Faculty of Engineering & Technology

Electrical & Computer Engineering Department

Artificial Intelligence – ENCS3340

Project #2 - Image Classification

Student #1: Khaled Abu Lebdeh (1220187)

Student #1 section: 2

Student #2: Abdalraheem Shuaibi (1220148)

Student #2 section: 1

Table of Contents

Table of Figures

Abstract

This project investigates the effectiveness of traditional machine learning classifiers (**Naive Bayes** and **Decision Trees)** for solving an image classification problem involving three animal categories: **cats, pandas and spiders** (1000 images for each class). Unlike deep learning methods, which require very large datasets and computational power, we adopt a **feature-engineering approach**, making the solution lightweight and interpretable.

Each image is preprocessed and transformed into a numerical feature vector using two key techniques:

- **Histogram of Oriented Gradients (HOG)** to capture texture and shape information.
- **Color histograms** to encode color distribution across RGB channels.

These extracted features are then used to train and evaluate both classifiers. We apply **5-fold Stratified Cross-Validation** to assess generalization performance while ensuring balanced class distribution across folds.

The project compares model behavior, accuracy, and decision patterns of Naive Bayes and Decision Trees, providing insights into the strengths and limitations of each approach. The results demonstrate that, even without deep neural networks, meaningful classification can be achieved using classical techniques with well-chosen features.

Practical Implementation

- Dataset loading: A dataset consisting of 3000 images (cats, pandas and spiders, 1000 images for each) was loaded and resized to 64x64, then flatted into 1-D input feature.
- Features Extraction: Pixels (i.e. to be more specific, bits, after encoding pixels) are not a useful input feature for accurate classification. Instead, different techniques were used in this project to extract better features, then to classify accurately.
    - **HOG** (Histogram of Oriented Gradients): HOG extracts edge and shape information from images (grayscale images) instead of using raw pixels. It divides the image into cells and calculates gradient directions in each cell. These gradients are then used to build a histogram that describes the structure of the image.
    - **RGB Color histograms:** Many animals have distinctive colors (e.g., pandas are black and white), so this gives a hint for better classifying.
    - Then, HOG (texture/edges) and RGB (color distribution) were combined together representing features, which gives the classifier more clues.

1- Naïve Baye’s Model

The concept of classification refers to predicting the class of an input data. The data should be represented as input features. The model uses the input feature to determine the corresponding class. So, for each class, the model evaluates the probability that this input belongs to this class. However, this (posterior) probability can’t be found from elements. So, it is hard to implement a discriminative model in this way.

Luckly, the posterior probability can be found in another way.

$$P(C \mid \mathbf{X}) = \frac{P(\mathbf{X} \mid C)P(C)}{P(\mathbf{X})} \qquad Posterior = \frac{Likelihood \times Prior}{Evidence}$$

