a dropout layer to prevent overfitting
model.add(Dropout(0.5))

# Add a final dense layer with 13 neurons and softmax activation for classification
model.add(Dense(13, activation='softmax'))
```

```
84
85 # Compile the model with categorical crossentropy loss and Adam
       optimizer
86 model.compile(loss='categorical_crossentropy', optimizer='adam',
       metrics=['accuracy'])
87
88 # Print the model summary
89 model.summary()
90
91 model.save('model.h5')
92
93 netron.start('model.h5')
94
95 history = model.fit(
96     train_generator,
97     epochs=epochs,
98     validation_data=validation_generator,
99     callbacks=[early_stopping, reduce_lr],
100    steps_per_epoch=train_generator.samples // batch_size,
101    validation_steps=validation_generator.samples // batch_size
102 )
```