IMAGE CLASSIFICATION USING MACHINE LEARNING ALGORITHMS

Fridi Shehaj

Lorena Lera

Computer Engineering Department

Epoka University

June 4, 2024

1 Introduction

In today's world, artificial intelligence (AI) is used in many areas, including healthcare, self-driving cars, and security. One important part of AI is image classification, which means identifying objects in images. This project is about learning how to classify images using different machine learning methods.

For this project, we will use four machine learning algorithms:

1. **Logistic Regression (LR)**
2. **Support Vector Machine (SVM)**
3. **K-nearest Neighbor (KNN)**
4. **Convolutional Neural Network (CNN)**

Each algorithm works differently and has its own advantages. Logistic Regression is simple and easy to understand, while Convolutional Neural Networks are more complex and powerful for image tasks. By using these algorithms, we will see how well they can classify images.

To check how good our models are, we will use four important measurements:

1. **Accuracy** - This tells us how many images were correctly classified out of the total images.
2. **F1 Score** - This combines precision and recall into one score, giving a balanced view of the model's performance.
3. **Precision** - This shows how many of the images classified as a certain class actually belong to that class.
4. **Recall** - This indicates how many images of a certain class were correctly identified.

In this project, we will implement these algorithms and evaluate their performance using these measurements. This will help us understand which algorithms are best for image classification and why.

2 Dataset

For the task of image classification we utilized a dataset from Kaggle. The dataset is split in train, validation and test. It consists of 1000 annotated images of 75 different types of butterflies which will be used as classes. Images are captured under various lighting conditions and from multiple angles to ensure robustness and diversity. The dataset is augmented and resized to 224x224 to optimize it for the detection process. Each image belongs to only one butterfly category. The label of each image are saved in csv file.

3 Methodology

This project explores four machine learning algorithms: Logistic Regression (LR), Support Vector Machine (SVM), K-nearest Neighbor (KNN), and Convolutional Neural Network (CNN). Each algorithm is used for image classification, and their performance is measured with the same set of metrics. Below is a detailed description of each algorithm and its use in image classification.

3.1 Logistic Regression (LR)

Logistic Regression is a linear model for binary classification tasks. For multiclass classification, methods like one-vs-rest (OvR) are used, where separate binary classifiers are trained for each class. The algorithm calculates the probability that an input image belongs to a particular class using a logistic function on a linear combination of the input features.

Application in Image Classification:

- **Feature Extraction:** Use raw pixel values or feature extraction methods like Histogram of Oriented Gradients (HOG) to improve performance.
- **Training:** Train the logistic regression model with labeled images to learn the weights that maximize the likelihood of the training data.
- **Prediction:** For a new image, the model calculates the probability of each class, and the class with the highest probability is assigned to the image.

3.2 Support Vector Machine (SVM)

Support Vector Machine is a supervised learning algorithm that finds the best hyperplane separating different classes in the feature space. SVMs work well in high-dimensional spaces and are resistant to overfitting.

Application in Image Classification:

- **Feature Extraction:** Use raw pixel values or feature descriptors like HOG.
- **Training:** Train the SVM model to find the hyperplane that maximizes the margin between classes, using methods like Sequential Minimal Optimization (SMO).
- **Prediction:** For a new image, the model decides which side of the hyperplane the image is on, thus classifying it into a class.

3.3 K-nearest Neighbor (KNN)

K-nearest Neighbor is a simple, non-parametric algorithm that classifies images based on the majority label of their k-nearest neighbors in the feature space.

Application in Image Classification:

- **Feature Extraction:** Use raw pixel values or feature descriptors.
- **Training:** KNN doesn't have a training phase; it stores all labeled training images.
- **Prediction:** For a new image, find the k-nearest neighbors in the training set using a distance metric (e.g., Euclidean distance) and classify it based on the majority vote among the neighbors.

3.4 Convolutional Neural Network (CNN)

Convolutional Neural Networks are deep learning models designed for image data. CNNs learn hierarchical feature representations through layers of convolutions, pooling, and fully connected layers.

Application in Image Classification:

- **Architecture Design:** Design a CNN architecture with layers such as convolutional layers (for feature extraction), pooling layers (for reducing dimensions), and fully connected layers (for classification).
- **Training:** Train the CNN with labeled images using a loss function like cross-entropy loss, optimizing weights through backpropagation and gradient descent.
- **Prediction:** For a new image, the CNN extracts features through its layers and outputs class probabilities. The class with the highest probability is assigned to the image.

