**Course:**

**Deep Learning – Spring 2023/2024**

**Course instructor:**

**Dr. Ala’a Al-Habashna**

**Prepared By:**

**Sama Al-Qaisi**

Developing a deep learning-based system

*Table of Contents*

[PART-1: Implementation of an MLP from Scratch 2](#_Toc168737875)

[Design and implementation of MLP network from scratch 2](#_Toc168737876)

[Design: 2](#_Toc168737877)

[Implementation: 3](#_Toc168737878)

[Training and testing of the MLP 4](#_Toc168737879)

[Dataset 4](#_Toc168737880)

[Training and Validation 4](#_Toc168737881)

[Testing 6](#_Toc168737882)

[Evaluation of the developed MLP 8](#_Toc168737883)

[PART-2: Developing a Deep-Learning System for a Computer Vision Application 10](#_Toc168737884)

[Problem statement 10](#_Toc168737885)

[Research on the Neural Networks and architectures 10](#_Toc168737886)

[Neural Networks used for the problem 10](#_Toc168737887)

[Modern architectures 10](#_Toc168737888)

[Modern Architecture Comparison 11](#_Toc168737889)

[Models’ development and training 11](#_Toc168737890)

[Dataset 11](#_Toc168737891)

[Training and Validation 12](#_Toc168737892)

[Models’ testing and evaluation 18](#_Toc168737893)

[Testing 18](#_Toc168737894)

[Over/under-fitting assessment 23](#_Toc168737895)

[Results analysis 24](#_Toc168737896)

[Effectiveness assessment 25](#_Toc168737897)

[Interface development 27](#_Toc168737898)

[Evaluation of models 28](#_Toc168737899)

[References 30](#_Toc168737900)

# **PART-1: Implementation of an MLP from Scratch**

## **Design and implementation of MLP network from scratch**

### Design:

The Multi-Layer Perceptron (MLP) is used for classifying breast cancer tumors as benign or malignant, which is known as binary classification. The inputs are 31 features including breast cancer cell nuclei measurements, while the outputs are single values implying 0 or 1 (indicating whether the tumor is benign or malignant).

UML Diagram:

A screenshot of a computer

Description automatically generated

The design of the MLP follows the architecture of loading and preprocessing the data including normalization and handling missing values, then it is divided into training, testing, and validation sets for model training, evaluation, and hyperparameter tuning.

The training set includes 30 features from the breast cancer dataset as an input (30 neurons), and the model outputs two neurons because we have a binary classification (distinguishing tumors between malignant and benign), activation functions used are Softmax for the output layer and ReLU function for hidden layers also the Loss function used is Cross Entropy Loss which is well suited for classification tasks. The training algorithm applied is Gradient Descent, a foundational optimization technique that updates the model parameters iteratively to minimize the loss function by descending along the steepest slope, enhancing the model's performance.

Multiple validation steps are performed, tuning architectures, epochs, and learning rates to determine the optimal model. Following these steps, the best-performing model is selected based on the validation results. A curve is then generated that depicts training and validation loss or accuracy across epochs to provide a visual representation of the model's performance.

The model with the highest performance is trained on the test set to measure its generalization ability. Next, a confusion matrix is ​​generated to evaluate the model's performance, providing a comprehensive overview of classification accuracy, precision, recall, and other key metrics.

Finally, the best-performing model architecture is implemented using PyTorch.

### Implementation:

The implementation of the MLP follows the structure of the Object-Oriented programming concept to design the architecture of our MLP. The MLP is implemented under the main class that includes our methods. The class structure is as follows:

**Class Data: This class encapsulates the Data process**

This Data class contains the Loading and preprocessing of the dataset, it contains methods for data loading from a CSV file, and processing stages of label encoding, standardization, and splitting the dataset.

**Class Main\_class: This class encapsulates the entire MLP**

**Methods:**

1. **\_\_init\_\_ Method:** This method works on initializing the MLP with the given input, hidden, and output layer sizes as well as initializing for each layer weights which are initialized randomly using (torch. rand), and biases which are initialized to zeros using (torch. zeros). By setting the requires\_grad=True flag for all weights and biases will enable automatic differentiation.
2. **softmax Method:** This method is for computing the softmax activation function for the output of the last hidden layer by normalizing the logits to probabilities and that is done by subtracting the maximum value from the logits and then computing the exponent of the adjusted logits and normalizing them by their sum.
3. **relu Method:** This method is for computing the ReLU activation function for the hidden layers by applying an element-wise operation that sets all negative values ​​in the input tensor to zero, leaving positive values ​​unchanged.
4. **cross\_entropy\_loss:** This method is for computing the cross-entropy loss between the true labels and the predicted probabilities by gathering the probabilities for the true classes and computing the negative log-likelihood then returning the mean loss.
5. **forward\_pro:** This method applies forward propagation by applying matrix multiplication and adding biases following that by applying the ReLU activation function in each hidden layer. After that, apply the softmax to the output of the last hidden layer to get the class probabilities.
6. **backward\_pro:** This method applies backward propagation; it begins by getting the predictions from the forward propagation and then computes the cross-entropy loss and then performs the backward propagation to calculate the gradients of the loss concerning the model parameters, these gradients are then used to update the weights and biases using the gradient descent within the “torch. no\_ grad ()” context to prevent the gradient tackling. And finishing by zeroing out the gradients to avoid accumulation.
7. **training Method:** This method trains the model by calling the backward propagation function to apply the forward and backward passes over a specific number of epochs, the number of times the entire dataset is passed forward and backward through the model during the training process.
8. **test Method:** This method evaluates the performance of the MLP trained on unseen test data. It applies forward propagation to obtain predictions, calculates metrics such as loss and accuracy, and returns an accuracy score.

A good object-oriented architecture is implemented throughout the implementation, where classes represent distinct MLP components and a clear separation of responsibilities. This makes the code standard, easy to manage, and understandable. The Main\_class embodies the full functionality of MLP, from forward propagation to backpropagation of weight updates via gradient descent. Through repetitive training, it improves its parameters to improve performance. Evaluating a test set provides insight into its accuracy. This approach demonstrates a concise structure of an MLP while maintaining clarity and adhering to object-oriented design principles.

