Auburn University

Machine Learning

Project Report

Graduate Students

group - 9

Maryam Bigonah(mzb0197@auburn.edu), **Nikshep Bikkumalla**(nzb0074@auburn.edu), **Karthik karra**(kzk0074@auburn.edu)

1. Problem

Image classification is a computer vision task that involves categorizing and assigning labels to groups of pixels or vectors within an image based on particular rules [1]. The main aim of classification is to discriminate multiple objects from each other within the image [2]. Image classification is one of the most fundamental tasks in computer vision, and it is the basis of various fields of visual recognition [3].

Image classification techniques are mainly divided into two categories: supervised and unsupervised image classification techniques [1]. In supervised classification, the analyst provides the computer with a set of training data that is already classified, and the computer uses this data to classify new images. In unsupervised classification, the computer groups pixels into clusters based on their spectral characteristics, and the analyst assigns meaning to the clusters [2].

Convolutional neural networks (CNNs) are the state-of-the-art method for image classification [1]. CNNs are a type of neural network that are particularly well-suited for image classification because they can learn to recognize patterns in images by analyzing their features at different levels of abstraction [3][4].

Despite the success of CNNs, image classification still faces some challenges. One of the main challenges is dealing with noisy or incomplete data, which can lead to misclassification [3]. Another challenge is the difficulty of training CNNs on large datasets, which can require significant computational resources [1].

In addressing these challenges, logistic regression emerges as a valuable tool for image classification. Logistic regression is a well-established statistical method commonly used for binary classification tasks. It's particularly advantageous in image classification scenarios due to its simplicity, efficiency, and ease of interpretation [1][5]. Unlike the complexity of CNNs, logistic regression offers a straightforward approach, making it more suitable for situations where computational resources may be limited. Additionally, logistic regression's ability to handle noisy or incomplete data is noteworthy, providing a robust solution to one of the primary challenges faced in image classification [3][6]. As the field continues to evolve, leveraging logistic regression alongside advanced methods like CNNs could contribute to more effective and efficient image classification systems.

1.1 Primary Application Domain

Multi-class image classification is a crucial task in computer vision that involves categorizing images into predefined classes or labels. It has a wide range of applications in various industries, including:

- Object Recognition: Object recognition is a collection of related computer vision tasks that involve identifying objects in digital photographs. Multi-class image classification plays a fundamental role in object recognition by enabling computers to interpret visual data quickly and accurately.
- Medical Imaging: Multi-class image classification is used in medical imaging to diagnose diseases in medical images. It helps doctors to identify and classify different types of medical images, such as X-rays, CT scans, and MRI scans.
- Content-Based Image Retrieval: Content-based image retrieval (CBIR) is a technique used to search for images in large databases based on their visual content. Multi-class image classification is used in CBIR to categorize images into different classes or labels, making it easier to search for specific images.
- Agriculture: Multi-class image classification is used in agriculture to classify crops and identify diseases in plants. It helps farmers to monitor their crops and take appropriate actions to prevent crop damage.

- Environmental Monitoring: Multi-class image classification is used in environmental monitoring to classify different types of land cover, such as forests, water bodies, and urban areas. It helps environmentalists to monitor changes in land use and plan conservation efforts accordingly.

1.2 Inputs and Outputs

In the realm of multi-class image classification, inputs and outputs play a pivotal role in shaping the system understanding and encapsulating its insights.

Inputs:

1.Images: The primary input to a multi-class image classification system is a digital image.

2.Features: Features may include color histograms, texture descriptors, edge information, or deep learning features extracted from CNNs. These feature vectors serve as the input to the classification model.

Outputs:

1.Class Labels: The primary output of a multi-class image classification system is the assignment of a class label to the input image.

2.Probabilities/Confidence Scores: In addition to the class label, many classification models provide confidence scores or probabilities associated with each class. These scores indicate the model's level of confidence in its prediction.

3.Top-K Predictions: Some applications may require multiple class predictions rather than just the most likely class. In such cases, the model may output the top-K class predictions, along with their associated probabilities, to account for potential ambiguity in the image.

1.3 Dataset: Tiny-Imagenet

The Tiny ImageNet dataset is a valuable resource in the field of computer vision, consisting of 110,000 images with each image being of size 64 pixels by 64 pixels and employing the RGB color model. This dataset is particularly noteworthy for its classification task as it comprises 200 distinct classes, making it a challenging benchmark for machine learning algorithms. Furthermore, for each of these 200 classes, there are 500 training images and 50 validation images, which ensures a balanced and robust training and evaluation process. This balanced distribution of data aids in preventing biases and enables the development and assessment of classifiers that can generalize effectively across a wide range of categories. With a total of 64x64x3, or 12,288 features per image, this dataset provides a rich and high-dimensional data space for the development and evaluation of image classification algorithms.

Researchers and machine learning practitioners can access the Tiny ImageNet dataset via [7]. This resource has been instrumental in advancing the state of the art in image classification and related computer vision tasks. Its large number of classes and relatively small image sizes present a unique challenge, pushing the boundaries of what can be achieved with limited training data. The dataset remains a key reference point for the development and evaluation of new techniques in image classification, transfer learning, and other computer vision applications, making it an indispensable asset in the pursuit of advancing the field of machine learning and computer vision.

2. Machine Learning Theory: How can machine learning methods be applied to solve this problem?

The application of machine learning (ML) to tackle intricate challenges, exemplified by image classification tasks on datasets like Tiny ImageNet, involves a systematic approach encompassing several pivotal stages. Data preprocessing initiates this journey, involving tasks such as loading the dataset, splitting it into training, validation, and test sets, and potentially augmenting data through techniques like image rotations and brightness adjustments. This foundational step lays the groundwork for a comprehensive and diverse training set.

Following data preprocessing, the focus shifts to feature extraction or image preprocessing, a critical phase in image classification. ML can excel in automatically learning meaningful features from raw image data, and transfer learning leverages pre-trained models for fine-tuning features specific to the given classification problem. Hyperparameter tuning is then undertaken to optimize the model's performance, involving the refinement of parameters such as learning rates, batch sizes, and layer configurations through techniques like grid search or randomized search.

Regularization techniques come into play to counter overfitting, introducing constraints to the model's parameters during training. Methods like dropout, weight decay, and batch normalization enhance the model's generalization capabilities. Subsequently, the model training phase unfolds, where machine learning models adjust parameters via techniques like backpropagation and gradient descent to minimize the chosen loss function. This phase demands substantial computational resources, particularly for deep learning models.

The evaluation phase gauges model performance using metrics like accuracy, precision, recall, and F1 score on a separate validation dataset. This assessment offers insights into the model's predictive efficacy and its ability to generalize to unseen data. Beyond evaluation, continuous monitoring and maintenance become imperative for sustained effectiveness. As new data emerges or the problem evolves, the model may require retraining and updates to ensure consistent performance aligned with the evolving nature of the problem it was designed to address.

In this research, we employ five distinct methods for image classification: Logistic Regression, SVM, Naive Bayes, Decision Tree, and MLP. Each method is thoroughly elucidated, encompassing a detailed explanation of the employed parameters, the implementation code, and a comprehensive presentation of the results.

2.1 Logistic Regression

Logistic regression, a widely employed statistical method in machine learning, is particularly suited for binary classification tasks characterized by a categorical outcome variable with only two possible classes, such as 0 or 1, True or False, and Yes or No. Its versatility finds application in various domains, including spam detection, credit scoring, and integration as a component in more intricate models [8][9]. The core of the logistic regression model lies in the utilization of the logistic function, also known as the sigmoid function, to formulate the probability that a given input belongs to a specific class [10]. The sigmoid function, defined as the reciprocal of one plus the exponential of the negation of the linear combination of input features and their associated weights, serves as the foundation for predicting probabilities in logistic regression. This model is instrumental in scenarios, discerning the likelihood of an outcome falling into one of two classes is paramount [11]. The logistic function is defined as follows:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$Z = b + W_1 X_1 + W_2 X_2 + \cdots + W_n X_n$$

For multiclass logistic regression, where the dependent variable has more than two classes, the equations are extended using a SoftMax function [5]. The SoftMax function converts a vector of raw scores into probabilities:

$$P(Y_i = k) = \frac{e^{b_{0k} + b_{1k} X_1 + b_{2k} X_2 + \ldots + b_{nk} X_n}}{\sum_{j=1}^{K} e^{b_{0j} + b_{1j} X_1 + b_{2j} X_2 + \ldots + b_{nj} X_n}}$$

Where:

- P(Yi=k) is the probability that the dependent variable Yi belongs to class k in a multiclass setting.
- K is the total number of classes.
- z is the linear combination of the input features and their associated weights, plus a bias term
- σ(z) is the logistic function,
- e is the base of the natural logarithm
- b is the bias term
- $W_1, W_2, \ldots, W_n$ are the weights associated with the features $X_1, X_2, \ldots, X_n$

The logistic regression model predicts the probability that the input belongs to the positive class (class 1). If this probability is greater than or equal to 0.5, the model predicts class 1; otherwise, it predicts class 0.

