[Thesis Title Here]

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of
Master of Science (MSc)
in
[Department Name]

[Your Name]
[Student ID]
[University Name]

Month, Year

Declaration

I hereby declare that the work presented in this thesis is my own and has not been submitted elsewhere for the award of any degree.

[Your Name]
[Date]

Abstract

Acknowledgement

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

1.2 Problem Statement

1.3 Objectives

1.4 Significance of the Study

1.5 Thesis Organization

Chapter 2

Theoretical Background

2.1 Overview

A collection of cells called colon polyps forms on the colon's epithelium. Colonic polyps can be

discovered and removed during a colonoscopy before they develop into cancer. However, around

one-fourth of the polyps can be missed due to their small size, location, or human mistake [10].

use a deep learning model capable of identifying and categorizing the polyp. One type of deep

learning methodology is convolutional neural networks. The theoretical underpinnings of colon

polyp identification and categorization were discussed in this chapter.

2.2 Deep Learning

One of the most common AI techniques used for processing big data is machine learning, a self-

adaptive algorithm that gets increasingly better analysis and patterns with experience or with newly

added data.

Traditional machine learning was confined is the way it processes data, as some functionalities

needed some exactly specific programming to perform some specific tasks. The traditional

machine-learning could not receive raw data as input and transform it into a suitable

understandable representation without the help of human brains. A machine-learning algorithm

required labeled/structured data to understand the differences between images of cats and dogs,

learn the classification and then produce output.

On the contrary, a subset of machine learning where algorithms are created and function similar

to those in machine learning, but there are numerous layers of these algorithms-each providing a

different interpretation to the data it feeds on. It did not require any labeled/structured data, as it

relied on the different outputs processed by each layer.

Deep learning is capable of using raw data and can automatically learn the features required to

perform the specific identification task. The learning method can be supervised, semi-supervised,

or unsupervised. This learning ability is based on stacking several non-linear modules as a stack

of multiple layers that convert the raw input data into a higher label more abstract representation

[5]. Each successive layer uses the output from the previous layer as input for the next layer. So,

it is like a cascade of multiple layers.

It requires high-end machines contrary to traditional Machine Learning algorithms. GPU has

become an integral part now to execute any Deep Learning algorithm. In traditional Machine

learning techniques, most of the applied features need to be identified by a domain expert in order

to reduce the complexity of the data and make patterns more visible to learning algorithms to work.

The biggest advantage of Deep Learning algorithms as discussed before are that they try to learn

high-level features from data in an incremental manner. This eliminates the need for domain

expertise and hardcore feature extraction.

Figure 2.1: Fig. 2.1 Machine Learning vs. Deep Learning

2.3 Neural Network

A neural network, more properly referred to as an 'artificial' neural network (ANN), is provided

by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielsen. He defines a neural

network as: "a computing system made up of a number of simple, highly interconnected processing

elements, which process information by their dynamic state response to external inputs." A

diagram of what one node might look like as shown in the graphic below.

Figure 2.2: Fig. 2.2 Simple Neural Network

Neural networks are typically organized in layers. Layers are made up of a number of

interconnected 'nodes' which contain an 'activation function'. Patterns are presented to the network

via the 'input layer', which communicates to one or more 'hidden layers' where the actual

processing is done via a system of weighted 'connections'. The hidden layers then link to an 'output

layer' where the answer is output as shown in the graphic below.

Figure 2.3: Fig. 2.3 Deep Neural Network with two hidden layers

A neural network is a set of connected neurons designed in layers:

a) Input Layer: Brings the initial data into the system for further processing by subsequent

layers of artificial neurons.

b) Hidden layer: A hidden layer is a layer between input layers and output layers. In the

hidden layer, artificial neurons take in a set of weighted inputs and give an output through

an activation function.

c) Output layer: In the program, the last layer of neurons that produces given outputs.

Neural networks may accomplish tasks that take hours or even days, such speech recognition and

picture identification, in a matter of minutes. A popular illustration of a neural network is the

search engine Google [11].