## **Training and testing of the MLP**

### Dataset

**Data Name:** Breast Cancer Dataset.

**Source:** The data was available and downloaded from the Kaggle website.

**The number of records & features:** 569 records, and 31 features.

**The features names and data types:** all 31 features have the float (numeric) data type, these features describe the characteristics of breast cancer tumors including (Radius of Lobes, Outer Perimeter of Lobes, Mean Area of Lobes, Mean of Smoothness Levels).

**The Target**: diagnosis (Target: M - Malignant B – Benign) which then are encoded as 0 (B) and 1 (M).

**The problem type:** Classification.

**You can access the data from the link below:**

<https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset>

### Training and validation

To make sure the evaluation process of the MLP is done correctly, the dataset was divided into three parts: Training Set (80%), Validation Set (10%), and Testing Set (10%), the model will be trained on a subset proportion of data while being evaluated on unseen data.

Table 1: Description of the hyperparameters considered and the best value for each.

|  |  |  |
| --- | --- | --- |
| **Hyper-parameter** | **Description** | **Value** |
| **Learning rate** | This hyperparameter is essential in the training processes which determine the step size the model takes while updating the weights. | **0.01** |
| **Number of epochs** | This hyperparameter determines the number of iterations that the model needs to iterate through the entire training dataset. | **5000** |
| **Neurons in layer1** | This hyperparameter determines the number of neurons in the first hidden layer. | **10** |
| **Neurons in layer2** | This hyperparameter determines the number of neurons in the second hidden layer. | **5** |

A graph of a graph

Description automatically generated with medium confidence

This learning curve shows the accuracy over the epochs to present the relation between the model during the training set over the epochs versus the accuracy of the model on the validation set over the epochs, what is observed from the curve is that the training accuracy increase in the first epochs and then stabilize around **0.93** while the validation accuracy increase in the first epochs and then stabilize right beneath the training accuracy curve at **0.91**, which show the good model performance while minimizing the occurrence of overfitting.

Table 2: Combination of the hyperparameter values and corresponding performance achieved.

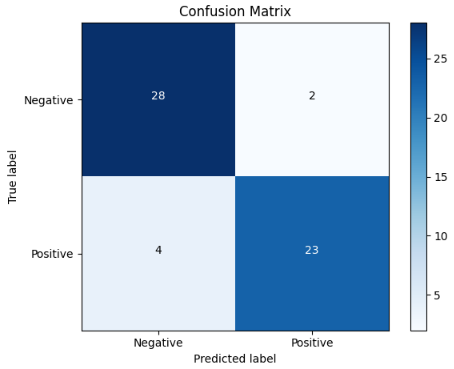
|  |  |  |
| --- | --- | --- |
| **Combination of hyperparameter values** | **Training performance** | **Validation performance** |
| **First Validation:**  Hidden Layer 1 size =10  Hidden Layer 2 size = 5  number of = 5000  learning rate = 0.01 | Final Train Accuracy: 0.9187 | Test Accuracy: 0.9123 |
| **Second Validation:**  Hidden Layer 1 size =10  Hidden Layer 2 size = 5  number of = 500  learning rate = 0.0001 | Final Train Accuracy: 0.3890 | Test Accuracy: 0.2807 |
| **Third Validation:**  Hidden Layer 1 size =10  Hidden Layer 2 size = 5  Hidden Layer 3 size = 2  number of = 5000  learning rate = 0.001 | Final Train Accuracy: 0.8088 | Test Accuracy: 0.7895 |
| **Fourth validation:**  Hidden Layer 1 size =10  Hidden Layer 2 size = 5  Hidden Layer 3 size = 15  number of = 500  learning rate = 0.0001 | Final Train Accuracy: 0.5604 | Test Accuracy: 0.4737 |
| **Fifth validation:**  Hidden Layer 1 size =5  Hidden Layer 2 size = 5  Hidden Layer 3 size = 5  Hidden Layer 4 size = 5  number of = 5000  learning rate = 0.001 | Final Train Accuracy: 0.8967 | Test Accuracy: 0.8947 |

### **Testing**

The testing process is an essential step that allows us to evaluate the applicability and performance of the trained model in the real world. This involves subjecting the model to a set of unseen data. By preparing this test dataset and loading the trained model later, practitioners can simulate how the model will perform on unseen data. During forward propagation, the model processes each test sample, leveraging its learned parameters to generate predictions. Performance metrics such as precision, precision, recall, and F1 score, along with confusion matrix construction, provide detailed insights into the model's classification capabilities. Through this careful evaluation, practitioners gain a deeper understanding of the strengths and weaknesses of the model.

Table 3: Best values for the evaluation metrics on the test set.

|  |  |  |
| --- | --- | --- |
| **Evaluation metric** | **Description** | **Value obtained** |
| **Test Accuracy** | It is the correct predictions made during the test.  The model correctly predicted the correct class by 89% of the data in the test set. | **0.8947** |
| **Precision** | It tells us how many cases that were predicted to be positive are positive 92% were positive out of all instances classified as positive. | **0.9200** |
| **Recall** | It measures the model's ability to find all relevant cases within the data. It tells us what proportion of actual positives are correctly classified by the model.  85% of all positive classes were correctly identified by the model. | **0.8519** |
| **F1-score** | It is the harmonic mean of precision and recall that provides the balance between precision and recall. A score of 88% indicates good agreement between accuracy and recall. | **0.8846** |



As observed from the confusion matrix, it shows a good performance with the classification problem having only a few misclassifications.

**True Negatives (TN-28**): Correct predictions for the negative class.

**False Positive (FP-2):** Wrong prediction (predicted positive while the actual class is negative).

**False Negative (FN-4):** Wrong prediction (predicted negative while the actual class is positive).

**True Positive (TP-23):** Correct predictions for the positive class.

|  |  |
| --- | --- |
| **Evaluation metric** | **Value obtained** |
| **Test Accuracy** | **0.9649** |
| **Precision** | **1.0000** |
| **Recall** | **0.9259** |
| **F1-score** | **0.9615** |

Table 4: Best values for the evaluation metrics for the implementation of MLP network using NN.

