OULUN AMMATTIKORKEAKOULU

Yingyu Mattila

MACHINE LEARNING METHODS IN IMAGE RECOGNITION

MACHINE LEARNING METHODS IN IMAGE RECOGNITION

Yingyu Mattila
Bachelor's Thesis
Spring 2024
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Web Development

Author: Yingyu Mattila
Title of the thesis: Machine Learning Methods in Image Recognition
Supervisors: Ilpo Virtanen
Term and year of thesis completion: Spring 2024 Pages: 52

In recent years, computer vision technology has played an important role in the field of wildlife monitoring. Machine learning technique can help to detect animals in the images collected in the field, so as to better understand the animal behaviour and the protection of key conserved animals. Machine learning algorithms can be trained iteratively on a large amount of image data to capture the features associated with the image labels, thus improving the recognition rate. The aim of this study is to classify animal images (including giant pandas, bears, lions and tigers) in the field background using machine learning methods.

This study is about evaluating the performance of various algorithms on animal image classification tasks. The algorithms that will be evaluated include the k-nearest neighbours algorithm, the decision tree algorithm, the random forest algorithm, the gradient boosting decision tree algorithm, and the support vector machine algorithm. The evaluation process involves clear model evaluation criteria as well as comparative analyses of model performance. For model evaluation, we used specifications such as accuracy, average precision, and other measures to comprehensively evaluate the performance of the models.

The results of the study demonstrate that the random forest and gradient boosting decision tree algorithms achieved high accuracy and precision in our tasks. However, the random forest algorithm demonstrated faster training and prediction speeds, making it more suitable for practical applications requiring the rapid processing of large datasets. This discovery offers an important reference for the future selection of machine learning models suitable for specific application scenarios.

Keywords: Machine Learning, Image Recognition, Animal Classification, Algorithmic Comparison

CONTENTS

VOCABULARY

K-NN	K-nearest neighbours
DT	Decision tree
GBDT	Gradient boosted decision tree
SVM	Support vector machine
PCA	Principal Component Analysis

1 INTRODUCTION

In recent years, computer vision has been rapidly developed, and accurate recognition and classification techniques for images have become increasingly important. With the advancement of computing technology, especially the significant increase in computing power, image recognition technology based on machine learning has begun to be widely used in our daily life. For example, in the field of autonomous driving, it ensures driving safety by accurately identifying pedestrians and vehicles on the road (Bachute & Subhedar, 2020); in medical image analysis, machine learning helps doctors diagnose diseases faster and more accurately, such as detecting early lung cancer by analysing X-ray images (Thallam et al., 2020). These applications not only enhance the practicality of the technology, but also greatly improve work efficiency and quality of life.

In this thesis, an insight into how machine learning algorithms can be used to identify and classify wild animals, specifically bears, lions, pandas and tigers, combined with the raw pixel feature vectors of the images. In the study, machine learning algorithms are used to train models by analysing the pixel-level data of these animals in an image so that they can accurately differentiate between the different species.

The primary motivation for this research is to apply machine learning techniques to wildlife identification with the aim of promoting wildlife conservation and biodiversity research. By using machine learning algorithms to identify different animal species, researchers can monitor and analyse animal behaviour more effectively, leading to more targeted conservation measures. Furthermore, the application of this technology can be used across a number of subject areas, particularly in bioinformatics, where it offers a way to provide technical assistance (Alharbi et al., 2019).

The main contribution of this thesis is to deepen our understanding of image recognition functions in the context of machine learning. It not only compares the performance of different models, but also provides an in-depth analysis of the practical applications of these models in real-life scenarios. Through such analysis, it helps us to recognise the challenges and limitations that we may encounter when deploying these models in real-world environments.

The structure of the thesis is as follows: after an introduction in Section 1, a literature review in Section 2 provides a comprehensive background to the classification problem and situates this

study within the broader context of machine learning in image recognition. In Section 3, we address the data preparation, model evaluation, and the details of how we applied the algorithm to our dataset. The results are displayed and analysed in Section 4. Finally, in the discussion parts, we compare our results with other studies horizontally and discuss the possibilities for improvement then summarise the results in the conclusions section.

2 LITERATURE

2.1 The overview of classification problems

Machine learning is an important part of artificial intelligence which utilises a series of computational technologies to enable a system to learn from data and make predictions or decisions. Supervised learning is also an important component of machine learning, with the idea that algorithms learn from labelled training data to predict results or classify the data.

The classification problem is a common task in supervised learning. The goal is to classify input data into predefined categories or labels. By training the samples in the dataset with their corresponding labels, the classification model is able to learn how to correctly classify new unlabelled data into the appropriate categories.

Classification tasks are widely used in real life, especially in the field of image recognition. According to Khan et al. (2021), it is one of the foundations in computer vision tasks and is widely used to identify and differentiate between different objects, scenes or features of images. The purpose is to enable computers to recognise and understand content from images. For instance, in the field of face recognition, classification tasks can be applied to security systems (Bagaa et al., 2020), identity verification (Srinivasan et al., 2005), and social media applications (Pennacchiotti & Popescu, 2021) by classifying face images into specific individuals. In medical image recognition, classification tasks are mainly used to automatically identify diseases, organs or abnormalities in images to assist doctors in making diagnoses and treatment plans (Olsen et al., 2020). And in e-commerce, classification tasks are used to categorise product images into different categories so that users can find their desired products more easily, thus improving the user experience (Xu et al., 2024).

2.2 Fundamentals of Machine Learning in Image Recognition

In the task of image recognition, algorithms for machine learning need to learn from a standard dataset, and they construct predictive models by finding relationships between data features and labels. It means that the algorithm needs to extract meaningful features from the raw pixel values and then use these features for a step of analysis and classification.

In addition to introducing how to convert images into data, several classical machine learning algorithms will also covered in detail in this chapter and explore their application to classification tasks, in particular how to train models to extract patterns and features from image data.

2.2.1 Image Feature Vectors in Image Recognition

The recognition process of an image usually involves the extraction of feature vectors in order to represent the key features of the image in a better way. Feature vectors are image abstractions used to characterise and numerically quantify the content of an image. In simple terms, feature vectors are lists of numbers used to represent an image, and these lists usually consist of real numbers, integers, or binary values.

In this thesis, the focus is on the classification task by extracting the most basic raw pixel image vectors in an image. In other words, it is the colour intensity values of each pixel in the image file that concerns us. In a colour image, the intensity of each pixel consists of a combination of red, green and blue values, which are stored digitally in the computer. Specifically, each colour channel usually has a value between 0 and 255, with 0 indicating the lowest intensity and 255 indicating the highest intensity (Yeung, 2015).

When the image is converted into data with the intensity of the colour channels of the pixels, we get a 3-dimensional array with dimensions of $m \times n \times 3$ which denoted as $A = (a_{ijk})$ with integers $i \in [1, m], j \in [1, n], k \in [1,3]$. Note that the pixel resolution is $m \times n$ and the blue, red, and green channels correspond to (a_{ij1}), (a_{ij2}) and (a_{ij3}), respectively. For the next step of data processing and analysis, the array converted from images needs to be further transformed into a 1-dimentional array with $3mn$ elements denoted as $B = (b_l)$ where l is also integer with $l \in [1,3mn]$. Two arrays comprise a one-to-one mapping

$$a_{ijk} \leftrightarrow b_l,$$

which satisfy the equation $3(i-1)n + 3(j-1) + k = l$. Given i, j, k, we can easily determine l by the above equation; and in turn, i, j, k can be determined by successively calculating the residue modulo 3 and n if $j < n$.

2.2.2 K-nearest neighbours

K-nearest neighbour (k-NN) is a lazy learning approach, which means that when building a model, it does not extract general features from the training data, but stores the entire training dataset in memory, waits for prediction to compute the distance between the test samples and the training samples by using the Euclidean distance and performs the classification task based on the labels of the k closest training samples. One of the notable k-NN's for image recognition advantage is its simplicity and intuitiveness (Cover & Hart, 1967).

In the k-NN algorithm, choosing an appropriate value of k is crucial because it directly affects the accuracy of classification. Choosing an overly large k may result in an overly simple model with underfitting problems, while choosing an underly small k may make the model too complex with the risk of overfitting. Usually, the best k value can be determined by means of cross-validation (Mullin & Sukthanka, 1999).

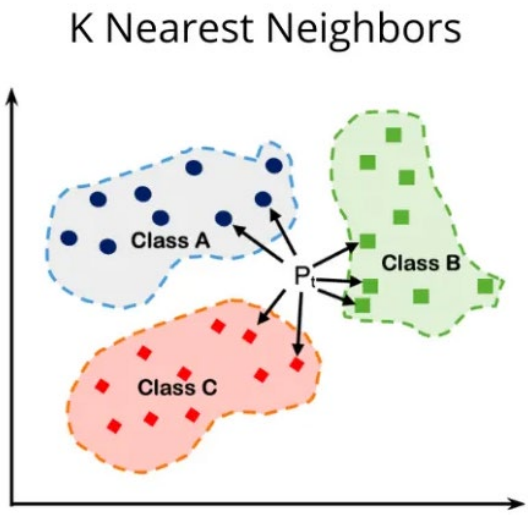

FIGURE 1. An example of the mechanism of k-NN algorithm (Sachinsoni, 2023). A new point P_t will be classified into one of the three classes A, B and C according to the distance.

2.2.3 Decision tree

Decision Tree is a supervised learning algorithm whose process is similar to a flow diagram and is very easy to understand and explain. By decomposing the data set into different decision nodes

and branches, the decision tree is able to make decisions step by step and generate a series of simple rules to explain the relationship between the data.

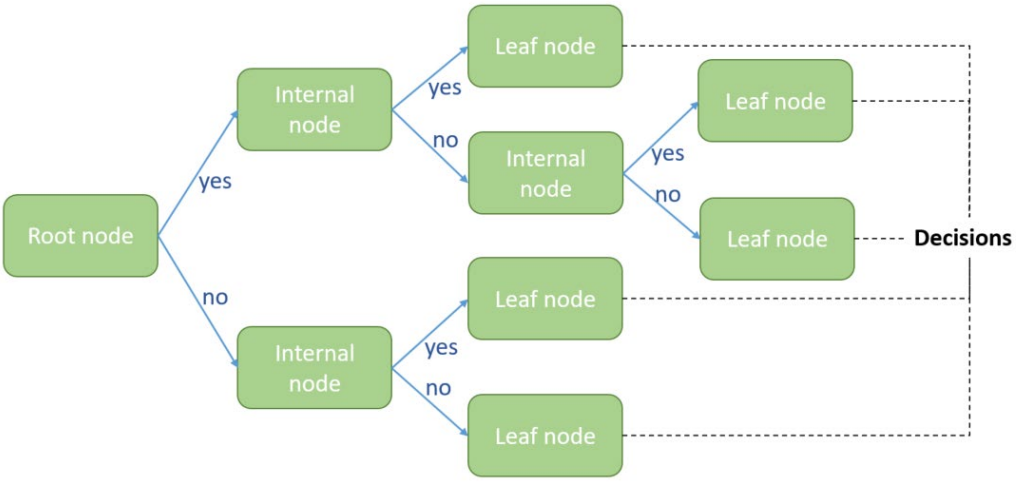

FIGURE 2. The structure of a decision tree. At root nodes and internal node, a condition is selected to split the sub data set of this node.

In Decision Trees, it is crucial to choose the right tree depth as it directly affects the performance and generalisation ability of the model. The depth of the tree determines how much detail the model learns about the data. If the tree depth is too shallow, it may result in underfitting as the model is unable to fully learn the patterns in the data. And a decision tree that is too deep may cause the model to overfit the training data, also making the model complex and difficult to understand (Johansson, 2021).

In practice, grid search is often used to optimise the depth of a decision tree. In this way, the effects of different depths can be systematically explored to determine the optimal depth also discussed in Johansson (2021).

2.2.4 Random forest

Random Forest is an integrated learning algorithm whose core idea is to combine the predictions of multiple decision trees as a way to enhance the accuracy of the overall model on classification tasks. Specifically, the Random Forest algorithm handles the feature vectors of an image by constructing multiple decision trees and training them independently. These decision trees then determine the final classification result by majority voting, i.e., each tree gives a prediction, and the final

classification result is determined by the prediction with the most occurrences. This majority voting mechanism effectively improves the accuracy and robustness of the predictions, as it reduces the biases and errors that may be introduced by relying on a single model.