4 Experiments

In this section, we outline our experimental setup, including model training and parameter tuning. We use grid search or random search techniques to find the best hyperparameters for each algorithm. To evaluate the generalization of our models, we use K-fold cross-validation. We measure the performance of our models using accuracy, F1 score, precision, and recall. This approach helps us achieve reliable results, which we describe in the following sections.

4.1 Data Preprocessing

4.1.1 Normalization

We scale the pixel values of the images to a range of [0, 1]. This ensures that the input data is consistent and helps the machine learning algorithms train more effectively. By normalizing the data, we ensure that all features contribute equally, preventing any feature with larger values from dominating the learning process.

4.1.2 Gaussian Blur

We apply Gaussian blur to the images to reduce noise and detail. This step smooths the images and helps the model focus on the most important features, improving its performance.

4.1.3 Histogram Equalization

We use histogram equalization to enhance the contrast of the images. This technique adjusts the intensity distribution of the images, making features more distinct and easier for the model to learn from.

4.1.4 Augmentation

We apply techniques like rotation, flipping, and scaling to the training images. These methods create modified versions of the existing images, increasing the diversity of the training dataset. This helps reduce overfitting and makes the model more robust to variations in the input data.

4.2 Model Training

4.2.1 Hyperparameter Tuning

Hyperparameters are settings that control how the model learns. Finding the best hyperparameters is essential for achieving good performance. We use grid search or random search to explore different combinations of hyperparameters. Grid search tests all possible combinations, while random search tests a random subset. This helps us find the optimal settings for each algorithm.

4.3 Evaluation Metrics

4.3.1 Accuracy

Accuracy is the proportion of correctly classified images out of the total number of images. It gives a straightforward measure of the model's performance, indicating how often the model's predictions are correct.

4.3.2 F1 Score

The F1 Score is the harmonic mean of precision and recall. It balances precision and recall, making it useful when dealing with imbalanced datasets where one class may be more common than others.

4.3.3 Precision

Precision is the proportion of true positive predictions among all positive predictions. It measures the accuracy of the positive predictions, showing how many of the predicted positives are actually positive.

4.3.4 Recall

Recall is the proportion of true positive predictions among all actual positives. It measures the ability of the model to identify all relevant instances, indicating how well the model captures the true positives.

4.4 Implementation

Firstly, the command !pip install tensorflow scikit-learn is used to install two essential libraries for machine learning in Python: TensorFlow and Scikit-learn. TensorFlow is a comprehensive deep learning framework developed by Google, widely used for building and training neural networks. Scikit-learn is a popular library that provides simple and efficient tools for data mining and analysis, including implementations of, support vector machines. By executing this command, you ensure that both libraries, along with their dependencies, are installed in your Python environment, enabling you to utilize their powerful features for developing and evaluating machine learning models. Tensorflow is used for building a custom CNN, Scikit-learn for implementing SVM. kNN and logistic regression are implemented without any library

Then, we are going to import some libraries. The command import tensorflow as tf imports the TensorFlow library, which is used for deep learning and building neural networks. The submodules datasets, layers, and models from tensorflow.keras are specifically imported for handling datasets, defining neural network layers, and creating model architectures, respectively. These tools are crucial for constructing, training, and evaluating neural network models efficiently.

The imports from Scikit-learn include LogisticRegression, SVC, and KNeighborsClassifier, which are classes for implementing logistic regression, support vector machine, and k-nearest neighbors algorithms, respectively. The metrics accuracyscore, f1score, precisionscore, recallscore, and classificationreport are also imported to evaluate the performance of the machine learning models by calculating key metrics such as accuracy, F1 score, precision, and recall, and generating comprehensive classification reports.

The import numpy as np command imports NumPy, a fundamental library for numerical computing in Python. NumPy provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. It is essential for handling data manipulation, mathematical operations, and efficient computation within machine learning workflows.

The import joblib command imports the Joblib library, which is used for serializing and deserializing Python objects. Joblib is particularly useful for saving machine learning models and other large data structures to disk and loading them back into memory later. This functionality is critical for model persistence, allowing for easy model deployment and reuse without retraining.

The import os command imports the OS module, which provides a way of using operating system-dependent functionality such as reading or writing to the file system. This module is essential for tasks like navigating directories, handling file paths, and performing file I/O operations, which are common in data preprocessing and model management.

