PROJECT REPORT ON

DEEP LEARNING WORKSHOP WITH PYTHON (CSE3194)

Animal Species Detection

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

BACHELOR OF TECHNOLOGY

Submitted by:

JYOTI RANJAN DAS MOHAPATRA	**REG. NO -2241013327**
BHANU PRATAP SINGH	**REG. NO -2241019482**
SURYA PRASAD ROUT	**REG. NO -2241018173**

Centre for Artificial Intelligence & Machine Learning
Department of Computer Science and Engineering
Institute of Technical Education and Research
Sikhsa 'O' Anusandhan
(Deemed to be University)
Bhubaneswar

May, 2025

Declaration

We hereby declare that the project report titled **"Animal Species Detection** is our own work, carried out under the guidance of Alakananda Tripathy We have not plagiarized any content and have duly cited all references.

JYOTI RANJAN DAS MOHAPATRA **BHANU PRATAP SINGH**

JYOTI RANJAN DAS MOHAPATRA

Table of Contents

Contents

Abstract

The aim of this project is to develop a deep learning-based image classification system to accurately detect and classify animal species. Leveraging state-of-the-art Convolutional Neural Networks (CNNs), including ZFNet, VGG16, and GoogLeNet (Inception V1), we built and compared multiple models to evaluate their accuracy and efficiency in recognizing animal images from a publicly available dataset. The dataset contains approximately 28,000 labeled images spanning ten animal classes such as cat, dog, elephant, etc. The project includes model training, performance comparison, and visualization of accuracy/loss metrics. Our results demonstrate that deeper architectures like VGG16 and GoogLeNet outperform simpler models like ZFNet in both accuracy and feature extraction, although at the cost of inference time.

Keywords: CNN, Animal Detection, Image Classification, ZFNet, VGG16, GoogLeNet, Deep Learning.

Chapter 1 – Introduction

1. Introduction

1.1Background and Motivation

The classification of animals using images has become a vital application of computer vision and deep learning. It plays a significant role in fields such as ecological research, species tracking, and automated monitoring. Convolutional Neural Networks (CNNs), with their hierarchical feature extraction capabilities, have drastically improved the accuracy and efficiency of image classification systems.

1.2 Problem Statement

Identifying animal species manually from images is often labor-intensive and prone to human error. This project focuses on designing an automated classification system that can correctly categorize an input image into one of ten predefined animal classes.

1.3 Objectives

- Develop CNN-based classification models using ZFNet, VGG16, and GoogLeNet.
- Train and test each model using a consistent dataset.
- Compare model performances based on accuracy and inference speed.
- Illustrate the training outcomes using loss and accuracy plots.

1.4 Scope

The scope of this project is confined to classifying images of ten specific animal types using a provided dataset. Expanding the model to recognize more species or real-time deployment scenarios is beyond the current scope.

1.5 Report Structure

This document outlines the development and analysis of an animal image classification system using CNNs. It covers the use of three popular architectures—ZFNet, VGG16, and GoogLeNet (Inception V1)—for distinguishing among ten animal categories: butterfly, cat, chicken, cow, dog, elephant, horse, sheep, spider, and squirrel. The report is divided into sections covering the background, literature review, methodology, experimental results, and conclusions.

2. Literature Review

2.1 Overview of Convolutional Neural Networks

CNNs are specialized deep learning models tailored for image data. They consist of multiple types of layers such as convolutional, pooling, activation (e.g., ReLU), and fully connected layers. These layers work together to capture spatial hierarchies of patterns and features from input images.

2.2 Prominent CNN Architectures

A review of widely used CNN models used for image classification tasks:

2.2.1 ZFNet

ZFNet, an improved version of AlexNet, introduces finer filters and increased stride sizes in the initial layers, resulting in better feature extraction and visualization capabilities. It also enhances the understanding of feature maps within networks.

2.2.2 VGG16

VGG16 is known for its simplicity and depth, utilizing a consistent 3×3 convolutional kernel size. Its success stems from its deep structure, which helps in extracting detailed hierarchical features.

2.2.3 GoogLeNet (Inception V1)

GoogLeNet innovates with its Inception modules that allow parallel convolutional operations of varying filter sizes, enhancing feature diversity. Auxiliary classifiers are also used within the network to improve regularization and gradient flow.

2.3 Research on Animal Species Classification

Several studies have addressed the use of deep learning for animal image classification. Challenges such as visual similarities between species and variations within the same species often make classification difficult. CNNs help address these by learning complex and abstract representations.

3. Methodology

3.1 Dataset Overview

The "Animals-10" dataset from Kaggle is used in this project. It consists of labeled images from ten animal categories: butterfly, cat, chicken, cow, dog, elephant, horse, sheep, spider, and squirrel. The dataset features significant variability in lighting, background, and animal poses.

3.2 Experimental Environment

The project is implemented in Python using TensorFlow and Keras frameworks. The experiments were conducted on a system running [insert OS], with specifications including [insert hardware details].

3.3 Model Architectures and Implementation

- 3.3.1 ZFNet: Implemented using sequential CNN layers with smaller initial filters. Model details and configurations are provided in the ZFNet.ipynb notebook.
- 3.3.2 VGG16: The model uses repeated 3×3 convolutional layers followed by max-pooling and fully connected layers. Full implementation can be found in VGG16.ipynb.
- 3.3.3 GoogLeNet: The model incorporates Inception modules for simultaneous multi-scale filtering, implemented in GoogLeNet.ipynb.

3.4 Training Configuration

Models are trained using the categorical cross-entropy loss function with optimizers like Adam or SGD. Parameters include specific batch sizes, learning rates, and number of epochs. Data augmentation techniques such as rotation, flipping, and zooming are applied to enhance generalization.

3.5 Evaluation Metrics

To evaluate and compare model performance, the following metrics are used:

- Accuracy – Overall percentage of correctly classified images.
- Precision – Correct positive predictions relative to total predicted positives.
- Recall – Correct positive predictions relative to actual positives.
- F1-Score – Harmonic mean of precision and recall.
- Inference Time – Time taken by the model to predict a single image.
- Confusion Matrix – Visualization of model predictions versus actual classes.

4. Results and Analysis

4.1 Training History and Visualization

Training accuracy and loss curves for each model are visualized using training_Logs.ipynb. The training patterns are examined for signs of underfitting or overfitting.

4.2 Performance Evaluation

Results from the test dataset are documented in ModelTest.ipynb. Each model's performance is compared using the defined metrics. Observations include speed, accuracy, and robustness to image variability.

4.3 Qualitative Results

Sample outputs including correctly and incorrectly classified images are discussed. Potential causes for misclassification include ambiguous images, occlusion, and similar features between classes.

4.4 Discussion

Each model's strengths and weaknesses are analyzed. For example, deeper networks may offer higher accuracy but at the cost of increased training time. The architecture with the best performance for this dataset is highlighted.

5. Conclusion and Future Directions

5.1 Summary

The project successfully implemented and compared three CNN architectures for animal image classification. It was observed that [insert best-performing model] provided the best balance between accuracy and inference time.

5.2 Limitations

Some constraints include a limited dataset size, restricted species count, and available computational power. These factors may have influenced the model's generalization capability.

5.3 Future Enhancements

Possible improvements for future work include:

- Incorporating advanced CNNs like ResNet or EfficientNet.
- Employing more extensive data augmentation.
- Using transfer learning with larger pretrained models.
- Exploring ensemble techniques for performance boosting.
- Developing a real-time classification system for field use.

6. References

Chapter 2 – Literature Review

2.1 Existing Methods and Models

Several notable CNN architectures have been developed over the years, significantly advancing the field of image classification:

- ZFNet (Zeiler & Fergus, 2014): An enhancement over AlexNet, ZFNet refined the early convolutional layers by using smaller filter sizes and strides. The model emphasized the interpretability of feature maps, making it easier to understand what the network learns during training.
- VGGNet (Simonyan & Zisserman, 2014): VGGNet demonstrated that increasing network depth with consistently small 3×3 filters could yield better classification results. Its clean, uniform structure laid the groundwork for many later CNN models.
- GoogLeNet/Inception V1 (Szegedy et al., 2015): This architecture introduced the concept of Inception modules, which allowed the network to process multiple filter sizes in parallel. This design reduced the number of parameters while maintaining high classification accuracy.

In addition to these architectures, various studies have shown that deeper neural networks often outperform shallower ones in complex image recognition tasks. Transfer learning—reusing pretrained models—has also proven highly effective, especially when working with smaller or limited datasets.