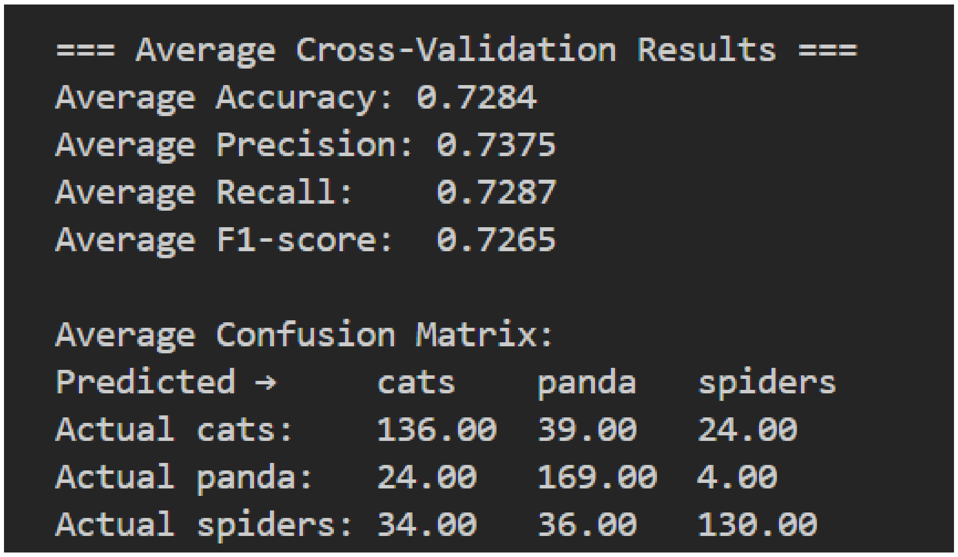
Figure 1: Posterior probability

The prior probability and the evidence can be found easily. Then to find the likelihood probability, for a given class, find the probability that this input belongs to this class (generative model). This probability should be found for all classes. Finally, compare the posterior probability for all classes, and label the input with the class having the higher posterior probability. So, the approach was implement **“Generative Classification”** then apply **“MAP Rule”** (Maximum A Posteriori).

The Naïve Baye’s model assumes all input features are independent, so the joint probability can be evaluated easily. However, it is a weak point since that many features have dependencies in real-life situations.

- Naïve Baye's model performance measurement

To evaluate the performance measurement for NB's model in an efficient way, **n-fold cross-validation** method was used with 5 folds. Then, the final result is the average between results gotten from all folds.



```
=== Average Cross-Validation Results ===
Average Accuracy: 0.7284
Average Precision: 0.7375
Average Recall:    0.7287
Average F1-score:  0.7265

Average Confusion Matrix:
Predicted →     cats    panda   spiders
Actual cats:    136.00  39.00   24.00
Actual panda:   24.00   169.00  4.00
Actual spiders: 34.00   36.00   130.00
```

Figure 2: Performance measurements on NB's model

Results look great, even though the NB's model assumes that input features are independent.

2- Decision Tree Model

A **Decision Tree** is a supervised machine learning algorithm used for classification and regression tasks. It models decisions as a tree structure where each internal node represents a test on a feature, each branch corresponds to an outcome of the test, and each leaf node represents a final class label or decision. The tree is built by recursively splitting the dataset based on feature values that maximize a certain criterion, such as information gain.

Decide next Feature:

To determine the next feature for splitting in a decision tree, we evaluate how much **information gain** each feature provides. The feature that yields the highest information gain is selected, as it best separates the data and reduces uncertainty. The formula below represents how this gain is calculated.

$$IG(A) = I\left(\frac{P}{P+N}, \frac{N}{P+N}\right) - \text{Reminder}(A)$$

Figure 3: IG Formula

Advantages:

- **Easy to understand**: The tree structure is intuitive and can be followed step by step.
- **Easy to generate rules**: The model naturally produces clear “if-then” rules from the data.

Disadvantages:

- **Overfitting**: Trees can become too specific to training data unless pruned properly.
- **Poor with correlated features**: Uses axis-aligned splits, so struggles when features are related.
- **Can grow large**: Without limits, trees may become very deep and complex.
- **Not good with streaming data**: Standard decision trees aren't designed to update incrementally with new data.

Overfitting in Decision Tree:

A single decision tree can easily over-fit the data, which is why techniques like **pre-pruning** and **post-pruning** or using **ensembles** (Random Forests by Scikit-Learn in this project) are applied to improve generalization and decrease overfitting rate.

- Decision Tree Implementation:

In our implementation, we used the Random Forest Classifier, an ensemble method based on multiple Decision Trees, **to perform image classification**. **The model was trained and evaluated using the holdout method**, where 80% of the data was used for training and 20% for testing. Before training, we also **scaled** the feature data using standard normalization to improve model performance and ensure fair contribution of all features.