In the Random Forest algorithm, each decision tree is trained independently, and two random methods are used: one is using self-sampling in the selection of the training set, and the other is randomly selecting a subset of features during the node splitting process of the decision tree. This double randomness allows each tree in the forest to capture different aspects of the data, thus enhancing the robustness of the overall model. Since image data often has a high-dimensional feature space, random forests can effectively handle high-dimensional data in this way (Qi, 2012).

Although Random Forest is computationally more complex than a single decision tree, its remarkable performance makes it a very valuable algorithm in the field of image recognition in recent works (Subasi & Subasi, 2016; Zhu et al., 2017)

2.2.5 Gradient boosted decision tree

Gradient Boosting Decision Tree (GBDT) is an integrated learning algorithm which is similar to Random Forest in that it also builds powerful predictive models based on multiple decision trees. However, unlike Random Forest that integrate multiple independently trained trees through a simple average or majority voting mechanism, GBDT corrects the prediction error of the previous tree by sequentially adding trees, with each new tree built to improve the overall model performance (Hastie et al., 2009). There is evidence that GBDT performs better than in random forests (Caruana & Niculescu-Mizil, 2006).

In the scenario of gradient boosting algorithm for multi-class classification problems, we need to transform the problem into probabilistic prediction as the tree model predicts continuous values. In our problem, the samples come from K categories, so the gradient boosting algorithm need to be extended from binary classification by building K trees at each iteration stage, thus giving the probability that each sample belongs to the k-th category.

2.2.6 Support Vector Machine

Support Vector Machine (SVM) is powerful supervised learning algorithm that perform classification tasks by separating samples with different labels by drawing hyperplanes in an N-dimensional feature space. SVMs are able to perform classification efficiently in complex datasets and perform well especially when the feature dimensionality is high (Howley et al., 2007).

SVM is particularly effective in image recognition tasks because of its ability to handle high-dimensional data and accurately classify complex patterns in the data. In image classification, Radial Basis Function (RBF) kernel, which is an efficient kernel function capable of handling nonlinear data distributions, is widely used. The main idea is to use the "kernel trick" to map the input data to a high-dimensional space where the data can be separated by category, and then use a linear classifier when the input dataset doesn't allow it. By feeding the feature vectors extracted from the images into the SVM model with the RBF kernel, the algorithm learns how to efficiently distinguish between different classes of images as discussed by Chapelle et al. (1999).

3 DATA AND MODELS

In this section, the dataset used and the data processing methods are presented, as well as the theoretical details of the machine learning approach using this dataset.

3.1 Data processing

In this study, two existed datasets are combined, which includes four animals: bears, pandas (Op, 2022), lions and tigers (Mehta, 2023), to evaluate the performance of different machine learning methods in classifying these animals. In the bear and panda dataset, a total of 500 images were used, while 563 images were used in the lion and tiger dataset.

To make the image processable by the algorithm, the image is transformed into an array of raw pixel feature vectors. In detail, the cv2.imread() function is used in the OpenCV library in Python to read the image files. Then, to improve the efficiency of data processing, images are resized to 64x64 pixels using the cv2.resize() function. Next, to simplify the process of further data processing and analysis, the flatten() function is used in the OpenCV library to convert the 3-dimensional image arrays with dimensions $64 \times 64 \times 3$ into one-dimensional arrays, and these one-dimensional arrays is added to the list image_data to store all the data. The detail of this conversion is described in Section 2.2.1. In addition, lists of tags are created to store the categories of images.

In the practical part, in order to facilitate the subsequent training and processing of the machine learning model, the labels are converted into numeric form as shown in Figure 3 and changed all the input data into the form of NumPy arrays.

In machine learning, the entire dataset needs to be divided into a training set and a test set to evaluate the performance of the model objectively. Training data is used to train the model and adapt the model to the pattern of the data. Meanwhile, the test data is a separate dataset from the training data which is used to evaluate the performance of the model on new data to ensure that our evaluation is not biased due to the fact that the model is familiar with the training data. In this way, it can be tested whether the model can be applied in new situations.

Thus, given our dataset, the data is split into training (75% of total) and test datasets (25% of total) randomly in order to ensure that both subsets are balanced and unbiased.

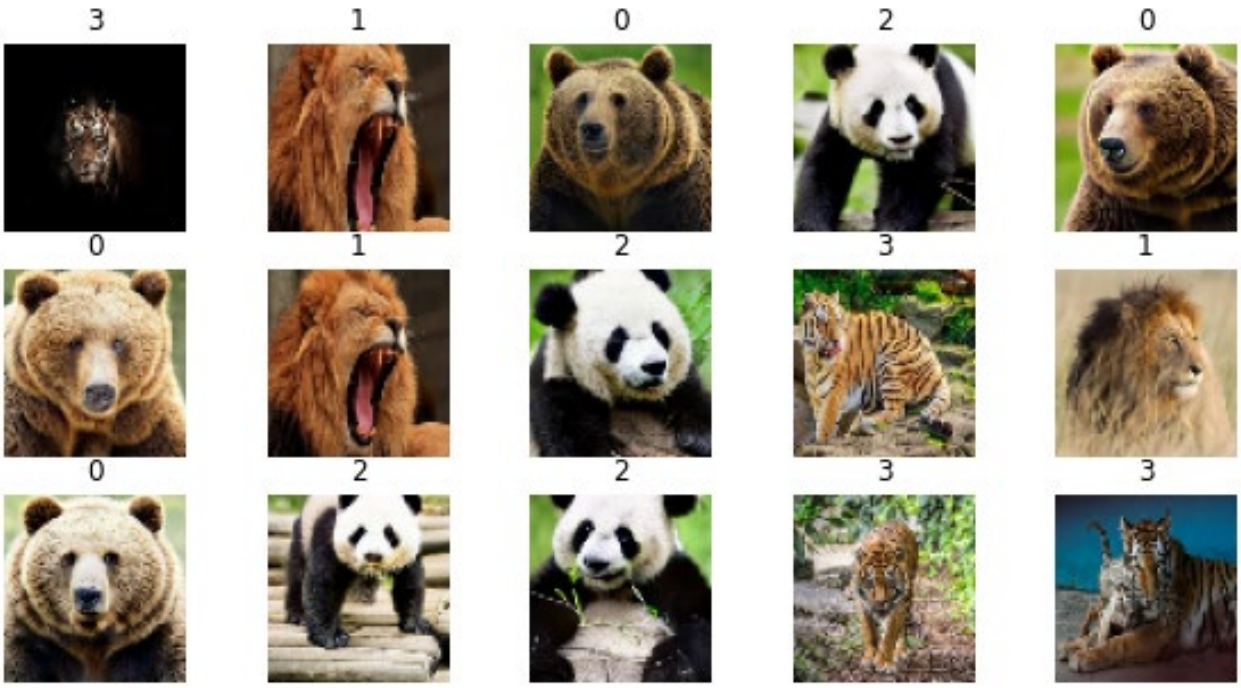

FIGURE 3. Visualize a subset of images, in which numbers are used to indicate the different categories of animals: 0-Bear, 1-Lion, 2-Panda, 3-Tiger.

3.2 Model evaluation

In classification models, the main goal is to give reasonable classes for new samples by training set fitting. And there are various evaluation criteria to assess the performance of the model. In supervised learning, the confusion matrix is a powerful tool to learn about the performance of classification models. It is usually defined as a contingency table with two dimensions of actual labels and predicted labels. In our example, the model gives a prediction of the class $\hat{y}_i$ for each sample $\{(\boldsymbol{x_i}, y_i), i = 1, \dots, N\}$ in the test set, where y_i is the real label of the sample. Then, the confusion matrix can be expressed as Figure 4.

The confusion matrix not only provides by-class and global accuracy of the classifier, but also allows further assessment of other metrics of the model. In this thesis, the global accuracy of the model is used and can be computed as

$$accuracy = \frac{\sum_{i=1}^{N} \mathbf{1}_{\{y_i = \hat{y}_i\}}}{N},$$

which is the proportion of correct labelling given for the model.

Another metric of interest to us is the precision for each class, which means the proportion of samples in each class for which the classifier gives the correct labels. The precision for class j has the form

$$precision_j = \frac{\sum_{i=1}^{N} \mathbf{1}_{\{y_i=\hat{y}_i=j\}}}{\sum_{i=1}^{N} \mathbf{1}_{\{y_i=j\}}}.$$

It is assumed that the importance of all classes is equal. On this basis, the macro average precision is an important indicator to reflect the overall level of precision of each class, which is the arithmetic mean of the precision rates of the classes.

		Predicted label			
		Bear (label = 0)	Lion (label = 1)	Panda (label = 2)	Tiger (label = 3)
Actual label	Bear (label = 0)	$\sum_{i=1}^{N} \mathbf{1}_{\{y_i=0,\hat{y}_i=0\}}$	$\sum_{i=1}^{N} \mathbf{1}_{\{y_i=0,\hat{y}_i=1\}}$	$\sum_{i=1}^{N} \mathbf{1}_{\{y_i=0,\hat{y}_i=2\}}$	$\sum_{i=1}^{N} \mathbf{1}_{\{y_i=0,\hat{y}_i=3\}}$
	Lion (label = 1)	$\sum_{i=1}^{N} \mathbf{1}_{\{y_i=1,\hat{y}_i=0\}}$	$\sum_{i=1}^{N} \mathbf{1}_{\{y_i=1,\hat{y}_i=1\}}$	$\sum_{i=1}^{N} \mathbf{1}_{\{y_i=1,\hat{y}_i=2\}}$	$\sum_{i=1}^{N} \mathbf{1}_{\{y_i=1,\hat{y}_i=3\}}$
	Panda (label = 2)	$\sum_{i=1}^{N} \mathbf{1}_{\{y_i=2,\hat{y}_i=0\}}$	$\sum_{i=1}^{N} \mathbf{1}_{\{y_i=2,\hat{y}_i=1\}}$	$\sum_{i=1}^{N} \mathbf{1}_{\{y_i=2,\hat{y}_i=2\}}$	$\sum_{i=1}^{N} \mathbf{1}_{\{y_i=2,\hat{y}_i=3\}}$
	Tiger (label = 3)	$\sum_{i=1}^{N} \mathbf{1}_{\{y_i=3,\hat{y}_i=0\}}$	$\sum_{i=1}^{N} \mathbf{1}_{\{y_i=3,\hat{y}_i=1\}}$	$\sum_{i=1}^{N} \mathbf{1}_{\{y_i=3,\hat{y}_i=2\}}$	$\sum_{i=1}^{N} \mathbf{1}_{\{y_i=3,\hat{y}_i=3\}}$

FIGURE 4. Example of confusion matrix. In each cell, the value is the number of samples in the dataset that satisfy the particular actual and predicted labels.

3.3 Models

In this study, multiple supervised learning algorithms are employed to accomplish the task of animal image classification. The purpose of this study is to compare the performance of different algorithms in this task, which includes comparative criteria such as confusion matrix, accuracy, etc. To

describe the following algorithm on the dataset more precisely, our dataset is abstracted into the following form:

We denote the 1-dimentional vector corresponding to each image as $\boldsymbol{x_i} = (x_{i1}, x_{i2}, \dots, x_{iK})^T$, and the label of each image as $y_i, i = 1, \dots, N$, where N is the number of images, and K is the number of elements in the 1-dimensional vector.

3.3.1 K-nearest neighbours algorithm

Given the straightforward principles of the k-NN algorithm, it is worthwhile to use it for image classification tasks. Given the training set, i.e., a set of datasets with known labels, a new sample will be classified. Specifically, it will evaluate the similarity between the images by calculating the Euclidean distance between them and then select the k nearest neighbours. The labels corresponding to these most similar neighbours will be used to classify the images to be identified. We compute the distance between a new image sample x_{new} and samples in the training data set, for which the Euclidean distance is calculated, with the form of

$$D(\boldsymbol{x_i}, \boldsymbol{x_{new}}) = \sqrt{\sum_{k=1}^{K} (x_{ik} - x_{new\,k})^2}, \qquad i = 1, \dots, N.$$

We rank the distances between the samples in the training set and the new sample and pick the first k neighbours that are closest to the new sample. According to the majority rule, the new sample is given the label $\hat{y}_{new}$ which is the majority of the k neighbours. Here the elements of the sample vector are the flattened pixel values. In Section 4.1, the algorithm is implemented by using the KNeighborsClassifier function from the sklearn.neighbors library in Python.