The neural network (NN) implementation outperforms the MLP implementation from scratch on several performance criteria, including precision, recall, and F1 score. With a test accuracy of **96%,** the NN model outperforms the model-from-scratch accuracy of **89%.** This significant difference highlights the greater ability of NN to classify breast cancer tumors as benign or malignant correctly. Moreover, the NN model's accuracy, recall, and F1-score significantly outperform MLP from scratch, proving its effectiveness in correctly detecting positive and negative cases with greater accuracy and sensitivity. This improvement demonstrates the NN's greater ability to detect subtle patterns in data and produce more accurate classifications. Overall, the NN implementation excels in terms of overall accuracy and precision, recall, and F1 score, highlighting its ability for accurate classification of breast cancer tumors and its real-world potential for use in medical diagnosis.

## **Evaluation of the developed MLP**

Designing and implementing a multilayer perceptron (MLP) from scratch for breast cancer tumor classification demonstrates a strong understanding of both the problem domain and neural network concepts. MLP achieves a good test accuracy of about **89%,** demonstrating its effectiveness in distinguishing between benign and malignant tumors compared to the neural network (NN) implementation with a test accuracy of **96%,** and there are suggested improvements in the performance of MLP.

MLP performs well and is consistent, but it may not take advantage of advanced techniques or architectures to achieve higher accuracy. While achieving around **89%** accuracy is acceptable, exploring more complex architectures, such as adding additional layers or neurons, can enhance the model's ability to capture complex patterns in the data. In addition, experimenting with different activation functions outside of ReLU and Softmax, implementing leakage regularization to prevent overflow, and using advanced optimization algorithms such as Adam can lead to improved performance.

The method of tuning hyperparameters appears to be relatively restricted since only a few hyperparameter combinations have been investigated. While recording approach and results is important, more comprehensive research may reveal higher performing groups. Implementing methods such as grid search or random search to systematically investigate a wide range of hyperparameter combinations may increase model performance and generalizability.

Using object-oriented programming (OOP) to organize the code base is valuable, however, there are some suggested improvements. It would be helpful to break down the code further to improve readability and reusability. Each component, such as data processing, model generation, and training/testing operations, can be contained within separate modules or categories.

While the structure of MLP provides a solid foundation for the classification of breast cancer tumors, there are opportunities for improvement in the model structure following improvements suggested in theses in different fields. The performance and effectiveness of MLP in classifying tumors can be enhanced, leading to better medical diagnostic results.

# **PART-2: Developing a Deep-Learning system for a Computer Vision Application**

## **Problem statement**

Problem statement: MSS vs MSI classification in Gastrointestinal Cancer using Histopathology Image analysis with CNN, The aim of this project is to develop a deep learning model, specifically a Convolutional Neural Network (CNN), to classify histopathology images of gastrointestinal cancer into two categories: MSI and MSS.

Microsatellite instability (MSI) and microsatellite stability (MSS) are important biomarkers in gastrointestinal cancers. Detecting MSI is crucial as it can influence the choice of treatment and prognosis. The objective of this project is to develop a deep learning model, specifically a Convolutional Neural Network (CNN), to classify histopathology images of gastrointestinal cancer into two categories: MSI (Microsatellite Instability) and MSS (Microsatellite Stability). The objective is to accurately classify histopathology images of gastrointestinal cancers into MSI and MSS categories. This model will assist pathologists in diagnosing and tailoring treatments based on the MSI/MSS status of the tumor. The type of problem is a classification problem where the goal is to categorize histopathology images into one of two classes: MSI or MSS.

## **Research on the Neural Networks and architectures**

### Neural Networks used for the problem

Convolutional neural networks (CNNs) are the most often used and appropriate neural networks for classifying histopathological pictures into MSI and MSS classes. CNNs are extremely effective in image classification uses because they can capture spatial hierarchy in pictures using convolutional layers.

### Modern architectures

I have used 4 modern architectures in my project which are as follows:

1. **AlexNet:**

It debuted in 2012 and is a groundbreaking convolutional neural network (CNN) architecture noted for its exceptional performance in classification tasks. It was one of the first deep networks to show substantial performance in the classification of image tasks. AlexNet can successfully learn hierarchical features from raw pixel data. Its use of rectified linear units (ReLU) for activation and dropout regularization during training addresses frequent issues including the vanishing gradient problem and overfitting.

1. **VGGNet:**

Created by the Visual Geometry Group at the University of Oxford in 2014, is noted for its simple design, which includes a consistent layout of convolutional layers. VGGNet's simplicity and efficacy have made it a popular choice in deep learning, especially for picture classification. Its pre-trained models are commonly utilized for transfer learning.

1. **GoogleNet:**

Also known as Inception v1, revolutionized CNN architectures with its inception module in 2014. This module applies multiple-size filters concurrently, allowing the network to efficiently collect various characteristics. By concatenating results from various filter sizes, it catches both small details and wider context.

1. **ResNet:**

Residual Networks is a revolutionary deep learning architecture developed by Microsoft Research in 2015. It incorporates residual connections, which let gradients skip certain layers during training, so addressing the vanishing gradient problem. This breakthrough permits the creation of very deep networks, including variations such as ResNet50 and ResNet101, which contain 50 and 101 layers.

## **Modern architectures comparison**

Table 4: Modern architectures are used to solve the problem.

|  |  |  |  |
| --- | --- | --- | --- |
| **Architecture** | **Description and number/types of layers** | **Advantages** | **Disadvantages** |
| **AlexNet (8)** | 5 convolutional layers  3 fully connected layers | Straightforwardness and importance in the context of history | Higher error rates compared to other architectures |
| **VGGNet (16)** | 13 convolutional layers  3 fully connected layers | Small filters, Deeper networks | Higher computational cost |
| **GoogleNet** | 22 layers | Deeper networks, with computational efficiency | Computational complexity  Very expensive compute |
| **ResNet (18)** | 16 convolutional layers  2 fully connected layers | It helps address the problem of vanishing gradients, allowing much deeper networks to be created. | Complex design and tuning process |

## **Models’ development and training**

### Dataset

The dataset utilized to train and test CNNs was composed of histopathology pictures labeled for microsatellite instability (MSI) and microsatellite stability (MSS) in gastrointestinal cancer. Originally, the collection had 5 million images. However, the number of images has been greatly decreased to improve training and assessment.