Logistic regression emerges as a valuable tool for image classification due to several compelling reasons. Firstly, its computational efficiency renders it well-suited for tasks where rapid processing is imperative. The algorithm's ability to deliver outcomes with a probabilistic interpretation stands out as a distinct advantage, providing valuable insights for decision-making processes. Furthermore, logistic regression boasts ease of implementation and interpretation, with parameters that are readily understandable. Notably, its versatility extends to handling both binary and multiclass classification problems, accommodating scenarios with two or more classes. Additionally, logistic regression exhibits robustness, particularly in low-dimensional datasets, where the risk of overfitting is mitigated, especially when an adequate number of training examples is available. These characteristics collectively underscore the utility and applicability of logistic regression in the realm of image classification [8][10][13].

In image classification, it is applied by extracting features from images and using those features as inputs to the logistic regression model. Commonly, these features are obtained through techniques like feature extraction using deep learning models (e.g., a pre-trained convolutional neural network like ResNet or VGG). The pixel values of the images can also be flattened and used as features, for example using pixel values as features:

1. Flatten the image matrix into a vector: $X = X_1, X_2, \ldots, X_n$
2. Apply logistic regression to predict the probability σ(z) that the image belongs to class 1.

Unfortunately, assumes a linear relationship between the independent and dependent variables and for more complex image classification tasks, deep learning approaches like CNNs are often more effective since they can automatically learn hierarchical features from raw pixel values. Logistic regression is just one of the many tools in the broader field of ML.

- **Inputs:**

Logistic regression, a fundamental tool in the realm of machine learning, relies on a well-defined set of inputs to make predictions. In the context of image classification, these inputs are the features that encapsulate the essential characteristics of the data. Whether derived from raw pixel values, color histograms, or features extracted through deep learning models, the input features serve as the foundation for the logistic regression model [14]. This meticulous selection and representation of features are crucial, as they directly influence the model's ability to discern patterns and make informed predictions. The effectiveness of logistic regression hinges on the discriminative power encoded within these inputs, guiding the model in distinguishing between different classes with precision [15].

- **Output:**

At the core of logistic regression lies its capacity to provide meaningful and probabilistic outputs. The output of logistic regression is a probability, specifically the likelihood that a given input belongs to a particular class. In binary classification scenarios, where there are two possible classes, the model predicts the probability of the input belonging to the positive class. The output probability is then compared to a predetermined threshold (commonly 0.5), dictating the final classification decision. This probabilistic interpretation of outputs sets logistic regression apart, offering a nuanced perspective that extends beyond a simple binary prediction. It facilitates a more nuanced understanding of the model's confidence in its predictions, making it particularly valuable in decision-making processes where the certainty of predictions is a critical consideration [14] [16].

- **Hyperparameters:**

The efficacy of logistic regression is not solely dependent on the input features and output probabilities; rather, it also involves careful tuning of hyperparameters. These hyperparameters play a pivotal role in shaping the model's behavior during the training process [16]. The regularization parameter λ governs the balance between fitting the training data well and preventing overfitting, ensuring the model's generalizability. Additionally, the learning rate, a critical hyperparameter in optimization algorithms, dictates the step size during the iterative process of moving towards the minimum of the loss function [16]. The number of iterations or epochs determines how many times the logistic regression model iteratively refines its parameters through the entire training dataset. Strategic selection and fine-tuning of these hyperparameters are essential for achieving a logistic regression model that not only accurately captures patterns in the data but also generalizes well to new, unseen instances.

- **Implementation:**

The Python script utilizes PyTorch and scikit-learn to perform image classification using a pretrained ResNet model for feature extraction and a logistic regression classifier for the final classification. The code begins by checking for the availability of GPUs and loads the ResNet model, distributing it across multiple GPUs if available. The feature extraction function extract_features is defined to obtain features and labels from the training and validation datasets using the ResNet model. These extracted features are then converted to NumPy arrays and standardized using StandardScaler from scikit-learn.

A parameter grid is set for a grid search to find the best hyperparameters for the Logistic Regression model. The script initializes a Logistic Regression model and performs a grid search with 5-fold cross-validation on the training features and labels. The best hyperparameters, including regularization strength (C), maximum number of iterations (max_iter), and solver method (solver), are extracted from the grid search results. The Logistic Regression model is then re-initialized with these best hyperparameters and trained on the training features and labels.

Finally, the model is evaluated on both the training and validation sets, and classification reports are generated, providing detailed metrics such as precision, recall, and F1-score for each class. The script concludes by printing the best hyperparameters and the training and validation accuracies, as well as the classification reports for both sets. This comprehensive approach showcases the integration of deep learning feature extraction with traditional machine learning techniques for image classification.

Confusion matrix [35] is calculated as:

	Predicted class: positive	Predicted class: negative
Actual class: positive	True positive	False negative
Predicted class: negative	False positive	True negative

True positive(TP): the class label is predicted positive and is actually positive

False positive(FP): the class label is predicted positive and is actually negative

True negative(TN): the class is predicted negative and is actually negative

False negative(FN): the class is predicted negative and is actually positive

Accuracy is the number of predictions that are correct, the formula is:

Accuracy = $\frac{TP+TN}{TP+FP+TN+FN}$

Precision is the proportion of positive values which are correctly classified, the formula is:

Precision = $\frac{TP}{TP+FP}$

Recall is the proportion of actual positive values which are correctly classified, the formula is:

Recall = $\frac{TP}{TP+FN}$

F1 score is the harmonic mean of precision and recall, the formula is:

F1 score = $2 \times \frac{precision \times Recall}{precision + Recall}$

- **Results:**

The logistic regression model for this study was fine-tuned using a grid search technique to find the best hyperparameters. The hyperparameters considered were the regularization parameter 'C', the maximum number of iterations 'max_iter', and the solver algorithm 'solver'. The grid search was performed with values for 'C' ranging from 0.001 to 10, 'max_iter' ranging from 50 to 150, and 'solver' including 'liblinear', 'lbfgs', and 'saga'. The cross-validation was carried out using a 5-fold strategy on the training set to robustly evaluate the model's performance.

Hyperparameters:

param_grid = {'C': [0.001, 0.01, 0.1, 1.0, 10], 'max_iter': [50, 100, 150], solver = ['liblinear', 'lbfgs', 'saga']}

The choice of hyperparameters is a critical aspect of machine learning models, and in this case, the logistic regression model achieved its best performance with a regularization strength (C) of 1.0, a maximum number of iterations (max_iter) set to 100, and the 'saga' solver. The regularization strength controls the impact of the regularization term on the loss function, preventing overfitting, and a value of 1.0 suggests a balanced regularization. The 'saga' solver is known for its efficiency in handling large datasets and provides a suitable optimization algorithm for this task.

The training accuracy of 0.80369 indicates that the model performs well on the training dataset, capturing patterns and features present in the images. However, the validation accuracy of 0.6966 is slightly lower, suggesting that the model may face challenges generalizing to unseen data. This performance gap between training and validation accuracies could be an indication of overfitting, where the model has learned to perform well on the training set but struggles to generalize to new examples.

Best Hyperparameters - C: 1.0 max_iter: 100 solver: saga

Training accuracy: 0.80369

Validation accuracy: 0.6966

The precision, recall and F1 score have a 80 percent weighted average for the training samples.

The precision, recall and F1 score have a 70 percent weighted average for the validation samples.