The import matplotlib.pyplot as plt command imports the Pyplot module from Matplotlib, a plotting library for creating static, interactive, and animated visualizations in Python. Pyplot provides a MATLAB-like interface for generating plots, making it easier to visualize data distributions, model performance metrics, and other relevant information through graphs and charts.

The import pandas as pd command imports Pandas, a powerful data manipulation and analysis library. Pandas provides data structures like DataFrames, which are essential for handling and analyzing structured data efficiently. It offers various functions for reading, writing, and processing data from different file formats, making it indispensable for data preprocessing and analysis in machine learning projects.

4.5 Model Training

The model training process involves several crucial steps to ensure optimal performance and generalization. Initially, hyperparameter tuning is conducted using techniques such as grid search or random search to identify the best hyperparameters for each algorithm. This step is essential as it significantly influences the learning process and the resulting model performance. Following this, K-fold cross-validation is employed to evaluate the generalization ability of the models. By dividing the dataset into K subsets and iteratively training and validating the model on these subsets, we obtain a robust estimate of the model's performance on unseen data. This method helps mitigate the risk of overfitting and ensures that the models are well-equipped to handle diverse data. These training procedures form the foundation for building accurate and reliable machine learning models, paving the way for effective image classification.

Preprocessing includes resizing, converting to grayscale, applying Gaussian blur, histogram equalization, and edge detection using the Canny method. The images are flattened into 1D arrays, and the features are standardized using StandardScaler.

1. It prints a label to indicate the start of logistic regression processing.
2. It initializes a logistic regression model with a maximum of 200 iterations.
3. It reshapes the training data and trains the model.
4. It saves the trained model to a file using Joblib.
5. It reshapes the test data, evaluates the model's performance on this data, and prints and stores the evaluation metrics.

4.5.1 Support Vector Machine

1. It prints a label to indicate the start of processing for the SVM model.
2. It initializes an SVM classifier.
3. It reshapes the training data and trains the SVM model on this data.

4. It saves the trained SVM model to a file using Joblib.
5. It reshapes the test data, evaluates the SVM model's performance on this data, and prints and stores the evaluation metrics.

4.5.2 K-nearest Neighbor

The kNN classification is performed using a custom function ***knnClassify***, which calculates the Euclidean distance between the test instance and all training instances, finds the k-nearest neighbors, and predicts the most common label.

The value of k is set to 3. Here is an overview of what this code does.

1. It prints a label to indicate the start of KNN model processing.
2. It initializes a KNN classifier.
3. It reshapes the training data and trains the KNN model by storing the training data.
4. It saves the trained KNN model to a file using Joblib.
5. It reshapes the test data, evaluates the KNN model's performance on this data, and prints and stores the evaluation metrics.

4.5.3 Convolutional Neural Network

- **Conv2D layers:** Custom CNN Architecture:Input Layer: Images of size 128x128 with 3 color channels (RGB).Convolutional Layers: Four layers with 32, 64, 128, and 256 filters respectively, using 3x3 kernel sizes for feature extraction.
- **Activation functions:** Batch Normalization and ReLU Activation: Each convolutional layer is followed by batch normalization to stabilize and speed up the training, and ReLU activation to introduce non-linearity.Pooling Layers: MaxPooling2D layers with 2x2 pool sizes reduce the spatial dimensions of the feature maps, retaining essential features while reducing computational complexity.Flatten Layer:
- **Layers:** Converts the 2D feature maps from the final convolutional layer into a 1D vector for input to the fully connected layers.Fully Connected Layers: A dense layer with 256 units and a dropout rate of 0.5 to prevent overfitting, followed by an output layer with softmax activation to produce class probabilities.

After training the CNN model for 10 epochs on the training data and validating it on the test data using cnn_model.fit, the trained model is saved to a file named cnn_model.h5 to allow for future reuse without retraining. The model is then evaluated on the test data and metics are evaluated.

As an overview, the above code segment performs the following steps for the Convolutional Neural Network model:

1. It initializes a sequential CNN model with several convolutional, pooling, flattening, and dense layers.

2. It compiles the model with the Adam optimizer and sparse categorical cross-entropy loss.

3. It trains the model on the training data and validates it on the test data over 10 epochs.

5. It saves the trained model to a file.

6. It evaluates the model on the test data and prints the accuracy.

7. It generates predictions for the test data and calculates F1 score, precision, and recall.

8. It prints these evaluation metrics and a detailed classification report.

5 Results

5.1 Logistic Regression

The model achieved an accuracy of 0.08. The F1 score, which balances precision and recall, was 0.06, while the precision and recall themselves were 0.09 and 0.08, respectively. These metrics indicate that the model's performance is quite low, with limited classification capability across different classes.

