**National College of Computer Studies**

**Tribhuvan University**

**Institute of Science and Technology (IoST)**



Project Report On

**Animal Species Classification**

**Submitted To:**

National College of Computer Studies

Department of Bachelor of Science in Computer Science and Information Technology (BSc.CSIT)

**In partial fulfillment of the requirement for the degree of Bachelor of Science in Computer Science and Information Technology (BSc.CSIT)**

**Submitted By:**

Atullya Maharjan

(BSc.CSIT 6th Semester)

# **Abstract**

This project focuses on using deep learning with Convolutional Neural Networks (CNNs) to automatically identify and classify different animal species from images. The model is trained on a collection of labeled animal pictures and uses techniques like data augmentation, batch normalization, and dropout to improve its performance and avoid overfitting. The CNN learns to recognize important features in the images, starting from simple shapes and moving to more complex details. To measure how well the model classifies the animals, we use performance metrics such as accuracy, precision, recall, and F1-score. The results show that the model can classify animal species accurately, highlighting the potential for real-time applications in areas like wildlife research, education, and conservation efforts. This project demonstrates how effective CNNs can be for image classification and lays the groundwork for future developments in automatic animal recognition systems.

# **Acknowledgement**

I would like to express my sincere gratitude to our supervisor, Mr. Mausam Rajbanshi, for providing us with the wonderful opportunity to work on the “**Animal Species Classification**” project. His valuable guidance and support have been instrumental in our project's success, enabling us to explore new tools and technologies.

I also extend my heartfelt thanks to NCCS College for their ongoing supervision, guidance, and the essential resources that were crucial for completing this project. The assistance from the library and the staff members at NCCS has been incredibly helpful, and I appreciate their cooperation and encouragement. I wish to acknowledge everyone who contributed, directly or indirectly, to making this study possible.

Finally, I am thankful to all those who read this project and hope that it will benefit both current and future audiences.

Yours sincerely,

Atullya Maharjan

**TABLE OF CONTENT**

Contents

[**Abstract** i](#_Toc191063151)

[**Acknowledgement** ii](#_Toc191063152)

[**1. Introduction** 1](#_Toc191063153)

[**2. Objective** 2](#_Toc191063154)

[**3. Methodology** 3](#_Toc191063155)

[**3.1. Data** 3](#_Toc191063156)

[**3.1.1. Data Collection** 3](#_Toc191063157)

[**3.1.2. Data Properties** 3](#_Toc191063158)

[**3.2. Neural Network Architecture** 3](#_Toc191063159)

[**3.2.1. Block Representation** 3](#_Toc191063160)

[**3.2.2. Layer Explanation** 5](#_Toc191063161)

[**3.2.3. Input and Output** 7](#_Toc191063162)

[**3.3. Activation Functions** 7](#_Toc191063163)

[**3.4 Loss Function** 8](#_Toc191063164)

[**3.4.1. Importance** 8](#_Toc191063165)

[**3.4.2. Function Used** 8](#_Toc191063166)

[**3.5. Forward Propagation** 9](#_Toc191063167)

[**3.6. Backpropagation** 9](#_Toc191063168)

[**4. Testing** 10](#_Toc191063169)

[**5. Results** 11](#_Toc191063170)

[**6. Conclusion** 13](#_Toc191063171)

[**7. References** 14](#_Toc191063172)

# **1. Introduction**

Animal species classification involves identifying and categorizing different animal species, such as zebra, tiger, and elephants, from images. Using deep learning techniques like Convolutional Neural Networks (CNNs), this approach enables models to automatically learn and recognize patterns, textures, and features unique to various species. By training on large datasets of labeled animal images, these models achieve high accuracy and efficiency in classification tasks. This technology is especially valuable for applications in wildlife conservation, ecological monitoring, and research, where automated image analysis can save time and improve precision. Deep learning-based animal classification systems demonstrate significant potential for advancing biodiversity studies and real-time species identification.

# **2. Objective**

The objectives of this project are

* Create a neural network model that can automatically identify and classify images of different animal species.
* Use deep learning methods, especially Convolutional Neural Networks (CNNs), to build the model.
* Improve the model's accuracy by fine-tuning its settings and boosting its performance.
* Enhance the ability to detect important features that help distinguish between different animal species.

# **3. Methodology**

## **3.1. Data**

### **3.1.1. Data Collection**

The dataset is made up of images organized into folders, with each folder representing a different animal species. Each folder contains images that belong to a specific species. The images were gathered from various sources, including open datasets like Kaggle.