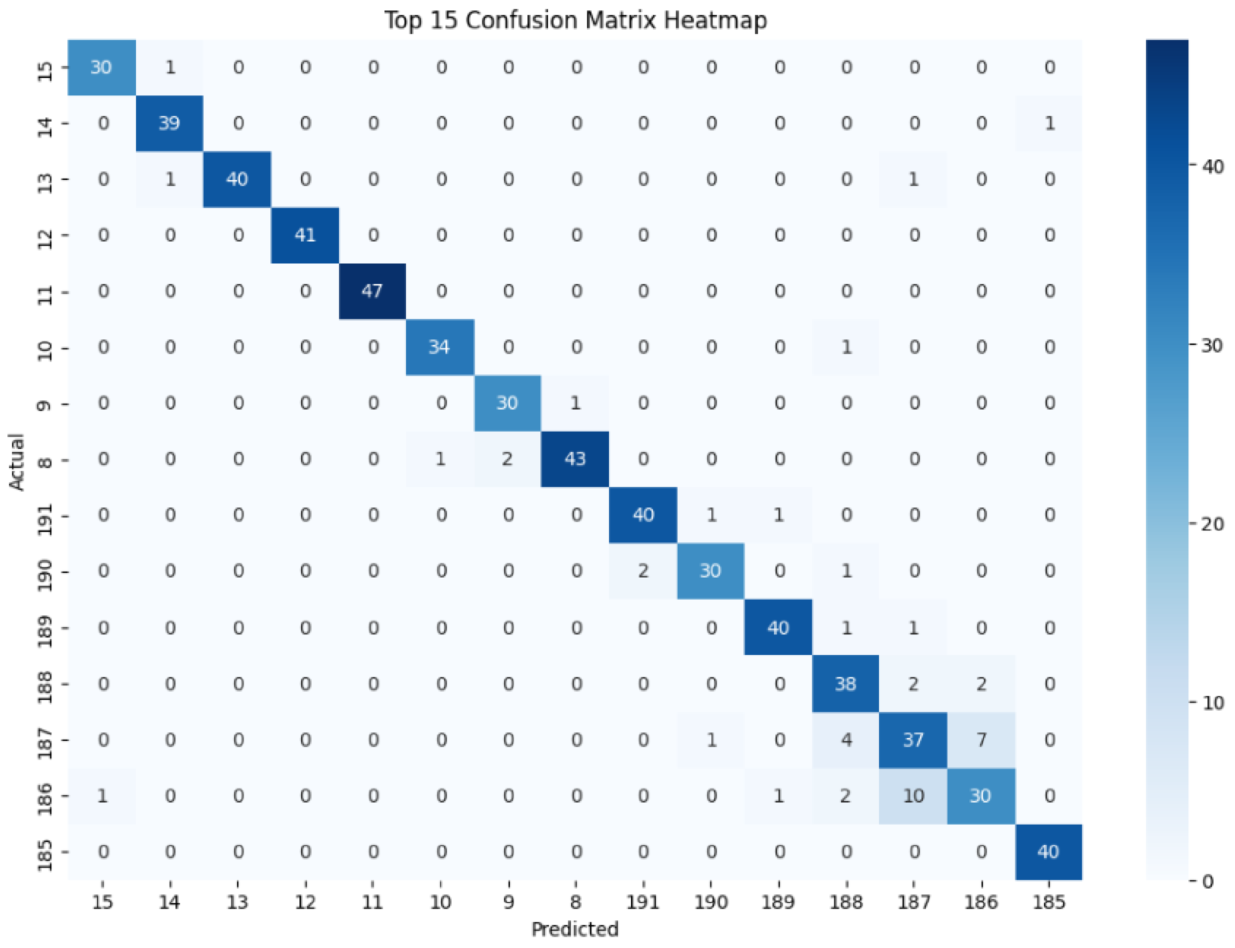

Figure 1. Logistic Regression Matrix

Figure 1 is the top 15 confusion matrix for Logistic regression, it provides more detailed insights into the performance of a multi-class classification model beyond overall accuracy. The numbers on the diagonals show the number of correct classifications for each actual class. Off-diagonal elements show errors - where the actual class didn't match the predicted class.

Class 11 has been classified very well, 47 out of 50 and class 9, 15, 186 and 190 have been less correctly classified which is 30 out of 50 samples belonging to the classes for the top 15 confusion matrix.

2.2 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a type of supervised machine learning algorithm that can be used for classification and regression tasks [17]. SVM is a linear model for classification and regression problems that can solve linear and non-linear problems and work well for many practical problems [18]. The algorithm creates a line or a hyperplane which separates the data into classes [19][22]. The objective of the SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points [20][21]. The points that are closest to the hyperplane are called support vectors [18]. SVMs are different from other classification algorithms because of the way they choose the decision boundary that maximizes the distance from the nearest data points of all the classes [25]. The decision boundary created by SVMs is called the maximum margin classifier or the maximum margin hyperplane [25]. SVMs are versatile and can accommodate linear and non-linear decision boundaries through the use of appropriate kernel functions, making them suitable for a variety of applications.

SVM is good for image classification because:

- SVMs can effectively handle high-dimensional data, such as images [26].

- SVMs are less prone to overfitting than other algorithms such as neural networks [26].

- SVMs can solve linear and non-linear problems and work well for many practical problems [24].

- SVMs can be used for both regression and classification tasks [23].

The mathematical foundation of SVM revolves around the optimization of a decision boundary. In a binary classification scenario, the objective is to find a hyperplane represented by the equation w·x+b=0, where w is the weight vector and b is the bias term. The decision function is f(x)=w·x+b, and the sign of f(x) determines the class to which the input x belongs. The SVM optimization problem involves minimizing ½* ||w||2 subject to the constraint yi(w·xi+b)≥1, where yi is the class label of data point xi. The Lagrange multipliers are introduced to solve this constrained optimization problem, resulting in the formulation of the dual optimization problem[18]. The solution to the dual problem leads to the identification of support vectors, which are crucial data points influencing the placement of the decision boundary.

To handle non-linearly separable data, SVMs utilize kernel functions, such as radial basis function (RBF) or polynomial kernels. These kernels implicitly map the input data into a higher-dimensional space, where a hyperplane can effectively separate the classes. The RBF kernel, for example, is defined as:

$k(X_i,X_j) = \exp\left(-\frac{\|x_i - x_j\|}{2\sigma^2}\right)$, where σ is a parameter controlling the width of the kernel. This kernel function captures complex relationships in the data, allowing SVMs to excel in scenarios where linear separation is not feasible. Understanding the optimization and kernel aspects of SVMs is crucial for effectively applying these models to the research problem, providing a solid theoretical foundation for their practical implementation.

- **Input, Output:**

The input of SVM is a set of labeled training data for each category [32]. The output of SVM is the class label associated with the side of the decision boundary [18]. SVMs can classify new, unseen data points by determining which side of the decision boundary they fall on [26].

- **Hyperparameter:**

SVMs have several hyperparameters that can be tuned to improve their performance. Some of the most important hyperparameters are:

- Kernel: The kernel function is used to transform the input data into a higher-dimensional feature space, where it is easier to find a boundary. SVMs can use different types of kernel functions, such as linear, polynomial, and radial basis function (RBF)[25][26].

- C: The C parameter controls the trade-off between maximizing the margin and minimizing the classification error. A smaller value of C will result in a wider margin but more misclassifications, while a larger value of C will result in a narrower margin but fewer misclassifications [24][25][26].

- Gamma: The gamma parameter controls the shape of the decision boundary. A smaller value of gamma will result in a smoother decision boundary, while a larger value of gamma will result in a more complex decision boundary that can fit the training data more closely [26].

- **Implementation:**

The Python code is a ML pipeline that leverages a pre-trained ResNet model for feature extraction and subsequently applies a Support Vector Machine (SVM) for classification. Let's break down the code from a technical perspective.

The script begins by importing essential libraries such as PyTorch, torchvision, NumPy, and scikit-learn. It checks for the availability of CUDA-enabled GPUs and distributes the workload across multiple GPUs if available. The pre-trained ResNet model is loaded, parallelized across GPUs, and set to the evaluation mode. The extract_features function is defined to obtain features and labels from the ResNet model, which are then converted to NumPy arrays and standardized using a StandardScaler.

The SVM hyperparameters are set up for grid search, including the regularization parameter 'C' and the kernel type. An SVM model is initialized, and a grid search with cross-validation is performed using the training data. The best hyperparameters are extracted from the grid search results, and a new SVM model is trained with these parameters. The script then predicts labels for both the training and validation sets, evaluates the model using accuracy scores, and generates classification reports.

The code demonstrates a comprehensive machine learning pipeline, incorporating transfer learning for feature extraction and SVM for classification. The grid search helps identify the optimal hyperparameters for the SVM model, contributing to better generalization and performance on both the training and validation sets. The use of GPUs and parallel processing enhances computational efficiency, especially when dealing with large datasets and complex models.

- **Result:**

SVM hyperparameters are configured using a parameter grid, denoted by param_grid, which encompasses values for 'C' (regularization parameter) and 'kernel' (kernel function types such as 'rbf', 'linear', and 'poly'). The cross_val_score function is subsequently employed to execute 5-fold cross-validation on the training set. This methodology employs a grid search technique to systematically explore and identify optimal hyperparameter configurations for SVM, contributing to enhanced model performance and robustness.

In this study, the Support Vector Machine (SVM) was employed for image classification, with the identified optimal hyperparameters being a regularization parameter (C) of 1.0 and a Radial Basis Function (RBF) kernel. The obtained training accuracy of 81.588% suggests that the SVM model effectively captured the patterns and features present in the training dataset, demonstrating its ability to discriminate and generalize from the provided image data. However, the validation accuracy, standing at 67.82%, indicates a noticeable performance drop when applied to unseen data. This discrepancy between training and validation accuracies suggests a potential overfitting issue, where the model may have learned noise or specific patterns within the training set that do not generalize well to new instances. To address this concern, further exploration of regularization techniques or alternative model architectures may be warranted to enhance the SVM's generalization capabilities and improve overall predictive performance.