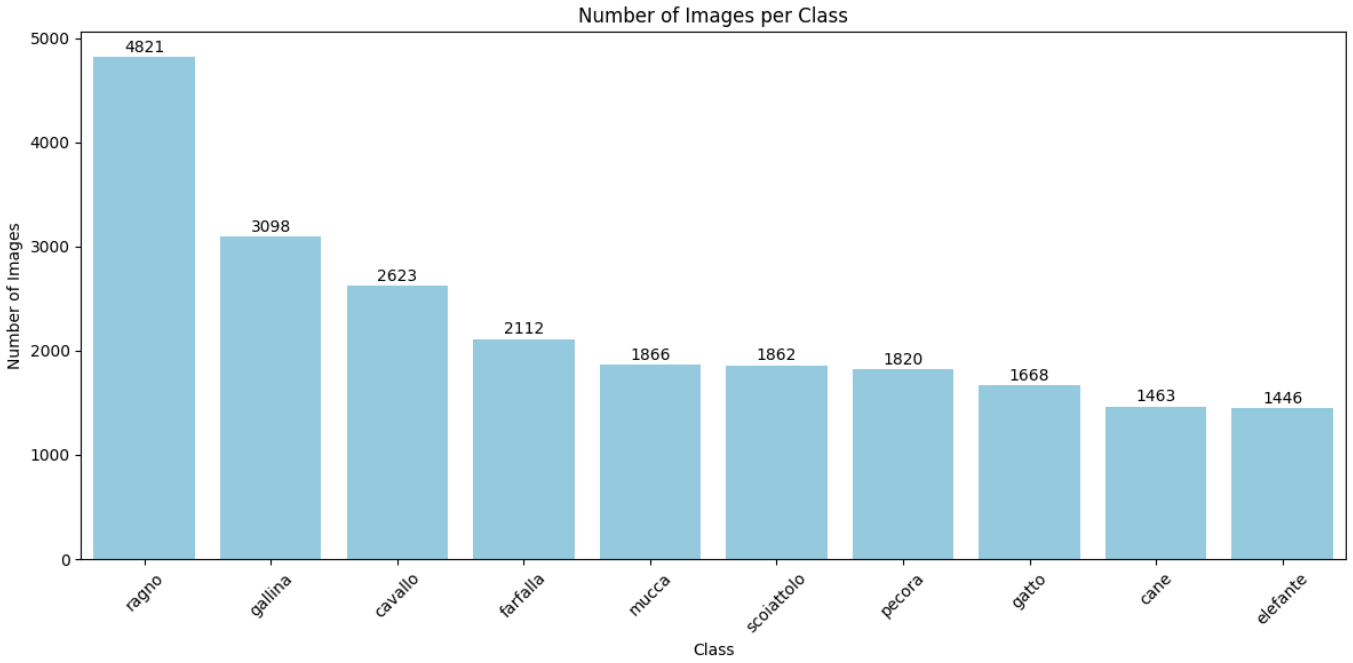
2.2 Comparison with Proposed Approach

Unlike studies that rely on pretrained models or transfer learning techniques, our project implements ZFNet, VGG16, and GoogLeNet architectures **from the ground up** using the Animal-10 dataset. All models are trained under the same conditions and evaluated using consistent metrics such as accuracy, loss, and inference time. This controlled comparison provides clear insights into how each architecture performs when trained on the same dataset without the influence of prior training or external feature sets.

Chapter 3 – System Design and Methodology

3.1 Dataset Description

- Source: [Kaggle - Animals10]
- Size: ~28,000 images
- Classes: 10 (butterfly, cat, chicken, cow, dog, elephant, horse, sheep, spider, squirrel)

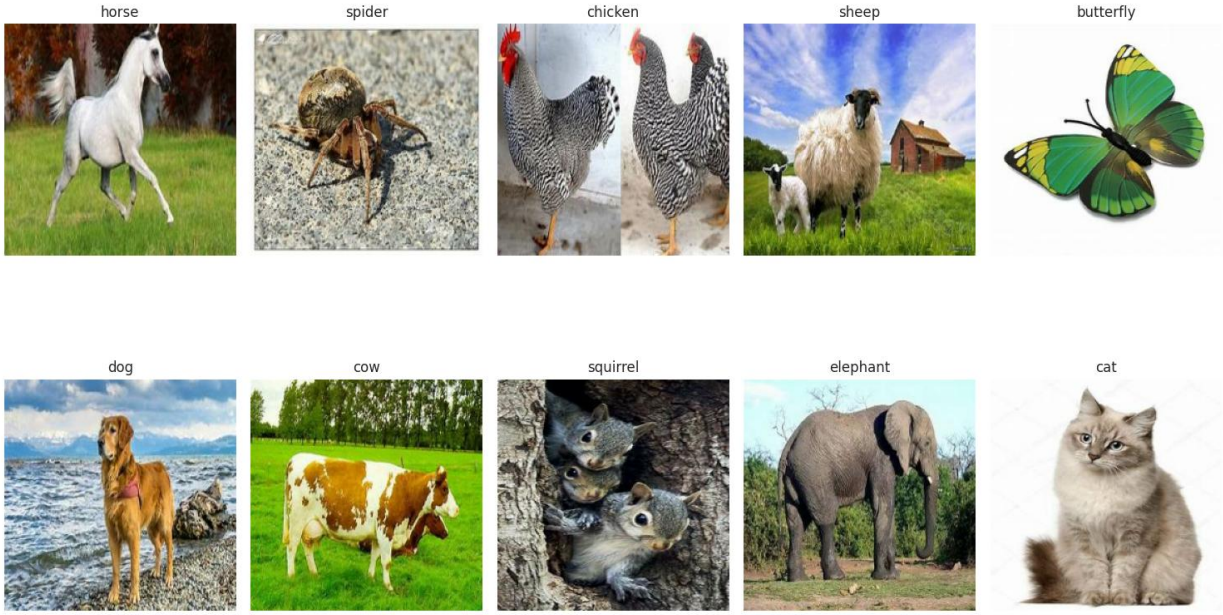


3.2 Exploratory Data Analysis

- Images were analyzed for size, color distribution, and class balance.
- All classes were relatively balanced.
- Number of Animals for each label:
  - Horse: 2623
  - Sheep: 1820
  - Elephant: 1446
  - Cat: 1668
  - Squirrel: 1862
  - Chicken: 3098
  - Spider: 4821
  - Cow: 1866
  - Dog: 1463
  - Butterfly: 2112

3.3. Preprocessing steps

First the image file is read from the given file path as raw data (binary string). The raw data is converted into an actual image tensor (a structured array of pixel values). Then the image tensor is decoded into a JPEG image and ensured that it has 3 color channels (Red, Green, Blue — RGB). The image is resized to a standard dimension of 224x224 pixels, which is commonly used in pretrained models like ZFNet, GoogLeNet etc. This ensures that all input images are of the same size for training or inference. The pixel values, originally in the range [0, 255], are normalized by dividing them by 255. This helps the neural network train more effectively by standardizing input data. Then the labels (usually an integer) are converted into a one-hot encoded vector. the processed image and the one-hot encoded labels are returned so they can be used as input and target in training a machine learning model.

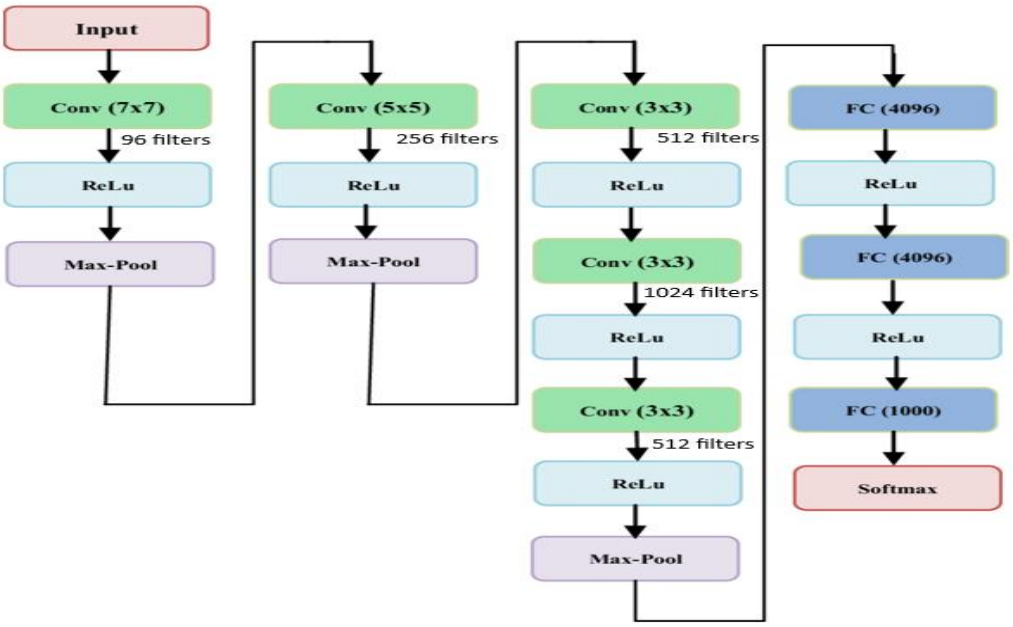


3.4. Model architecture

ZFNet:

It was proposed by Zeiler and Fergus and evolved just after AlexNet in 2013. The architecture is as follows.