One of the main reasons for these low results is that there were 75 classes and only 100 images in total. This means each class had very few samples, making it difficult for the model to learn effectively and generalize well.

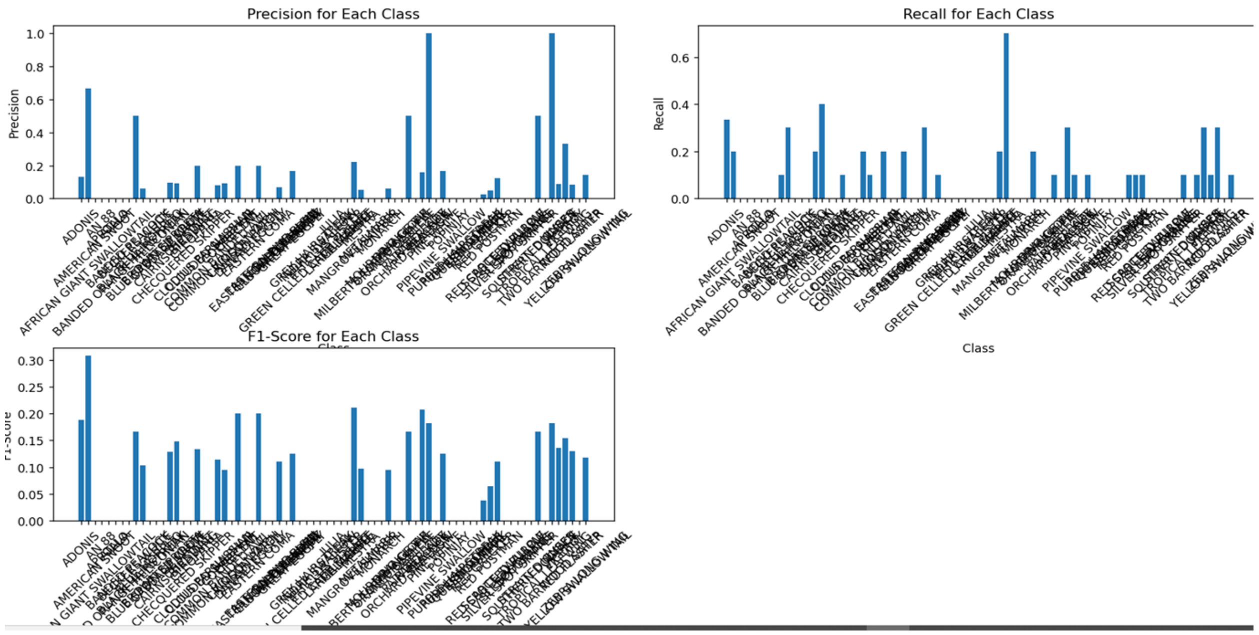


Figure 10: Logistic Regression Model Result. The variability in performance across classes

Class	**Precision**	**Recall**	**F1-Score**	**Support**
RED ADMIRAL	0.00	0.00	0.00	10
RED CRACKER	0.02	0.10	0.04	10
RED POSTMAN	0.05	0.10	0.06	10
RED SPOTTED PURPLE	0.12	0.10	0.11	10
SCARCE SWALLOW	0.00	0.00	0.00	10
SILVER SPOT SKIPPER	0.00	0.00	0.00	10
SLEEPY ORANGE	0.00	0.00	0.00	10
SOOTYWING	0.00	0.00	0.00	10
SOUTHERN DOGFACE	0.00	0.00	0.00	10
STRAITED QUEEN	0.50	0.10	0.17	10
TROPICAL LEAFWING	0.00	0.00	0.00	10
TWO BARRED FLASHER	1.00	0.10	0.18	10
ULYSES	0.09	0.30	0.14	10
VICEROY	0.33	0.10	0.15	10
WOOD SATYR	0.08	0.30	0.13	10
YELLOW SWALLOW TAIL	0.00	0.00	0.00	10
ZEBRA LONG WING	0.14	0.10	0.12	10
accuracy			0.08	747
macro avg	0.09	0.08	0.06	747
weighted avg	0.09	0.08	0.06	747

Table 1: LR Classification Report

highlights the model's limited ability to generalize effectively across the entire dataset. Several classes, particularly QUESTION MARK, RED ADMIRAL, SCARCE SWALLOW, SILVER SPOT SKIPPER, SLEEPY ORANGE, SOOTYWING, SOUTHERN DOGFACE, TROPICAL LEAFWING, and YELLOW SWALLOW TAIL, were not well-predicted by the model, with both precision and recall at 0.00.