### **3.1.2. Data Properties**

* **Total images**: Extracted from the dataset and consists of Elephant, Tiger, Zebra, and Panda dataset. Each dataset consists of 100 images each.
* **Image dimensions**: Varies, resized to a standard size 180X180 pixels for consistency.
* **Color space**: RGB (3 channels) to retain color information.
* **Number of categories: 4 animal classes** – Elephant, Tiger, Zebra, Panda
* **Augmentation Techniques**: Rotation (30°), horizontal flipping, width & height shifting (20%), shearing, zooming (20%), and nearest fill mode to enhance model generalization.

## **3.2. Neural Network Architecture**

### **3.2.1. Block Representation**

* **Input Layer**: The input layer receives the image data, typically with dimensions of **150×150 pixels**. Since image data can come in various formats (RGB, grayscale, etc.), the input size is standardized to 150×150 pixels with 3 color channels (for RGB). This ensures that all input images have consistent dimensions, which is crucial for the neural network to process them properly. The images are resized as needed, and pixel values are normalized to a range between 0 and 1 using rescale=1./255 to improve model convergence during training.
* **Convolutional Layers**: Convolutional layers extract features like edges, textures, and patterns from images. The first layer detects simple features, while deeper layers capture complex shapes and objects. In this model, the first **Conv2D layer** applies **32 filters (3×3)** to identify basic patterns, followed by a second **Conv2D layer** with **64 filters** for more detailed features. **MaxPooling layers (2×2)** reduce the feature map size while preserving essential information.
* Example feature map sizes:
* After first Conv2D: (None, 150, 150, 32)
* After second Conv2D**:** (None, 74, 74, 64) (after pooling)
* **Batch Normalization**: Batch normalization improves training stability and speed by normalizing activations within each layer. It maintains a mean of 0 and variance of 1 at the mini-batch level, reducing internal covariate shifts and leading to faster convergence. Additionally, it acts as a regularizer, potentially reducing the reliance on dropout. By keeping activations stable, batch normalization enhances the model’s ability to generalize effectively.
* **MaxPooling Layers**: MaxPooling reduces the spatial dimensions of feature maps by selecting the maximum value from each region, retaining the most important features. A 2x2 pooling operation reduces the feature map size by half, lowering computational cost and improving generalization. It also enhances translation invariance, making the model more robust to small shifts in the image. Eg: After the first MaxPooling2D layer: (None, 90, 90, 16)
* **Dropout Layers**: Dropout is a regularization technique that prevents overfitting by randomly deactivating a fraction of neurons during training. This forces the model to learn more robust and generalized features. The dropout rate (e.g., 0.5) determines the proportion of neurons dropped. In this model, dropout is applied after the third MaxPooling layer, before the fully connected layers, enhancing generalization and reducing overfitting.
* **Fully Connected Layers (Dense)**: Fully connected layers process the extracted features from the convolutional and pooling layers for final classification. The Flatten layer converts the 2D feature maps into a 1D vector, which is then passed to a Dense layer with 128 neurons, each capturing key patterns from the input images. This layer applies a ReLU activation function to introduce non-linearity. Finally, the Softmax activation function in the output layer assigns probabilities to each class, allowing the model to classify the images accurately.
* **Output Layer**: The output layer is responsible for making the final classification decision. It consists of a Dense layer with neurons equal to the number of classes, each representing a category. The Softmax activation function is applied to convert the logits into probability scores, ensuring that the sum of all class probabilities equals 1. This allows the model to assign a confidence score to each class, making predictions more interpretable and suitable for multi-class classification.