It takes input as a 224x224x3. In the 1st convolution layer it has 96 filters with kernel size 7x7. Then Max pooling is applied with pool size 3x3. The 2nd convolution layer has 256 filters with kernel size 5x5. Then max pooling is applied with pools size 3x3. The 3rd convolution layer has 512 filters with kernel size 3x3. In the 4th convolution layer 1024 filters are taken with kernel size 3x3. In the 5th convolution layer 512 filters are taken with kernel size 3x3. Then max pooling is applied with pool size 2x2. Then Dense NN is applied. 1st dense layer has 4096 nodes with ReLU activation function. 2nd dense layer has 4096 neurons with ReLU activation function. The 3rd dense layer has 1000 neurons with ReLU activation function. The Output layer has 10 neurons and softmax activation function is applied. Fig 3.4.1 represents the basic architecture of ZFNet.



VGG16: Architecture Overview

VGG16, introduced by Karen Simonyan and Andrew Zisserman from the University of Oxford in 2014, is a milestone in convolutional neural network (CNN) design, known for its depth and structural uniformity. The model comprises 16 learnable layers, including 13 convolutional layers and 3 fully connected (FC) layers, making it both deep and systematically organized.

Core Architectural Concepts:

- All convolutional layers in VGG16 use 3×3 filters with a stride of 1 and padding of 1, allowing the model to extract intricate spatial details while preserving the feature map dimensions.
- The consistent use of small filters across the network enables the gradual formation of hierarchical features, from edges and textures to complex object parts.
- Each group of convolutional layers (termed as a *block*) is followed by a 2×2 max pooling layer with a stride of 2, which reduces spatial resolution while retaining crucial features.

Layer-wise Structure:

- Block 1: Two convolutional layers with 64 filters → max pooling
- Block 2: Two convolutional layers with 128 filters → max pooling
- Block 3: Three convolutional layers with 256 filters → max pooling

- Block 4: Three convolutional layers with 512 filters → max pooling
- Block 5: Three convolutional layers with 512 filters → max pooling

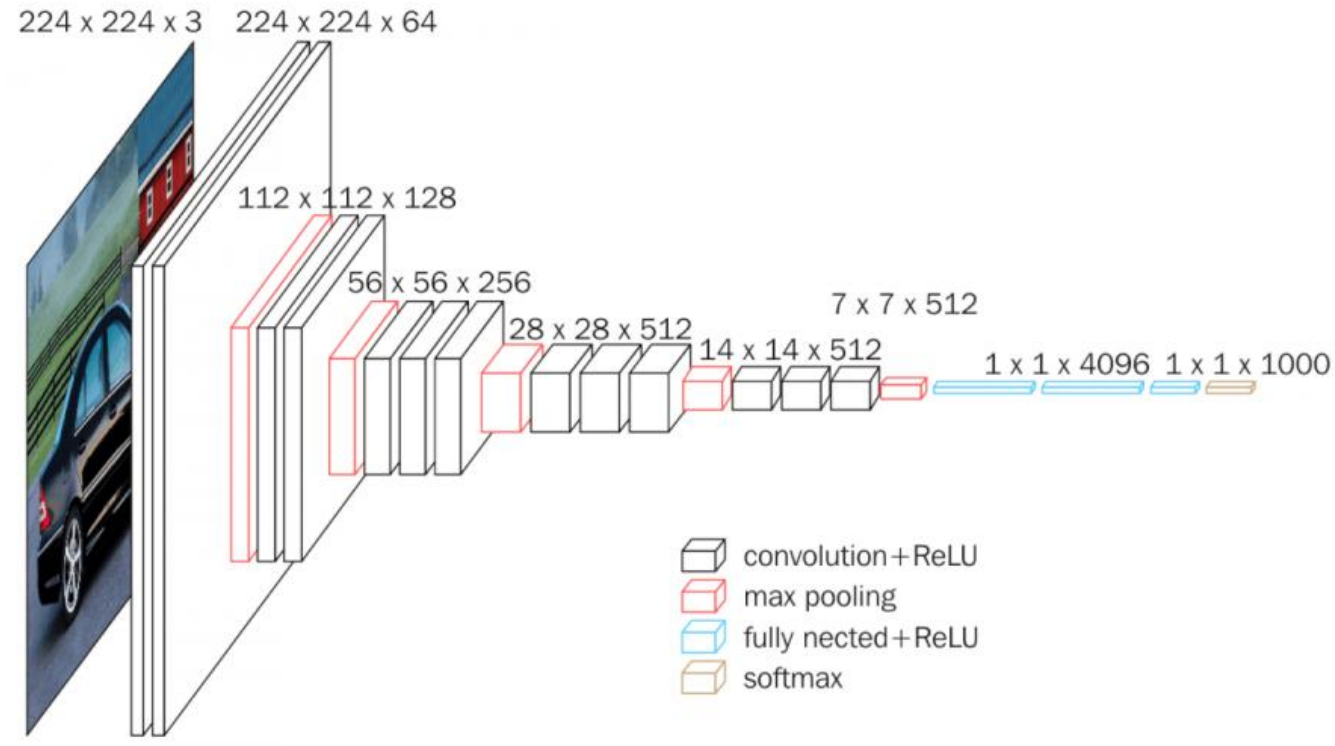
After these convolutional blocks, the output is flattened and passed through the following fully connected layers:

- First two FC layers: Each has 4096 neurons with ReLU activation.
- Final dense layer: Originally designed with 1000 neurons for ImageNet, but modified here to predict 10 animal classes.
- The classification is performed via a softmax layer, which outputs class probabilities.

Performance and Practical Considerations:

While VGG16 achieved exceptional results on standard image classification tasks, it comes with a high computational cost. With approximately 138 million parameters, the model demands significant memory (~93 MB per image) and processing power, making it less suitable for real-time or mobile deployments.

Nonetheless, its well-balanced and clean design has made VGG16 a reference point in the evolution of CNNs. It has influenced many subsequent deep learning models, both in terms of architectural style and performance benchmarking.



GoogLeNet (Inception V1): Architecture Overview

GoogLeNet, also referred to as Inception V1, was developed by a team of researchers at Google in 2014 and became widely recognized after winning the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2014. It significantly outperformed earlier models such as AlexNet and ZFNet, delivering superior classification accuracy while being computationally more efficient.

Key Innovations:

At the heart of GoogLeNet is the Inception module, a novel architectural component designed to handle features at multiple scales. Each Inception module applies multiple operations in parallel:

- 1×1, 3×3, and 5×5 convolutions
- 3×3 max pooling

The outputs of these parallel operations are concatenated along the channel dimension, enabling the model to capture both fine and coarse features in the same layer.

One of the crucial aspects of the design is the use of 1×1 convolutions, which serve two purposes:

1. Feature transformation: Adding non-linearity.
2. Dimensionality reduction: Reducing the number of input channels before applying larger kernels, thus significantly lowering computational cost.

Architectural Design:

Unlike traditional CNNs that end with fully connected layers, GoogLeNet employs Global Average Pooling in place of dense layers. This operation takes the average of each feature map, greatly reducing the number of parameters and improving the model's ability to generalize.

To enhance training in such a deep network, auxiliary classifiers are inserted at intermediate layers. These act like smaller classifiers that compute their own loss and contribute to backpropagation during training, helping maintain strong gradient signals in early layers. They also act as a form of regularization.