2.4 Convolutional Neural Network

Convolutional Neural Networks are very similar to ordinary Neural Networks from the previous

section: they are made up of neurons that have learnable weights and biases. Each neuron receives

some inputs, performs a dot product and optionally follows it with a non-linearity. The whole

network still expresses a single differentiable score function: from the raw image pixels on one

end to class scores at the other. And they still have a loss function (e.g., SVM/Softmax) on the last

(fully-connected) layer and all the tips/tricks we developed for learning regular Neural Networks

still apply.

A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of

the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of

neuron activations. In this example, the red input layer holds the image, so its width and height

would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume

of activations to another through a differentiable function. Here three main types of layers are used

to build ConvNet architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer

(exactly as seen in regular Neural Networks).

Figure 2.4: Fig. 2.4 Convolutional Neural Network

CNNs are used in autonomous number plate reading, photo recognition, and self-driving car

software. Convolutional neural networks have gained a lot of popularity because of their

adaptability to changes in data size, rotation, distortion, and other factors [12].

2.4.1 Convolutional Layer

The Conv layer is the core building block of a Convolutional Network that does most of the

computational heavy lifting. Convolution is the first layer to extract features from an input image.

Convolution preserves the relationship between pixels by learning image features using small

squares of input data. It is a mathematical operation that takes two inputs such as image matrix

and a filter or kernel.

Figure 2.5: Fig. 2.5 Convolution operation on an MxNx3 image matrix with a 3x3x3 Kernel

2.4.2 Activation Functions

• Step function: A step function is defined as

Figure 2.6: Fig. 2.6 Step function

Where the output is 1 if the value of x is greater than equal to zero and 0 if the value of x is less

than zero. As one can see a step function is non-differentiable at zero. Since the step function is

non-differentiable at zero hence it is not able to make progress with the gradient descent approach

and fails in the task of updating the weights.

To overcome, this problem sigmoid functions were introduced instead of the step function.

• Non-Linearity (ReLU): ReLU stands for the Rectified Linear Unit. This is the equation

for ReLU:

$\text{ReLU}(y) = \max(0, x)$ (2.1)

$$\text{ReLU}(y) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

The ReLU equation tells us this: If the input z is less than 0, set input equal to 0 and if the input is

greater than 0, set input equal to the input.

Figure 2.7: Fig. 2.7 ReLU activation function

• Softmax: Softmax is a very interesting activation function because it not only maps our

output to a [0,1] range but also maps each output in such a way that the total sum is 1. The

output of Softmax is, therefore, a probability distribution. The softmax function is often

used in the final layer of a neural network-based classifier. Such networks are commonly

trained under a log loss (or cross-entropy) regime, giving a non-linear variant of

multinomial logistic regression.

2.4.3 Batch Normalization

Batch normalization is a technique for training very deep neural networks that standardizes the

inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and

dramatically reducing the number of training epochs required to train deep networks. During

training time, a batch normalization layer does the following:

1) Calculate the mean and variance of the layer's input

$$\text{Batch mean:} \quad \mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \quad (2.3)$$

$$\text{Batch variance:} \quad \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2 \quad (2.4)$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad (2.5)$$

$$y_i = \gamma\,\hat{x}_i + \beta \quad (2.6)$$

2.4.4 Pooling Layer

It is common to periodically insert a Pooling layer in-between successive Conv layer in a ConvNet

architecture. Its function is to progressively reduce the spatial size of the representation to reduce

the number of parameters and computation in the network, and hence to also control overfitting.

The Pooling Layer operates independently on every depth slice of the input and resizes it spatially,

using the MAX operation. The most common form is a pooling layer with filters of size 2x2 applied

with a stride of 2 down samples every depth slice in the input by 2 along both width and height,

discarding 75

4 numbers (little 2x2 region in some depth slice). The depth dimension remains unchanged.

• Max-Pooling: Max pooling is used to reduce the image size by mapping the size of a given

window into a single result by taking the maximum value of the elements in the window.

Figure 2.8: Fig. 2.8 Max-Pooling

• Average-Pooling: It's the same as max-pooling except that it averages the windows

instead of picking the maximum value.