In this case, cross-validation is used to find the optimal values of the hyperparameter k. The training set is divided into 5 parts and determined the number of neighbours k based on the average of the accuracy.

3.3.2 Decision tree

Since our method distinguishes animals based on the colour of the image, in addition to a selection of pixel points, it is required to set a threshold for the colour channel value of the pixels in order to classify the images based on colour type and intensity. A decision tree starts at the root node at the bottom and will divide the dataset in two based on the intensity values of the original pixels of the image, repeating the operation at each layer until it reaches the leaf node, which is the result of the decision. Each node represents a feature test, e.g., a node may test whether the feature vector of an image exceeds a certain threshold.

For our pixel dataset, since each pixel value is continuous, we need to determine a pixel and a segmentation threshold (k^*, t^*) at each node of the tree. Here the objective function is defined as (Breiman et al., 1984, pp. 18-58)

$$\mathrm{G(D, k, t)} = \frac{\mathrm{n^{left}}}{\mathrm{n_D}} \mathrm{Impurity}\left(\mathrm{D^{left}}\right) + \frac{\mathrm{n^{right}}}{\mathrm{n_D}} \mathrm{Impurity}\left(\mathrm{D^{right}}\right),$$

where D and n_D is the data set and the number of samples at this node, $\mathrm{D^{left}}$ and D^{right} are defined as $D^{left} = \{D|x_{.k} \leq t\}$ and $D^{right} = \{D|x_{.k} > t\}$, respectively, while n^{left} and n^{right} are the corresponding number of samples in D^{left} and D^{right}. The Gini impurity is used in the implementation and defined as

$$Impurity(D) = \sum_{b} \frac{n_{y=b}}{n_D}\left(1 - \frac{n_{y=b}}{n_D}\right).$$

Low Gini impurity usually implies a more homogeneous subset, i.e., a higher percentage of data with the same label. In each step, the algorithm aims to find the optimal pixel and thereshold to obtain a smaller Gini impurity, which improves the classification accuracy. Therefore, (k^*, t^*) can be derived by

$$(k^*, t^*) = \underset{k,t}{\operatorname{argmin}}\, G(D, k, t).$$

We keep repeating the above steps to choose the best pixel and the best segmentation threshold at each node until the required tree depth is reached. The algorithm was implemented by using the DecisionTreeClassifier function in sklearn.tree and searching for the best tree depth using a grid search as shown in Section 4.2,.

In a decision tree, we can capture the level of importance of each feature, which depends on how much each feature contributes to reducing the uncertainty of the target variable. In the algorithm used, this is measured by the amount of reduction in Gini impurity achieved by making cuts to specific features. Thus, we can learn which pixels in the image give the most information for the classification.

3.3.3 Random forest

The fundamentals of random forests are explained in Section 2.2.4. For the implementation of the algorithm, the RandomForestClassifier function from sklearn.ensemble is used, where a default number of trees of 100 is set and the maximum depth of the tree is not defined specifically, i.e. up to the point where the subset of all nodes is of a single class or the subset capacity is less than three.

For each sample in the test set, the label can be predicted by selecting the majority vote among the trees. Here, decision trees in the forest are used to identify the most important pixel points, providing a basis for classification of the sample. In the decision tree, the feature importance in a tree is computed by Gini Impurity, whereas in the random forest, the mean decrease in impurity needs to be computed for the whole random forest to get the feature importance.

3.3.4 Gradient boosted decision tree

The basic idea of the algorithm can be found in Section 2.2.5. For our K-category classification problem, GBDT performs classification by analysing the feature vectors of an image. In each iteration, the algorithm determines which features best reduce classification errors and adjusts the construction of the decision tree in accordance with this. As the number of trees in the model increases, it gradually captures more complex image features, thereby improving the classification accuracy of the image. Friedman (2001) gave the principles of the algorithm for this scenario. For the i-th sample, the probability that it belongs to the k-th group can be calculated by

$$\hat{y}_{i,k} = F_{M,k}(x_i) = \sum_{m=1}^{M} \upsilon \, h_{m,k}(x_i),$$

where $h_{m,k}$ comes from the k-th tree of the m-th iteration and is called the 'weak learner' for the prediction update. After each iteration, the prediction will be updated by

$$F_{m,k}(x_i) = F_{m-1,k}(x_i) + \upsilon\, h_{m,k}(x_i),$$

where the parameter ν is referred as the learning rate, which scales the step size of the gradient descent process and thus the contribution of the weak learner. The weak learner $h_{m,k}$ is determined by minimizing the loss function $L_{m,k}$ at that stage. In the current scenario, the multinomial negative log-likelihood loss function is used as a general form of the loss function. Specifically, the weak learner has the form

$$h_{m,k} = \underset{h}{\operatorname{argmin}}\ L_{m,k} = \underset{h}{\operatorname{argmin}} \sum_{i=1}^{N}\sum_{k=1}^{K} L\left(y_{i,k}, F_{m-1,k}(x_i) + \upsilon\, h_{m,k}(x_i)\right),$$

where $y_{i,k} = 1_{\{y_i=k\}}$ and $L(y_{\cdot,k}, F_{\cdot,k}(x)) = -y_{\cdot k}\ \log p_k$, in which

$$p_k = exp(F_{\cdot,k}(x)) \Big/ \sum_{l=1}^{K} exp(F_{\cdot,l}(x)).$$

The GradientBoostingClassifier method in sklearn.ensemble are used to implement this process. To avoid overfitting, we retain the default maximum depth of each tree in the method, which is 3. Here we do not specifically assign an initial value for the probability of a sample belonging to each class, hence the initial value is given by the a priori probability of each class. Therefore, the algorithm will perform the classification task according to the above process and improve the performance of the classifier consistently.

3.3.5 Support vector machine

Despite the support vector machines (SVMs) have the ability to handle high-dimensional data efficiently, since each feature vector in our dataset contains 12,288 elements, it is reasonable to preserve as much information as possible in the dataset while reducing the data dimensionality in order to process the data more efficiently. The original dataset is pre-processed by using Principal Component Analysis (PCA).

PCA transforms the original dataset to a new coordinate system consisting of principal components by means of a linear transformation. In this case, the different pixels is recombined. The first principal component is a mapping of the original data in the direction of the maximum variance. The second principal component explains the maximum variance after the effects of the first principal component have been eliminated, and so on until all the variance is explained.

The covariance matrix of the data set is defined as

$$C = \frac{1}{N-1}\sum_{i=1}^{N}(\boldsymbol{x_i} - \overline{\boldsymbol{x}})(\boldsymbol{x_i} - \overline{\boldsymbol{x}})^T,$$

Here, $\overline{\boldsymbol{x}}$ is the mean of each feature across all samples $\{\boldsymbol{x_i}\}$. The eigenvalues of the covariance matrix $\{\lambda_i,\ i = 1, \ldots, N | \lambda_1 > \lambda_2 > \ldots > \lambda_K\}$ represent the variance of the data along the corresponding principal component directions. Thus, the importance of the components, or the explained variance ratio, has the form

$$ratio_i = \frac{\lambda_i}{\sum_{k=1}^{K}\lambda_k}.$$

In order to achieve an effective dimensionality reduction, it is plausible to retain the number of principal components with a cumulative explained variance greater than 80%. We use the principal component vector $\{\boldsymbol{z_j}\}$ obtained by linear transformation as the input feature vector of SVM.

The purpose of SVM is to find the optimal hyperplane that maximises the distance between classes. To avoid overfitting, soft margin techniques are used in our case, which include penalty terms for misclassified instances, aiming to strike a balance between smoothing the hyperplane as much as possible and allowing some misclassification. Mathematical details are omitted here due to the complexity and can be found in (Cortes & Vapnik, 1995). Due to the nature of the data, the RBF kernel is used to handle nonlinear boundaries, and the SVC function in sklearn.svm accomplishes this by specifying the kernel.

4 RESULTS

In this chapter, we focus on the results of the model fitting and the models used are described in Section 3. The entire dataset was split randomly into a training set of 75% samples, and a test set of 25% samples. Meanwhile, the sample's label were numericized, i.e. 0 – bear, 1 – Lion, 2 – Panda, 3 – Tiger.

To prevent predictions from being affected by differences in the dataset used to fit the model, in all methods, the same random numbers were used in the training-test set split; similarly, this operation was performed in some of the algorithms that require cross-validation to find the best hyperparameters.

4.1 K-nearest neighbours algorithm

The optimal number of neighbours k was derived by cross validation, as shown in Figure 5. To prevent overfitting, we select $k = 3$ instead of 1. With $k = 3$ and the training dataset described in Section 3.1, the model was then trained and predictions were made on the test dataset, with the accuracy of 0.77 and the confusion matrix as in Figure 6.

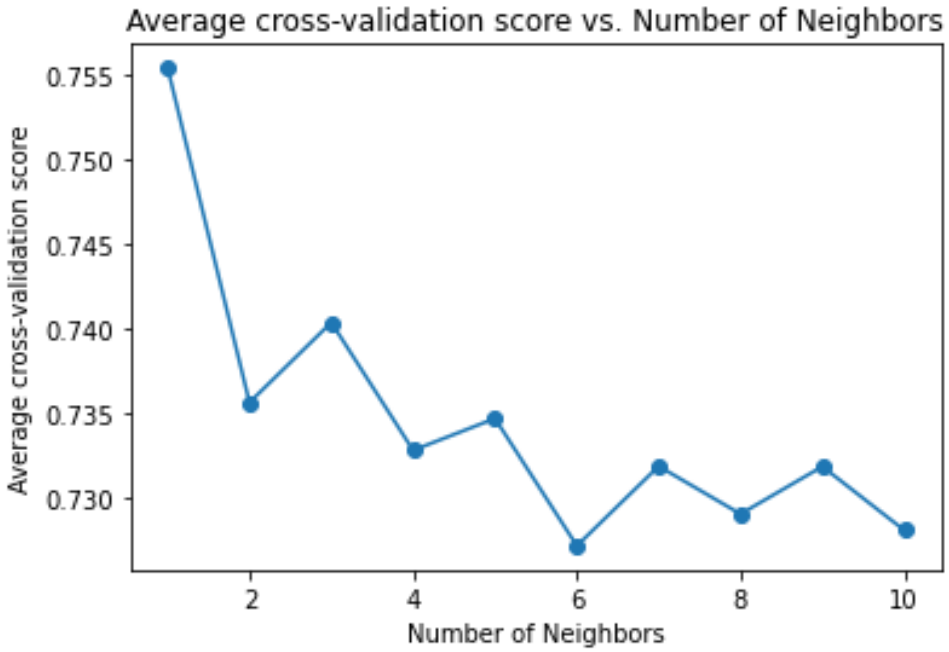

FIGURE 5. The average accuracy with different number of neighbours by cross validation.

Further calculation can be done from the confusion matrix in Figure 6 and find the precision rates of bear (label = 0), lion (label = 1) and Panda (label = 2) reached over 85%. However, the precision of the tiger (label = 3) is only about 24%, which means that there is a high probability that the tiger will be classified as lion, but the situation is not likely to be reversed, for only 3 out of 74 samples of lion is misclassified as tiger.