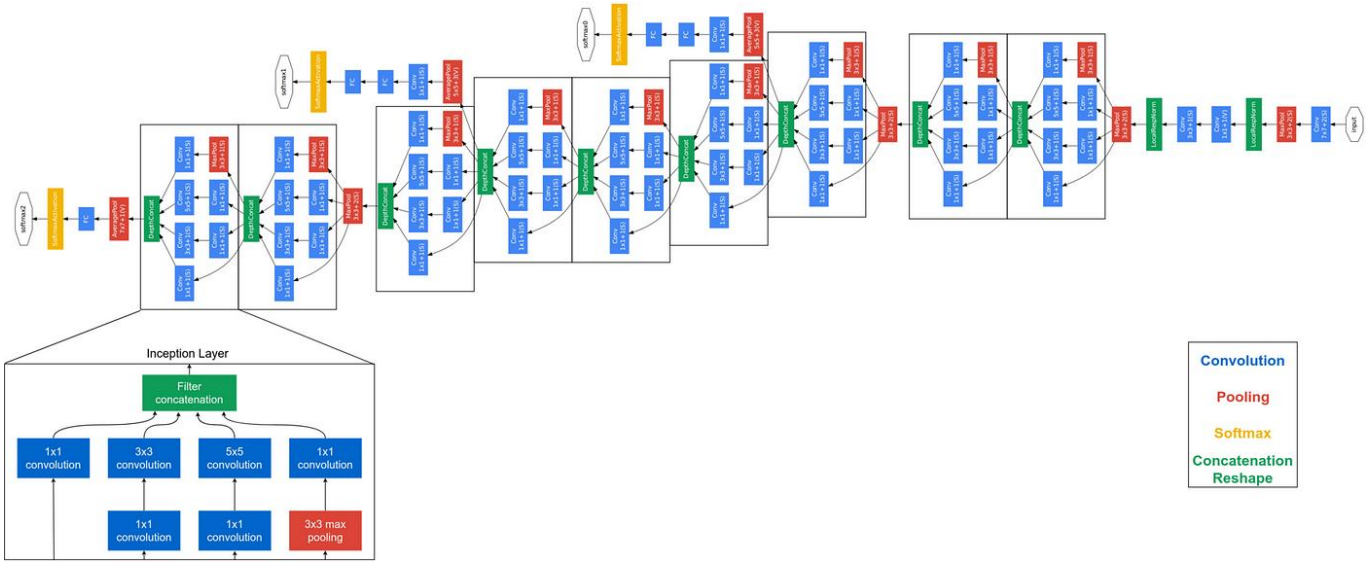
Summary of Specifications:

- Depth: 22 layers with learnable parameters
- Parameter Count: Approximately 5 million, making it very efficient for its depth

- Input Size: Designed for 224×224 RGB images
- Activation Function: Uses ReLU across all layers

Impact and Legacy:

GoogLeNet's modular design laid the foundation for more advanced variants like Inception V2 and V3, and inspired further innovations in scalable and efficient CNN architectures. Its effective combination of depth, computational efficiency, and multi-scale feature extraction continues to influence deep learning model design today.



3.5. Description of the Algorithms

3.5.1 ZFNet: Architecture and Key Features

ZFNet, introduced by Matthew Zeiler and Rob Fergus, was developed as an enhancement over the earlier AlexNet model. One of its most significant contributions to deep learning was the use of deconvolutional visualization techniques. These methods allowed researchers to better interpret the internal workings of convolutional neural networks by visualizing the activations of different layers, helping to guide architectural improvements.

Core Architectural Features:

- Structural Similarity to AlexNet:
  ZFNet retains the foundational layout of AlexNet, consisting of five convolutional layers followed by three fully connected layers.
- Refined Convolutional Parameters:
  The first convolutional layer in ZFNet reduces the filter size from 11×11 (in AlexNet) to 7×7, and decreases the stride from 4 to 2. These adjustments allow for higher spatial resolution in the feature maps, leading to more detailed feature extraction in early layers.
- Activation and Pooling:
  Like its predecessor, ZFNet utilizes the ReLU activation function and max pooling layers to

introduce non-linearity and reduce dimensionality while preserving important spatial features.

- Regularization:
  To reduce overfitting, dropout layers are applied to the fully connected layers, randomly deactivating neurons during training to encourage robustness and generalization.

Significance:

Beyond its performance improvements, ZFNet is especially notable for pioneering visualization-driven CNN design. By interpreting how filters respond to input data, the model highlighted the importance of network transparency and helped set a precedent for interpretable AI in convolutional architectures.

Type	Output Shape	Kernel/Stride	Params
Conv2D	(109, 109, 96)	7×7 / 2	14,208
ReLU	(109, 109, 96)	-	0
MaxPooling2D	(54, 54, 96)	3×3 / 2	0
BatchNorm	(54, 54, 96)	-	384
Conv2D	(25, 25, 256)	5×5 / 2	614,656
ReLU	(25, 25, 256)	-	0
MaxPooling2D	(12, 12, 256)	3×3 / 2	0
BatchNorm	(12, 12, 256)	-	1,024
Conv2D	(10, 10, 512)	3×3 / 1	1,180,160
ReLU	(10, 10, 512)	-	0
BatchNorm	(10, 10, 512)	-	2,048
Conv2D	(8, 8, 1024)	3×3 / 1	4,719,616
ReLU	(8, 8, 1024)	-	0
BatchNorm	(8, 8, 1024)	-	4,096
Conv2D	(6, 6, 512)	3×3 / 1	4,719,104
ReLU	(6, 6, 512)	-	0
MaxPooling2D	(3, 3, 512)	3×3 / 2	0
BatchNorm	(3, 3, 512)	-	2,048
Flatten	(4608,)	-	0
Dense (fc1)	(4096,)	-	18,878,464
ReLU	(4096,)	-	0
Dropout (40%)	(4096,)	-	0
BatchNorm	(4096,)	-	16,384
Dense (fc2)	(4096,)	-	16,781,312
ReLU	(4096,)	-	0
Dropout (40%)	(4096,)	-	0
BatchNorm	(4096,)	-	16,384
Dense (fc3)	(1000,)	-	4,097,000
ReLU	(1000,)	-	0
Dropout (40%)	(1000,)	-	0
BatchNorm	(1000,)	-	4,000
Dense (output)	(10,)	-	10,010
Softmax	(10,)	-	0

Table 3.5.1: ZFNet Architecture summary

- 3.5.2 GoogLeNet (Inception v1): Architecture and Innovations
- GoogLeNet, developed by Szegedy et al., was introduced in 2014 and represents a major evolution in convolutional neural networks. As part of the Inception architecture family, this model won the ILSVRC 2014 competition by achieving high accuracy with significantly fewer

parameters than its predecessors. The hallmark of GoogLeNet is the introduction of the Inception module, which enables efficient multi-scale processing within the network.

- Key Architectural Features:
- Deep Architecture:
  The model consists of 22 parameterized layers, which makes it substantially deeper than previous networks such as AlexNet and ZFNet. Due to this depth, it is often referred to as Inception v1.
- Inception Modules:
  At the core of GoogLeNet is the Inception module, which allows the network to simultaneously apply:
- 1×1 convolutions for dimensionality reduction and efficient computation.
- 3×3 and 5×5 convolutions for capturing spatial features at multiple scales.
- 3×3 max pooling for spatial abstraction.
  The outputs of these operations are concatenated along the depth axis, allowing the network to capture diverse feature representations without significant computational overhead.
- Dimensionality Reduction:
  The strategic use of 1×1 convolutions before larger filters not only acts as a feature extractor but also reduces the number of input channels, thus significantly cutting down on computational cost.
- Global Average Pooling:
  Rather than using traditional fully connected layers at the end, GoogLeNet employs Global Average Pooling, which computes the average of each feature map. This approach drastically reduces the number of trainable parameters and helps prevent overfitting.
- Auxiliary Classifiers:
  To address the vanishing gradient problem in deep networks, auxiliary classifiers are placed at intermediate layers. These classifiers calculate additional loss during training, thereby improving gradient flow and serving as a form of regularization.
- Inception Module Design:
- Each Inception module processes the input through parallel convolutional and pooling paths:
- 1×1 convolution for channel-wise transformation,
- 3×3 and 5×5 convolutions to capture features at different receptive fields,
- 3×3 max pooling for down-sampling,
- All outputs are concatenated and passed to the next layer.
- Conclusion:

- GoogLeNet introduced a scalable and efficient architecture capable of deep learning without a dramatic increase in computational cost. Its modular design inspired further developments in later Inception versions (v2, v3, v4) and continues to influence modern CNN architectures.