Figure 2.9: Fig. 2.9 Average-Pooling

2.4.5 Dropout

Dropout is a technique used to prevent a model from overfitting. Dropout works by randomly

setting the outgoing edges of hidden units. A fully connected layer occupies most of the

parameters, and hence, neurons develop co-dependency amongst each other during training which

curbs the individual power of each neuron leading to over-fitting of training data.

2.4.6 Flatten Layer

Flatten is the function that converts the pooled feature map to a single column that is passed to the

fully connected layer. Dense adds the fully connected layer to the neural network. Once the pooled

featured map is obtained, the next step is to flatten it. Flattening involves transforming the entire

pooled feature map matrix into a single column which is then fed to the neural network for

processing.

2.4.7 Fully-Connected layer

Neurons in a fully connected layer have full connections to all activations in the previous layer, as

seen in regular Neural Networks. Their activations can hence be computed with a matrix

multiplication followed by a bias offset. Matrix is flattened into the vector and fed it into a fully

connected layer like a neural network.

The fully connected (FC) layer in the CNN represents the feature vector for the input. This feature

vector/tensor/layer holds information that is vital to the input. When the network gets trained, this

feature vector is then further used for classification, regression, or input into other networks like

RNN for translating into other types of output, etc. It is also being used as an encoded vector.

During training, this feature vector is being used to determine the loss, and help the network to get

train.

The convolution layers before the FC layer(s) hold information regarding local features in the input

image such as edges, blobs, shapes, etc. Each conv layer holds several filters that represent one of

the local features. The FC layer holds composite and aggregated information from all the conv

layers that matters the most.

Figure 2.10: Fig. 2.10 Flatten layer and Fully Connected layer

2.5 VGG16

ConvNets are a kind of ANN. A network of CNN is composed of many hidden layers, an output

layer, and an input layer. CNN like the VGG16 model, are considered to be among the top

computer vision models (CVM) currently in use. The model's creators evaluated the networks and

employed a tiny (3 x 3) convolution filter architecture to increase the depth, demonstrating a

significant advancement over the prior state-of-the- art configuration [13].

Figure 2.11: Fig. 2.11 VGG16 network architecture

2.5.1 VGG16 Architecture

• Input: The VGGNet model takes in an image of size 224×224. To ensure consistency

during the ImageNet competition, it's cut 224×224 section from the middle of each image

as the input.

• Convolutional layers: VGG makes use of 3×3 filters in its convolutional layers, which is

the smallest possible size. Furthermore, the input undergoes 1×1 convolution filter.

• ReLU activation: One of the most significant advancements in reducing training time for

AlexNet was the implementation of the Rectified Linear Unit Activation Function (ReLU).

This innovative component behaves as a linear function, producing a zero value for negative inputs and an identical output for positive inputs. To preserve the spatial resolution during convolution, VGG employs a constant stride of 1 pixel.

• Hidden Layers: All of the hidden layers in VGG network utilize ReLU, as opposed to the

implementation of Local Response Normalization in AlexNet. While the latter has minimal

effect on overall accuracy, it significantly increases training time and memory consumption.

• Pooling layers: By including a pooling layer after a set of convolutional layers, the feature

maps generated during each stage of convolution undergo a reduction in the number of

parameters and dimensionality. This is crucial, especially considering the rapid progression

from 64 to 128, then to 256, and eventually to 512 filters in the concluding layers. Therefore, utilizing pooling is vital for optimizing the performance of the model.

• Fully connected layers: VGGNet consists of three interconnected layers, each with a

distinct purpose. The first and second layers boast an impressive 4096 channels, while the

third layer contains exactly 1000 channels, one for every class.

Figure 2.12: Fig. 2.12 VGG16 architecture Map

2.5.2 Object Localization in Image

We must replace the class score with the bounding box location coordinates during localization.