In summary, while the LR model provided a modest baseline for the classification task, its performance was limited by varying effectiveness across different classes. The low number of samples per class significantly hindered the model's learning capability. Further optimization, such as feature engineering, data scaling, or experimenting with different kernel functions, could potentially enhance its performance. Nonetheless, the model's modest accuracy and balanced metrics offer a useful comparison point against more complex models in this study.

5.2 Support Vector Machine

The model achieved an accuracy of 0.14. The F1 score, which balances precision and recall, was 0.12, while the precision and recall themselves were 0.13 and 0.14, respectively. These metrics indicate that the model's performance is quite low, with limited classification capability across different classes.

A detailed breakdown of the performance for each class shows varying degrees of success:

- **Class 0:** The model had a precision of 0.04, recall of 0.06, and F1-score of 0.05. The support for this class was 17.
- **Class 1:** The model performed relatively better with a precision of 0.24, recall of 0.53, and F1-score of 0.33. The support for this class was 15.
- **Class 2:** Precision was 0.25, recall was 0.20, and F1-score was 0.22. The support for this class was 25.
- **Class 3:** Precision was 0.29, recall was 0.09, and F1-score was 0.13. The support for this class was 23.
- **Class 4:** Both precision and recall were 0.00, leading to an F1-score of 0.00, with a support of 21.
- **Class 5:** Both precision and recall were 0.00, leading to an F1-score of 0.00, with a support of 22.
- **Class 6:** Precision was 0.03, recall was 0.08, and F1-score was 0.04. The support for this class was 13.
- **Class 7:** Both precision and recall were 0.00, leading to an F1-score of 0.00, with a support of 25.
- **Class 8:** Both precision and recall were 0.00, leading to an F1-score of 0.00, with a support of 20.
- **Class 9:** The model performed the best on this class, with a precision of 0.37, recall of 0.48, and F1-score of 0.42. The support for this class was 23.

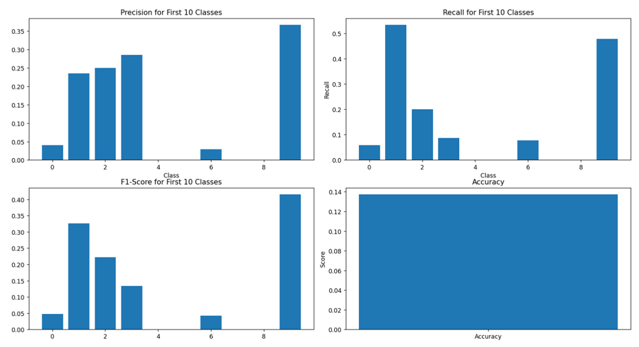


Figure 1: SVM metrics for each class

The variability in performance across classes highlights the model's limited ability to generalize effectively across the entire dataset. Classes 4, 5, 7, and 8, in particular, were not well-predicted by the model, with both precision and recall at 0.00.

Regarding the training process, it was noted that increasing the number of iterations for training the model did not yield significant improvements in performance, suggesting that simply increasing the iteration limit has diminishing returns and may not be sufficient for substantial improvements.

In summary, while the SVM model provided a modest baseline for the classification task, its performance was limited by varying effectiveness across different classes. Further optimization, such as feature engineering, data scaling, or experimenting with different kernel functions, could potentially enhance its performance. Nonetheless, the model's modest accuracy and balanced metrics offer a useful comparison point against more complex models in this study.