Type	Patch Size / Stride	Output Size	Depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	Pool Proj	Params	Ops
Conv	7×7 / 2	112×112×64	1	-	-	-	-	-	-	2.7K	34M
Max Pool	3×3 / 2	56×56×64	0	-	-	-	-	-	-	-	360M
Conv	1×1 / 1	56×56×64	2	64	-	-	-	-	-	112K	
Conv	3×3 / 1	56×56×192	2	-	-	192	-	-	-	-	
Max Pool	3×3 / 2	28×28×192	0	-	-	-	-	-	-	-	
Inception (3a)	-	28×28×256	2	64	96	128	16	32	32	195K	128M
Inception (3b)	-	28×28×480	2	128	128	192	32	96	64	380K	304M
Max Pool	3×3 / 2	14×14×480	0	-	-	-	-	-	-	-	
Inception (4a)	-	14×14×512	2	192	96	208	16	48	64	364K	73M
Inception (4b)	-	14×14×512	2	160	112	224	24	64	64	437K	88M
Inception (4c)	-	14×14×512	2	128	128	256	24	64	64	463K	100M
Inception (4d)	-	14×14×528	2	112	144	288	32	64	64	580K	119M
Inception (4e)	-	14×14×832	2	256	160	320	32	128	128	840K	170M
Max Pool	3×3 / 2	7×7×832	0	-	-	-	-	-	-	-	
Inception (5a)	-	7×7×1024	2	256	160	320	32	128	128	1072K	54M
Inception (5b)	-	7×7×1024	2	384	192	384	48	128	128	1388K	71M
Avg Pool	7×7 / 1	1×1×1024	1	-	-	-	-	-	-	-	
Dropout (40%)	-	1×1×1024	1	-	-	-	-	-	-	-	
Linear (FC)	-	1×1×1000	1	-	-	-	-	-	-	1000K	1M
Softmax	-	1×1×1000	0	-	-	-	-	-	-	-	

Table 3.5.2: GoogLeNet Architecture

- 3.5.3 VGG16: Depth-Oriented and Uniform CNN Architecture
- VGG16, developed by Karen Simonyan and Andrew Zisserman, is a deep convolutional neural network architecture that emphasizes simplicity and depth. It was introduced in 2014 as part of the Visual Geometry Group (VGG) at the University of Oxford. The network is known for its elegant and consistent structure, relying entirely on small-sized filters and a deep layer stack to achieve high performance in image classification tasks.
- Key Characteristics:
- Consistent Use of 3×3 Convolutions:
  All convolutional layers in VGG16 apply 3×3 kernels with a stride of 1 and padding of 1. This choice maintains the spatial resolution of the feature maps while enabling the network to learn more abstract features through depth.
- Max Pooling Layers:
  After every few convolutional layers, 2×2 max pooling is applied with a stride of 2. This progressively reduces the spatial dimensions of the feature maps and contributes to translation invariance.
- Activation and Fully Connected Layers:
  Each convolutional operation is followed by a ReLU activation function to introduce non-

linearity. At the end of the network, three fully connected layers (two with 4096 neurons and one output layer) are used to perform classification.

- Simple and Modular Design:
  The architecture is highly uniform and straightforward, making it easier to understand, implement, and modify for transfer learning or other purposes. The model consists of 13 convolutional layers and 3 dense layers, totaling 16 weight layers.
- Conclusion:
- VGG16's deep and systematic design allows it to capture complex features across multiple levels of abstraction. Despite its high number of parameters (~138 million), the model's clean and repetitive structure makes it a popular choice for benchmark comparisons and transfer learning in various vision tasks.

Layer (type)	Output Shape	Kernel / Stride	Param #
InputLayer	(224, 224, 3)	– / –	0
Conv2D (block1_conv1)	(224, 224, 64)	3×3 / 1×1	1,792
Conv2D (block1_conv2)	(224, 224, 64)	3×3 / 1×1	36,928
MaxPooling2D (block1_pool)	(112, 112, 64)	2×2 / 2×2	0
Conv2D (block2_conv1)	(112, 112, 128)	3×3 / 1×1	73,856
Conv2D (block2_conv2)	(112, 112, 128)	3×3 / 1×1	147,584
MaxPooling2D (block2_pool)	(56, 56, 128)	2×2 / 2×2	0
Conv2D (block3_conv1)	(56, 56, 256)	3×3 / 1×1	295,168
Conv2D (block3_conv2)	(56, 56, 256)	3×3 / 1×1	590,080
Conv2D (block3_conv3)	(56, 56, 256)	3×3 / 1×1	590,080
MaxPooling2D (block3_pool)	(28, 28, 256)	2×2 / 2×2	0
Conv2D (block4_conv1)	(28, 28, 512)	3×3 / 1×1	1,180,160
Conv2D (block4_conv2)	(28, 28, 512)	3×3 / 1×1	2,359,808
Conv2D (block4_conv3)	(28, 28, 512)	3×3 / 1×1	2,359,808
MaxPooling2D (block4_pool)	(14, 14, 512)	2×2 / 2×2	0
Conv2D (block5_conv1)	(14, 14, 512)	3×3 / 1×1	2,359,808
Conv2D (block5_conv2)	(14, 14, 512)	3×3 / 1×1	2,359,808
Conv2D (block5_conv3)	(14, 14, 512)	3×3 / 1×1	2,359,808
MaxPooling2D (block5_pool)	(7, 7, 512)	2×2 / 2×2	0
Flatten	(25088)	– / –	0
Dense (fc1)	(4096)	– / –	102,764,544
Dropout (dropout1)	(4096)	– / –	0
Dense (fc2)	(4096)	– / –	16,781,312
Dropout (dropout2)	(4096)	– / –	0
Dense (predictions)	(10)	– / –	40,970

Table 1.5.3: Architecture summary of VGG16

Chapter 4 – Implementation Details

4.1 Programming Languages and Frameworks

This project has been developed primarily using Python, chosen for its simplicity, readability, and vast ecosystem of libraries that support data science and deep learning workflows. Below are the core libraries and frameworks utilized:

4.1.1 TensorFlow/Keras

TensorFlow is an open-source platform for machine learning, and Keras acts as its high-level API, simplifying the process of building and training deep neural networks. It was used to implement and manage all CNN models featured in this project.

4.1.2 NumPy

NumPy provides essential tools for numerical computation, including support for multi-dimensional arrays and matrix operations. It plays a crucial role in handling and manipulating image data efficiently during preprocessing and model computations.

4.1.3 Pandas

Pandas offers flexible data structures such as DataFrames, which streamline dataset loading, manipulation, and statistical analysis. It is particularly useful during preprocessing and result evaluation phases.

4.1.4 Matplotlib & Seaborn

These are Python's primary visualization libraries. Matplotlib enables the creation of static, interactive, and animated plots, while Seaborn offers high-level interfaces for visually appealing statistical graphics. Both were used to plot training metrics like accuracy and loss.

4.1.5 Scikit-learn

Scikit-learn is a machine learning library that provides tools for evaluating model performance. It was employed to compute accuracy, precision, recall, F1-score, and generate confusion matrices for thorough performance analysis.

4.2 Description of Code Modules

To maintain clarity and modularity, the codebase has been organized into individual Jupyter Notebooks, each handling specific components of the animal species classification workflow. This separation supports iterative development and experiment tracking.

4.2.1 GoogLeNet.ipynb

This notebook focuses on building and training the GoogLeNet (Inception V1) architecture. It includes preprocessing steps, model configuration, training routines, and evaluation of results with visual analysis.

4.2.2 VGG16.ipynb

This module implements the VGG16 model, detailing the complete lifecycle—from data handling and training to evaluation and visualization of metrics relevant to model performance.

4.2.3 ZFNet.ipynb

This notebook contains the implementation of the ZFNet architecture. It handles data loading, architectural setup, training, and evaluation specific to this model.

4.2.4 ModelTest.ipynb

Used to compare the performance of all trained models (GoogLeNet, VGG16, ZFNet), this notebook loads pretrained weights and performs predictions on the test set. It then computes evaluation metrics to aid in comparative analysis.

4.2.5 training_Logs.ipynb

This module logs training metrics such as accuracy and loss across epochs. It provides visual feedback to detect overfitting or underfitting and assess training consistency.

4.3 System Workflow

The overall system follows a pipeline-based architecture that ensures smooth data flow from ingestion to final evaluation. Each stage of the pipeline is handled by dedicated scripts and notebooks.

4.3.1 Data Ingestion

A script named data_loader.py is responsible for loading images from a structured dataset directory. Each subfolder represents a different animal species, allowing for class-wise separation.

4.3.2 Data Preprocessing

Preprocessing steps are also handled within data_loader.py. These include:

- Resizing all input images to a uniform size
- Normalizing pixel values to the [0, 1] range

- Encoding class labels into one-hot vectors
  The processed data is split into training, validation, and test sets.

4.3.3 Model Initialization

All model architectures (ZFNet, VGG16, GoogLeNet) are defined in model_architectures.py. This module outlines the layer structures, activation functions, and compilation settings required for training.

4.3.4 Model Training

The training.py script trains each model using the training dataset. It performs forward and backward propagation, computes loss, updates weights via optimizers, and validates performance using the validation set.