A bounding box's location may be expressed as a 4-dimensional vector with the height, width, and

center values corresponding to x and y. Both class-specific bounding boxes (which produce a 4-

thousand-parameter vector) and shared bounding boxes (which produce a 4-parameter vector) are

examples of the two forms of localization architecture. The article tested both strategies using the

VGG16 (D) architecture. Additionally, in this case, we must switch from classification loss

functions to regression loss functions, such as mean square error (MSE), which punish the

difference between the predicted and actual losses.

2.5.3 Results

VGG16 performed among the top designs in the 2014 ILSVRC competition. It was only surpassed

by GoogLeNet, which had a classification error of 6.66

a remarkable 7.32

localization error of 25.32

2.5.4 Limitation

• The original VGG model requires two to three weeks of training on an Nvidia Titan GPU.

• VGG16 prepared picture. The size of net weights is 528 MB. It is therefore inefficient

because it uses a large quantity of storage space and bandwidth.

• An explosion of gradient problems arises with 138 million parameters.

2.6 ResNet50

Convolutional neural networks (CNNs) of the ResNet50 variety have completely changed the way

we approach deep learning. Kaiming He et al. initially presented it at Microsoft Research Asia in

2015. ResNet, an acronym for residual network, describes the remaining building components that

comprise the network's design. Deep residual learning is the foundation of ResNet50, which

enables the training of extremely deep networks with hundreds of layers. A startling finding in

deep learning research led to the creation of the ResNet architecture: a neural network's

performance does not necessarily improve with the addition of more layers. This was surprising

because a network should be able to learn more information in addition to what the prior network

knew when a new layer is added. The ResNet group, under the direction of Kaiming He, created a

unique design using skip connections to solve this problem. The network was able to learn more

accurate representations of the input data thanks to these connections, which allowed knowledge

from previous layers to be preserved. They were able to train networks with up to 152 layers using

the ResNet design. With a 3.57

contests, such as the ILSVRC and COCO object detection tasks, ResNet's results were

revolutionary. This illustrated the ResNet architecture's strength and promise for use in deep

learning studies and applications [14].

2.6.1 ResNet50 Architecture

ResNet50 is made up of 50 layers split up into 5 blocks, each of which has a collection of residual

blocks. The network may learn more accurate representations of the input data by using the residual

Figure 2.13: Fig. 2.13 ResNet50 Architecture

blocks, which enable the retention of information from previous levels [14]. The primary ResNET

components are as follows.

• Convolutional Layers: Convolution on the input picture is carried out by the network's

first layer, the convolutional layer. A max-pooling layer that down samples the

convolutional layer's output comes next. Following the max-pooling layer, a number of

residual blocks are applied to the output.

• Residual Blocks: The two convolutional layers that comprise a residual block are followed

by a batch normalization layer and a rectified linear unit activation function. Next, the

output of the second convolutional layer is mixed with the input of the residual block,

which is then exposed to an additional ReLU activation function. The output of the residual

block is then passed on to the next block.

• Fully Connected Layer: The fully connected layer, the final layer in the net- work, maps

the output of the final residual block to the output classes. In the completely connected

layer, the number of neurons and the number of output classes are equal.

2.6.2 Concept of Skip Connection

One important component of ResNet50 is skip connections, sometimes referred to as identity

connections. They make it possible to keep data from previous layers, which aids in the network's

ability to learn more accurate representations of the input. By combining the output of an earlier

layer with the output of a later layer, skip connections are created.

Figure 2.14: Fig. 2.14 Skip Connection

2.6.3 Advantages of ResNet50 Over Other Networks

ResNet50 has several advantages over other networks. One of its main advantages is its ability to

train extraordinarily intricate networks with hundreds of layers. It is possible to retain data from

earlier levels by using skip connections and remaining blocks. Another advantage of the model is

ResNet50's ability to generate state-of-the-art results in a range of image-related tasks, such as

object recognition, image classification, and picture segmentation.