### **3.2.2. Layer Explanation**

* **Sequential Layer:** The output shape is (None, 150, 150, 3). The Sequential layer acts as a container for the model, defining the input image size of 150x150 pixels with 3 color channels (RGB). The None represents the variable batch size. This layer holds the sequence of layers that follow but does not perform operations itself..
* **Rescaling Layer:** The output shape is (None, 150, 150, 3). This layer normalizes the pixel values by scaling them between 0 and 1, dividing all values by 255. It helps improve training stability and convergence by reducing sensitivity to large input variations.
* **Conv2D (First Convolutional Layer):** The output shape is (None, 150, 150, 32). The param is 896. This layer applies 32 convolutional filters (3×3) to extract features like edges and textures from the input image. Each filter processes all three RGB channels, generating a set of feature maps. The number of parameters (896) includes weights and biases for each filter.
* **MaxPooling2D (First Max Pooling Layer):** The output shape is (None, 75, 75, 32). The params is 0. This layer applies max pooling with a 2x2 filter to the feature maps generated by the first convolutional layer. It reduces the spatial dimensions from 150x150 to 75x75 while keeping the 32 feature maps unchanged. This operation helps in downsampling, reducing computational complexity, and retaining the most important features for further processing.
* **MaxPooling2D (Second Max Pooling Layer):** The output shape is (None, 37, 37, 64). The params is 0. This layer applies a 2x2 pooling operation to downsample the feature map from the second convolutional layer, reducing its dimensions from 75x75 to 37x37. By selecting the maximum value in each 2x2 region, it preserves essential spatial features while reducing computational complexity and improving model efficiency.
* **Conv2D (Third Convolutional Layer):** The output shape is (None, 35, 35, 128). The params is 73,856. This layer applies 128 convolutional filters to the output of the second max-pooling layer, allowing the model to learn more detailed patterns and textures. Each filter extracts different features, increasing the depth of the feature map. The number of parameters (73,856) is calculated based on the kernel size (3x3), the number of input channels (64), and the biases.
* **MaxPooling2D (Third Max Pooling Layer):** The output shape is None, 22, 22, 64)**.** The params is 0.This max-pooling layer reduces the size of the feature map generated by the third convolutional layer. Using a 2x2 pooling window, it decreases the spatial dimensions from 45x45 to 22x22 while maintaining the important features. The depth remains 64.
* **Dropout Layer:** The output shape is (None, 512). The params is 0. This layer randomly drops a fraction of neurons during training to prevent overfitting. By disabling certain neurons, it forces the model to generalize better rather than relying too much on specific patterns. Since Dropout does not have trainable parameters, the number of parameters remains 0. The output shape remains the same as the previous Dense layer, but some neurons are set to zero during training.
* **Flatten Layer:** The output shape is (None, 30976). The params is 0. This layer converts the 3D feature map (22x22x64) into a 1D vector with 30,976 values. This transformation allows the data to be fed into fully connected layers for classification. Since it only reshapes the data without learning any parameters, the number of parameters remains 0.
* **Dense (Fully Connected Layer):** The output shape is (None, 128). The params is 3,965,056. This fully connected layer takes the flattened 1D vector (30,976 values) and connects it to 128 neurons. Each neuron has a weight for every input value, along with a bias term, leading to a total of 3,965,056 trainable parameters. This layer helps the model learn high-level representations and combine extracted features for classification.
* **Dense (Output Layer):** The params is 645. This final dense layer consists of 5 neurons, each representing one class in the classification task. It applies a softmax activation function to generate a probability distribution over the 5 classes. The total parameters (645) come from the connections between the 128 neurons in the previous layer and the 5 output neurons, including the bias terms.

### **3.2.3. Input and Output**

* **Input**: The input to the model consists of resized animal images, which are fed into the Convolutional Neural Network (CNN) after undergoing preprocessing. Each image is resized to a consistent dimension of 150x150 pixels to ensure uniformity across all input data. The images are also normalized (scaled to a range of 0 to 1) by dividing the pixel values (originally ranging from 0 to 255) by 255. This normalization improves the efficiency and stability of the training process, allowing the neural network to learn features more effectively. Additionally, data augmentation techniques such as rotation, shifting, zooming, and flipping are applied to the training images to enhance model generalization.
* **Output**: The output of the model is a class prediction for the input image. The final Dense (Output) Layer consists of 4 neurons, each corresponding to one of the four animal categories (Tiger, Panda, Zebra, and Elephant). The softmax activation function produces a probability distribution, where each value represents the likelihood of the image belonging to a particular class. The model selects the class with the highest probability as the final classification result.

## **3.3. Activation Functions**

* **ReLU (Rectified Linear Unit)**: The ReLU activation function is used in the hidden layers of the Convolutional Neural Network (CNN). It introduces non-linearity by returning the input value directly if it is positive and zero otherwise. This function helps the model learn complex patterns and ensures that negative values do not pass forward, improving the efficiency of training.

In the given model, ReLU is applied in the Conv2D and Dense (hidden) layers, enabling the network to learn better representations of the input images. One key advantage of ReLU is that it mitigates the vanishing gradient problem, which often occurs with older activation functions like sigmoid and tanh. By keeping gradients either 0 or 1, ReLU helps the model train faster and more effectively, especially in deep networks

* **Softmax**: The softmax activation function is used in the final output layer of the model, which consists of 4 neurons, each representing one of the animal classes (Tiger, Panda, Zebra, and Elephant). Softmax converts the raw output values (logits) from the last Dense layer into a probability distribution The function works by computing the exponential of each logit and then normalizing the values so that their sum equals 1.

## **3.4 Loss Function**

### **3.4.1. Importance**

A loss function is a fundamental component of the animal classification model, as it measures the difference between the predicted class and the actual class labels. The goal of the model is to minimize this loss to improve its classification accuracy over time.

In this model, categorical cross-entropy is used as the loss function because the problem involves multi-class classification (i.e., classifying images into one of four animal categories).