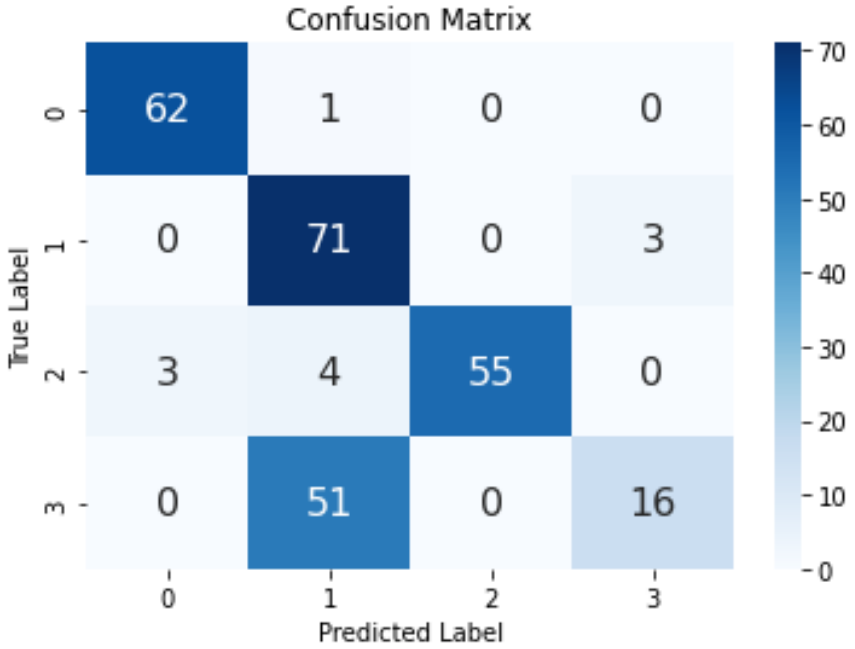

FIGURE 6. The confusion matrix of the test data set in the k-NN algorithm. The tags correspond to the following classes: 0 – Bear, 1 – Lion, 2 – Panda, 3 – Tiger.

4.2 Decision tree

In the decision tree, we found that the optimal depth of the tree is 4 by grid searching and the tree is visualized as shown in Figure 17 in Appendix. The splitting criteria for each branch of the tree are given in the nodes of the graph. With the test data set, the accuracy given by such a tree is 70% and the confusion matrix is shown in Figure 7.

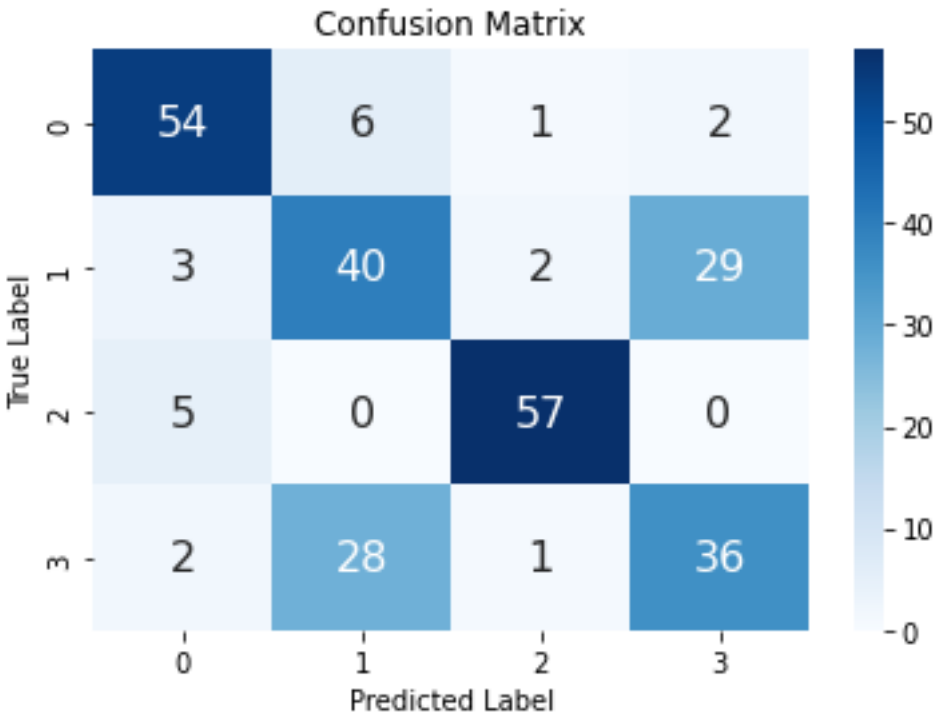

FIGURE 7. The confusion matrix of the test data set from the decision tree. The tags correspond to the following classes: 0 – Bear, 1 – Lion, 2 – Panda, 3 – Tiger.

We still care about the precision of the classifier for each class of images. For bear (label = 0) and panda (label = 2), decision tree could give high precision (> 85%). However, the precision is about 54% for both lion (label = 1) and tiger (label = 2), which is plausible since the lion and tiger have similar colours.

The pixels used in the decision tree are shown in Figure 18 and the importance of these features was ranked and the top ten most important features are displayed in Figure 9. The locations of all 14 features used in the tree are indicated in Figure 8. It can be seen that most of the important features are located at the edges of the pictures and only a few are located in the centre of the pictures, i.e., for most of the pictures, the area of the animal's face. It is due to the fact that the way the data and the decision tree algorithm are processed makes it impossible to differentiate between the animal and the background on a single image, whereas in a single decision tree, the features of the background portion in which the animal is located are also taken into account to distinguish between the animal classes. This leads to potential misidentification in some of the images, such as the lion and tiger in Figure 8, due to the similarity of the colours of their background parts.

FIGURE 8. The location of the 14 pixels used in the decision tree.

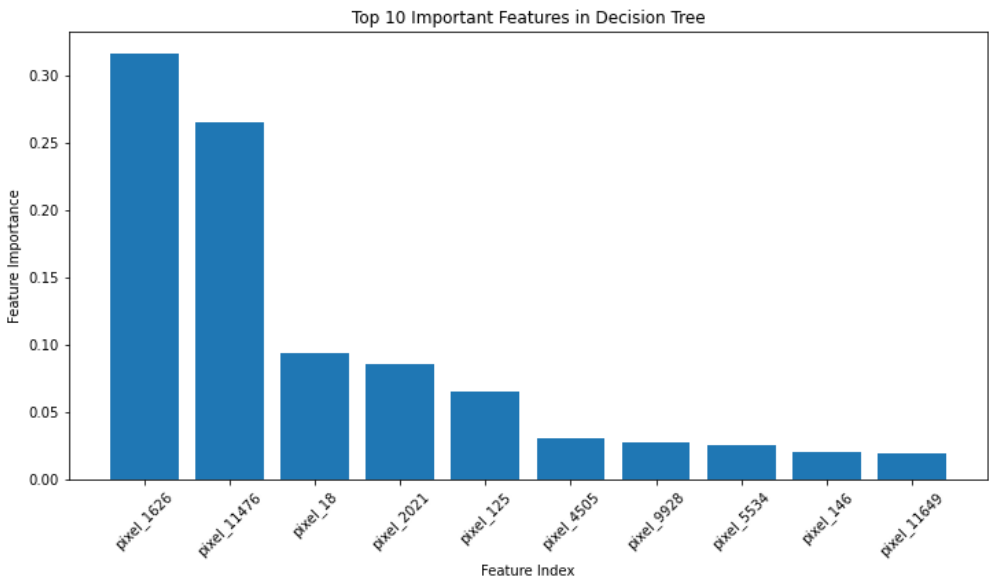

FIGURE 9. Top 10 most important features/index of pixels in the decision tree and their feature importance values.

4.3 Random forest

In the random forest, 100 trees were generated with no specified max depth. Compared to a single tree, the performance of a random forest is significantly enhanced. With the test data set, the accuracy is 85%, which is improved greatly compared to a single decision tree. For bear (label = 0) and panda (label = 2), the random forest could give 97% and 100% of precision, respectively. There is still a problem of confusion between the lion and tiger categories, but for lion (label = 1) and tiger (label = 2), the precision is increased to 78% and 66%, respectively.

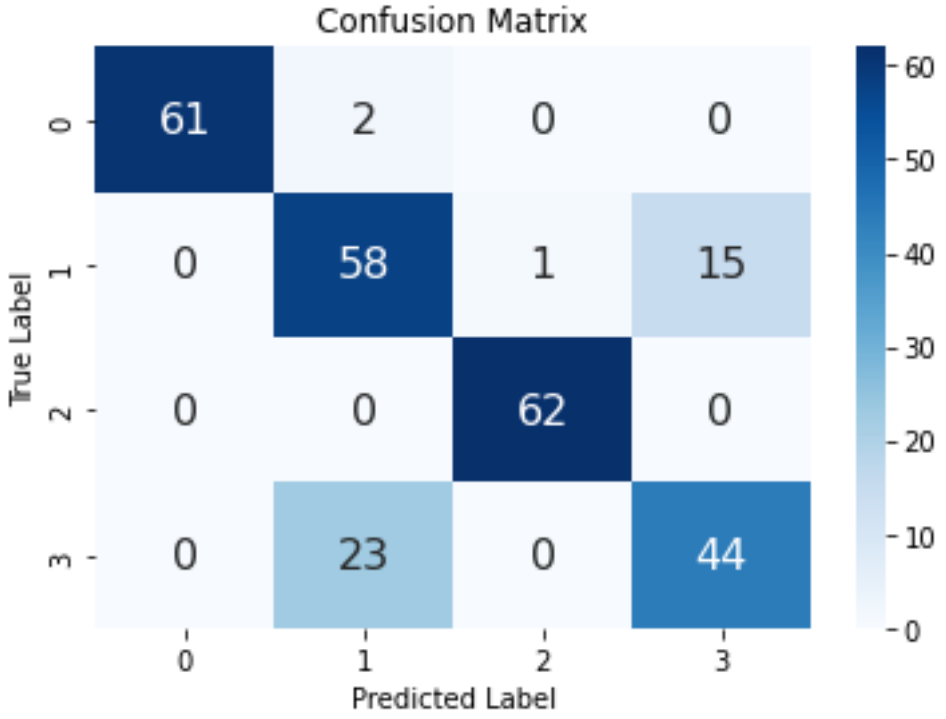

FIGURE 10. The confusion matrix of the test data set from the random forest. The tags correspond to the following classes: 0 – Bear, 1 – Lion, 2 – Panda, 3 – Tiger.

During model training, the importance of features was also calculated. We sort the feature importance and select the top ten most important features (Figure 11). The locations of these features/pixels are labeled in Figure 12. It is found that the most important pixels in the random forest are mostly located in the center of the image, where the animal's face is located in most of the images, and this region is important for recognizing animal images with high accuracy. Multiple trees make it significantly less likely that background is used as a source for identifying animal species, as occurs in decision tree algorithms, thus enhancing the robustness of the algorithm.

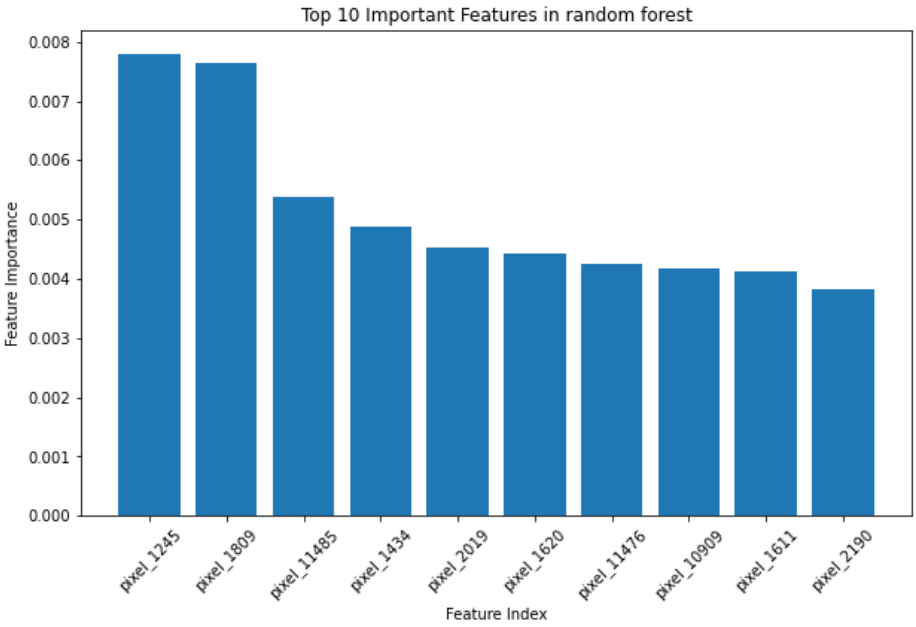

FIGURE 11. Top 10 most important features/index of pixels in the random forest and their feature importance values.

FIGURE 12. The location of the top 10 most important pixels in the random forest.

4.4 Gradient boosted decision tree

In the gradient boosted decision tree, the number of iterations is 100, and the maximum depth of each tree is 3. With the test data set, the accuracy is 84%, with precision values 97% in bears, 80% in lions, 98% in pandas, and 64% in tigers.