2.7 DenseNetv3

DenseNet V3 is a convolutional neural network architecture that is based on the idea of densely

connected layers. Each layer in a DenseNet V3 receives input from all the previous layers, which

allows for better feature reuse and information flow. DenseNet V3 consists of several dense blocks,

each containing a fixed number of convolutional layers, followed by transition layers that reduce

the spatial dimensions and the number of feature maps [15]. DenseNet V3 is an improved version

of DenseNet. DenseNet V3 uses a modified version of DenseNetv3, which is a 121-layer DenseNet

with four dense blocks and three transition layers. DenseNet V3 also incorporates some techniques

from YOLO- V3, a state-of-the-art object detection model, to enhance its performance on multi-

scale remote sensing target detection. Some of these techniques include:

• Using a larger input size of 608 x 608 pixels, instead of the original 224 x 224 pixels, to

capture more details of the targets.

• Adding a fourth detection scale at the end of the network, which outputs bounding boxes

at a finer resolution of 76 x 76, instead of the original 19 x 19.

• Replacing the 1 x 1 convolution layer in the transition layer with a DenseNet layer, which

increases the number of feature maps and the diversity of features.

• Applying a leaky ReLU activation function with a negative slope of 0.1, instead of a regular

ReLU, to avoid gradient vanishing and improve the non-linearity of the network.

2.7.1 DenseNetv3 Architecture

In a DenseNet architecture, each layer is connected to every other layer, hence the name Densely

Connected Convolutional Network. For L layers, there are L(L+1)/2 direct connections. For each

layer, the feature maps of all the preceding layers are used as inputs, and its own feature maps are

used as input for each subsequent layer.

This is really it, as simple as this may sound, DenseNets essentially connect every layer to every

other layer. This is the main idea that is extremely powerful. The input of a layer inside DenseNet

is the concatenation of feature maps from previous layers.

Figure 2.15: Fig. 2.15 DenseNetv3 Architecture

2.7.2 Advantages of DenseNetv3

Here are the advantages of DenseNetv3:

• Improved Gradient Flow: Alleviates the vanishing gradient problem.

• Efficient Parameter Use: Requires fewer parameters, enhancing memory efficiency.

• Feature Reuse: Allows richer feature representation by connecting all layers.

• Strong Performance: Achieves high accuracy on benchmark datasets.

• Reduced Overfitting: Minimizes overfitting risk, especially with small datasets.

• Versatility: Adaptable for various tasks, including segmentation and detection.

• Modular Design: Facilitates easy modification of network depth.

• Context Awareness: Captures multi-scale information for better object recognition.

• Faster Inference: Generally faster during inference compared to similar architectures.

2.7.3 Disadvantages of DenseNetv3

Here are the disadvantages of DenseNetv3:

• Increased Computational Complexity: Dense connections result in higher memory and

processing power requirements.

• Longer Training Times: More complex architecture leads to extended training durations

compared to simpler models.

• Difficulties in Implementation: The unique connectivity patterns can complicate implementation and tuning.

• Sensitivity to Hyperparameters: Requires careful adjustment of hyperparameters, which

may necessitate extensive experimentation.

• High Memory Consumption: Storing multiple feature maps from all layers can lead to

excessive memory usage during training.

• Stage Bottlenecks: Potential bottlenecks at certain stages due to processing large numbers

of features, affecting efficiency.

• Limited Interpretability: The model's decisions may be difficult to interpret, making it

hard to understand feature importance.

• Complexity in Transfer Learning: May complicate fine-tuning when adapting the model

for new tasks or datasets.

2.8 MobileNetv3

MobileNetv3 is an advanced deep learning architecture designed specifically for mobile and edge

devices. It builds upon the successful foundations of its predecessors (MobileNet V1 and V2),

introducing optimizations that improve performance and efficiency while maintaining low latency

and high accuracy [16]. Leveraging techniques such as neural architecture search (NAS),

lightweight operations, and improved activation functions, MobileNetv3 is well-suited for real

time applications like image classification, object detection, and semantic segmentation on

resource-constrained devices.

2.8.1 MobileNetv3 Architecture

MobileNetv3 incorporates several key design choices and features, including:

• Depthwise Separable Convolutions: MobileNetv3 continues to use depth wise separable

convolutions, which split the convolution operation into two smaller operations: a depth

wise convolution (applying a single filter to each input channel) followed by a pointwise

convolution (1x1 convolution to combine the outputs).