Here are the number of images in each set:

**Training Set:**

MSS images:596

MSI images:603

Total Train Set: 1,199

**Testing Set:**

MSS images:403

MSI images:463

Total Test Set: 866

**Validation Set:**

MSS images:210

MSI images:205

Total Validation Set: 415

I have applied Data augmentation and normalization by implementing the following:

* All images were resized to 224×224 pixels to maintain consistency and compatibility with conventional CNN systems, although the images' original sizes are 224×224 I have resized them in case I add images to training and testing sets with varying sizes I will always ensure consistency between across my dataset.
* The pixel values ​​of the images were normalized using the mean and standard deviation defined as [0.485, 0.456, 0.406] and [0.229, 0.224, 0.225] in this order.

### Training and validation

Splitting Data: My histological images Dataset was split into three splits as follows:

* Training Set: **50%**
* Validation Set: **35%**
* Testing Set: **15%**

***AlexNet:***

Table 5: Description of the hyperparameters considered and the best value for each

|  |  |  |
| --- | --- | --- |
| **Hyper-parameter** | **Description** | **Value** |
| **Learning rate** | This hyperparameter is essential in the training processes which determine the step size the model takes while updating the weights. | **0.001** |
| **Weight decay** | Weight decay is one of the fundamental hyperparameters in the training process that applies a regularization penalty to the weights, which helps prevent overfitting by encouraging the model to keep the weights small. | **0.005** |
| **momentom** | Momentum is an essential hyperparameter in the training process of image classification architectures, as it accelerates the gradient vectors in the right directions, leading to faster convergence and more stable updates while improving the model weights. | **0.7** |
| **batch size** | Batch size is a key hyperparameter in the training process that determines how many training samples are processed before the model weights are updated. It greatly affects the training dynamics, affecting the model's convergence speed and stability. | **32** |

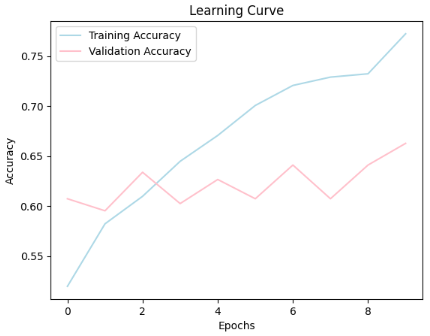


Table 6: Combination of the hyperparameter values and corresponding performance achieved

|  |  |  |
| --- | --- | --- |
| **Combination of hyperparameter values** | **Training performance** | **Validation performance** |
| **First combination**  Learning rate= 0.003  Weight decay= 0.001  momentom= 0.5  batch size= 64 | Training Accuracy: **0.7273** | Overall Accuracy on val Set: **0.6169** |
| **Second combination**  Learning rate= 0.001  wd= 0.005  momentom= 0.7  batch size= 32 | Training Accuracy: **0.7723** | Overall Accuracy on val Set: **0.6627** |
| **Third combination**  Learning rate= 0.001  wd= 0.01  momentom= 0.7  batch size= 64 | Training Accuracy: **0.7389** | Overall Accuracy on val Set: **0.6530** |
| **Fourth combination**  Learning rate= 0.003  wd= 0.001  momentom= 0.5  batch size= 128 | Training Accuracy: **0.7089** | Overall Accuracy on val Set: **0.6096** |

***VGGNet:***

Table 5: Description of the hyperparameters considered and the best value for each

|  |  |  |
| --- | --- | --- |
| **Hyper-parameter** | **Description** | **Value** |
| **Learning rate** | This hyperparameter is essential in the training processes which determine the step size the model takes while updating the weights. | **0.001** |
| **Weight decay** | Weight decay is one of the fundamental hyperparameters in the training process that applies a regularization penalty to the weights, which helps prevent overfitting by encouraging the model to keep the weights small. | **0.005** |
| **momentom** | Momentum is an essential hyperparameter in the training process of image classification architectures, as it accelerates the gradient vectors in the right directions, leading to faster convergence and more stable updates while improving the model weights. | **0.7** |
| **batch size** | Batch size is a key hyperparameter in the training process that determines how many training samples are processed before the model weights are updated. It greatly affects the training dynamics, affecting the model's convergence speed and stability. | **32** |

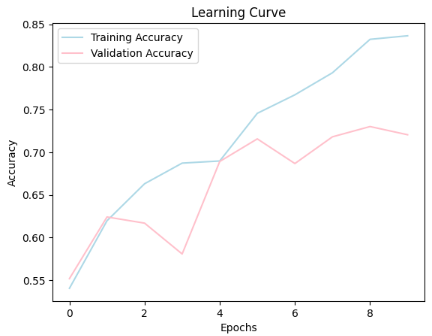


Table 6: Combination of the hyperparameter values and corresponding performance achieved

|  |  |  |
| --- | --- | --- |
| **Combination of hyperparameter values** | **Training performance** | **Validation performance** |
| **First combination**  Learning rate= 0.003  Weight decay= 0.001  momentom= 0.5  batch size= 64 | Training Accuracy: **0.8524** | Overall Accuracy on val Set: **0.7133** |
| **Second combination**  Learning rate= 0.001  wd= 0.005  momentom= 0.7  batch size= 32 | Training Accuracy: **0.8365** | Overall Accuracy on val Set: **0.7205** |
| **Third combination**  Learning rate= 0.001  wd= 0.01  momentom= 0.7  batch size= 64 | Training Accuracy: **0.7531** | Overall Accuracy on val Set: **0.6843** |
| **Fourth combination**  Learning rate= 0.003  wd= 0.001  momentom= 0.5  batch size= 128 | Training Accuracy: **0.7298** | Overall Accuracy on val Set: **0.6386** |

***GoogleNet:***

Table 5: Description of the hyperparameters considered and the best value for each

|  |  |  |
| --- | --- | --- |
| **Hyper-parameter** | **Description** | **Value** |
| **Learning rate** | This hyperparameter is essential in the training processes which determine the step size the model takes while updating the weights. | **0.001** |
| **Weight decay** | Weight decay is one of the fundamental hyperparameters in the training process that applies a regularization penalty to the weights, which helps prevent overfitting by encouraging the model to keep the weights small. | **0.005** |
| **momentom** | Momentum is an essential hyperparameter in the training process of image classification architectures, as it accelerates the gradient vectors in the right directions, leading to faster convergence and more stable updates while improving the model weights. | **0.7** |
| **batch size** | Batch size is a key hyperparameter in the training process that determines how many training samples are processed before the model weights are updated. It greatly affects the training dynamics, affecting the model's convergence speed and stability. | **32** |