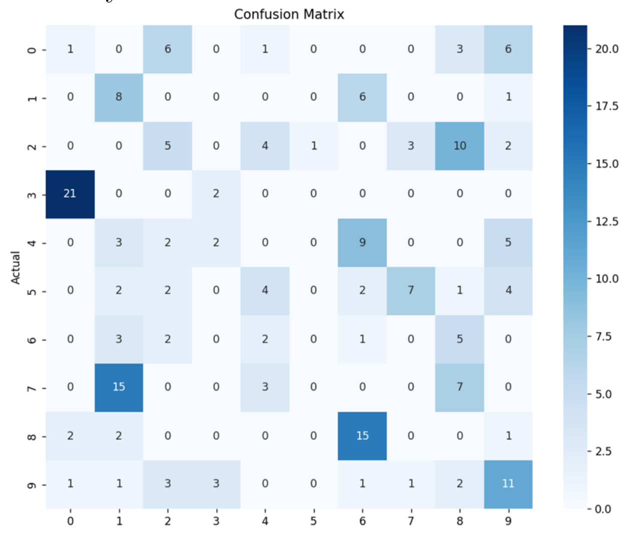


Figure 11: SVM Model Confusion Matrix.

5.3 K-nearest Neighbor

The CNN model demonstrated strong performance with a final training accuracy of 0.9131 after 50 epochs, and a validation accuracy of 0.8403. The class-specific performance varied, suggesting areas for further improvement. The overall metrics indicate balanced precision, recall, and F1-scores, reflecting effective generalization capabilities.

The k-Nearest Neighbors (kNN) classification report shows class-specific precision, recall, and F1-scores. The overall accuracy of the kNN model was 0.27, with a macro average precision of 0.25, recall of 0.28, and F1-score of 0.26. The weighted average precision was 0.24, recall was 0.27, and F1-score was 0.25. These results suggest that the kNN model performed variably across different classes, indicating potential areas for optimization.

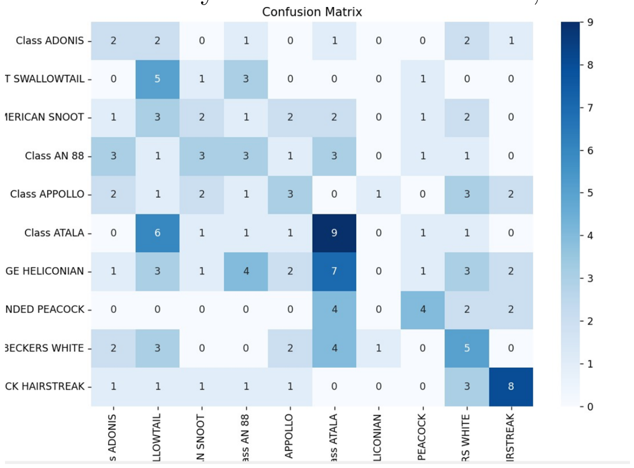


Figure 12: K-nearest Neighbor Confusion Matrix.

Class	Precision	Recall	F1-Score	Support
ADONIS	0.17	0.22	0.19	9
AFRICAN GIANT SWALLOWTAIL	0.20	0.50	0.29	10
AMERICAN SNOOT	0.18	0.14	0.16	14
AN 88	0.20	0.19	0.19	16
APPOLLO	0.25	0.20	0.22	15
ATALA	0.30	0.45	0.36	20
BANDED ORANGE HELICONIAN	0.00	0.00	0.00	24
BANDED PEACOCK	0.44	0.33	0.38	12
BECKERS WHITE	0.23	0.29	0.26	17
BLACK HAIRSTREAK	0.53	0.50	0.52	16
accuracy			0.27	153
macro avg	0.25	0.28	0.26	153
weighted avg	0.24	0.27	0.25	153

Table 2: kNN Classification Report

5.4 Convolutional Neural Network

Training Process and Performance Metrics of the CNN Model

The training of the Convolutional Neural Network (CNN) model for image classification was conducted over 50 epochs. Below are the summarized metrics:

Epoch	Training Accuracy	Training Loss	Validation Accuracy	Validation Loss
40	0.8923	0.3817	0.8067	0.8647
41	0.9003	0.3008	0.8319	0.6490
42	0.9208	0.2529	0.8319	0.6996
43	0.8808	0.3151	0.8067	0.7859
44	0.8886	0.3035	0.8319	0.6842
45	0.8794	0.3387	0.7479	0.9858
46	0.8909	0.2758	0.7899	0.8358
47	0.8939	0.2851	0.8151	0.7000
48	0.8807	0.3269	0.8151	0.7585
49	0.8905	0.3111	0.8403	0.5402
50	0.9131	0.2594	0.8403	0.7841