• Neural Architecture Search (NAS): The architecture of MobileNetv3 was optimized

using NAS, leading to an intelligent design that balances performance and computational

efficiency.

• Inverted Residual Blocks: Similar to MobileNet V2, MobileNetv3 employs inverted

residual blocks, which consist of a lightweight linear bottleneck structure that enhances

feature extraction.

• Activation Functions: MobileNetv3 introduces the Swish activation function, which

provides better performance compared to ReLU. Additionally, it utilizes Hard-Swish for

faster computation while retaining benefits of Swish.

• Squeeze-and-Excitation Blocks: These blocks help to recalibrate channel-wise feature

responses, improving the model's representational power.

• Multi-Branch Architecture: MobileNetv3 employs a multi-branch design in its

architecture, allowing it to capture more diverse features while keeping the overall model

lightweight.

• Version Variants: MobileNetv3 comes in two variants:

o MobileNetv3-Large: Optimized for higher accuracy but requires more computational resources.

o MobileNetv3-Small: A more lightweight version, optimized for faster performance and efficiency.

2.8.2 Advantages of MobileNetv3

• High Efficiency: MobileNetv3 achieves a strong balance between accuracy and computational efficiency, making it ideal for mobile devices.

• Reduced Model Size: The architecture is designed to minimize the number of parameters,

facilitating faster downloads and lower memory usage.

• Real-Time Performance: Optimized for lower latency, enabling real-time processing for

applications like image classification and object detection on edge devices.

• Versatile Applications: Suitable for various tasks, including image recognition, object

detection, and segmentation, due to its efficient design.

• Flexible Design: The ability to choose between MobileNetv3-Large and MobileNetv3-

Small allows developers to tailor the model based on specific application requirements.

2.8.3 Disadvantages of MobileNetv3

• Limited Performance on Complex Tasks: While efficient, MobileNetv3 might not perform as well as larger architectures (e.g., ResNet, DenseNet) in handling highly complex tasks that require deeper networks.

• Sensitivity to Data Quality: The model's performance may degrade significantly with

poor quality or imbalanced datasets, necessitating careful data handling.

• Interpretability Issues: Like many deep learning models, MobileNetv3 can be seen as a

"black box," making it difficult to interpret the reasoning behind its predictions.

• Hardware Limitations: Although designed for mobile devices, performance may vary

depending on specific hardware capabilities, impacting the feasibility of some applications.

2.9 Summary

Finally, an examination of VGG16, ResNet50, MobileNetV3, and DenseNetv3 demonstrates the

distinctive qualities of each network architecture and the advances they enable in computer vision.

Our selection of models for certain tasks is influenced by our understanding of deep layer

topologies, skip connections, inception modules, depth-wise separable convolutions, and dense

connectivity. Our empirical inquiry is guided by this theoretical framework, which also helps with

model selection and result interpretation. Recognizing the advantages of each architecture places

our study in the context of deep learning's ongoing evolution and highlights its contributions to

the advancement of computer vision and image processing.

Chapter 3

Literature Review

3.1 Antimicrobial Resistance Overview

3.2 Machine Learning in Genomics

3.3 Related Works

Table 3.1: **Summary of the main characteristics of the study.**

Authors and Country	Number of Microbial Strains	Antibiotics Studied	Critical and High-Priority Pathogens Studied	Machine Learning Model	Assessr Perforr
This Study (2025), Bangladesh	6,100	Meropenem, Doripenem, Imipenem, Ertapenem	*Escherichia coli*, *Klebsiella pneumoniae*, *Pseudomonas aeruginosa*	Logistic Regression, Decision Tree, Random Forest, XGBoost, SVM, R-Blend Ensemble	Accurac cision, F1-score Confusic trix
Sunuwar & Azad (2021), USA	724	Doripenem, Ertapenem, Imipenem, Meropenem	*Escherichia coli*, *Klebsiella pneumoniae*, *Pseudomonas aeruginosa*	LR, gNB, SVM, DT, RF, KNN, LDA, mNB, AdaBoost, GBDT, ETC, BG	Recall, cision, k-fold validatic