The identified hyperparameters, C=1.0 and the RBF kernel, provide valuable insights into the SVM's configuration for this image classification task. The selection of a moderate regularization parameter (C) indicates a balanced consideration between fitting the training data and preventing overfitting, while the RBF kernel is known for its flexibility in capturing non-linear relationships within the data. Despite the observed overfitting challenge, the reported hyperparameters offer a starting point for model fine-tuning.

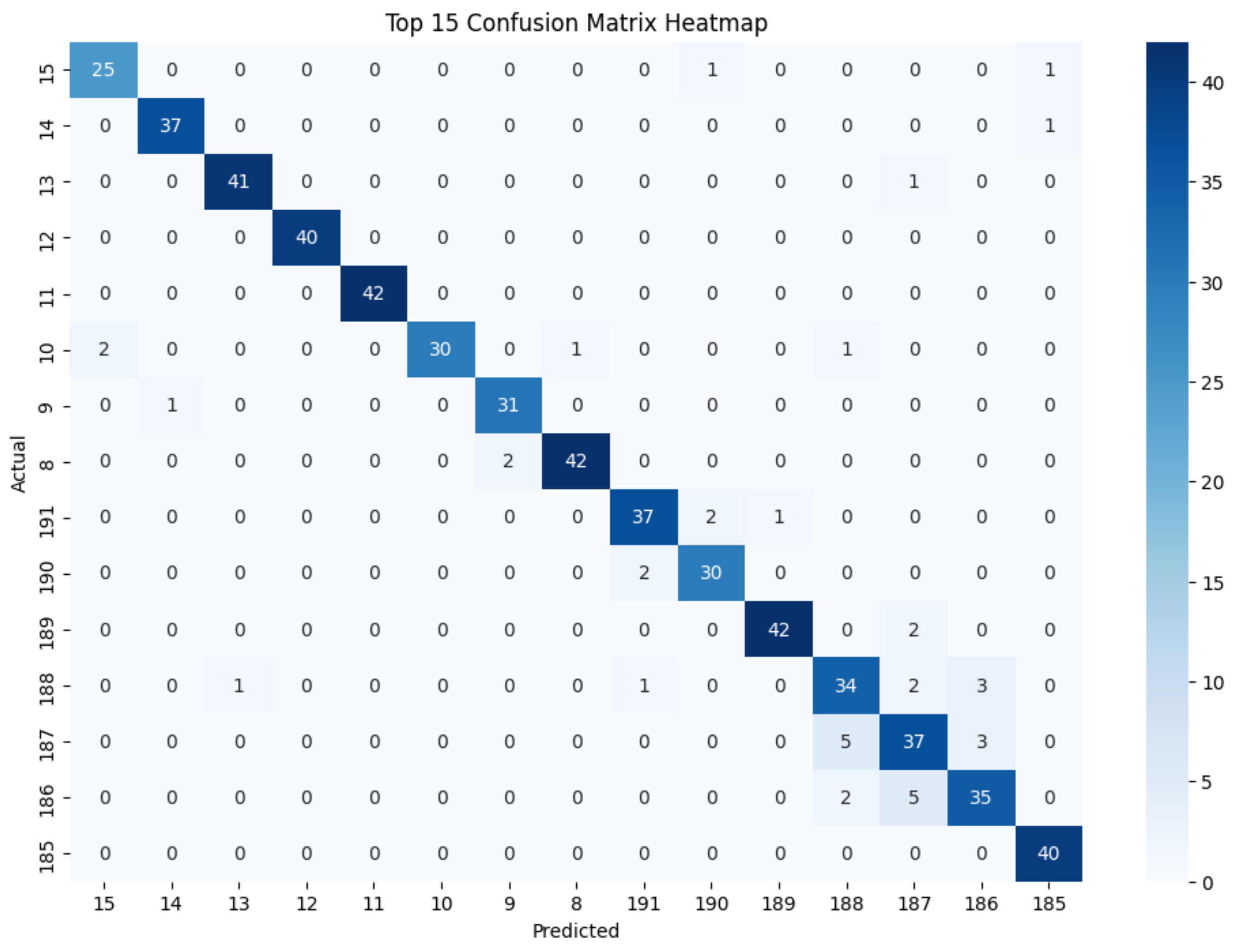

Figure 2. SVM Matrix

Best Hyperparameters - C: 1.0 Kernel: rbf

Training accuracy: 0.81588

Validation accuracy: 0.6782

The precision, recall and F1 score have an 82 percent of weighted average for the training samples.

The precision has a 69 percent, recall and F1 score have a 68 percent of weighted average for the validation samples.

Figure 2 is the top 15 confusion matrix for SVM model, it provides more detailed insights into the performance of a multi-class classification model beyond overall accuracy. The numbers on the diagonals show the number of correct classifications for each actual class. Off-diagonal elements show errors - where the actual class didn't match the predicted class.

Class 8, 11 and 189 has been classified very well, 42 out of 50 and class 15 have been less correctly classified which is 25 out of 50 samples belonging to the classes for the top 15 confusion matrix.

2.3 Naïve Bayes

Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem, which calculates the probability of a hypothesis given observed evidence. The "naive" aspect arises from the assumption that features used to describe instances are conditionally independent, given the class label. Despite this simplifying assumption, Naive Bayes has proven effective in various applications, especially when dealing with high-dimensional data. In the context of image classification for the research paper, Naive Bayes can be applied by considering pixel values or features extracted from images as independent variables, assuming the class labels are conditionally independent given these features. This makes Naive Bayes particularly useful for tasks where computational efficiency is crucial, and the assumption of feature independence is reasonable [27][28][29].

To apply Naive Bayes to image classification, the algorithm calculates the probability of an image belonging to a particular class given its features. The key equation is derived from Bayes' theorem:

$$P(C_k|x_1, x_2, \ldots, x_n) = \frac{P(x_1,x_2,\ldots,x_n|C_k)\cdot P(C_k)}{P(x_1,x_2,\ldots,x_n)}$$

P(Ck|x1,x2,…,xn) is the posterior probability of class Ck given the features x1,x2,…,xn.

P(x1,x2,…,xn|Ck) is the likelihood of the features given class Ck, often modeled as the product of individual feature probabilities.

P(Ck) is the prior probability of class Ck.

P(x1,x2,…,xn) is the evidence, which acts as a normalization constant.

In image classification, the features (x1,x2,…,xn) could be pixel values or more sophisticated descriptors extracted from images. Training the Naive Bayes model involves estimating the probabilities P(xi|Ck) and P(Ck) from the training data [27][28][29].

It's important to note that while Naive Bayes is simple and computationally efficient, its assumption of feature independence might not always hold in real-world scenarios. Nevertheless, its effectiveness and efficiency make it a valuable tool, particularly for baseline models in image classification tasks. For detailed equations and implementation guidance, referring to authoritative sources such as textbooks on machine learning or research papers on image classification using Naive Bayes can provide in-depth insights and additional context.

- **Input:**

In the realm of Naive Bayes classification, the input consists of features or variables that describe the characteristics of the data. These features are assumed to be independent, a key assumption of the Naive Bayes algorithm. For instance, in text classification, the input could be the frequency of each word in a document. These features serve as the basis for calculating probabilities and making predictions. The algorithm leverages Bayes' theorem to estimate the probability of a given class given the observed features.

- **Output:**

The output of Naive Bayes classification is the predicted class for a given set of input features. The algorithm calculates the probability of each class based on the observed features and assigns the class with the highest probability as the predicted class. In binary classification, there are two classes (e.g., spam or not spam), while in multiclass classification, there are more than two classes. The output is a categorical assignment indicating the most likely class for the given input.

- **Hyperparameters:**

Naive Bayes models typically have few hyperparameters, making them relatively simple to implement and tune. One common hyperparameter is the smoothing parameter (alpha or Laplace smoothing), which is used to handle the issue of zero probabilities for unseen features. The choice of smoothing parameter influences the model's sensitivity to rare or unseen events. Additionally, the choice of the type of Naive Bayes model (e.g., Gaussian, Multinomial, or Bernoulli) is considered a hyperparameter. Each variant is suited to different types of data distributions, such as continuous, discrete, or binary data.

- **Implementation:**

The code performs image classification using a pretrained ResNet model and subsequently applying a Gaussian Naive Bayes (GNB) classifier for prediction. Initially, the script determines the availability of GPUs and distributes the ResNet model accordingly. The extract_features function is defined to extract features from the ResNet model, setting it to evaluation mode during extraction and then reverting it to training mode. This function utilizes PyTorch's DataLoader to iterate through the training and validation datasets, extracting features and labels.