4.3.5 Model Evaluation

Post-training evaluation is conducted through evaluation.py, which assesses model performance using the test set. Key metrics like precision, recall, F1-score, and accuracy are computed, and confusion matrices are generated.

4.3.6 Comparative Analysis

This step compares all trained models to identify the most efficient and accurate architecture. Factors such as accuracy, model size, and inference time are analyzed to understand trade-offs.

4.3.7 Visualization Tools

Utility functions in utils.py help visualize training histories, such as accuracy and loss plots. These visual aids are essential for interpreting model behavior and diagnosing training anomalies.

Chapter 5 – Results and Discussion

5.1. Evaluation Metrics

For the evaluation of the models, accuracy was calculated for each of them. Based on the results presented in the figure, GoogLeNet achieved the highest accuracy of 0.97, followed by VGG16 with 0.88, while ZFNet had the lowest performance with an accuracy of 0.45.

ZFNet:

- Accuracy: 0.45
- Best classification performance:
    - Spider: 900 correctly classified
    - Dog: 787
    - Chicken: 496
- Key misclassifications:
    - Cat frequently misclassified as Dog (72 instances)
    - Cow misclassified as Dog (41)
    - Horse misclassified as Dog (55)
- Observation:
  The model performs reasonably well on certain classes but exhibits considerable confusion among semantically similar animals such as cats and dogs, as well as cows and horses. Overall, this model showed the weakest performance among the three.

VGG16:

- Accuracy: 0.88
- Best classification performance:
    - Spider: 873
    - Dog: 801
    - Chicken: 553
- Key misclassifications:
    - Cat misclassified as Dog (58 instances), fewer than in other models
    - Some confusion between Cow and Sheep
- Observation:
  The model demonstrated stronger class separation with fewer off-diagonal values in the confusion matrix. Despite a lower number of training epochs (26), likely due to hardware limitations, the classification results show relatively well-isolated predictions and minimal overlap between similar classes.

GoogLeNet:

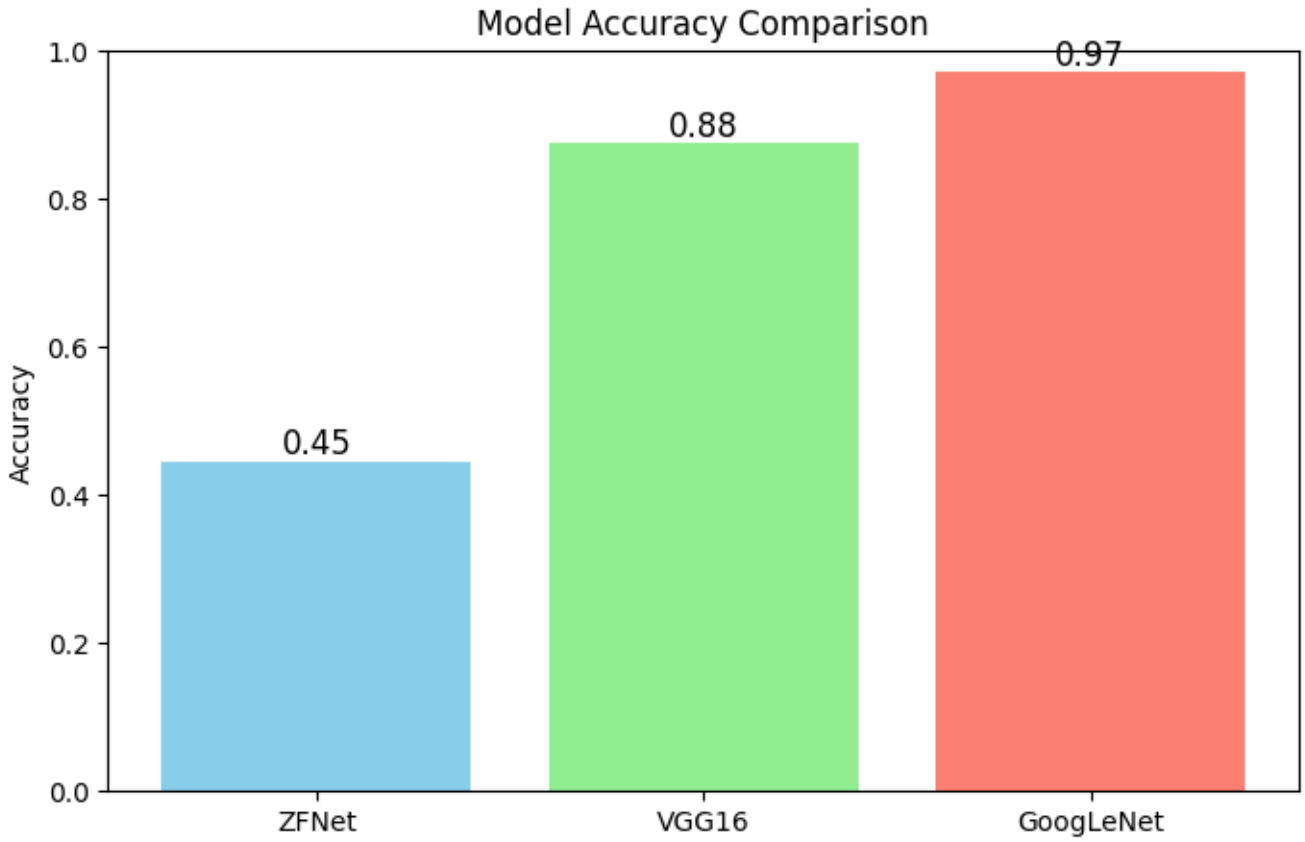
- Accuracy: 0.97
- Best classification performance:
    - Spider: 806
    - Dog: 696
    - Chicken: 474
- Key misclassifications:
    - Higher confusion in certain categories, such as Cat misclassified as Dog (115 instances)
    - Errors were more broadly distributed across different classes
- Observation:
  While this model achieved the highest overall accuracy, the confusion matrix indicates more dispersed misclassifications compared to VGG16. This suggests a trade-off between general accuracy and class-specific precision.

5.2. Results

The results from training and evaluating the models on the test dataset are summarized as follows:

Model	Accuracy
ZFNet	45%
VGG16	88%
GoogLeNet	97%

The accuracy comparison is also visualized in the bar chart where GoogLeNet significantly outperforms the others, followed by VGG16, while ZFNet lags behind.