Chapter 4

Materials and Methods

4.1 Research Methodology

4.1.1 Data Collection

This study utilized publicly available genomic and phenotypic data on antimicrobial resistance (AMR) from the NCBI Pathogen Detection database [4], as curated and preprocessed by Sunuwar et al. [8]. The dataset consists of multiple subsets, each corresponding to a specific bacterial species and antibiotic combination (e.g., *Escherichia coli*–meropenem, *Klebsiella pneumoniae*–doripenem, *Pseudomonas aeruginosa*–imipenem, etc.). Each subset contained approximately 10^2–10^3 isolates with known antimicrobial susceptibility test (AST) outcomes (resistant or susceptible), consistent with prior AMR prediction studies [7].

Genomic features were represented in a binary gene presence–absence format following the procedures described by Sunuwar and Azad [7]. The union of all annotated genes across isolates formed the feature space, typically containing several hundred genes per dataset. Each isolate's entry consisted of 1s and 0s reflecting gene presence or absence. The final column represented the AST phenotype, with resistant = 1 and susceptible = 0. Any intermediate susceptibility states were treated as resistant, following the conservative classification approach in Sunuwar et al. [8].

All data were provided as CSV files, one file per antibiotic–species combination. Preprocessing steps included: (1) removing isolates lacking genomic features or phenotype labels; (2) standardizing gene names and renaming hypothetical proteins following Sunuwar et al. protocols [7]; (3) verifying class imbalance for model selection; (4) ensuring no missing values remained (binary features imply absence).

Thus, the curated dataset consisted of clean binary feature matrices suitable for machine learning analysis.

4.1.2 Feature Engineering

Feature engineering followed two complementary strategies: (1) gene presence features, and (2) k-mer–based alignment-free features.

First, binary features representing known antimicrobial resistance genes (e.g., *aac(3)-IIa*, *tet(B)*) were included directly as in prior AMR studies [7]. These features capture canonical resistance determinants documented in the literature.

Second, genomic sequences were represented using fixed-length nucleotide k-mers, an approach widely used in alignment-free genomic analyses [1]. K-mer frequencies were computed by scanning each assembled genome, treating the genome analogously to a document in text mining. Because raw k-mer vectors can be extremely sparse and high-dimensional, term frequency–inverse document frequency (TF-IDF) weighting was applied, a technique validated in multiple genomic machine learning studies [9, 2].

TF-IDF weighting emphasizes k-mers that appear frequently within an isolate but are relatively rare globally across genomes, making them useful for distinguishing resistant vs. susceptible isolates. Formally:

$$\text{TF-IDF}(k\text{-mer}) = \text{TF}(k\text{-mer}) \times \log\left(\frac{N}{n_k}\right),$$

where TF(k-mer) is the normalized count in a genome, N is total genomes, and n_k is number of genomes containing the k-mer.

Additionally, an aggregate feature named **R-Score** was engineered to capture the total "AMR gene burden" of each isolate. The R-Score was defined as the row sum of all known resistance gene indicators and then min–max normalized to $[0, 1]$. Previous work has shown that such row-sum AMR metrics may improve classifier robustness [8].

All feature engineering steps—including TF-IDF fitting—were conducted inside training folds only to prevent data leakage.

4.1.3 Model Selection

A broad spectrum of machine learning algorithms was evaluated using scikit-learn [5], XGBoost, and LightGBM.

Models included:

- **Logistic Regression (L2-regularized)** — baseline linear model widely used in genomic ML [8].
- **Support Vector Machine (RBF kernel)** — effective in high-dimensional spaces, often used in AMR prediction tasks [7].

- **Decision Tree (CART)** — interpretable but prone to overfitting [3].
- **Random Forest** — ensemble method robust to feature noise; used extensively in AMR gene prediction [3].
- **XGBoost and LightGBM** — state-of-the-art gradient boosting algorithms shown to perform well on genomic datasets [2, 6].
- **Naïve Bayes models** — fast baselines included for completeness.
- **k-Nearest Neighbors, LDA** — evaluated but not optimal for sparse genomic matrices.