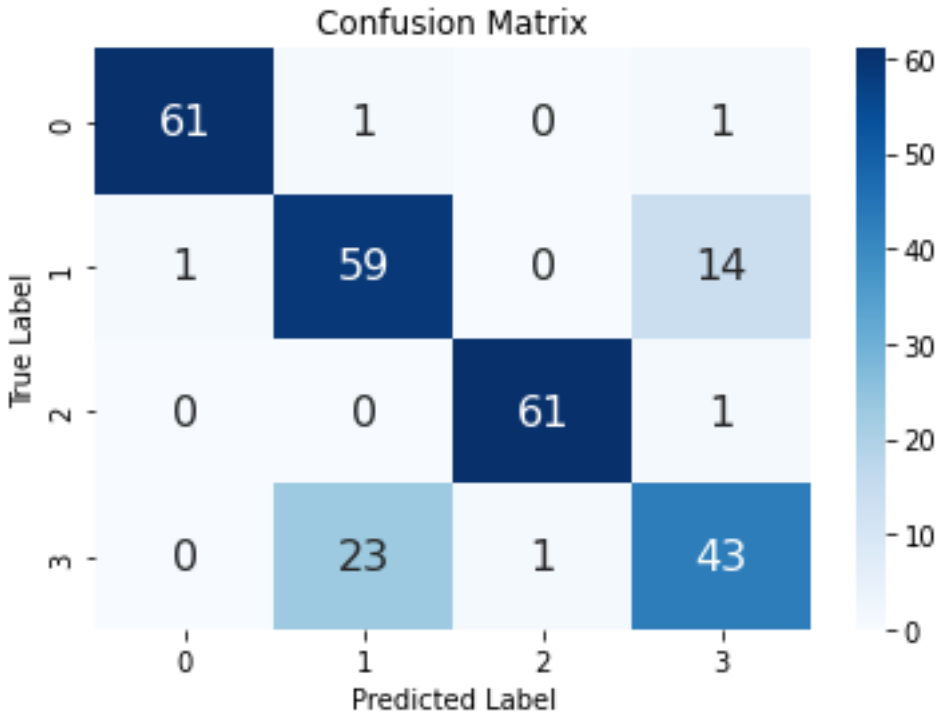

FIGURE 13. The confusion matrix of the test data set from the gradient boosted decision tree. The tags correspond to the following classes: 0 – Bear, 1 – Lion, 2 – Panda, 3 – Tiger.

Similarly, we ranked the feature importance and selected the top ten most important features as shown in Figure 14. The locations of these features/pixels are labeled in Fig. 15. It is found that most of the most important pixels are located in the center of the image, i.e., the face of the animal in most of the samples. The gradient boosted decision tree uses this portion of pixels as the basis for recognizing the animal images.

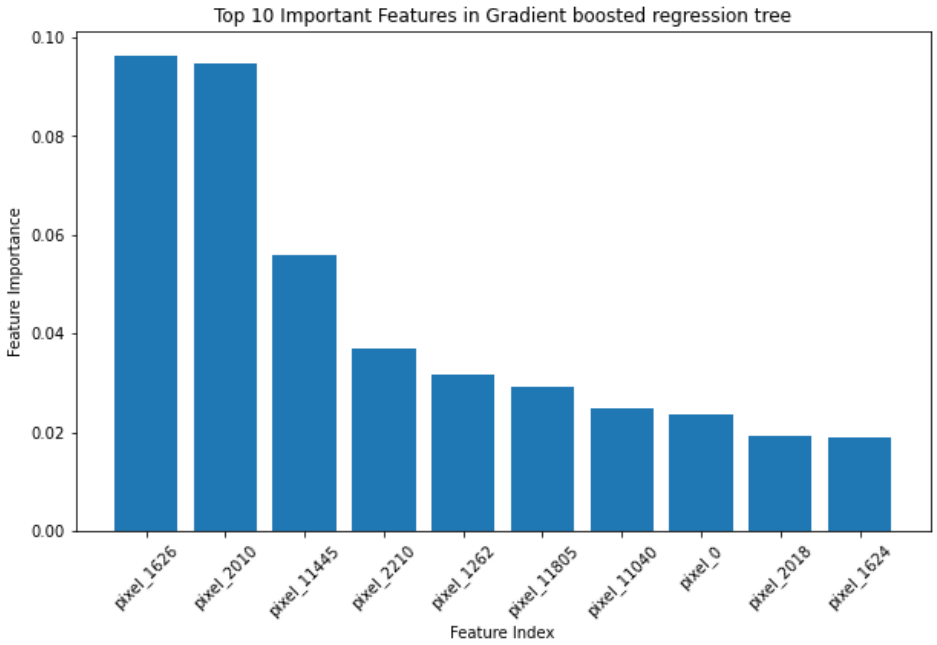

FIGURE 14. Top 10 most important features/index of pixels in the gradient boosted decision tree and their feature importance values.

FIGURE 15. The location of the top 10 most important pixels in the gradient boosted decision tree.

4.5 Support vector machine

In Support Vector Machine, we have used PCA as a dimensionality reduction tool. With the training set, the cumulative accuracy contributions of the principal components are shown in Figure 16. To prevent overfitting, we keep the number of principal components whose cumulative explained variance exceeds 80%, i.e., the first 6 principal components.

The SVM model was trained using data that had been downscaled by the PCA method. With the test data set, the accuracy is 80%, with precision values 98% in bears, 73% in lions, 97% in pandas, and 55% in tigers after further computation according to Figure 17.

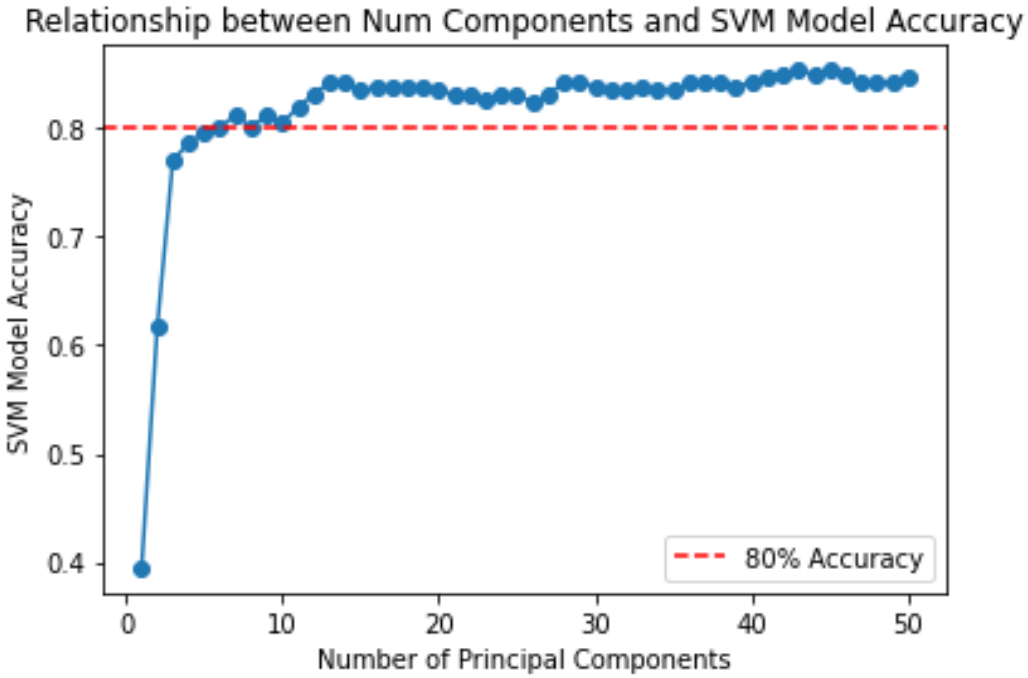

FIGURE 16. Cumulative Model Accuracy vs. Number of Principal Components using PCA in Support Vector Machines.

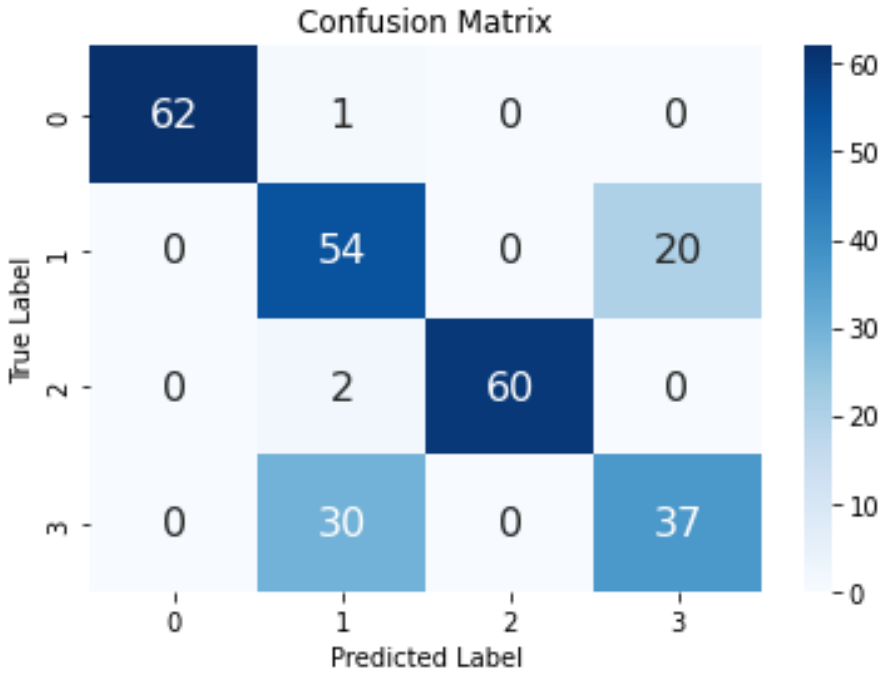

FIGURE 17. The confusion matrix of the test data set from support vector machine. The tags correspond to the following classes: 0 – Bear, 1 – Lion, 2 – Panda, 3 – Tiger.

4.6 Comparison

In this section, we integrate and compare the performance of the above five methods for classification on our dataset. The accuracy comparison is shown in Table 1 and it can be seen that the random forest model provides the highest accuracy score, while the single tree, i.e., the decision tree, provides the lowest accuracy score.

	K-NN	DT	RF	GBDT	SVM
ACCURACY	0.77	0.70	0.85	0.84	0.80

TABLE 1. The accuracy comparison between different methods including K-NN (K-nearest neighbours), DT (Decision tree), RF (Random forest), GBDT (Gradient boosted decision tree) and SVM (Support vector machine).

Furthermore, in addition to accuracy, we are also concerned with the precision in each class in the image classification with different methods, which is shown in Table 2. As in practical situations we need to treat all classes equally, we thereby compute the macro-mean score, which is the arithmetic average of the accuracies of the four classes, to evaluate the overall performance of the classifier.

As discussed in Section 4.1-4.5, in classifying bears and pandas, the precision remained high in all methods. Whereas confusion exists in labeling lions and tigers. In k-NN methods, the classifier's tendency to categorize images from both categories into lions, on the other hand, resulted in a high precision for lions but low precision for tigers. Whereas in other classifiers, lions and tigers are categorized to achieve a kind of equilibrium.

When we focus on the macro average precision, random forest, gradient boosted decision tree and support vector machine achieve high levels (>0.8), while the low precision on tigers makes the average precision of k-NN low despite the high precision of the other three classifications.

Another interesting indicator is the time required to run the algorithm. We compute the time from model training to predicting the test dataset, which includes the time taken to compute the confusion matrix and output images. The algorithm was run on a device with processor of 12th Gen Intel(R) Core(TM) i7-1255U 1.70 GHz. From Table 3, it is shown that k-NNs and decision trees take less time due to their simple form.

For the other better performing generally but more complex models, random forest took the least amount of time, and the support vector machine after data dimensionality reduction took slightly more time, but it was also acceptable. Although gradient boosted decision tree performs well on several metrics, it took tens of times longer than the other two models. Thus, for the image categorization problem discussed, random forest and the support vector machine perform better.