### **3.4.2. Function Used**

* **Categorical Crossentropy**: The Categorical Crossentropy loss function is used for this multi-class classification model, as it effectively measures the difference between the actual one-hot encoded labels and the predicted probability distribution. Paired with the softmax activation, it helps the model assign higher probabilities to the correct classes. The Adam optimizer updates the weights based on the computed loss, improving classification accuracy over time.

## **3.5. Forward Propagation**

In forward propagation, the input image passes through multiple layers of the neural network, where each layer extracts and refines features. Convolutional layers detect patterns like edges and textures, activation functions introduce non-linearity, and pooling layers reduce dimensionality. As the data flows through the network, learned weights transform the image into meaningful feature representations. Finally, the softmax activation in the output layer produces a probability distribution, with the highest probability indicating the predicted animal class.

## **3.6. Backpropagation**

Backpropagation is the process of refining the model’s weights to minimize prediction errors. After forward propagation, the loss function calculates the error between predicted and actual outputs. The model then computes gradients—measuring how each weight affects the loss—and updates them using an optimizer like gradient descent. By iteratively adjusting the weights, the model gradually improves its accuracy. This learning process continues until the loss reaches a minimal value, ensuring the network effectively classifies images.

# **4. Testing**

The trained model is evaluated on a separate test dataset to measure its performance. The evaluation includes may metrics such as:

* **Accuracy**: Accuracy is a widely used metric that shows how well the model performs by calculating the percentage of images correctly classified. It represents the overall correctness of the model's predictions.
* **Precision**: Precision, also called positive predictive value, measures how many of the model’s positive predictions are actually correct. It indicates the accuracy of the model’s positive classifications.
* **Recall**: Recall, also called sensitivity or true positive rate, measures how effectively the model detects all relevant positive cases. It is especially important when missing positive instances is more costly than having false positives, such as in medical diagnoses.
* **F1-score**: The F1-score is the harmonic mean of precision and recall, offering a balanced evaluation of both metrics. It helps ensure that the model performs well in cases where both precision and recall are important.

# **5. Results**

The model's performance is analyzed using:

* **Confusion Matrix**: The confusion matrix is a key tool for evaluating the classification model's performance. It provides a breakdown of true positives, false positives, true negatives, and false negatives for each class, offering insights into the model’s accuracy. By analyzing misclassified instances, we can identify patterns of confusion between similar categories and refine the model accordingly. This helps in diagnosing weaknesses and improving overall classification performance.
* **Accuracy Graphs**: Accuracy graphs track the model's performance during training and validation across multiple epochs. They help analyze how well the model is learning and whether it is generalizing to new data. If training accuracy is much higher than validation accuracy, it suggests overfitting, meaning the model memorizes the training data instead of learning meaningful patterns. If both training and validation accuracy are low, the model may be underfitting, indicating it is too simple to capture the underlying structure of the data. These graphs provide valuable insights for improving model performance.
* **Loss Graphs**: Loss graphs show how the model's error decreases over time during training. A steadily decreasing loss indicates effective learning, while a stagnant or increasing loss may signal issues like a poor learning rate or difficulty in learning patterns. Monitoring both training and validation loss helps detect overfitting—if training loss decreases but validation loss rises, the model may be memorizing the training data instead of generalizing to new data. These graphs are essential for evaluating model performance and making necessary adjustments.
* **Experimentation with Hyperparameters**: Adjusting hyperparameters like learning rate, batch size, and optimizer choice plays a crucial role in improving model performance. The learning rate determines how quickly the model updates its weights, while batch size affects how much data is processed at a time. Choosing the right optimizer, such as Adam or SGD, impacts how efficiently the model learns. Experimenting with different combinations helps find the best settings to maximize accuracy and reduce overfitting or underfitting. Monitoring performance through metrics and graphs ensures effective tuning.

# **6. Conclusion**

This project successfully implements a Convolutional Neural Network (CNN) for animal species classification, demonstrating its effectiveness in image recognition tasks. Data augmentation enhanced model robustness by making it more adaptable to variations in animal images, while batch normalization stabilized training and dropout reduced overfitting, improving generalization. Hyperparameter tuning, including adjustments to the learning rate and optimizer, further optimized model performance. Evaluation metrics such as accuracy, precision, recall, and F1-score confirm the model’s reliability in accurately classifying different animal species. This project showcases the potential of CNNs for species identification and can be extended to broader wildlife recognition applications.

# **7. References**

* <https://www.kaggle.com/datasets/utkarshsaxenadn/animal-image-classification-dataset>
* <https://github.com/hocuf/Animal-Species-Recognition--Classification-Project/blob/main/.ipynb_checkpoints/animal-species-recognition-classification-checkpoint.ipynb>
* <https://chatgpt.com/>
* https://www.youtube.com/watch?v=rZwq7J2QETw