A specialized soft-voting ensemble, named **R-Blend**, was constructed to combine strengths of three high-performing models:

1. Logistic Regression,
2. Decision Tree,
3. XGBoost.

Weights (1:1.5:1) were assigned to emphasize the well-calibrated probabilities of Logistic Regression.

Hyperparameter optimization was performed via stratified k-fold cross-validation. SMOTE, random oversampling, and class-weight adjustments were employed to handle imbalance, consistent with existing best practices [7, 8].

4.1.4 Evaluation

Each dataset was split into 80% training and 20% held-out testing sets, using stratified sampling to preserve label proportions [8]. Model tuning used stratified k-fold cross-validation (typically $k = 5$). In selected cases, nested cross-validation and leave-one-out cross-validation (LOOCV) were explored to reduce variance, similar to methods used in prior genomic AMR studies [3].

Performance evaluation employed multiple metrics:

- Accuracy,
- Precision,
- Recall,

- F1-score (primary metric),
- Area Under ROC Curve (AUROC),
- Area Under Precision–Recall Curve (AUPRC),
- Confusion matrix visualization.

AUPRC was emphasized due to class imbalance, as recommended in multiple AMR machine learning works [6]. ROC and PR curves were generated for diagnostic visualization. Standard deviation across folds and statistical tests (e.g., paired t-test on F1-scores) were used to confirm robustness.

All experiments were executed with fixed random seeds and version-controlled scripts to ensure full reproducibility, following reproducibility standards in computational biology [5].

Chapter 5

Results

5.1 Descriptive Statistics

5.2 Model Performance

5.3 Feature Importance

Chapter 6

Discussion

6.1 Interpretation of Findings

6.2 Comparison with Previous Studies

6.3 Limitations

Chapter 7

Conclusion and Future Work

7.1 Conclusion

7.2 Future Research Directions

Bibliography

[1] Y. Chang, D. Lee, and H. Kim. Alignment-free identification of clones in b-cell receptor repertoires using tf-idf weighted k-mers. *Frontiers in Immunology*, 12: 791377, 2021. doi: 10.3389/fimmu.2021.791377.

[2] A. Kumar, S. Banerjee, and P. Roy. Bioset2vec: Extraction of k-mer dictionaries from biological sequences via tf-idf. *BMC Bioinformatics*, 24:62, 2025. doi: 10.1186/s12859-025-06261-7.

[3] Pavel Májek, Fajfr Oldřich, Náplavová Jana, Milan Kolář, and Jansa Petr. Genome-wide mutation scoring for machine-learning-based antimicrobial resistance prediction. *International Journal of Molecular Sciences*, 22(23):13049, 2021. doi: 10.3390/ijms222313049.

[4] NCBI Pathogen Detection. Pathogen detection isolate browser. URL `https://www.ncbi.nlm.nih.gov/pathogens`. Accessed: 2025-12-01.

[5] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[6] E. Rannon, T. Smith, and J. Lee. Dramma: A multifaceted machine-learning approach for novel antimicrobial resistance gene detection. *Microbiome*, 13(1):67, 2025. doi: 10.1186/s40168-025-1567-9.

[7] J. Sunuwar and R. Azad. Identification of novel antimicrobial resistance genes using machine learning, homology modeling, and molecular docking. *Microorganisms*, 10 (11):2102, 2022. doi: 10.3390/microorganisms10112102.

[8] Jitendra Sunuwar and Rajeev Azad. A machine learning framework to predict antibiotic resistance traits and yet unknown genes. *Briefings in Bioinformatics*, 22 (6):bbab179, 2021. doi: 10.1093/bib/bbab179.

[9] R. Zhang and L. Wang. Tf-idf based alignment-free methods for dna sequence comparison. *Computational Biology and Chemistry*, 94:107583, 2021. doi: 10.1016/j.compbiolchem.2021.107583.

Appendix A

Appendix A

Appendix B

Appendix B