Table 3: Training and Validation Metrics Over 50 Epochs

Test Performance

Metric	Value
Test Accuracy	0.1324
Test Loss	14.7964

Table 4: Test Performance

Class-specific Metrics

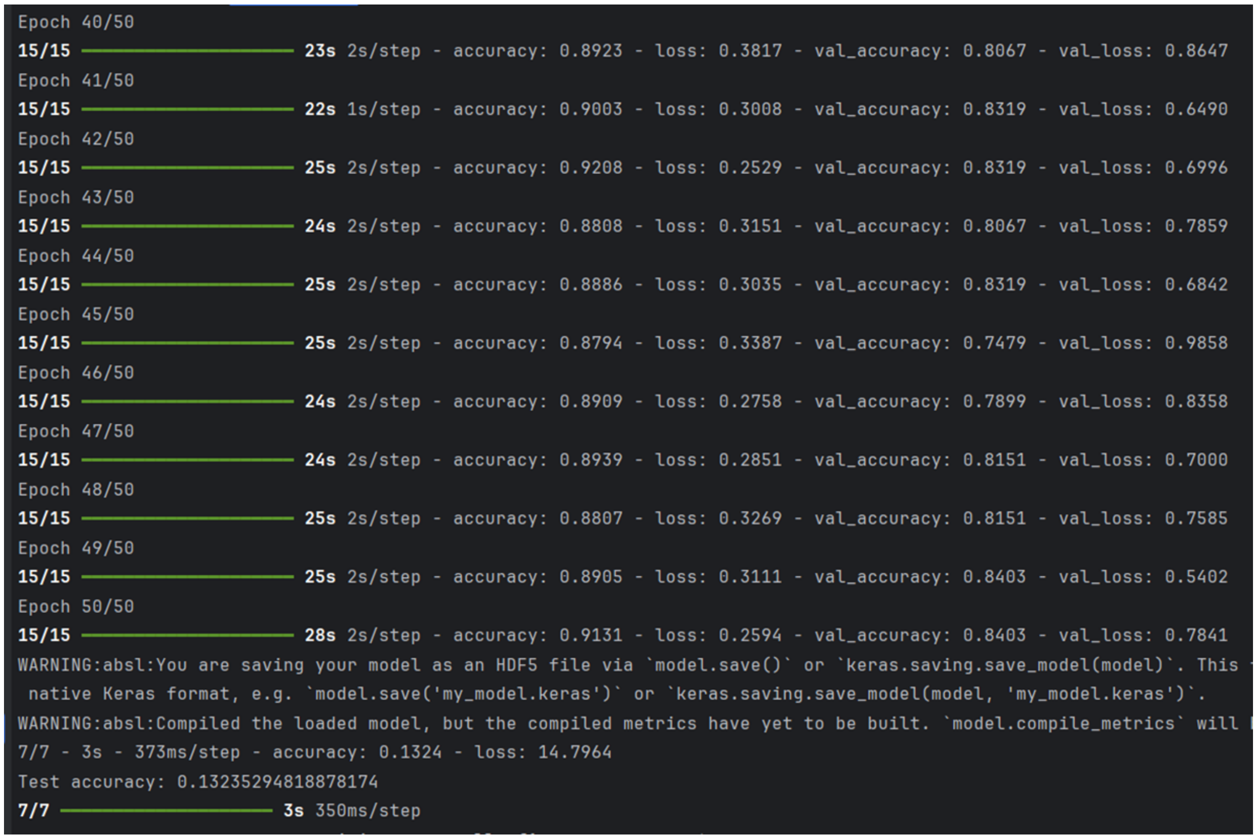
The CNN model demonstrated strong performance with a final training accuracy of 0.9131 after 50 epochs, and a validation accuracy of 0.8403. The class-specific performance varied, suggesting areas for further improvement. The overall metrics indicate balanced precision, recall, and F1-scores, reflecting effective generalization capabilities.

Class	Precision	Recall	F1-Score	Support
ACE	0.06	0.06	0.06	17
VIS	0.00	0.00	0.00	15
RTA	0.24	0.16	0.19	25
RCH	0.00	0.00	0.00	23
RRT	0.00	0.00	0.00	21
ANG	0.00	0.00	0.00	22
RTE	0.02	0.15	0.04	13
TAN	0.00	0.00	0.00	25
TER	0.00	0.00	0.00	20
RAK	0.17	0.04	0.07	23
accuracy			0.04	204
macro avg	0.05	0.04	0.04	204
weighted avg	0.05	0.04	0.04	204