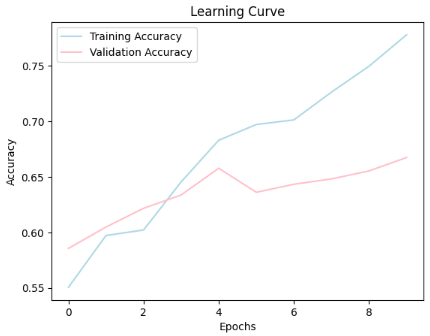


Table 6: Combination of the hyperparameter values and corresponding performance achieved

|  |  |  |
| --- | --- | --- |
| **Combination of hyperparameter values** | **Training performance** | **Validation performance** |
| **First combination**  Learning rate= 0.003  Weight decay= 0.001  momentom= 0.5  batch size= 64 | Training Accuracy: **0.7723** | Overall Accuracy on val Set: **0.6410** |
| **Second combination**  Learning rate= 0.001  wd= 0.005  momentom= 0.7  batch size= 32 | Training Accuracy: **0.7781** | Overall Accuracy on val Set: **0.6675** |
| **Third combination**  Learning rate= 0.001  wd= 0.01  momentom= 0.7  batch size= 64 | Training Accuracy: **0.7106** | Overall Accuracy on val Set: **0.6651** |
| **Fourth combination**  Learning rate= 0.003  wd= 0.001  momentom= 0.5  batch size= 128 | Training Accuracy: **0.7023** | Overall Accuracy on val Set: **0.6482** |

***ResNet:***

Table 5: Description of the hyperparameters considered and the best value for each

|  |  |  |
| --- | --- | --- |
| **Hyper-parameter** | **Description** | **Value** |
| **Learning rate** | This hyperparameter is essential in the training processes which determine the step size the model takes while updating the weights. | **0.001** |
| **Weight decay** | Weight decay is one of the fundamental hyperparameters in the training process that applies a regularization penalty to the weights, which helps prevent overfitting by encouraging the model to keep the weights small. | **0.005** |
| **momentom** | Momentum is an essential hyperparameter in the training process of image classification architectures, as it accelerates the gradient vectors in the right directions, leading to faster convergence and more stable updates while improving the model weights. | **0.7** |
| **batch size** | Batch size is a key hyperparameter in the training process that determines how many training samples are processed before the model weights are updated. It greatly affects the training dynamics, affecting the model's convergence speed and stability. | **32** |

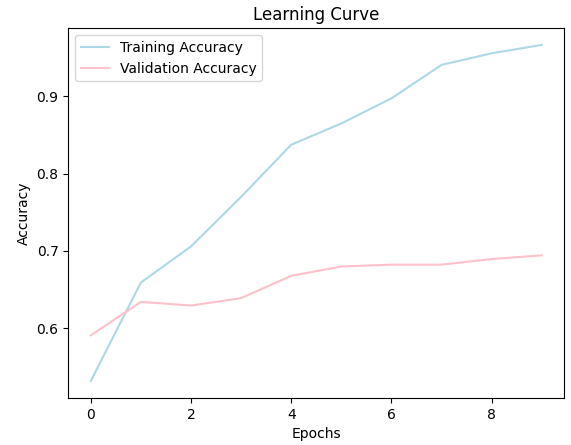


Table 6: Combination of the hyperparameter values and corresponding performance achieved

|  |  |  |
| --- | --- | --- |
| **Combination of hyperparameter values** | **Training performance** | **Validation performance** |
| **First combination**  Learning rate= 0.003  Weight decay= 0.001  momentom= 0.5  batch size= 64 | Training Accuracy: **0.9725** | Overall Accuracy on val Set: **0.6747** |
| **Second combination**  Learning rate= 0.001  wd= 0.005  momentom= 0.7  batch size= 32 | Training Accuracy: **0.9666** | Overall Accuracy on val Set: **0.6940** |
| **Third combination**  Learning rate= 0.001  wd= 0.01  momentom= 0.7  batch size= 64 | Training Accuracy: **0.9049** | Overall Accuracy on val Set: **0.6554** |
| **Fourth combination**  Learning rate= 0.003  wd= 0.001  momentom= 0.5  batch size= 128 | Training Accuracy: **0.8924** | Overall Accuracy on val Set: **0.6723** |

## **Models’ testing and evaluation**

### Testing

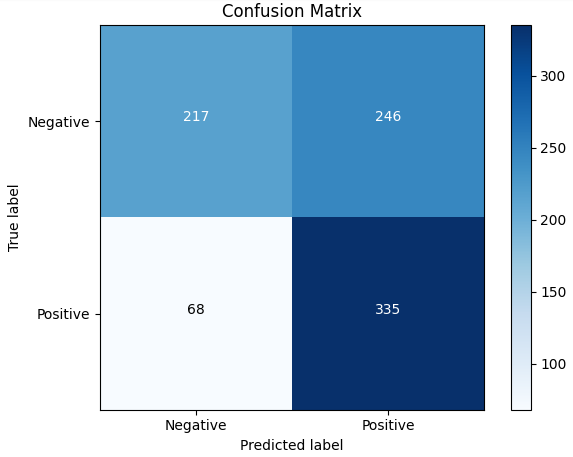
During the testing phase, four architectures (AlexNet, VGGNet, GoogleNet, and ResNet) were implemented and assessed using the best-performing hyperparameter combinations, which included the learning rate, weight decay, momentum, and batch size. Each of these architectures went through various training and validation processes before selecting the top-performing model for each based on the validation outcomes. These models were then evaluated on an independent test set to determine their predictive accuracy. To evaluate I used measures such as accuracy score, precision score, recall score, and f1 score, which were produced for each model to evaluate the success of classifying MSI and MSS.

Then I created a confusion matrix to represent the models' performance, showing true positives, true negatives, false positives, and false negatives. This comprehensive review intends to determine the most effective architecture based on overall performance and the ability to appropriately classify essential 2 classes.

***AlexNet:***

Table 7: Best values for the evaluation metrics on the test set.