	K-NN	DT	RF	GBDT	SVM
BEAR	0.98	0.86	0.97	0.97	0.98
LION	0.96	0.54	0.78	0.80	0.83
PANDA	0.89	0.92	1.00	0.98	0.97
TIGER	0.21	0.54	0.66	0.64	0.55
MACRO AVERAGE	0.76	0.72	0.85	0.85	0.83

TABLE 2. The precision of different label and the macro-averaged precision, i.e., arithmetic average of the precision of the different categories with different training methods which includes K-NN (K-nearest neighbours), DT (decision tree), RF (random forest), GBDT (gradient boosted decision tree) and SVM (support vector machine).

	K-NN	DT	RF	GBDT	SVM
TIME	5.92	6.60	7.10	543.25	24.86

TABLE 3. The time used (in seconds) of different methods including K-NN (K-nearest neighbours), DT (Decision tree), RF (Random forest), GBDT (Gradient boosted decision tree) and SVM (Support vector machine).

5 DISSCUSION

In this thesis, we focus on the performance of different machine learning methods in image classification. Overall, more complex models provide better performance in accuracy. With our dataset, Random Forest displayed the best overall performance, with 85% accuracy and macro-averaged precision, having a fast-operating speed while guaranteeing performance. In addition, both GBDT and SVM provide acceptable accuracy and precision, but there is a big difference in the running speed between the two models. And the two simple models – k-NN and decision tree - perform much worse than the above models.

The research of Stehman and Czaplewski (1998), Gislason et al. (2006) and Baatuuwie and Leeuwen (2011) stated that Random Forest can give the highest accuracy in classification, compared with other models, which is consistent with our findings in this study. Meanwhile, Jin et al. (2014) pointed out that the performance of the Random Forest is better with a balanced number of samples in each category of the data, which is exactly what happens in our dataset. As for unbalanced datasets, the study of Noi and Kappas (2018) showed that when the number of samples is large enough, there is no significant difference between the performance of k-NN, Random Forest and SVM on balanced and unbalanced datasets; on the contrary, if the number of samples is small, the classifier is more sensitive to the data, and Random Forest is more advantageous in dealing with unbalanced data. Therefore, for different datasets, we should adopt slightly different model training strategies based on the performance of different classifiers (Noi and Kappas, 2018).

In our study, we used only the most basic model structure for various approaches. Whereas in existing studies, a variety of techniques are used to improve the classifiers. For example, in the study by Man et al. (2016), the splitting rules of nodes were improved to enhance the accuracy of image classification. And Demir and Erturk (2008) proposed to combine standard SVM classification with a hierarchical approach to improve the accuracy of SVM classification and reduce the computational load of the algorithms. Hence, in the future study, we can consider improving the existing algorithms to enhance the overall performance of the model.

6 CONCLUSION

In this thesis, we discussed ways to convert wildlife images into pixel data and classify animals using machine learning algorithms. We have used k-NN, Decision Tree, Random Forest, GBDT, SVM to classify a set of animal images and have evaluated and compared the performance of different methods. The accuracy of each method is comparably high, from 70% to 85%, and the average precision is between 72% and 85%. The decision trees had the lowest accuracy and average precision which were 70% and 72% respectively, while the integration of multiple decision trees, i.e. Random Forest, gave the highest accuracy and average precision which were both 85%. Overall, models with simpler structures, i.e., k-NN and Decision Tree, perform poorly, while complex models, i.e., Random Forest, GBDT, SVM, can give better performance. Whereas, considering the efficiency of model running, Random Forest could perform the model training with comparable running speed as the simple model among all the complex models. Hence, Random Forest gives the best overall performance in our dataset scenario.

REFERENCES

Alharbi, Fahad, Alharbi, Arar, & Kamioka, Eiji 2019. Animal species classification using machine learning techniques. *2018 International Joint Conference on Metallurgical and Materials Engineering (JCMME 2018)*, *277*. https://doi.org/10.1051/matecconf/201927702033

Baatuuwie, Bernard N., & Leeuwen, Louise Van 2011. Evaluation of three classifiers in mapping forest stand types using medium resolution imagery: A case study in the Offinso Forest District, Ghana. *African Journal of Environmental Science and Technology*, *5*(1).

Bachute, Mrinal, & Subhedar, Javed 2020. Autonomous Driving Architectures: Insights of Machine Learning and Deep Learning Algorithms. *Machine Learning With Applications*, *6*. https://doi.org/10.1016/j.mlwa.2021.100164

Bagaa, Miloud, Taleb, Tarik, Bernabe, Jorge Bernal, & Skarmeta, Antonio 2020. A Machine Learning Security Framework for Iot Systems. *IEEE Access*, *8*, 114066-114077. https://doi.org/10.1109/ACCESS.2020.2996214

Breiman, Leo, Friedman, Jerome, Olshen, Richard. A., & Stone, Charles. J. 1984. *Classification and Regression Trees* (1st ed., pp. 18-58). Chapman and Hall/CRC. https://doi.org/10.1201/9781315139470

Caruana, Rich, & Niculescu-Mizil, Alexandru 2006. An empirical comparison of supervised learning algorithms. *23rd International Conference on Machine Learning*, 161–168. https://doi.org/10.1145/1143844.1143865

Chapelle, Olivier, Haffner, Patrick, & Vapnik, Vladimir 1999. Support vector machines for histogram-based image classification. *IEEE Transactions on Neural Network*, *10*(5), 1055-1064. https://doi.org/10.1109/72.788646

Cortes, Corinna, & Vapnik, Vladimir 1995. Support-vector networks. *Machine Learning*, *20*, 273–297. https://doi.org/10.1007/BF00994018

Cover, Thomas, & Hart, Peter 1967. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, *13*(1), 21-27. https://doi.org/10.1109/TIT.1967.1053964

Demir, Begm, & Ertürk, Sarp 2009. Improving SVM classification accuracy using a hierarchical approach for hyperspectral images. *2009 16th IEEE International Conference on Image Processing (ICIP)*, 2849-2852. https://doi.org/10.1109/ICIP.2009.5414491

Friedman, Jerome H. 2001. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, *29*(5), 1189-1232. https://doi.org/10.1109/ICIP.2009.5414491

Gislason, Pall Oskar, Benediktsson, Jon Atli, & Sveinsson, Johannes R. 2006. Random Forests for land cover classification. *Pattern Recognition Letters*, *27*(4), 294-300. https://doi.org/10.1016/j.patrec.2005.08.011

Hastie, Trevor, Tibshirani, Robert, & Friedman, Jerome 2009. 10. Boosting and Additive Trees. *The Elements of Statistical Learning (2nd ed.).* New York: Springer. pp. 337–384. ISBN 978-0-387-84857-0.

Howley, Tom, Madden, Michael G., O'Connell, Marie-Louise, & Ryder, Alan G. 2007. The Effect of Principal Component Analysis on Machine Learning Accuracy with High Dimensional Spectral Data. *Applications and Innovations in Intelligent Systems XIII*. https://doi.org/10.1007/1-84628-224-1_16

Jin, Huiran, Stehman, Stephen V., & Mountrakis, Giorgos 2014. Assessing the impact of training sample selection on accuracy of an urban classification: A case study in Denver, Colorado. *International Journal of Remote Sensing*, *35*(6), 2067-2081. https://doi.org/10.1080/01431161.2014.885152

Johansson, Richard 2021. *Decision trees: Some additional notes* [Tutorials]. Retrieved March 25, 2024 from https://www.cse.chalmers.se/~richajo/dit866/lectures/l1/decision_trees.pdf

Khan, Abdullah Ayub, Laghari, Asif Ali, & Awan, Shafique Ahmed 2021. Machine Learning in Computer Vision: A Review. EAI Endorsed Transactions on Scalable Information Systems, 8(32). https://doi.org/10.4108/eai.21-4-2021.169418

Man, Weishi, Ji, Yuanyuan, & Zhang, Zhiyu 2018. Image classification based on improved random forest algorithm. *2018 IEEE 3rd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, 346-350. https://doi.org/10.1109/ICCCBDA.2018.8386540

Mehta, Anshul 2023. Wildlife Animals Images. Retrieved March 1, 2024 from https://www.kaggle.com/datasets/anshulmehtakaggl/wildlife-animals-images/data?select=lion-resize-224

Mullin, M. D., & Sukthanka, R. 1999. Complete Cross-Validation for Nearest Neighbor Classifiers. *International Conference on Machine Learning*. https://api.semanticscholar.org/CorpusID:1774187

Navlani, Avinash 2023. *Decision Tree Classification in Python Tutorial*. Datacamp. Retrieved March 20, 2024 from https://www.datacamp.com/tutorial/decision-tree-classification-python

Noi, Phan Thanh, & Kappas, Martin 2018. Comparison of Random Forest, k-Nearest Neighbor, and Support Vector Machine Classifiers for Land Cover Classification Using Sentinel-2 Imagery. *Sensors*, *18*(1). https://doi.org/10.3390/s18010018

Olsen, Cameron R., Mentz, Robert J., Anstrom, Kevin J., Page, David, & Patel, Priyesh A. 2020. Clinical applications of machine learning in the diagnosis, classification, and prediction of heart failure. *American Heart Journal*, *229*, 1-17. https://doi.org/10.1016/j.ahj.2020.07.009

Op, Matt 2022. Panda or Bear Image Classification. Retrieved March 1, 2024 from https://www.kaggle.com/datasets/mattop/panda-or-bear-image-classification/data

Pennacchiotti, Marco, & Popescu, Ana-Maria 2021. A Machine Learning Approach to Twitter User Classification. *Proceedings of the International AAAI Conference on Web and Social Media*, *5*(1), 281-288. https://doi.org/10.1609/icwsm.v5i1.14139

Qi, Yanjun 2012. Random Forest for Bioinformatics. In: Zhang, Cha & Ma, Yunqian (eds) Ensemble Machine Learning. Springer, New York, NY. https://doi.org/10.1007/978-1-4419-9326-7_11

Sachinsoni 2023. *K Nearest Neighbours — Introduction to Machine Learning Algorithms*. Medium. Retrieved March 12, 2024 from https://medium.com/@sachinsoni600517/k-nearest-neighbours-introduction-to-machine-learning-algorithms-9dbc9d9fb3b2

Srinivasan, Harish, Beal, Matthew J., & Srihari, Sargur N. 2005. Machine learning approaches for person identification and verification. *Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense IV*, *5778*. https://doi.org/10.1117/12.601987

Stehman, Stephen V., & Czaplewski, Raymond L. 1998. Design and Analysis for Thematic Map Accuracy Assessment: Fundamental Principles. *Remote Sensing of Environment*, *64*(3), 331-344. https://doi.org/10.1016/S0034-4257(98)00010-8

Subasi, Emir, & Subasi, Abdulhamit 2016. Performance of Random Forest and SVM in Face Recognition. *The International Arab Journal of Information Technology*, *13*(2). https://api.semanticscholar.org/CorpusID:7820835

Thallam, Chinmayi, Peruboyina, Aarsha, Raju, Sagi Sai Tejasvi, & Sampath, Nalini 2020. Early Stage Lung Cancer Prediction Using Various Machine Learning Techniques. *2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, 1285-1292. https://doi.org/10.1109/ICECA49313.2020.9297576

Xu, Kangming, Zhou, Huiming, Zheng, Haotian, Zhu, Mingwei, & Xin, Qi 2024. Intelligent Classification and Personalized Recommendation of E-commerce Products Based on Machine Learning. *ArXiv:2403.19345 [Preprint]*. https://doi.org/10.48550/arXiv.2403.19345

Yeung, Serena 2015 Tutorial 1: Image Filtering. *Computer Vision* [Tutorials]. Retrieved March 15, 2024 from https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html

Zhu, Li, Wu, Minghu, Wan, Xiangkui, Zhao, Nan, & Xiong, Wei 2017. Image Recognition of Rapeseed Pests Based on Random Forest Classifier. *International Journal of Information Technology and Web Engineering (IJITWE)*, *12*(3). https://doi.org/10.4018/IJITWE.2017070101

APPENDICES

Appendix 1. Source code

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
import time

# 1. Loading image data
image_data = []
labels = []
```

```
# Load images of Bears
bears_path = "PandasBearsLions/Train/Bears/"
for filename in os.listdir(bears_path):
    img = cv2.imread(os.path.join(bears_path, filename))
    img = cv2.resize(img, (64, 64))
    image_data.append(img.flatten())
    labels.append("Bear")