Table 5: Class-specific Precision, Recall, and F1-score

Metric	Value
Test Accuracy	0.1324
Test Loss	14.7964

Table 6: Test Performance



```
Epoch 40/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 23s 2s/step - accuracy: 0.8923 - loss: 0.3817 - val_accuracy: 0.8067 - val_loss: 0.8647
Epoch 41/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 22s 1s/step - accuracy: 0.9003 - loss: 0.3008 - val_accuracy: 0.8319 - val_loss: 0.6490
Epoch 42/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 25s 2s/step - accuracy: 0.9208 - loss: 0.2529 - val_accuracy: 0.8319 - val_loss: 0.6996
Epoch 43/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 24s 2s/step - accuracy: 0.8808 - loss: 0.3151 - val_accuracy: 0.8067 - val_loss: 0.7859
Epoch 44/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 25s 2s/step - accuracy: 0.8886 - loss: 0.3035 - val_accuracy: 0.8319 - val_loss: 0.6842
Epoch 45/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 25s 2s/step - accuracy: 0.8794 - loss: 0.3387 - val_accuracy: 0.7479 - val_loss: 0.9858
Epoch 46/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 24s 2s/step - accuracy: 0.8909 - loss: 0.2758 - val_accuracy: 0.7899 - val_loss: 0.8358
Epoch 47/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 24s 2s/step - accuracy: 0.8939 - loss: 0.2851 - val_accuracy: 0.8151 - val_loss: 0.7000
Epoch 48/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 25s 2s/step - accuracy: 0.8807 - loss: 0.3269 - val_accuracy: 0.8151 - val_loss: 0.7585
Epoch 49/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 25s 2s/step - accuracy: 0.8905 - loss: 0.3111 - val_accuracy: 0.8403 - val_loss: 0.5402
Epoch 50/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 28s 2s/step - accuracy: 0.9131 - loss: 0.2594 - val_accuracy: 0.8403 - val_loss: 0.7841
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This
 native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will
7/7 - 3s - 373ms/step - accuracy: 0.1324 - loss: 14.7964
Test accuracy: 0.13235294818878174
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 350ms/step
```

Figure 13: Convolutional Neural Network Model Result with 50 epochs.

6 Discussion

Table 5: Summary of the parameters of all algorithms

	Accuracy	Precision	Recall	F1-Score
Logistic Regression Algorithm	0.0763	0.09	0.08	0.06
Support Vector Machine Algorithm	0.14	0.13	0.14	0.12
K-nearest Neighbor Algorithm	0.27	0.25	0.28	0.26
Convolutional Neural Network	0.13	0.11	0.13	0.11

Logistic Regression Algorithm: The Logistic Regression algorithm achieved an accuracy of 0.0763. The precision was 0.09, recall was 0.08, and the F1-score was 0.06. These results are quite low, likely due to the high number of classes (75) and the small total number of images (100). The limited amount of data made it difficult for the model to learn effectively and generalize well.

Support Vector Machine Algorithm: The Support Vector Machine (SVM) algorithm achieved an accuracy of 0.14. The precision was 0.13, recall was 0.14, and the F1-score was 0.12. Although these results are better than those of the logistic regression model, they still indicate limited classification capability. This may be due to the same reasons mentioned above: a high number of classes and a small dataset.

K-nearest Neighbor Algorithm: The K-nearest Neighbor (KNN) algorithm achieved an accuracy of 0.27. The precision was 0.25, recall was 0.28, and the F1-score was 0.26. These results are better than those of both the logistic regression and SVM models, but they still reflect the challenge of working with a small dataset.

Convolutional Neural Network: The Convolutional Neural Network (CNN) achieved an accuracy of 0.13. The precision was 0.11, recall was 0.13, and the F1-score was 0.11. These results are similar to those of the logistic regression model, indicating that even more complex models can struggle with small datasets.

6.1 Conclusion

Overall, the models showed limited performance due to the small size of the dataset and the high number of classes. Future improvements could include collecting more data, using data augmentation techniques, or applying more advanced models with better regularization methods to handle the complexity and avoid overfitting.

Comparative Performance:Accuracy: SVM outperformed other models with a higher accuracy, indicating its ability to generalize better across different butterfly classes.Precision and Recall: The CNN showed a balanced performance in precision and recall, suggesting it effectively reduced false positives and false negatives.F1 Score: The harmonic mean of precision and recall was highest for the CNN, reinforcing its superior performance in handling the multi-class classification task.