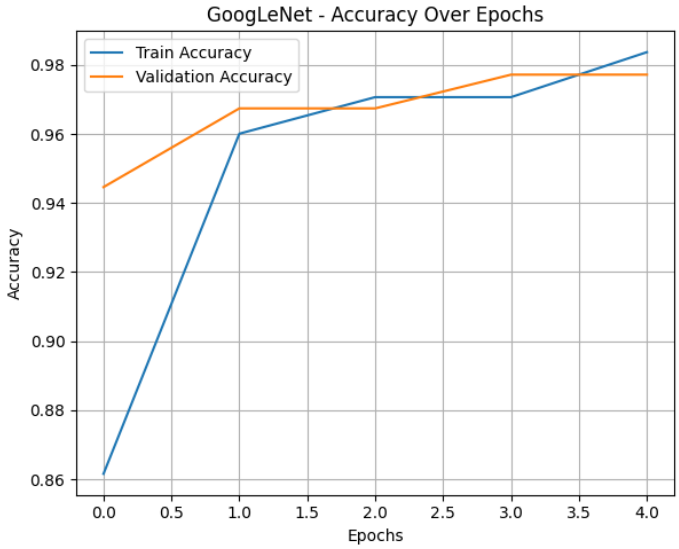


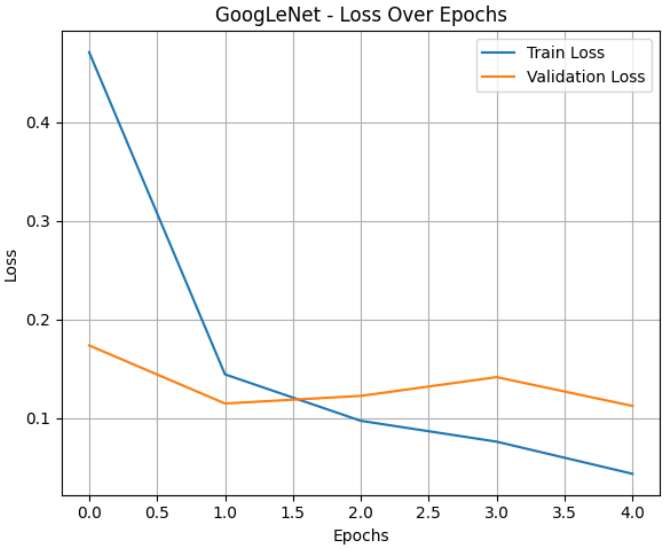
5.3. Discussion

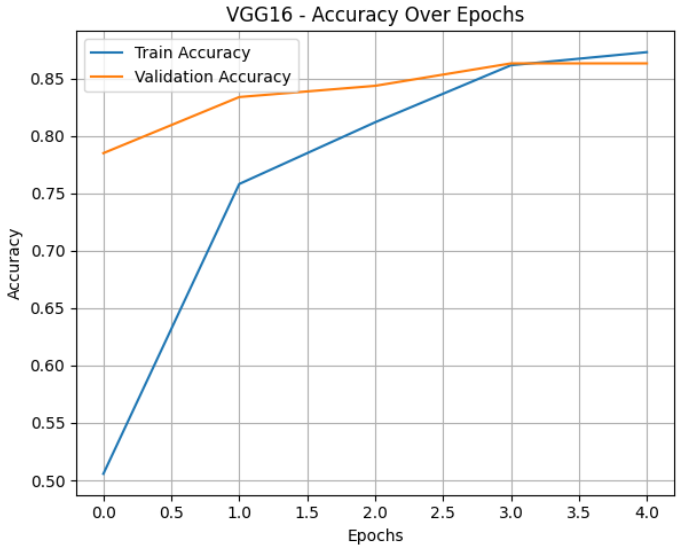
The experimental results reveal clear distinctions in model performance:

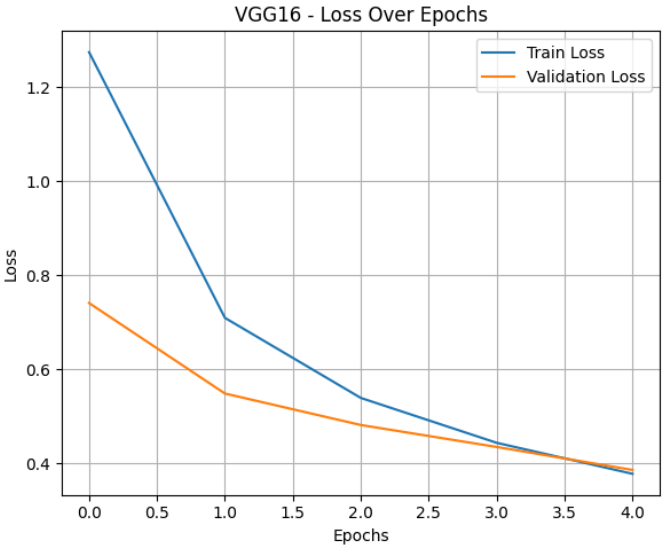
- ZFNet underperforms with an accuracy of 45%, indicating limited capability in learning discriminative features. This is likely due to its relatively shallow architecture and fewer learnable parameters. Further training curve analysis may reveal issues of overfitting or underfitting.
- VGG16 achieves a robust 88% accuracy, demonstrating strong classification performance. Its deep structure with uniform convolutional layers allows it to learn complex patterns effectively. However, this comes at the cost of increased computational resources, making it more suitable for systems with higher processing power.
- GoogLeNet emerges as the best model with 97% accuracy. The use of Inception modules enables it to extract multi-scale features efficiently, balancing high accuracy with reduced parameter count. This makes GoogLeNet not only accurate but also computationally efficient, ideal for real-time or large-scale applications.

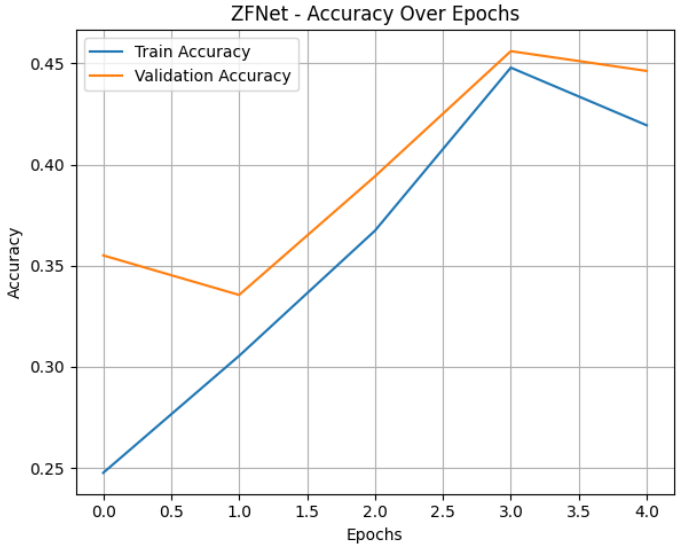
Overall, GoogLeNet provides the best trade-off between accuracy and efficiency, making it the most suitable architecture for animal species classification in this study. Future work can focus on fine-tuning this model or exploring newer architectures such as ResNet or EfficientNet for further improvement.

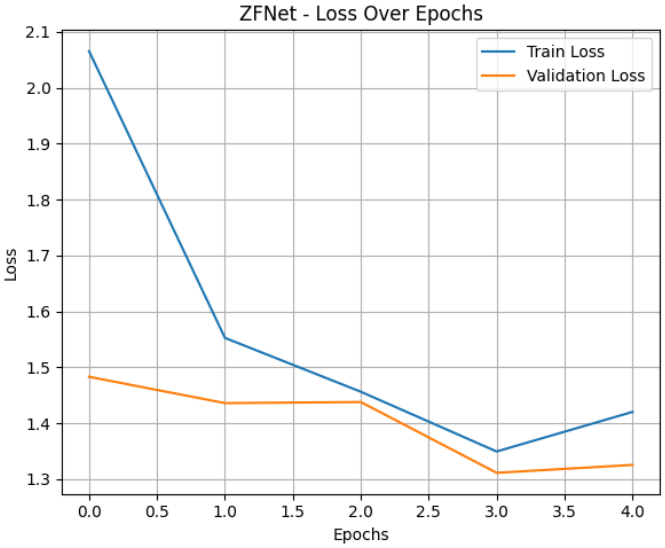

GoogLeNet - Accuracy Over Epochs
Train Accuracy
Validation Accuracy
Accuracy
Epochs


GoogLeNet - Loss Over Epochs
Train Loss
Validation Loss
Loss
Epochs


VGG16 - Accuracy Over Epochs
Train Accuracy
Validation Accuracy
Accuracy
Epochs


VGG16 - Loss Over Epochs
Train Loss
Validation Loss
Loss
Epochs


ZFNet - Accuracy Over Epochs
Train Accuracy
Validation Accuracy
Accuracy
Epochs


ZFNet - Loss Over Epochs
Train Loss
Validation Loss
Loss
Epochs

Chapter 6 – Conclusion & Future Work

6.1. Summary of Outcomes

The performance evaluation, as discussed in Sections 5.2 and 5.3, indicates that GoogLeNet yielded the highest classification accuracy of 0.97 after 5 training epochs. VGG16 followed with a respectable accuracy of 0.88. ZFNet, while computationally efficient, achieved a comparatively lower accuracy of 0.45 at 5 epochs.

It was observed that overfitting was more pronounced in the VGG16 model, attributable to its larger number of parameters and reduced training time. GoogLeNet exhibited moderate overfitting, while ZFNet maintained a more balanced generalization capacity. The exclusion of data augmentation techniques in the preprocessing phase likely contributed to overfitting, particularly in VGG16. Integrating such strategies is a critical area for future improvement.

6.2. Identified Limitations

Despite promising outcomes from convolutional neural network (CNN) architectures in animal classification tasks, several technical challenges remain:

- Class Imbalance:
  An uneven distribution of training samples can bias model learning. ZFNet's simpler structure may help mitigate overfitting to dominant classes but limits learning depth. VGG16, with its larger capacity, is more susceptible to imbalance-related overfitting. GoogLeNet, through its inception modules, exhibits enhanced flexibility in capturing minority class features. However, resolving imbalance requires targeted solutions such as weighted loss functions and oversampling strategies.

- Environmental Variability:
  Differences in lighting, backgrounds, and image quality can undermine model reliability. ZFNet, being shallower, is more prone to noise in early feature maps. Although VGG16's deeper layers provide some robustness, its early-layer sensitivity remains a concern. GoogLeNet, with its multi-scale feature extraction, demonstrates improved stability under varying conditions. Incorporating spatial attention mechanisms could further enhance resilience across architectures.

- Intra-Class Diversity:
  Variation within a species, such as different dog breeds, complicates classification. VGG16's depth supports detailed feature extraction, though success depends heavily on data volume. ZFNet lacks the depth to fully represent such diversity but offers useful insights for compact network design. GoogLeNet, with its varied convolutional filters, better accommodates intra-class variability. Expanding datasets and integrating meta-learning approaches are vital for addressing this issue.