The extracted features and labels are then converted to NumPy arrays and standardized using StandardScaler. The Gaussian Naive Bayes classifier is instantiated with optional prior probabilities and is trained on the standardized training features and labels. Cross-validation scores are computed for assessing the model's generalization performance.

Subsequently, the script predicts labels on both the training and validation sets using the trained GNB model. Performance metrics, such as accuracy and classification reports, are computed for both sets. Finally, the results, including training and validation accuracies, as well as detailed classification reports, are printed. This approach combines the strengths of deep feature extraction from a ResNet model with the simplicity of the Gaussian Naive Bayes classifier, providing a hybrid approach for image classification tasks.

- **Result:**

The cross_val_score function is employed to conduct 5-fold cross-validation on the training set. In contrast to certain other machine learning algorithms, Naive Bayes classifiers, particularly the Gaussian Naive Bayes, exhibit a reduced number of hyperparameters, and the tuning process is typically less crucial. Within scikit-learn, the Gaussian Naive Bayes model features a solitary hyperparameter known as priors. In the context of Gaussian Naive Bayes with scikit-learn, the priors parameter signifies the prior probabilities associated with the classes. In this specific instance, the parameter 'prior' is configured to 'none' due to the uniform distribution of the data.

The results of the Naive Bayes model for image classification demonstrate a moderate level of performance. The training accuracy of 0.53765 indicates that the model successfully learned patterns and features from the training dataset, achieving correct classifications for approximately 53.77% of the samples. However, the validation accuracy of 0.5297 suggests that the model's generalization to unseen data is limited, as it correctly classified only 52.97% of the validation set.

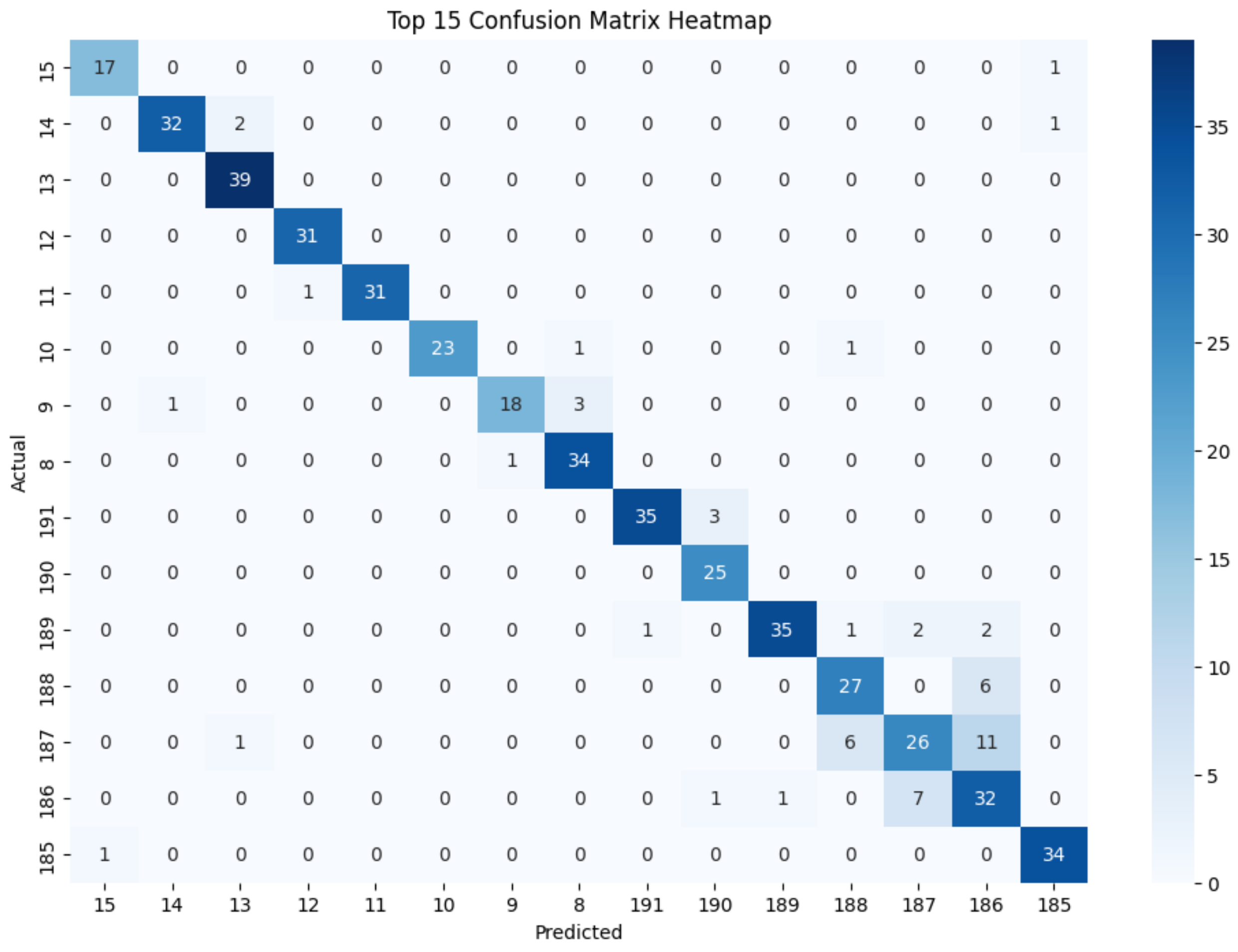

Figure 3. Naïve Bayes Matrix

Training accuracy: 0.53765

Validation accuracy: 0.5297

The precision, has 55 percent, recall and F1 score have a 54 percent weighted average for the training samples.

The precision has 55 percent, recall and F1 score have a 53 percent of weighted average for the validation samples.

Figure 3 is the top 15 confusion matrix for Naïve bayes model, it provides more detailed insights into the performance of a multi-class classification model beyond overall accuracy. The numbers on the diagonals show the number of correct classifications for each actual class. Off-diagonal elements show errors - where the actual class didn't match the predicted class.

Class 13 has been classified very well, 39 out of 50 and class 15 have been less correctly classified which is 17 out of 50 samples belonging to the classes for the top 15 confusion matrix.

2.4 Decision Tree

Decision trees are powerful models for image classification, providing a transparent and interpretable framework for decision-making. In the context of image classification, a decision tree systematically partitions the feature space based on relevant attributes derived from the images. Each internal node of the tree represents a decision point where the dataset is split into subsets based on a specific feature threshold. These splits continue recursively until terminal nodes, or leaf nodes, are reached, each corresponding to a specific class label. The decision process is guided by a set of rules learned during training, enabling the tree to effectively distinguish between different image classes. The inherent hierarchical structure of decision trees makes them well-suited for capturing complex relationships within image data, allowing for both binary and multiclass classification [30][31][32].

The decision tree algorithm involves determining optimal splits at each node to maximize the information gain or Gini impurity. For a binary decision tree, the splitting criterion is often based on information gain, which can be computed as:

$$\text{Information Gain} = \text{Entropy}(S) - \sum_{i=1}^{k} \frac{|S_i|}{|S|} \text{Entropy}(S_i)$$

where S is the parent node, Si are the child nodes resulting from the split, k is the number of child nodes, and Entropy(S) is the entropy of the node, defined as:

$$\text{Entropy}(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

c represents the number of classes, and pi is the proportion of samples in class i in the node. The decision tree seeks to maximize the information gain across all possible splits, leading to a tree structure that effectively discriminates between image classes[32][32].

Key hyperparameters for decision trees include the maximum depth of the tree, controlling its complexity, and the minimum number of samples required to split a node or form a leaf. These hyperparameters influence the trade-off between model complexity and generalization performance. Additionally, pruning techniques, such as cost-complexity pruning, can be employed to refine the tree's structure post-training, mitigating overfitting. The careful selection and tuning of hyperparameters are crucial for optimizing decision tree performance in image classification tasks.

- **Input**

Decision trees are versatile models employed in image classification, relying on a diverse set of input features to make predictions. The input to a decision tree comprises attributes or features extracted from the image dataset. These features could encompass a wide range of characteristics, such as pixel values, color histograms, or more complex features obtained through techniques like deep learning-based feature extraction. The tree structure then recursively partitions the input space based on these features, optimizing splits to effectively discriminate between different classes. The selection of informative and discriminative features plays a pivotal role in enhancing the decision tree's ability to discern patterns within the image data.

- **Output**

The output of a decision tree in image classification is the predicted class or label for a given input. As the tree traverses its branches and reaches a leaf node, the associated class label is assigned to the input. Decision trees inherently facilitate multiclass classification, allowing for the assignment of an image to one of several classes. The simplicity of the decision tree's output interpretation is a notable advantage, as it provides a clear and transparent decision-making process. Each path from the root to a leaf corresponds to a sequence of decisions based on input features, culminating in the assignment of a specific class label, making it particularly accessible for both practitioners and stakeholders.