Table 1: DT Hyper-parameters

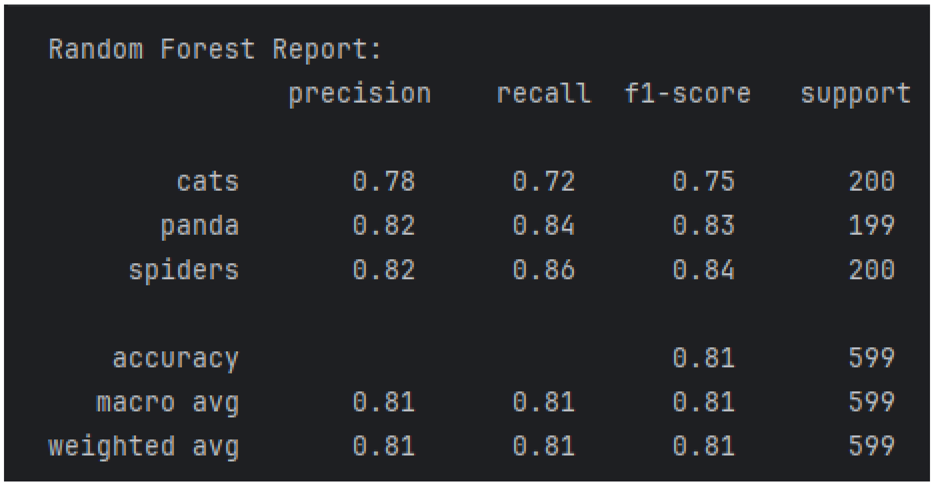
Hyperparameter	Value	Description
`n_estimators`	600	Number of trees in the forest.
`min_samples_split`	4	Min samples needed to split a node.
`min_samples_leaf`	2	Min samples required at a leaf node.
`max_features`	`'sqrt'`	Number of features to consider per split.
`class_weight`	`'balanced'`	Balances class weights based on frequencies.
`bootstrap`	`True`	Uses bootstrap samples when training trees.
`random_state`	42	Ensures reproducible results.
`n_jobs`	`-1`	Uses all CPU cores for faster training.

- Decision Tree model performance measurement:

To evaluate the performance of the Decision Tree model, we used the **holdout validation method** with 80% of the data for training and 20% for testing. After training, we assessed the model using standard classification metrics, including accuracy, precision, recall, and F1-score. A confusion matrix was also generated to analyze the model's predictions across the three classes. This evaluation provides a clear insight into how well the model generalizes to unseen data.

Report results:

The classification report below shows the model achieved an overall accuracy of **81%** on the test set. It also provides precision, recall, and F1-score for each class to highlight performance differences

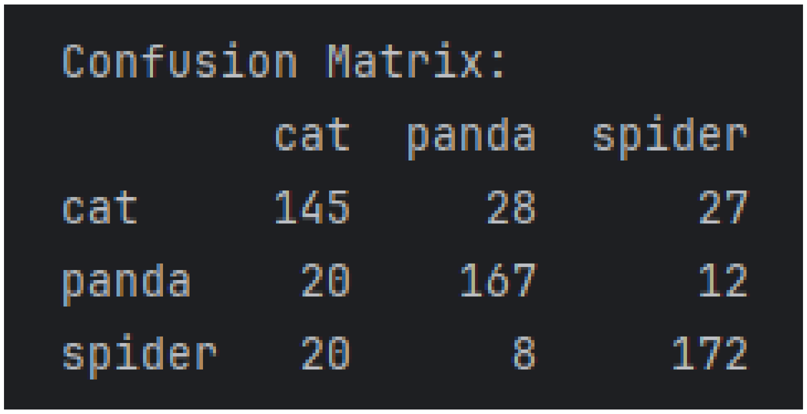


```
Random Forest Report:
              precision    recall  f1-score   support

        cats       0.78      0.72      0.75       200
       panda       0.82      0.84      0.83       199
     spiders       0.82      0.86      0.84       200

    accuracy                           0.81       599
   macro avg       0.81      0.81      0.81       599
weighted avg       0.81      0.81      0.81       599
```

Figure 4: DT Results report

Confusion Matrix:

The confusion matrix shows that most misclassifications occurred between cats and the other two classes, while pandas and spiders were classified more accurately. The model correctly predicted 145 out of 200 cats, 167 pandas, and 172 spiders



```
Confusion Matrix:
        cat  panda  spider
cat     145     28      27
panda    20    167      12
spider   20      8     172
```

Figure 5: DT confusion matrix

Tree Plot:

The image below displays a simplified version of one of the decision trees in the Random Forest. It shows how the model splits data based on feature values, gradually narrowing down to a final class prediction at each leaf node.