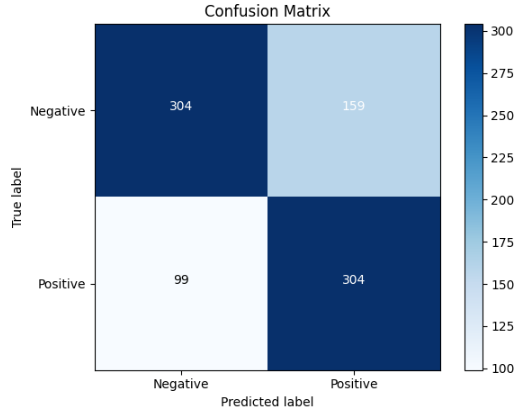
|  |  |  |
| --- | --- | --- |
| **Evaluation metric** | **Description** | **Value obtained** |
| **Test Accuracy** | It is the correct predictions made during the test.  The model correctly predicted the correct class by 64% of the data in the test set. | **0.64** |
| **Precision** | It tells us how many cases that were predicted to be positive are positive 67% were positive out of all instances classified as positive. | **0.67** |
| **Recall** | It measures the model's ability to find all relevant cases within the data. It tells us what proportion of actual positives are correctly classified by the model.  65% of all positive classes were correctly identified by the model. | **0.65** |
| **F1-score** | It is the harmonic mean of precision and recall that provides the balance between precision and recall. A score of 63% indicates good agreement between accuracy and recall. | **0.63** |



***VGGNet:***

Table 7: Best values for the evaluation metrics on the test set.

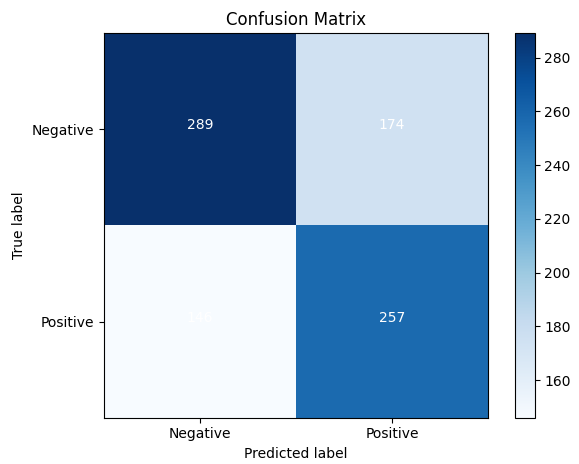
|  |  |  |
| --- | --- | --- |
| **Evaluation metric** | **Description** | **Value obtained** |
| **Test Accuracy** | It is the correct predictions made during the test.  The model correctly predicted the correct class by 70% of the data in the test set. | **0.70** |
| **Precision** | It tells us how many cases that were predicted to be positive are positive 71% were positive out of all instances classified as positive. | **0.71** |
| **Recall** | It measures the model's ability to find all relevant cases within the data. It tells us what proportion of actual positives are correctly classified by the model.  71% of all positive classes were correctly identified by the model. | **0.71** |
| **F1-score** | It is the harmonic mean of precision and recall that provides the balance between precision and recall. A score of 70% indicates good agreement between accuracy and recall. | **0.70** |



***GoogleNet:***

Table 7: Best values for the evaluation metrics on the test set.

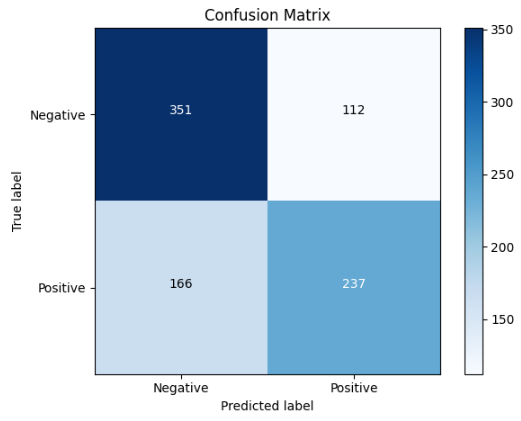
|  |  |  |
| --- | --- | --- |
| **Evaluation metric** | **Description** | **Value obtained** |
| **Test Accuracy** | It is the correct predictions made during the test.  The model correctly predicted the correct class by 63% of the data in the test set. | **0.63** |
| **Precision** | It tells us how many cases that were predicted to be positive are positive 63% were positive out of all instances classified as positive. | **0.63** |
| **Recall** | It measures the model's ability to find all relevant cases within the data. It tells us what proportion of actual positives are correctly classified by the model.  63% of all positive classes were correctly identified by the model. | **0.63** |
| **F1-score** | It is the harmonic mean of precision and recall that provides the balance between precision and recall. A score of 63% indicates good agreement between accuracy and recall. | **0.63** |



***ResNet:***

Table 7: Best values for the evaluation metrics on the test set.

|  |  |  |
| --- | --- | --- |
| **Evaluation metric** | **Description** | **Value obtained** |
| **Test Accuracy** | It is the correct predictions made during the test.  The model correctly predicted the correct class by 68% of the data in the test set. | **0.68** |
| **Precision** | It tells us how many cases that were predicted to be positive are positive 68% were positive out of all instances classified as positive. | **0.68** |
| **Recall** | It measures the model's ability to find all relevant cases within the data. It tells us what proportion of actual positives are correctly classified by the model.  67% of all positive classes were correctly identified by the model. | **0.67** |
| **F1-score** | It is the harmonic mean of precision and recall that provides the balance between precision and recall. A score of 67% indicates good agreement between accuracy and recall. | **0.67** |



### Over/under-fitting assessment

I will be comparing the training accuracy with the validation set and test accuracies, for further information about the performance of the models during the training set over the epochs versus the accuracy of the model on the validation set over the epochs you can go back to the plots of the learning curves for the models, here is the assessment for each architecture:

* AlexNet:

AlexNet has a training accuracy of **78%** and an overall accuracy of **64%** on the test set. The lower training accuracy compared to the test accuracy indicates that AlexNet does not perform well on the training data, but the overall accuracy on the test set indicates moderate performance. AlexNet performs well with great generalization ability, but there is potential for improvement in terms of accuracy.