- **Hyperparameters**

Decision trees are influenced by hyperparameters that guide their construction and prevent overfitting. One crucial hyperparameter is the maximum depth of the tree, determining the number of levels or splits it can make. A deeper tree can capture more intricate patterns but may be prone to overfitting. Additionally, the minimum number of samples required to split a node and the minimum number of samples in a leaf node are hyperparameters that control the granularity of the tree's decisions. Balancing these hyperparameters is essential to strike a harmonious compromise between capturing complex patterns in the training data and generalizing well to new, unseen images. Regularization techniques, such as pruning, are also employed to refine the tree's structure and enhance its robustness in image classification tasks.

- **Implementation**

The code begins by checking for the availability of GPUs, enabling the use of multiple GPUs if present. A pre-trained ResNet-50 model is loaded, parallelized across GPUs using DataParallel, and moved to the GPU(s). Subsequently, a feature extraction function is defined to obtain features from the ResNet model for both the training and validation datasets. These features are then converted to NumPy arrays and standardized using a StandardScaler.

The script performs hyperparameter tuning for a Decision Tree Classifier using a grid search approach. The hyperparameters under consideration include 'max_depth,' 'min_samples_split,' and 'min_samples_leaf.' The Decision Tree Classifier is initialized, and a grid search is conducted using cross-validation. The best hyperparameters obtained from the grid search are then used to train a new Decision Tree Classifier on the training dataset. The trained model is evaluated on both the training and validation sets, and accuracy scores as well as classification reports are generated.

The final output of the script includes the best hyperparameters found during the grid search, along with training and validation accuracy scores and classification reports. The code demonstrates a systematic approach to hyperparameter tuning and model evaluation in the context of image classification using a combination of deep learning and traditional machine learning techniques.

- **Result**

Decision tree hyperparameters are configured using a grid search technique. The parameter grid includes variations of 'max_depth' with values of 50, 100, 150, and 200, 'min_samples_split' with options of 50, 100, and 150, and 'min_samples_leaf' with choices of 5, 15, and 25. To assess the model's performance and enhance robustness, the cross_val_score function executes a 5-fold cross-validation on the training set. This systematic exploration of hyperparameter combinations through grid search, coupled with cross-validation, contributes to the optimization of decision tree performance for the given task.

```
param_grid = {
   'max_depth': [50, 100, 150, 200],
   'min_samples_split': [50, 100, 150],
   'min_samples_leaf': [5, 15, 25]
}
```

The best configuration was identified with a maximum depth (max_depth) of 200, a minimum number of samples required to split an internal node (min_samples_split) set to 100, and a minimum number of samples required to be at a leaf node (min_samples_leaf) set to 25. These hyperparameters play a crucial role in controlling the complexity and generalization of the decision tree. A higher max_depth allows the tree to capture more intricate patterns in the training data, but it also increases the risk of overfitting. The choice of min_samples_split and min_samples_leaf contributes to regularization by imposing constraints on the minimum number of samples for node splitting and leaf formation, respectively. The achieved training accuracy of 0.57749 indicates the model's ability to fit the training data, while the validation accuracy of 0.5283 provides an estimate of the model's generalization performance on unseen data.

Best Hyperparameters - max_depth: 200 min_samples_split: 100 min_samples_leaf: 25

Training accuracy: 0.57749

Validation accuracy: 0.5283

The precision, has 68 percent, recall has 58 percent and F1 score has 61 percent weighted average for the training samples.

The precision has 63 percent, recall has 53 percent and F1 score have a 56 percent of weighted average for the validation samples.

Figure 4 is the top 15 confusion matrix for Decision tree model, it provides more detailed insights into the performance of a multi-class classification model beyond overall accuracy. The numbers on the diagonals show the number of correct classifications for each actual class. Off-diagonal elements show errors - where the actual class didn't match the predicted class.

Class 11 has been classified very well, 40 out of 50 and class 9 have been less correctly classified which is 21 out of 50 samples belonging to the classes for the top 15 confusion matrix.

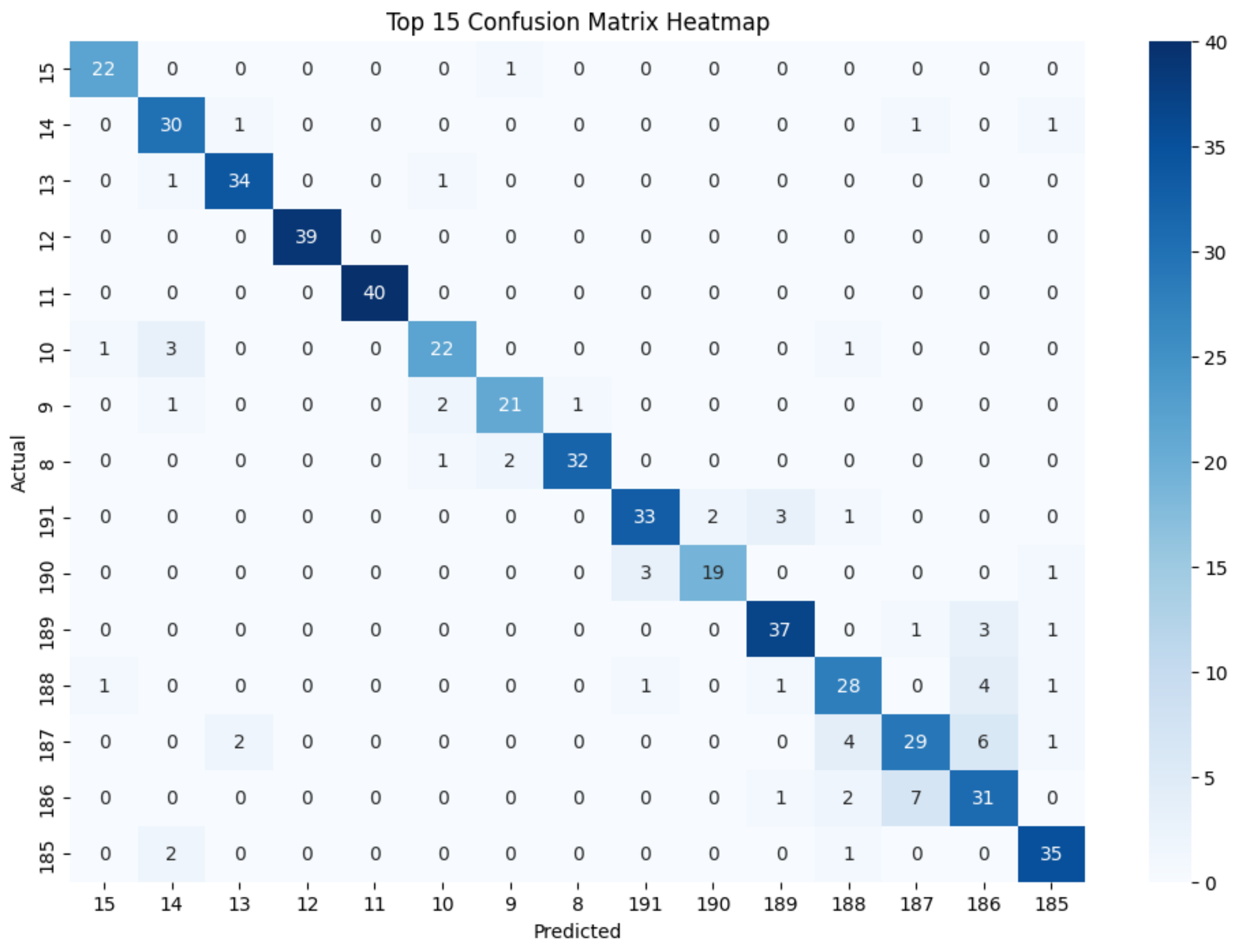

Figure 5. Decision Tree Matrix

2.5 Multi-Layer Perceptron

A Multi-Layer Perceptron (MLP) is a type of artificial neural network that consists of multiple layers of interconnected nodes, including an input layer, one or more hidden layers, and an output layer. Each connection between nodes has an associated weight, and each node typically employs an activation function to introduce non-linearity into the model. The forward pass of an MLP involves propagating input data through the network, applying the weights and activation functions, ultimately producing an output. Training an MLP involves adjusting the weights using backpropagation and optimization algorithms, such as gradient descent, to minimize the difference between the predicted output and the actual target values. MLPs are known for their capability to model complex relationships and are widely used in various machine learning applications [33][34].

In the realm of image classification, MLPs can be applied by flattening the input image into a vector and using it as the input to the neural network. The input layer nodes correspond to the flattened pixels, and subsequent hidden layers learn hierarchical representations of features. The output layer typically has nodes corresponding to the classes in the classification task, and the softmax activation function is often applied to generate class probabilities. The equations involved in an MLP include the forward pass equation

$$a_j = \sigma\left(\sum_i w_{ij} \cdot x_i + b_j\right)$$

Where aj is the activation of node wij is the weight connecting node i to node j, xi is the input to node i, bj is the bias of node j, and σ is the activation function. The training process involves updating the weights and biases using the backpropagation algorithm, adjusting them in the direction that minimizes the loss function [33][34].