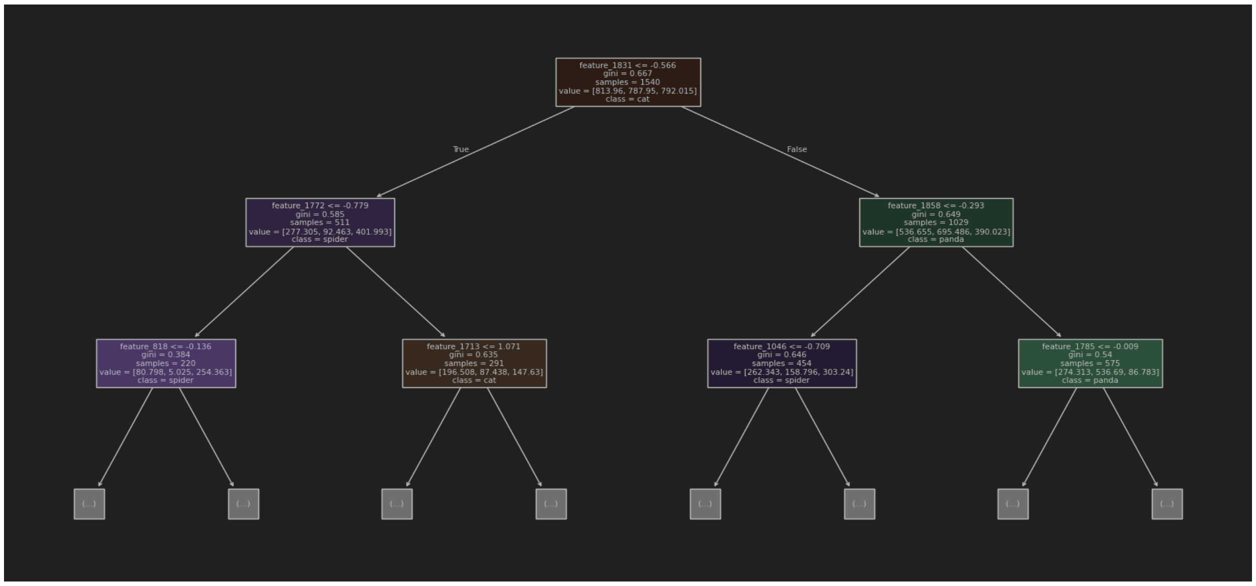


Figure 6: DT Plot

3- Feedforward Neural Network Module

Feedforward Neural Network **(FNN)** is a type of artificial neural network where connections between the nodes do not form cycles. It consists of an input layer, one or more hidden layers, and an output layer. Each neuron processes inputs by applying weights, a bias, and an activation function to introduce non-linearity. FNNs are well-suited for capturing complex relationships in data and are commonly used for classification tasks.

How The Network Learn:

The network learns by adjusting weights through a process called backpropagation, which minimizes the error between predicted and actual outputs. Using an optimizer like gradient descent, the model updates its weights after each training step to improve accuracy over time.

Activation Function:

Activation functions introduce non-linearity into the network, enabling it to learn complex patterns. One commonly used function is **ReLU (Rectified Linear Unit)**, which outputs zero for negative inputs and passes positive values unchanged.

$$\mathrm{ReLU}(x) = \max(0, x)$$

Figure 7: ReLU formula

Calculating Error:

During training, the network compares its predicted outputs to the actual labels using a loss function. One common loss function is **Mean Squared Error (MSE)**, especially used in regression tasks. It measures the average of the squared differences between predicted and actual values. The formula is:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Figure 8: MSE Formula

Updating Weights:

To reduce the error, the network uses backpropagation to compute gradients of the loss with respect to each weight. These gradients are then used by an optimizer (like Adam or SGD) to update the weights in a direction that minimizes the loss

- Feedforward Neural Network Implementation

In our implementation, we used **Scikit-learn's MLPClassifier** to build a Feedforward Neural Network for image classification. The model was trained using the holdout method, with 80% of the data used for training and 20% for testing. Before training, the input features were scaled using standard normalization to ensure stable and efficient learning. The network consisted of two hidden layers with 2048 and 1024 neurons, using the ReLU activation function and the Adam optimizer