* VGGNet:

VGGNet reaches a high training accuracy of **88%** and an overall accuracy of **70%** on the test set. As can be observed, there is a gap between the training and test accuracies, demonstrating the possibility of overfitting the training data. However, the total accuracy on the test set is high, demonstrating that the VGGNet architecture is effective in class classification.

* GoogleNet:

GoogleNet obtained a training accuracy of **74%** and an overall accuracy of **63%** on the testing set. As with AlexNet, lower training accuracy relative to testing accuracy indicates balanced performance with good generalization. The total accuracy of the test set implies decent performance, however not as high as VGGNet16. GoogleNet performs quite well, demonstrating that it learns properly from training data without overfitting.

* ResNet:

ResNet shows a very high training accuracy of **95%** while obtaining an overall accuracy of **67%** on the test set. The large gap between training and testing accuracy shows that there is a high risk of overfitting the training data. Despite the potential overfit, ResNet performs well on the test set, displaying its ability to accurately classify classes.

Overall, AlexNet and GoogleNet display balanced performance, with testing accuracies somewhat greater or similar to training accuracies, indicating successful generalization. VGGNet16 and ResNet, on the other hand, perform better in training than in testing, indicating possible overfitting concerns. However, all models achieve average to good overall accuracy on the test set, with VGGNet16 topping the way at **70%.**

### Results analysis

* **AlexNet:**

Accuracy: 64%

Precision: 0.76 (MSS) / 0.58 (MSI)

Recall: 0.47 (MSS) / 0.83 (MSI)

These results indicate that AlexNet has a better performance on MSI than on MSS with moderate precision and recall, this imbalance between them shows that AlexNet is having a difficult time correctly identifying class MSS.

* **VGGNet16:**

Accuracy: 70%

Precision: 0.75 (MSS) / 0.66 (MSI)

Recall: 0.66 (MSS) / 0.75 (MSI)

These results indicate that VGGNet has a balanced precision and recall for the MSS and MSI classes, which will lead to a higher accuracy compared to AlexNet which shows that VGGNet generalizes better providing a more reliable process of classification.

* **GoogleNet:**

Accuracy: 63%

Precision: 0.66 (MSS) / 0.60 (MSI)

Recall: 0.62 (MSS) / 0.64 (MSI)

These results indicate that GoogleNet has balanced precision and recall for MSS and MSI classes and compared to AlexNet and VGGNet has slightly lower F1 scores which shows that this model requires further tuning to improve its performance.

* **ResNet:**

Accuracy: 68%

Precision: 0.68 (MSS) / 0.68 (MSI)

Recall: 0.76 (MSS) / 0.59 (MSI)

These results indicate that ResNet has a high training accuracy but has difficulty maintaining this performance on the test set, as I have explained in the previous section indicating overfitting, resulting in a drop in the F1-score on the test set. However, ResNet still gives a relatively good performance with balanced precision and recall, especially in classifying class MSS.

**The Over-all analysis:**

VGGNet16 outperformed the other CNN architectures evaluated for MSI/MSS classification in gastrointestinal cancer histopathology images, with an accuracy of up to **70%**. VGGNet16 not only has the greatest accuracy, but also the best balance of precision, recall, and F1 score in the test set. This shows that it can accurately detect MSI and MSS with high reliability. Following VGGNet16, ResNet shows promising performance with an accuracy of **68%**, although signs of overfitting are evident as evidenced by a noticeable drop in performance on the test set compared to training. AlexNet and GoogleNet have intermediate performance metrics (**64%** and **63%** accuracy, respectively) and might benefit from more optimization or more complex architectures to increase their efficacy in this classification problem. Finally, VGGNet16 is suggested for clinical application because of its high accuracy and balanced metrics, making it a dependable option for MSI/MSS classification in histopathology images. Further refinement through larger datasets and additional techniques would increase its accuracy for practical applications.

### Effectiveness assessment

To Assess the effectiveness of my 4 architectures, I need to reach and assess specific insights about memory usage, model complexity, and computational efficiency these insights will be generated by considering 3 main elements: the training time, the number of parameters, and the overall computational requirements.

* **Number of Parameters:**

AlexNet: **57,012,034** parameters

VGGNet: **134,268,738** parameters

GoogleNet: **5,601,954** parameters

ResNet: **11,177,538** parameters

Calculating the number of parameters for the 4 architectures is important when I want to evaluate the computational complexity. The large number of parameters requires more memory but can also provide an ability doe learn complex patterns.

VGGNet16 has the greatest number of parameters model. This wealth of parameters offers VGGNet16 a high capacity for learning complicated features but at the expense of increased memory consumption and computational complexity. In comparison, GoogleNet is efficient, proving that it's capable of delivering competitive performance while requiring less memory. AlexNet and ResNet are ranked in the middle in terms of parameter count. This takes AlexNet closer to VGGNet16 in terms of complexity, but ResNet reaches a balance with a reasonable number of parameters, proving its success in balancing model complexity with computing efficiency. These variations highlight how the number of parameters directly influences model memory usage and computational needs.

* **Training Time:**

AlexNet: **1216.92** seconds

VGGNet16: **853.29** seconds

GoogleNet: **798.23** seconds

ResNet: **593.72** seconds

Calculating training time is important when I want to evaluate the computational efficiency model, faster training time is usually what we look for, specifically for large datasets.

ResNet ranks as the most computationally efficient. This efficiency is related to the ResNet architecture, which employs residual connections to enable deeper networks while having minimal computing cost. GoogleNet follows closely behind, showing its efficient architecture while having fewer parameters than VGGNet16. VGGNet16's longer training time is likely because of its deeper layer structure and a bigger number of parameters. In comparison, AlexNet has the longest training time, which is consistent with its earlier design and relatively high parameter complexity.

These findings demonstrate ResNet's applicability for tasks requiring quick model training, as well as the trade-offs between architecture complexity and computing efficiency in deep learning models.

* **Overall Computational Requirements:**

AlexNet's huge number of parameters allows it to learn complicated patterns successfully. However, this comes at the expense of lengthier training times and increased memory use. VGGNet16 excels in accuracy and balanced exact recall performance, but it has a relatively large number of parameters, making it memory-intensive for reasonable training times. GoogleNet contains the fewest parameters among the models, resulting in reasonable training times and efficient memory usage, while it may have somewhat smaller accuracy than VGGNet16. ResNet provides an appropriate mix of accuracy and recall, with quicker training times and fewer parameters than VGGNet16. However, warning signs of excessive processing can arise, which may be reduced by using suitable regulation techniques. These considerations highlight the necessity of selecting a model that fulfills certain computational constraints while giving good performance across different metrics.