# Load images of Lions
lions_path = "PandasBearsLions/Train/Lions/"
for filename in os.listdir(lions_path):
    img = cv2.imread(os.path.join(lions_path, filename))
    img = cv2.resize(img, (64, 64))
    image_data.append(img.flatten())
    labels.append("Lions")
```

```
# Load images of Tigers
tigers_path = "PandasBearsLions/Train/Tigers/"
for filename in os.listdir(tigers_path):
    img = cv2.imread(os.path.join(tigers_path, filename))
    img = cv2.resize(img, (64, 64))
    image_data.append(img.flatten())
    labels.append("Tigers")

# Load images of Pandas
pandas_path = "PandasBearsLions/Train/Pandas/"
for filename in os.listdir(pandas_path):
    img = cv2.imread(os.path.join(pandas_path, filename))
    img = cv2.resize(img, (64, 64))
    image_data.append(img.flatten())
    labels.append("Panda")
```

```
# 2. Convert image data and labels into NumPy arrays for easier processing
image_data = np.array(image_data)
labels = np.array(labels)

# 3. Encode string labels to numeric values
le = LabelEncoder()
labels_encoded = le.fit_transform(labels)
```

```
# 4. Print the shape of image data array to verify
print(image_data.shape)

# 5. Print unique labels for verification
unique_labels = np.unique(labels)
print("Unique labels:", unique_labels)

# 6. Measure processing time
start = time.time()
```

```
# 7. Split data into training and test datasets
(
    X_train,
    X_test,
    y_train,
    y_test
) = train_test_split(
    image_data,
    labels_encoded,
    test_size=0.25,
    random_state=40,
    stratify=labels_encoded
)
```

```
# 8. Print shapes of the split datasets to check
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)


# 9. Visualize a subset of test images
fig, axes = plt.subplots(3, 5, figsize=(10, 5))
for i, ax in enumerate(axes.flat):
    # Reshape without flattening and ensure uint8 type
    image = X_test[i].reshape(64, 64, 3).astype(np.uint8)
    #Conversion from color BGR to RGB
    img_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    ax.imshow(img_rgb)
    ax.set_title(y_test[i])
    ax.axis('off')

plt.show()
```

```
# 10. Training accuracy visualization
neighbors_range = range(1, 11)
training_accuracies = []

for n_neighbors in neighbors_range:
    knn_classifier = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn_classifier.fit(X_train, y_train)
    cv_accuracy = cross_val_score(knn_classifier, image_data, labels, cv=5, scoring='accuracy')
    training_accuracies.append(cv_accuracy.mean())
```

```
# 11. Plot training accuracies
plt.plot(neighbors_range, training_accuracies, marker='o')
plt.title('Average cross-validation score vs. Number of Neighbors')
plt.xlabel('Number of Neighbors')
plt.ylabel('Average cross-validation score')
plt.show()
```

```
# 12.  Initialize and train the KNN classifier
knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_train, y_train)
```

```
# 13. Count different animal labels in the training set
unique_labels_train, counts_train = np.unique(y_train, return_counts=True)
animal_counts_train = dict(zip(le.inverse_transform(unique_labels_train), counts_train))
print("Number of different animal labels in X_train:")
print(animal_counts_train)

# 14. Prediction on the test set
y_pred = knn_classifier.predict(X_test)
print(y_pred)

# 15. Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
# 16.confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", annot_kws={"size": 16})
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# 17 Measure processing time
end = time.time()
print(end - start)
```

```
import os
import cv2
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.model_selection import GridSearchCV
import time

# 1. Loading image data
image_data = []
labels = []
```

```
# Load images of Bears
bears_path = "PandasBearsLions/Train/Bears/"
for filename in os.listdir(bears_path):
    img = cv2.imread(os.path.join(bears_path, filename))
    img = cv2.resize(img, (64, 64))
    image_data.append(img.flatten())
    labels.append("Bear")

# Load images of Lions
lions_path = "PandasBearsLions/Train/Lions/"
for filename in os.listdir(lions_path):
    img = cv2.imread(os.path.join(lions_path, filename))
    img = cv2.resize(img, (64, 64))
    image_data.append(img.flatten())
    labels.append("Lions")
```

```
# Load images of Tigers
tigers_path = "PandasBearsLions/Train/Tigers/"
for filename in os.listdir(tigers_path):
    img = cv2.imread(os.path.join(tigers_path, filename))
    img = cv2.resize(img, (64, 64))
    image_data.append(img.flatten())
    labels.append("Tigers")

# Load images of Pandas
pandas_path = "PandasBearsLions/Train/Pandas/"
for filename in os.listdir(pandas_path):
    img = cv2.imread(os.path.join(pandas_path, filename))
    img = cv2.resize(img, (64, 64))
    image_data.append(img.flatten())
    labels.append("Panda")
```

```
# 2. Convert image data and labels into NumPy arrays for easier processing
image_data = np.array(image_data)
labels = np.array(labels)

# 3. Encode string labels to numeric values
le = LabelEncoder()
labels_encoded = le.fit_transform(labels)
```

```
# 4. Output unique labels to verify all are processed
unique_labels = np.unique(labels)
print("Unique labels:", unique_labels)

# 5. Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(image_data, labels_encoded, test_size=0.25,
random_state=40, stratify=labels_encoded)

# 6. Print shapes of the split datasets to check
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
print(X_train.shape[1])
print("pixel1626", X_train[1][1626])

# 7. Initialize and perform grid search to find the best parameters for Decision Tree
testingTree = DecisionTreeClassifier(random_state=0)
param_grid = {'max_depth': [1, 2, 3, 4]}
grid_search = GridSearchCV(testingTree, param_grid, cv=5)
grid_search.fit(X_train, y_train)
print("Best parameters: {}".format(grid_search.best_params_))
```

```
# 8. Measure processing time
start = time.time()

# 9. Train Decision Tree classifier
tree = DecisionTreeClassifier(max_depth=4, random_state=0)
tree.fit(X_train, y_train)

# 10. Visualize the Decision Tree
plt.figure(figsize=(12, 8),dpi=300)
plot_tree(tree, filled=True, feature_names=[f'pixel_{i}' for i in range(X_train.shape[1])],
class_names=unique_labels.tolist(), rounded=True)
plt.title("Decision Tree Visualization")
plt.show()
```

```
# 11. Analyze and display important features in the tree
def extract_decision_features(tree):
    decision_features = []
    feature_importances = tree.feature_importances_
    non_zero_indices = np.nonzero(feature_importances)[0]
    decision_features.extend(non_zero_indices)
    return decision_features

# 12. Extract decision features from the tree
decision_features = extract_decision_features(tree)
```

```
# 13. Print the decision features
print("Decision features:")
print(len(decision_features))
print(decision_features)

sample_image_index = 24
sample_image = X_train[sample_image_index].reshape(64, 64, 3).astype(np.uint8)
sample_image_rgb= cv2.cvtColor(sample_image, cv2.COLOR_BGR2RGB)


# 14. Create a blank image for visualization
visualization_image = np.copy(sample_image_rgb)

# 15. Mark the top N important features on the visualization image
for feature_index in decision_features:
    # Convert the flattened feature index to 2D coordinates
    row, col = divmod(feature_index, 64)
    # Check bounds to prevent the "index out of bounds" issue
    row = min(row, 63)
    col = min(col, 63)
    # Mark the pixel on the visualization image
    visualization_image[row, col] = [255, 0, 0]  # Mark in red

# 16. Display the original image with marked important features
plt.imshow(visualization_image)
plt.axis('off')
plt.show()
```

```
# 17. Get feature importances from the trained Decision Tree
feature_importances = tree.feature_importances_

# 18. Sort feature importances in descending order
sorted_indices = np.argsort(feature_importances)[::-1]

# 19. Get the top N most important features
top_n = 10
top_feature_indices = sorted_indices[:top_n]

# 20. Visualize the importance of the top features
plt.figure(figsize=(10, 6))
plt.bar(range(top_n), feature_importances[top_feature_indices], align="center")
plt.xticks(range(top_n), [f'pixel_{i}' for i in top_feature_indices], rotation=45)
plt.xlabel("Feature Index")
plt.ylabel("Feature Importance")
plt.title("Top {} Important Features in Decision Tree".format(top_n))
plt.show()

# 21. Make predictions on the test set
y_pred = tree.predict(X_test)

# 22. confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", annot_kws={"size": 16})
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
# 23. Evaluate the accuracy
accuracy = (np.sum(np.diagonal(cm))) / np.sum(cm)
print("Accuracy:", accuracy)

# 24. Measure processing time
end = time.time()
print(end - start)
```

```
import os
import cv2
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
import time

# 1. Loading image data
image_data = []
labels = []
```

```
# Load images of Bears
bears_path = "PandasBearsLions/Train/Bears/"
for filename in os.listdir(bears_path):
    img = cv2.imread(os.path.join(bears_path, filename))
    img = cv2.resize(img, (64, 64))
    image_data.append(img.flatten())
    labels.append("Bear")

# Load images of Lions
lions_path = "PandasBearsLions/Train/Lions/"
for filename in os.listdir(lions_path):
    img = cv2.imread(os.path.join(lions_path, filename))
    img = cv2.resize(img, (64, 64))
    image_data.append(img.flatten())
    labels.append("Lions")
```

```
# Load images of Tigers
tigers_path = "PandasBearsLions/Train/Tigers/"
for filename in os.listdir(tigers_path):
    img = cv2.imread(os.path.join(tigers_path, filename))
    img = cv2.resize(img, (64, 64))
    image_data.append(img.flatten())
    labels.append("Tigers")

# Load images of Pandas
pandas_path = "PandasBearsLions/Train/Pandas/"
for filename in os.listdir(pandas_path):
    img = cv2.imread(os.path.join(pandas_path, filename))
    img = cv2.resize(img, (64, 64))
    image_data.append(img.flatten())
    labels.append("Panda")
```

```
# 2. Convert image data and labels into NumPy arrays for easier processing
image_data = np.array(image_data)
labels = np.array(labels)

# 3. Encode string labels to numeric values
le = LabelEncoder()
labels_encoded = le.fit_transform(labels)
```

```
# 4. Print unique labels for verification
unique_labels = np.unique(labels)
print("Unique labels:", unique_labels)

# 5. Measure processing time
start = time.time()

# 6. Split data into training and test datasets
X_train, X_test, y_train, y_test = train_test_split(image_data, labels_encoded, test_size=0.25,
random_state=40, stratify=labels_encoded)

# 7. Print shapes of the split datasets to check
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
print("pixel1626", X_train[1][2190])

# 8. Create a Random Forest classifier
forest = RandomForestClassifier(n_estimators=100, random_state=0)

# 9. Train the classifier
forest.fit(X_train, y_train)

# 10. Make predictions on the test set
y_pred = forest.predict(X_test)
```

```
# 11. Display feature importances
feature_importances = forest.feature_importances_
sorted_indices = np.argsort(feature_importances)[::-1]

# 12. Get the top N most important features
top_n = 10
top_feature_indices = sorted_indices[:top_n]

# 13. Visualize the importance of the top features
plt.figure(figsize=(10, 6))
plt.bar(range(top_n), feature_importances[top_feature_indices], align="center")
plt.xticks(range(top_n), [f'pixel_{i}' for i in top_feature_indices], rotation=45)
plt.xlabel("Feature Index")
plt.ylabel("Feature Importance")
plt.title("Top {} Important Features in random forest".format(top_n))
plt.show()

# 14. Sample image index and sample image
sample_image_index =24
sample_image = X_train[sample_image_index].reshape(64, 64, 3).astype(np.uint8)
sample_image_rgb = cv2.cvtColor(sample_image, cv2.COLOR_BGR2RGB)
```

```
# 15. Create a blank image for visualization
visualization_image = np.copy(sample_image_rgb)
# Mark the top N important features on the visualization image
for feature_index in top_feature_indices:
    # Convert the flattened feature index to 2D coordinates
    row, col = divmod(feature_index, 64)
    # Check bounds to prevent the "index out of bounds" issue
    row = min(row, 63)
    col = min(col, 63)
    # Mark the pixel on the visualization image
    visualization_image[row, col] = [255, 0, 0]  # Mark in red

plt.imshow(visualization_image)
plt.axis('off')
plt.show()
```

```
# 16.confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", annot_kws={"size": 16})
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()