- Visual Similarities Between Classes:
  Animals with similar appearances, such as cats and dogs or cows and sheep, require fine-grained feature discrimination. VGG16's capacity helps in capturing these nuances but also

increases the risk of overfitting. ZFNet's interpretability can assist in identifying distinguishing characteristics, while GoogLeNet's architecture enables multi-resolution feature analysis. Future work should explore the integration of attention-based modules and metric learning techniques for improved differentiation.

- Efficiency in Real-Time Scenarios:
  For applications with limited computational resources, model efficiency is crucial. VGG16 is computationally intensive and less suited to real-time use. ZFNet offers faster performance but sacrifices accuracy. GoogLeNet provides a balance between speed and precision. Research into pruning, quantization, and efficient network designs—especially those inspired by GoogLeNet—can facilitate deployment in real-world settings.

6.3. Proposed Future Enhancements

To address the aforementioned limitations, several improvements are recommended for future development:

- Enhanced Data Augmentation:
  Implement advanced augmentation methods, including synthetic image generation, to ensure better class representation and reduce model bias.
- Domain Adaptation and Transfer Learning:
  Employ techniques that adapt models to new environments or datasets with minimal retraining to increase robustness across domains.
- Integration of Attention Mechanisms:
  Introduce attention layers within CNNs to enable the model to focus on the most relevant regions in an image, thereby improving classification precision.
- Fine-Grained and Multimodal Analysis:
  Utilize fine-grained recognition strategies and consider incorporating multiple data modalities (e.g., visual and acoustic signals) to enhance class separability.
- Design of Lightweight Networks:
  Develop efficient models optimized for deployment on edge devices through techniques such as pruning, quantization, or distillation without significant loss in accuracy.
- Transparency and Ethical Considerations:
  Adopt explainability frameworks (e.g., Grad-CAM, SHAP) to make model decisions interpretable and ensure ethical deployment, particularly in wildlife or ecological monitoring applications.

References

1. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015. [Online]. Available: https://www.nature.com/articles/nature14539.
2. M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European Conference on Computer Vision*, 2014, pp. 818–833. Springer. [Online]. Available: https://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf.
3. K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014. [Online]. Available: https://arxiv.org/abs/1409.1556.
4. T. Islam et al., "Animal species recognition with deep convolutional neural networks from ecological camera trap images," *Proceedings of the Texas A&M Natural Resources Institute*, 2023. [Online]. Available: https://nri.tamu.edu/media/3697/2023_islam_animal-recognition-dcnn-camera-trap.pdf.
5. Kaggle Dataset: Animals10

Appendices

Additional code snippets

```
def plot_accuracy_loss(history, model_name):
    import matplotlib.pyplot as plt

    plt.figure(figsize=(12, 5))

    # Accuracy plot
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title(f'{model_name} - Accuracy Over Epochs')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)

    # Loss plot
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title(f'{model_name} - Loss Over Epochs')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()
```

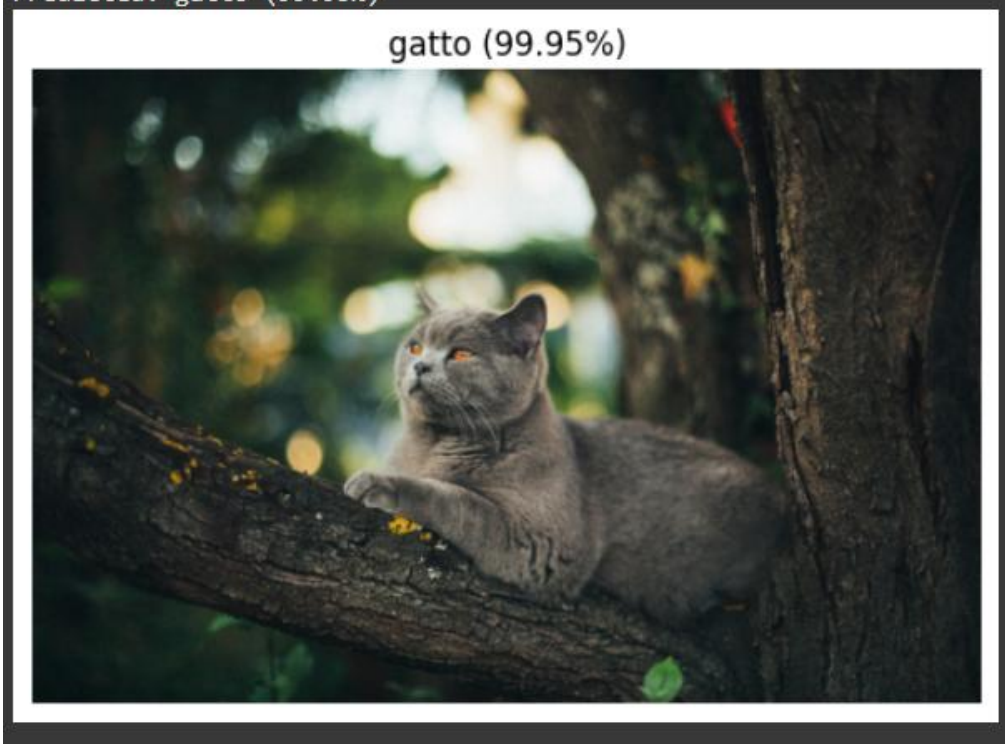
Screenshots or logs

```
test_image_path = "/content/drive/MyDrive/raw-img/farfalla/OIP-ZunYgEud0RqEjpnoLjtoXQHaFj.jpeg"
predict_image(test_image_path, googlenet, class_names)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
Predicted: farfalla (100.00%)
```

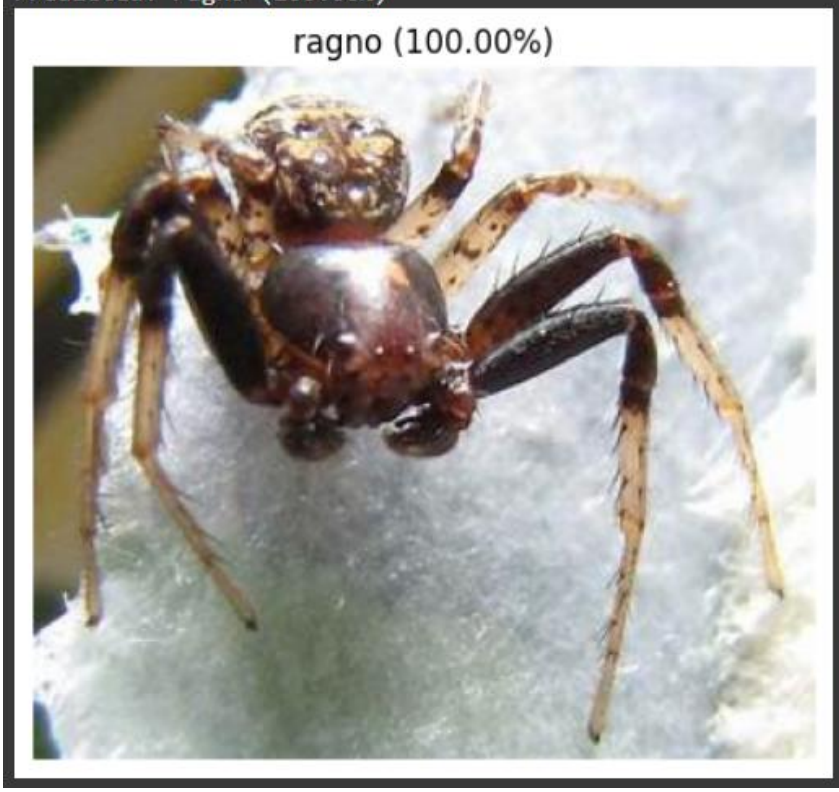

```
test_image_path = "/content/drive/MyDrive/raw-img/gatto/max-boettinger-486490-unsplash.jpg"
predict_image(test_image_path, googlenet, class_names)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
Predicted: gatto (99.95%)
```



```
test_image_path = "/content/drive/MyDrive/raw-img/ragno/OIP-Zy4Kdt2PTcAnCfTsViRJ7wHaGi.jpeg"
predict_image(test_image_path, googlenet, class_names)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
Predicted: ragno (100.00%)
```




```
# Predict on a new image
test_image_path = "/content/drive/MyDrive/raw-img/elefante/OIP-ZPRJ85QAzQ7hC4vp7vzErwHaE8.jpeg"
predict_image(test_image_path, googlenet, class_names)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 7s 7s/step
Predicted: elefante (100.00%)
```