**The Overall assessments:**

When comparing the efficacy of the 4 architectures, VGGNet16 outperforms the others in terms of performance parameters, including the greatest accuracy and balanced precision recall. However, because of the huge number of parameters, it requires a significant amount of memory. GoogleNet and ResNet provide an ideal balance between computational efficiency and performance. GoogleNet is the most memory efficient, although ResNet has the quickest training time with a decent set of parameters.

Best overall selection: VGGNet16 For applications that require accuracy and do not have limited memory.  
The most suitable alternative: is ResNet for applications that demand a balance of short training time and strong performance.

### Interface development

My interface preprocesses images, loads the model with specific classification adjustments, and predicts image classes inside a specified directory. The predictions are visually presented next to each image, indicating what is the predicted class.

I have uploaded a file with a test sample of images of total 12 images, 6 of them were labeled MSI and 6 of them were labeled MSS.

Here is the prediction of the models on the 12 test sample images:

* AlexNet:

Accuracy: **64%**

Predictions:

MSI class: **1/6**

MSS class: **2/6**

* VGGNet:

Accuracy: **70%**

Predictions:

MSI class: **5/6**

MSS class: **4/6**

* GoogleNet:

Accuracy: **63%**

Predictions:

MSI class: **2/6**

MSS class: **1/6**

* ResNet:

Accuracy: **68%**

Predictions:

MSI class: **2/6**

MSS class: **3/6**

### Evaluation of models

Based on the performance of the 4 architectures, I can critically evaluate their effectiveness in meeting end-user requirements. This evaluation considers several key aspects including prediction performance, User Requirements, and Future Improvements

Prediction Performance:

* **AlexNet:** It has obtained a relatively low accuracy. It has shown limited success in classifying MSI and MSS images.
* **VGGNet:** It showed the highest accuracy among the models and performed well in MSI and MSS classifications.
* **GoogleNet:** Moderate accuracy was demonstrated, with lower performance, especially in MSI image classification.
* **ResNet:** It demonstrated competitive accuracy, with balanced performance in MSI and MSS image classification.

According to the results of my project, VGGNet16 provides the best performance, however, it has the potential for overfitting. Similarly, ResNet performs well but displays evidence of overfitting. To address these concerns, future improvements might include improved regularization techniques and cross-validation techniques. These tactics will improve generalization while reducing overfitting, resulting in more trustworthy model performance.

In practice, VGGNet16 delivers the highest possible accuracy and performance in MSI and MSS classification of images. This makes it ideal for applications that need precise image classification. While AlexNet, GoogleNet, and ResNet18 have demonstrated a range of performance levels, their applicability will be determined by application-specific needs such as computational efficiency and memory restrictions.

User Requirements and Future Improvements:

* **Real-Time Application:**

According to the results of my project GoogleNet and ResNet are better suited for real-time applications since they have reduced the computation needs. To increase the practicality of VGGNet16 and AlexNet for real-time applications, future improvements should focus on decreasing their parameters using techniques like pruning. This enhancement will minimize memory use and computational needs, making VGGNet16 and AlexNet more suitable for real-time applications while preserving speed.

* **Memory Requirements:**

Based on the results of my project, VGGNet uses a lot of memory, making it difficult to install on devices with restricted capabilities. To remedy this, future improvements must concentrate on model compression techniques. These techniques may reduce memory usage while preserving performance, making VGGNet more suitable for deployment on resource-constrained devices.

* **Computational Efficiency:**

According to my project results, ResNet and GoogleNet currently give efficient training times, however VGGNet16 and AlexNet take longer. To address this, future improvements should look into newer architectures or modifications that preserve excellent accuracy while decreasing the computation needs.

* **User-Friendly Interface:**

Based on my user-friendly interface with image preprocessing and prediction display is in place. Future improvements should enhance the interface to include real-time prediction capabilities.

By addressing these aspects, the applied learning solution can better meet end-user requirements, providing a more efficient, and user-friendly interface for MSI/MSS classification in histopathology images of gastrointestinal cancer.

**The Overall evaluation**:

VGGNet16 exceeds the other CNN architectures, having the greatest accuracy and balanced precision-recall metrics for MSI and MSS classification. Despite its risk of overfitting, VGGNet16 is quite good at properly classifying histopathological images. GoogleNet and ResNet also score well in terms of computational efficiency and balanced metrics. Meanwhile, AlexNet shows potential but requires more optimization to improve classification accuracy. Future advancements in regularization approaches and model compression will be essential to improving generalization across all models.

## **References**

* Titoriya, A. and Sachdeva, S., 2019, November. Breast cancer histopathology image classification using AlexNet. In *2019 4th International conference on information systems and computer networks (ISCON)* (pp. 708-712). IEEE.
* Kaur, T. and Gandhi, T.K., 2019, December. Automated brain image classification based on VGG-16 and transfer learning. In *2019 international conference on information technology (ICIT)* (pp. 94-98). IEEE.
* Sharma, N., Jain, V. and Mishra, A., 2018. An analysis of convolutional neural networks for image classification. *Procedia computer science*, *132*, pp.377-384.
* Sarwinda, D., Paradisa, R.H., Bustamam, A. and Anggia, P., 2021. Deep learning in image classification using residual network (ResNet) variants for detection of colorectal cancer. *Procedia Computer Science*, *179*, pp.423-431.
* Lever, J., 2016. Classification evaluation: It is important to understand both what a classification metric expresses and what it hides. *Nature methods*, *13*(8), pp.603-605.
* Shi, B., 2021. On the hyperparameters in stochastic gradient descent with momentum. *arXiv preprint arXiv:2108.03947*.
* Li, H., Chaudhari, P., Yang, H., Lam, M., Ravichandran, A., Bhotika, R. and Soatto, S., 2020. Rethinking the hyperparameters for fine-tuning. *arXiv preprint arXiv:2002.11770*.
* Smith, L.N., 2018. A disciplined approach to neural network hyper-parameters: Part 1--learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*.