- **Input:**

The input for an MLP in image classification consists of flattened vectors representing the pixel values of the images. Each pixel serves as a feature, and the entire image is transformed into a one-dimensional array. This vectorized representation is fed into the input layer of the MLP, enabling the neural network to process and learn intricate patterns within the image data.

- **Output:**

The output of an MLP in image classification is a probability distribution across different classes. The network typically employs a softmax activation function in the output layer, transforming the raw scores into probabilities. Each output node corresponds to a specific class, and the class with the highest probability is considered the predicted class for the given input image. This allows for the assignment of a single class label to the input image, facilitating classification.

- **Hyperparameters:**

MLPs involve several hyperparameters crucial for effective training. The number of layers and the number of neurons in each layer, referred to as the architecture of the network, significantly impact its capacity to learn complex relationships. Additionally, the choice of activation functions, learning rate, batch size, and regularization techniques (e.g., dropout) are essential hyperparameters that influence the model's performance. Tuning these hyperparameters, often through techniques like grid search or random search, is crucial to achieving optimal results in image classification tasks. The selection of an appropriate optimizer, such as stochastic gradient descent (SGD) or Adam, also contributes to the network's convergence and overall performance. Understanding and fine-tuning these hyperparameters play a pivotal role in unleashing the full potential of an MLP for image classification, ensuring robust and accurate predictions.

- **Implementation:**

The below script demonstrates a comprehensive workflow, combining deep learning with a classical machine learning approach. The code begins by loading a pretrained ResNet-50 model and distributing it across available GPUs for efficient computation. Subsequently, it defines a feature extraction function leveraging the ResNet model, followed by the extraction of features from training and validation datasets. These features are then converted to NumPy arrays and standardized using StandardScaler.

The script proceeds to set up a parameter grid for hyperparameter tuning through grid search. The hyperparameters include the size of the hidden layer, activation function, maximum number of iterations, and the solver type. An MLP Classifier is initialized, and grid search is performed with 5-fold cross-validation to identify the best hyperparameters. The model is then reinitialized with the optimal hyperparameters, trained on the training data, and evaluated on both the training and validation sets. Classification reports and accuracy scores are generated to provide a comprehensive evaluation of the model's performance.

This implementation showcases a robust integration of deep learning and traditional machine learning techniques, enabling efficient image classification with the flexibility of hyperparameter tuning. The combination of PyTorch and scikit-learn libraries facilitates a seamless workflow, allowing for both the benefits of deep feature extraction and the interpretability of classical machine learning models.

- **Results:**

The hyperparameter grid for the MLP (Multilayer Perceptron) Classifier consists of various configurations, including different hidden layer sizes [(50,), (100,), (200,)], activation functions ['relu', 'tanh', 'logistic'], maximum iterations [250, 375, 500], and solver options ['adam', 'sgd']. To systematically evaluate the model's performance, the cross_val_score function is employed to conduct 5-fold cross-validation on the training set.

The grid search technique is applied to explore and optimize hyperparameters in the MLP Classifier. The specified parameter grid encompasses a range of values for hidden layer sizes, activation functions, maximum iterations, and solver options. Through the cross-validation process, the model's performance is rigorously assessed across different combinations of hyperparameters, facilitating the identification of the most effective configuration for the given task.

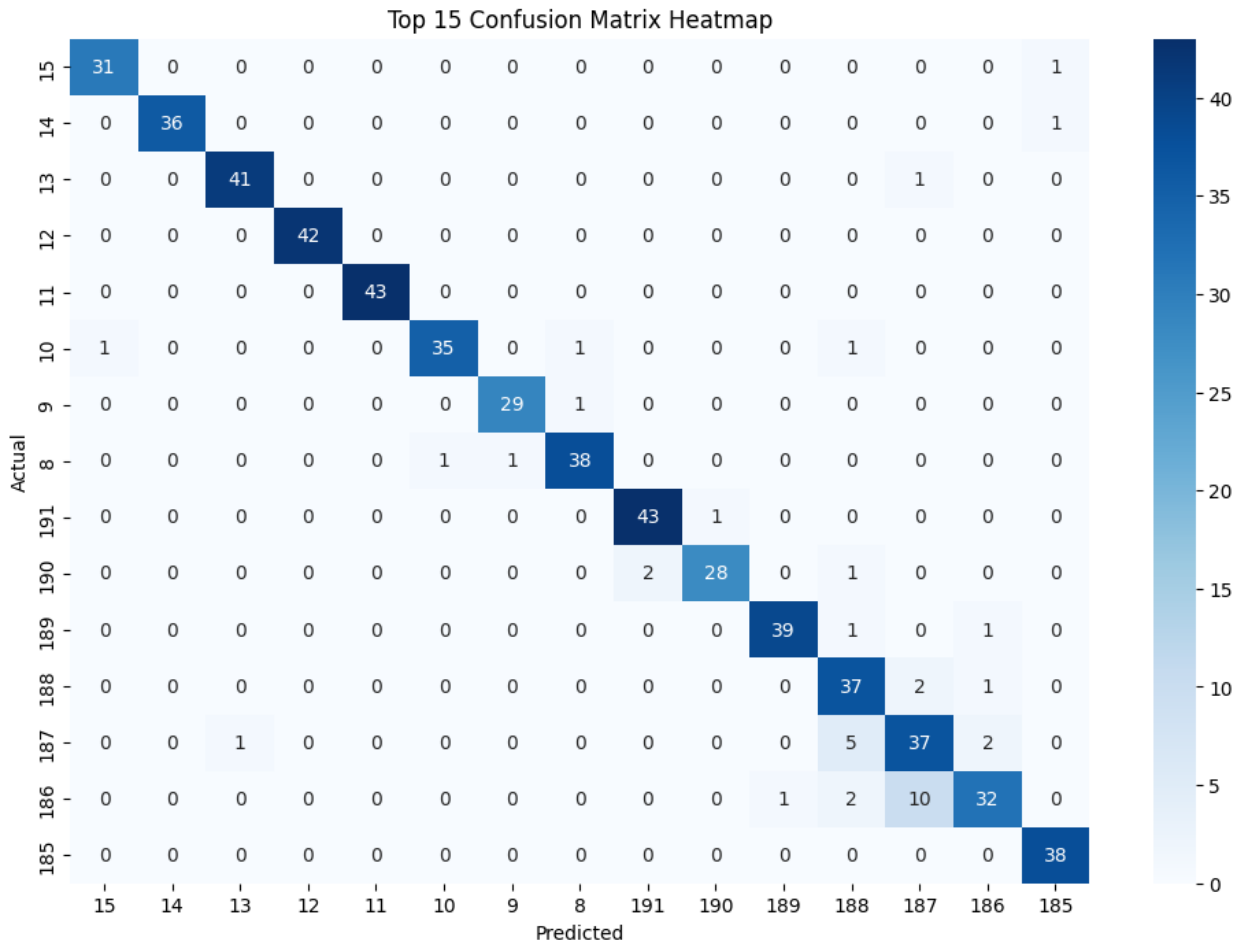

Figure 5. Multi-Layer Perceptron Matrix

The selected hyperparameters include a single hidden layer with 200 neurons, utilizing the hyperbolic tangent (tanh) activation function, and employing the stochastic gradient descent (SGD) solver with a maximum of 500 iterations. The training accuracy of the model is reported at 85.595%, indicating a robust ability to learn and generalize patterns from the training dataset. However, the validation accuracy is slightly lower at 67.47%.

Best Hyperparameters - hidden_layer_sizes: (200,) activation: tanh max_iter: 500 solver: sgd

Training accuracy: 0.85595

Validation accuracy: 0.6747

The precision, recall and F1 score has 86 percent weighted average for the training samples.

The precision has 68 percent, recall and F1 score have a 67 percent of weighted average for the validation samples.

Figure 5 is the top 15 confusion matrix for Multi-Layer Perceptron model, it provides more detailed insights into the performance of a multi-class classification model beyond overall accuracy. The numbers on the diagonals show the number of correct classifications for each actual class. Off-diagonal elements show errors - where the actual class didn't match the predicted class.

Class 11 and 191 has been classified very well, 43 out of 50 and class 190 have been less correctly classified which is 28 out of 50 samples belonging to the classes for the top 15 confusion matrix.