Table 2: FNN Hyper-parameters

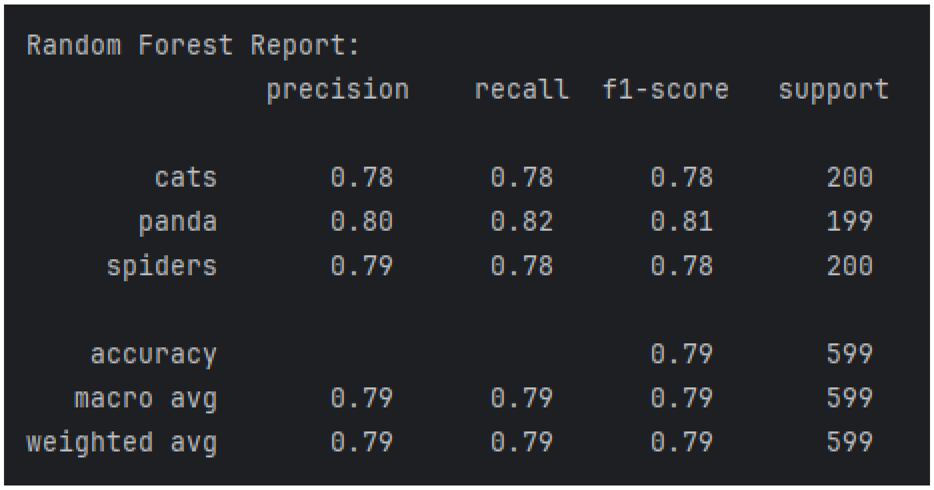
Hyperparameter	Value	Description
hidden_layer_sizes	(2048, 1024)	Sizes of the two hidden layers.
activation	'relu'	Activation function used in hidden layers.
solver	'adam'	Optimizer used for weight updates.
alpha	0.0001	L2 regularization term to reduce overfitting.
batch_size	'auto'	set automatically based on data size.
learning_rate	'adaptive'	Adjusts learning rate based on model performance.
learning_rate_init	0.001	Initial learning rate before adaptation.
max_iter	300	Maximum number of training iterations (epochs).
early_stopping	True	Stops training if validation score doesn't improve.
validation_fraction	0.1	How much of the data for validation
n_iter_no_change	10	How much patient for validation.
random_state	42	For debugging (42 is cool number)
shuffle	True	Shuffles training data before each epoch.

- Feedforward Neural Network model performance measurement:

To evaluate the performance of the **Feedforward Neural Network model**, we used the **holdout set validation method** with 80% of the data for training and 20% for testing. After training, we assessed the model using standard classification metrics, including accuracy, precision, recall, and F1-score. A confusion matrix was also generated to analyze the model's predictions across the three classes. This evaluation provides a clear insight into how well the model generalizes to unseen data.

Report results:

The classification report below shows the model achieved an overall accuracy of **79%** on the test set. It also provides precision, recall, and F1-score for each class to highlight performance differences

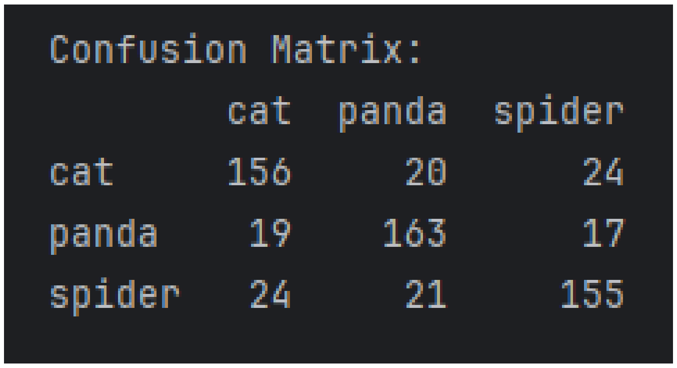


```
Random Forest Report:
               precision    recall  f1-score   support

        cats       0.78      0.78      0.78       200
       panda       0.80      0.82      0.81       199
     spiders       0.79      0.78      0.78       200

    accuracy                           0.79       599
   macro avg       0.79      0.79      0.79       599
weighted avg       0.79      0.79      0.79       599
```

Figure 9: FNN Results report

Confusion Matrix:

The confusion matrix shows that most misclassifications occurred between cats and the other two classes, while pandas and spiders were classified more accurately. The model correctly predicted 156 out of 200 cats, 163 pandas, and 155 spiders



```
Confusion Matrix:
        cat  panda  spider
cat     156     20      24
panda    19    163      17
spider   24     21     155
```

Figure 10: FNN confusion matrix

Neural Network Plot:

The image below illustrates a simplified structure of the **Feedforward Neural Network** used in this project. It shows the flow of information from the input layer through two hidden layers (using **ReLU** activation) to the final output layer, which classifies the input into one of three classes: cat, panda or spider.