# 17. Evaluate the model
accuracy = (np.sum(np.diagonal(cm))) / np.sum(cm)
print("Accuracy:", accuracy)

# 18. Measure processing time
end = time.time()
print(end - start)
```

```
import os
import cv2
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
import time

# 1. Loading image data
image_data = []
labels = []
```

```
# Load images of Bears
bears_path = "PandasBearsLions/Train/Bears/"
for filename in os.listdir(bears_path):
    img = cv2.imread(os.path.join(bears_path, filename))
    img = cv2.resize(img, (64, 64))
    image_data.append(img.flatten())
    labels.append("Bear")

# Load images of Lions
lions_path = "PandasBearsLions/Train/Lions/"
for filename in os.listdir(lions_path):
    img = cv2.imread(os.path.join(lions_path, filename))
    img = cv2.resize(img, (64, 64))
    image_data.append(img.flatten())
    labels.append("Lions")
```

```
# Load images of Tigers
tigers_path = "PandasBearsLions/Train/Tigers/"
for filename in os.listdir(tigers_path):
    img = cv2.imread(os.path.join(tigers_path, filename))
    img = cv2.resize(img, (64, 64))
    image_data.append(img.flatten())
    labels.append("Tigers")

# Load images of Pandas
pandas_path = "PandasBearsLions/Train/Pandas/"
for filename in os.listdir(pandas_path):
    img = cv2.imread(os.path.join(pandas_path, filename))
    img = cv2.resize(img, (64, 64))
    image_data.append(img.flatten())
    labels.append("Panda")
```

```
# 2. Convert image data and labels into NumPy arrays for easier processing
image_data = np.array(image_data)
labels = np.array(labels)

# 3. Encode string labels to numeric values
le = LabelEncoder()
labels_encoded = le.fit_transform(labels)
```

```
# 4. Print unique labels for verification
unique_labels = np.unique(labels)
print("Unique labels:", unique_labels)

# 5. Measure processing time
start = time.time()


# 6. Split data into training and test datasets
X_train, X_test, y_train, y_test = train_test_split(image_data, labels_encoded, test_size=0.25,
random_state=40, stratify=labels_encoded)


# 7. Print shapes of the split datasets to check
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

# 8. Create and fit the GradientBoostingClassifier
boosted_decision_trees = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=1,
random_state=0)
boosted_decision_trees.fit(X_train, y_train)

# 9. Make predictions on the test set
y_pred = boosted_decision_trees.predict(X_test)


# 10. Get feature importances from the trained random forest
feature_importances = boosted_decision_trees.feature_importances_
```

```
# 11. Sort feature importances in descending order
sorted_indices = np.argsort(feature_importances)[::-1]

# 12. Get the top N most important features
top_n = 10
top_feature_indices = sorted_indices[:top_n]

# 13. Visualize the importance of the top features
plt.figure(figsize=(10, 6))
plt.bar(range(top_n), feature_importances[top_feature_indices], align="center")
plt.xticks(range(top_n), [f'pixel_{i}' for i in top_feature_indices], rotation=45)
plt.xlabel("Feature Index")
plt.ylabel("Feature Importance")
plt.title("Top {} Important Features in Gradient boosted regression tree".format(top_n))
plt.show()

# 14. Sample image index and sample image
sample_image_index = 24
sample_image = X_train[sample_image_index].reshape(64, 64, 3).astype(np.uint8)
sample_image_rgb = cv2.cvtColor(sample_image, cv2.COLOR_BGR2RGB)
```

```
# 15. Create a blank image for visualization
visualization_image = np.copy(sample_image_rgb)
# Mark the top N important features on the visualization image
for feature_index in top_feature_indices:
    # Convert the flattened feature index to 2D coordinates
    row, col = divmod(feature_index, 64)
    # Check bounds to prevent the "index out of bounds" issue
    row = min(row, 63)
    col = min(col, 63)
    # Mark the pixel on the visualization image
    visualization_image[row, col] = [255, 0, 0]  # Mark in red

plt.imshow(visualization_image)
plt.axis('off')
plt.show()
```

```
# 16. confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", annot_kws={"size": 16})
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()


# 17. Evaluate the model
accuracy = (np.sum(np.diagonal(cm))) / np.sum(cm)
print("Accuracy:", accuracy)

# 18. Measure processing time
end = time.time()
print(end - start)
```

```
import os
import cv2
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
import seaborn as sns
import time

# 1. Loading image data
image_data = []
labels = []
```

```
# Load images of Bears
bears_path = "PandasBearsLions/Train/Bears/"
for filename in os.listdir(bears_path):
    img = cv2.imread(os.path.join(bears_path, filename))
    img = cv2.resize(img, (64, 64))
    image_data.append(img.flatten())
    labels.append("Bear")

# Load images of Lions
lions_path = "PandasBearsLions/Train/Lions/"
for filename in os.listdir(lions_path):
    img = cv2.imread(os.path.join(lions_path, filename))
    img = cv2.resize(img, (64, 64))
    image_data.append(img.flatten())
    labels.append("Lions")
```

```
# Load images of Tigers
tigers_path = "PandasBearsLions/Train/Tigers/"
for filename in os.listdir(tigers_path):
    img = cv2.imread(os.path.join(tigers_path, filename))
    img = cv2.resize(img, (64, 64))
    image_data.append(img.flatten())
    labels.append("Tigers")

# Load images of Pandas
pandas_path = "PandasBearsLions/Train/Pandas/"
for filename in os.listdir(pandas_path):
    img = cv2.imread(os.path.join(pandas_path, filename))
    img = cv2.resize(img, (64, 64))
    image_data.append(img.flatten())
    labels.append("Panda")
```

```
# 2. Convert image data and labels into NumPy arrays for easier processing
image_data = np.array(image_data)
labels = np.array(labels)

# 3. Encode string labels to numeric values
le = LabelEncoder()
labels_encoded = le.fit_transform(labels)
```

```
# 4. Measure processing time
start = time.time()

# 5. Iterate over a range of num_components from 1 to 100
num_components_values = range(1, 51)
accuracy_scores = []

for num_components in num_components_values:
    # Apply PCA for dimensionality reduction
    pca = PCA(n_components=num_components)
    image_data_pca = pca.fit_transform(image_data)

    # Divide the training set and test set
    X_train, X_test, y_train, y_test = train_test_split(image_data_pca, labels_encoded, test_size=0.25,
random_state=40, stratify=labels_encoded)

    # Train an SVM with RBF kernel
    svm_model = SVC(kernel='rbf', C=1, gamma='scale')
    svm_model.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = svm_model.predict(X_test)

    # Evaluate the accuracy
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_scores.append(accuracy)

    print(f"Num Components: {num_components}, Accuracy: {accuracy}")
```

```
# 6. Plot the relationship between num_components and SVM model accuracy
plt.plot(num_components_values, accuracy_scores, marker='o')
plt.xlabel('Number of Principal Components')
plt.ylabel('SVM Model Accuracy')
plt.title('Relationship between Num Components and SVM Model Accuracy')
```

```
# 7. Draw a line at 80% accuracy on the y-axis
plt.axhline(y=0.8, color='r', linestyle='--', label='80% Accuracy')
plt.legend()
plt.show()
```

```
# 8. Split data into training and test datasets
X_train, X_test, y_train, y_test = train_test_split(image_data, labels_encoded, test_size=0.25,
random_state=40, stratify=labels_encoded)

# 9. Apply PCA for dimensionality reduction
n_components = 6
pca = PCA(n_components=n_components)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# 10. Train an SVM with RBF kernel on the reduced feature space
svm_model = SVC(kernel='rbf', C=1, gamma='scale')  # You can adjust C and gamma parameters based on your
dataset
svm_model.fit(X_train_pca, y_train)

# 11. Make predictions on the test set
y_pred = svm_model.predict(X_test_pca)
```

```
# 12. confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", annot_kws={"size": 16})
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# 13. Print the evaluation metrics
accuracy = (np.sum(np.diagonal(cm))) / np.sum(cm)
print("Accuracy:", accuracy)

# 14. Measure processing time
end = time.time()
print(end - start)
```

Appendix 2. Visualisation of decision tree

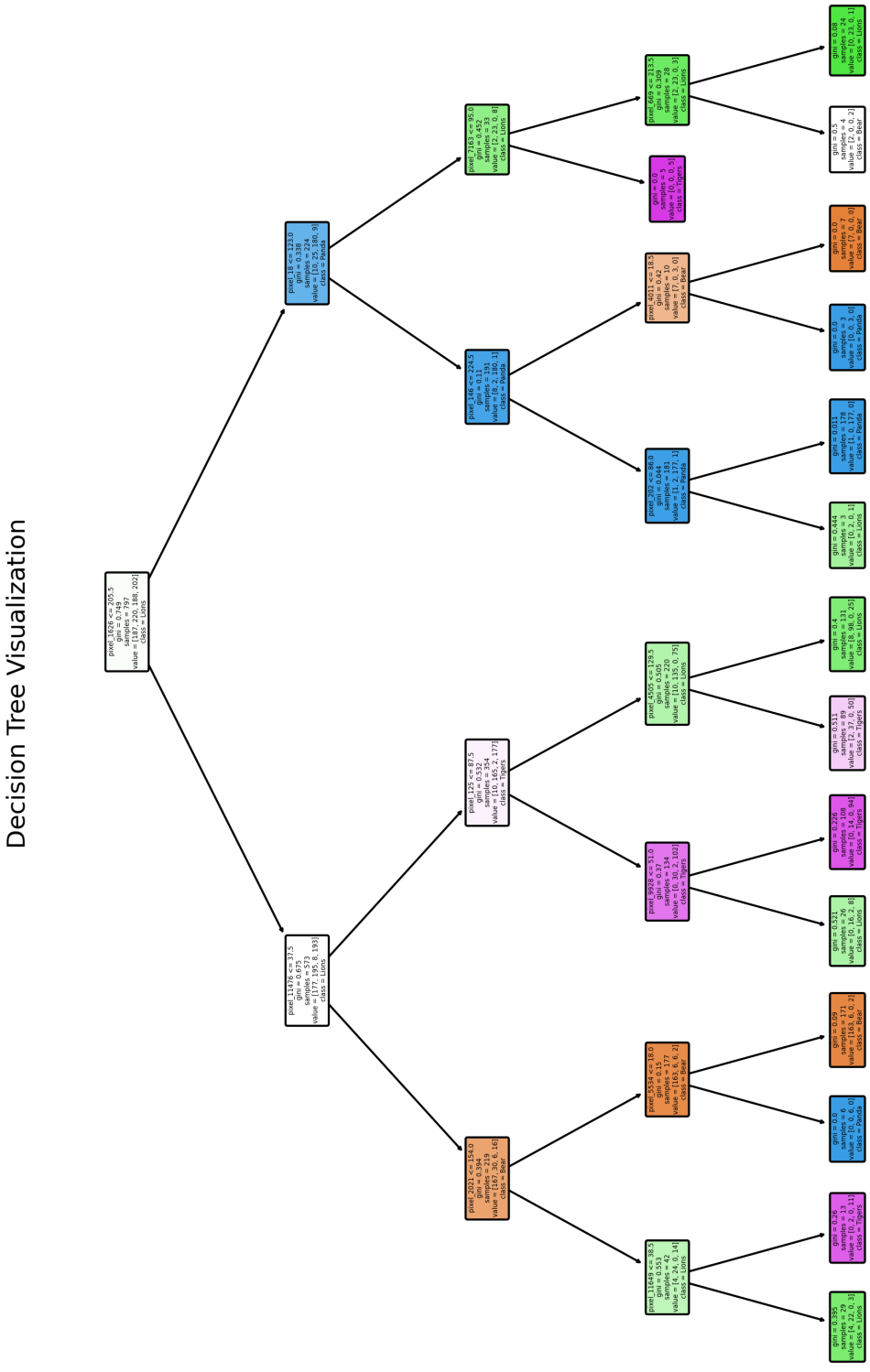

FIGURE 18. The fitted decision tree in Section 4.2. The depth is 4. At each node, the following information are given: the chosen pixel and the optimal threshold, the Gini impurity at this node; the sample size, the sample size of each class, and the category of the major samples from the array at the 4th line.