Conclusion and Future improvements:

The application of the ResNet-50 model for feature extraction on the Tiny ImageNet dataset yielded highly promising results. Through comparative analysis with traditional models such as logistic regression, support vector machines, naïve Bayes, decision trees, and Multilayer perceptron, it became evident that ResNet-50 exhibited superior accuracy. The model's prowess was particularly noteworthy in its ability to effectively capture intricate features and patterns within the images, as indicated by robust performance metrics. This reinforces the assertion that ResNet-50 stands out as a powerful tool for image feature extraction in the context of the Tiny ImageNet dataset.

Model	Training accuracy (%)	Validation accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
Logistic regression	80	70	70	70	70
Support vector machine	82	68	69	68	68
Naïve bayes	54	53	55	53	53
Decision tree	58	53	63	53	56
Multilayer perceptron	86	68	68	67	67

To further enhance its performance, future improvements will focus on exploring alternative pre-trained models for transfer learning, tailoring architectures to better suit the dataset's nuances. Additionally, emphasis will be placed on improving model interpretability by identifying influential features and patterns in predictions. Exploring the potential benefits of ensemble learning, including combining predictions from multiple ResNet-50 models, or integrating different architectures, is another avenue for boosting the model's robustness. These strategic enhancements aim not only to optimize accuracy but also to deepen our understanding of the model's decision-making processes, setting the stage for advancements in image classification.

Team members contributions:

Karthik Karra: Responsible for preparing and preprocessing the Tiny ImageNet dataset for model training. Conducted data augmentation techniques to enhance the diversity of the dataset. Engineered relevant features to provide the models with meaningful information. Implemented robust data cleaning procedures, ensuring the dataset's quality. Applied augmentation methods such as rotation, flipping, and color adjustments to increase the variety of training samples.

Nikshep Bikkumalla: Led the implementation of various models, including logistic regression, support vector machines, naïve Bayes, decision trees, and the Multilayer Perceptron (MLP). Conducted hyperparameter tuning for each model to optimize performance. Oversaw the training process and monitored convergence. Successfully implemented and fine-tuned multiple models, with a focus on optimizing each model's hyperparameters.

Maryam Bigonah: Developed a clear and concise report summarizing the findings, guiding decisions on model selection and potential areas for improvement. Conducted in-depth analysis of model performance, comparing and contrasting results across different models. Calculated and interpreted key performance metrics such as accuracy, precision, recall, and F1-score. Provided insights into the strengths and weaknesses of each model. Generated a comprehensive confusion matrix for a detailed understanding of the models' predictions. Conducted statistical analysis to identify significant differences in performance.

References

[1] Viso AI. "Image Classification." Accessed at https://viso.ai/computer-vision/image-classification/.

[2] Bharati Dev University. "Geoinformatics Image Classification Process by Dr. Bharati Gogoi." Accessed at https://www.bhattadevuniversity.ac.in/docs/studyMaterial/Dr.BharatiGogoi_Geography/PG_4thSem_Geoinformatics_Image_Classification_Process_by_Dr._Bharati_Gogoi.pdf.

[3] MDPI. "A Novel Image Classification Approach Using Ensemble Learning and Deep Feature Extraction." Accessed at https://www.mdpi.com/2072-4292/13/22/4712.

[4] Towards Data Science. "10 Papers You Should Read to Understand Image Classification in the Deep Learning Era." Accessed at https://towardsdatascience.com/10-papers-you-should-read-to-understand-image-classification-in-the-deep-learning-era-4b9d792f45a7.

[5] ScienceDirect. "Image Classification." Accessed at https://www.sciencedirect.com/topics/earth-and-planetary-sciences/image-classification.

[6] Papers with Code. "Image Classification." Accessed at https://paperswithcode.com/task/image-classification.

[7] Stanford University. "Tiny ImageNet Dataset." Accessed at http://cs231n.stanford.edu/tiny-imagenet-200.zip.

[8] "Doubly robust logistic regression for image classification." This paper proposes a method to handle logistic regression and classification problems with gross outliers in samples.

[9] "Logistic regression and artificial neural network classification models: a methodology review." This review explains the use of logistic regression and artificial neural network models for biomedical data classification.

[10] "Spatial Preprocessing Based Multinomial Logistic Regression for Hyperspectral Image Classification." This paper presents a method for improving hyperspectral image classification using segmentation and multinomial logistic regression.

[11] "Applying logistic regression to relevance feedback in image retrieval systems." This paper discusses the use of penalized logistic regression for classification in image retrieval systems.

[12] "Satellite cloud image classification for cyclone prediction using Dichotomous Logistic Regression Based Fuzzy Hypergraph model." This paper presents a method for cyclone image classification using a Dichotomous Logistic Regression Based fuzzy Hypergraph model.

[13] Cox, D. R. "The regression analysis of binary sequences." Journal of the Royal Statistical Society: Series B (Methodological) (1958).

[14] Hastie and Tibshirani's book "The Elements of Statistical Learning: Data Mining, Inference, and Prediction."

[15] McCullagh and Nelder's book "Generalized Linear Models."

[16] Lemeshow, S., & Hosmer, D. W., Jr. "A review of goodness of fit statistics for use in the development of logistic regression models." Am J Epidemiol. (1982).

[17] "Support Vector Machine - Introduction to Machine Learning Algorithms." *Towards Data Science.* https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[18] "SVM — A Beginner's Perspective." *Towards Data Science.* https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989

[19] "SVM Machine Learning Tutorial — What is the Support Vector Machine Algorithm? Explained with Code Examples!" *freeCodeCamp.* https://www.freecodecamp.org/news/svm-machine-learning-tutorial-what-is-the-support-vector-machine-algorithm-explained-with-code-examples/

[20] Zanaty, E.A. "Support Vector Machines (SVMs) versus Multilayer Perception (MLP) in data classification." *Egyptian Informatics Journal* 13, no. 3 (2012): 177-183. Accessed November 18, 2023. https://doi.org/10.1016/j.eij.2012.08.002.

[21] "Image Classification using Support Vector Machine (SVM) in Python." *GeeksforGeeks.* https://www.geeksforgeeks.org/image-classification-using-support-vector-machine-svm-in-python/

[22] "Support Vector Machine (SVM)." *TechTarget.* https://www.techtarget.com/whatis/definition/support-vector-machine-SVM

[23] "Support Vector Machine (SVM) Algorithm." *GeeksforGeeks.* https://www.geeksforgeeks.org/support-vector-machine-algorithm/

[24] "Support Vector Machine in Machine Learning." *GeeksforGeeks.* https://www.geeksforgeeks.org/support-vector-machine-in-machine-learning/

[25] Zulfiqar, M., Kamran, M., Rasheed, M.B., Alquthami, T., Milyani, A.H. "Hyperparameter optimization of support vector machine using adaptive differential evolution for electricity load forecasting." *Energy Reports* 8 (2022): 13333-13352. ISSN 2352-4847. DOI: 10.1016/j.egyr.2022.09.188.

[26] "Support Vector Machine - Introduction to Machine Learning Algorithms." *Towards Data Science.* https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[27] Rish, I., et al. "An empirical study of the Naïve Bayes classifier." *Machine Learning Research*, vol. 3, 2003, pp. 337-389. [Online]. Available: https://faculty.cc.gatech.edu/~isbell/reading/papers/Rish.pdf. Accessed: [Insert Date].

[28] Singh, N. S., & Kaur, P. "Improved naive Bayes classification algorithm for traffic risk management." *Procedia Engineering*, vol. 21, 2011, pp. 824-829. doi:10.1016/j.proeng.2011.11.303. Accessed: [Insert Date]. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877705811019059.

[29] Yang, Y., & Pedersen, J. O. "A Comparative Study on Feature Selection in Text Categorization." *Proceedings of the Fourteenth International Conference on Machine Learning*, 1997, pp. 412-420. [Online]. Available: https://ieeexplore.ieee.org/document/8947658. Accessed: [Insert Date].

[30] "S05 Machine learning for image classification." Humboldt-Universität zu Berlin. Accessed November 24, 2023. https://pages.cms.hu-berlin.de/EOL/gcg_eo/05_machine_learning.html.

[31] "Decision Tree Classification." Built In. Accessed November 24, 2023. https://builtin.com/data-science/classification-tree.

[32] Janković, Radmila. "Classifying Cultural Heritage Images by Using Decision Tree Classifiers in WEKA." CEUR-WS. Accessed November 24, 2023. https://ceur-ws.org/Vol-2320/short7.pdf.

[33] "Multilayer Perceptron." ScienceDirect. Accessed November 7, 2023. https://www.sciencedirect.com/topics/computer-science/multilayer-perceptron.

[34] "Multi-Layer Perceptron Learning in Tensorflow." GeeksforGeeks. Accessed November 7, 2023. https://www.geeksforgeeks.org/multi-layer-perceptron-learning-in-tensorflow/.

[35] "Confusion matrix" ScienceDirect. Accessed October 25, 2023. https://www.sciencedirect.com/topics/engineering/confusionmatrix.