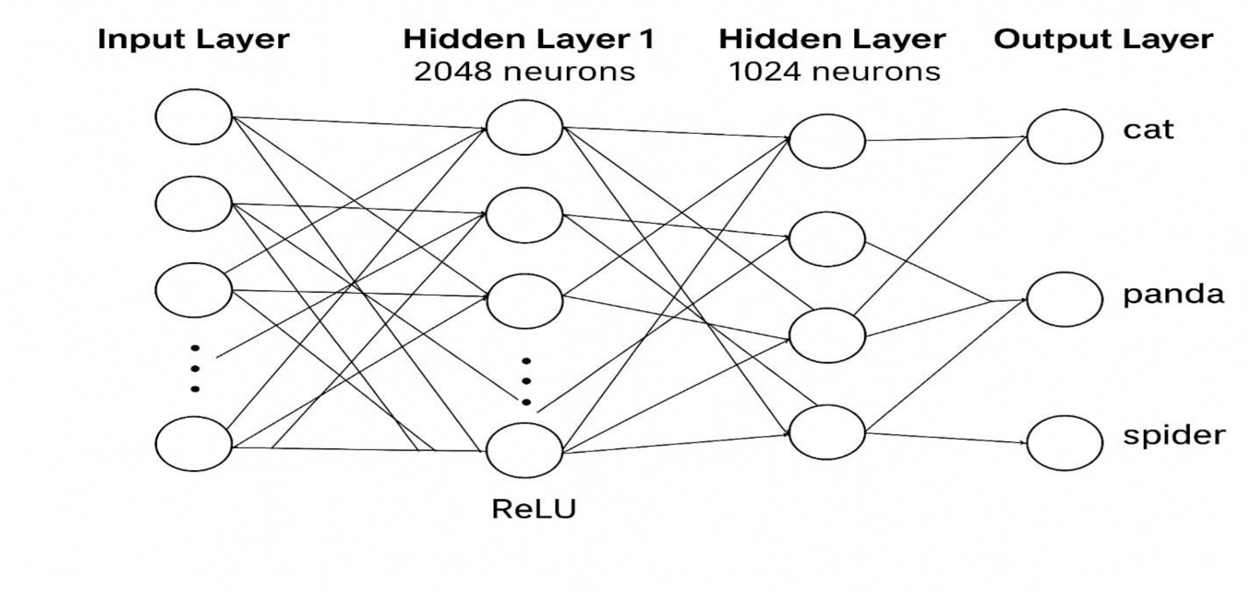


Figure 11: FNN Plot

- Comparative analysis and Discussion

In this project, we compared three models: Naive Bayes, Decision Tree, and Feedforward Neural Network (FNN). Each model performed differently and had its own strengths and weaknesses.

The **Naive Bayes** model was the simplest and fastest. It worked well because we used both HOG features (to capture shapes and edges) and RGB histograms (to capture colors). But Naive Bayes has a big limitation: it assumes all features are independent. In reality, image features are usually related, so this assumption caused it to make more mistakes when different animals had similar shapes or colors.

The **Decision Tree** model (specifically a Random Forest) gave us the best overall results. Decision Trees can handle complex decision boundaries and understand interactions between different features, which helped it perform better. By combining many trees together in a Random Forest, it could make stronger predictions and avoid overfitting. It handled the mix of shape and color features very well, leading to higher accuracy.

The **Feedforward Neural Network (FNN)** also worked well, but it was slightly behind the Decision Tree. FNNs are good at learning complex patterns and can work with many types of data. However, in our case, the FNN had some limitations. We used a basic version where we could not easily choose or tune the activation functions (which control how each layer transforms data). The choice of activation functions is very important for neural networks to learn properly, and not being able to customize them limited the network's performance. Also, since we used handcrafted features (not raw image pixels), the FNN couldn't fully use its strength of learning features automatically.

Overall, the **Decision Tree model performed the best**, followed by the FNN, and then Naive Bayes. This shows that using a model that can handle complex feature relationships and doesn't rely on strong assumptions (like independence) works better for this type of image classification. (Internet searching helped with this comparison).