KYUSHU

UNIVERSITY

PROJECT REPORT

BUTTERFLY & MOTH CLASSIFICATION

Student: Anh Minh Tran

Supervisor: Assoc. Prof. Ta Viet Ton

Fukuoka - 2024

KYUSHU UNIVERSITY

PROJECT REPORT

BUTTERFLY & MOTH CLASSIFICATION

Student: Tran Anh Minh
Supervisor: Assoc. Prof. Ta Viet Ton

Fukuoka - 2024

PREFACE

This project represents a focused effort to apply advanced convolutional neural networks (CNNs) to the classification of butterfly and moth species based on their physical features.

Butterflies and moths are essential to our ecosystems, serving as indicators of environmental health and biodiversity. Traditional classification techniques often involve time-consuming manual processes that can be prone to inaccuracies. To overcome these limitations, this project employs the Xception model, a cutting-edge CNN architecture that excels in feature extraction and classification tasks.

The Xception model stands out due to its use of depthwise separable convolutions, which significantly enhance computational efficiency and model performance. By separating convolution operations into depthwise and pointwise convolutions, Xception reduces the number of parameters and computational complexity while maintaining high accuracy. This architectural innovation allows the model to capture fine-grained details and complex patterns in butterfly and moth images more effectively than traditional CNNs.

In this project, we preprocess a butterfly image dataset from ***Kaggle*** and apply the Xception model for training and evaluation. The model's advanced feature extraction capabilities make it particularly well-suited for the nuanced task of insect classification, enabling more accurate and efficient identification of different species.

I appreciate your understanding as this project is part of an ongoing learning process, and I welcome any feedback on its execution.

My sincere thanks go to Professor Ta Viet Ton for his steadfast support and guidance throughout this endeavor.

Sincerely,

Minh

Fukuoka, August 20, 2024

Table of Content

Chapter 1

DATA INFORMATION

1.1 Introduction to Data

The dataset titled ”Butterfly & Moths Image Classification 100 species” comprises a comprehensive collection of images for classifying butterfly and moth species. This dataset includes a total of 13,594 images, meticulously categorized into 100 distinct species.

- **Resolution:** All images are uniformly sized at **224 x 224 pixels**, ensuring consistency in input dimensions for model training.
- **Channels:** The images are in **RGB format** with **3 color channels**, capturing the vibrant hues and patterns of butterfly and moth species.
- **Format:** Each image is stored in **JPEG format**, a widely used format for image compression and quality preservation.

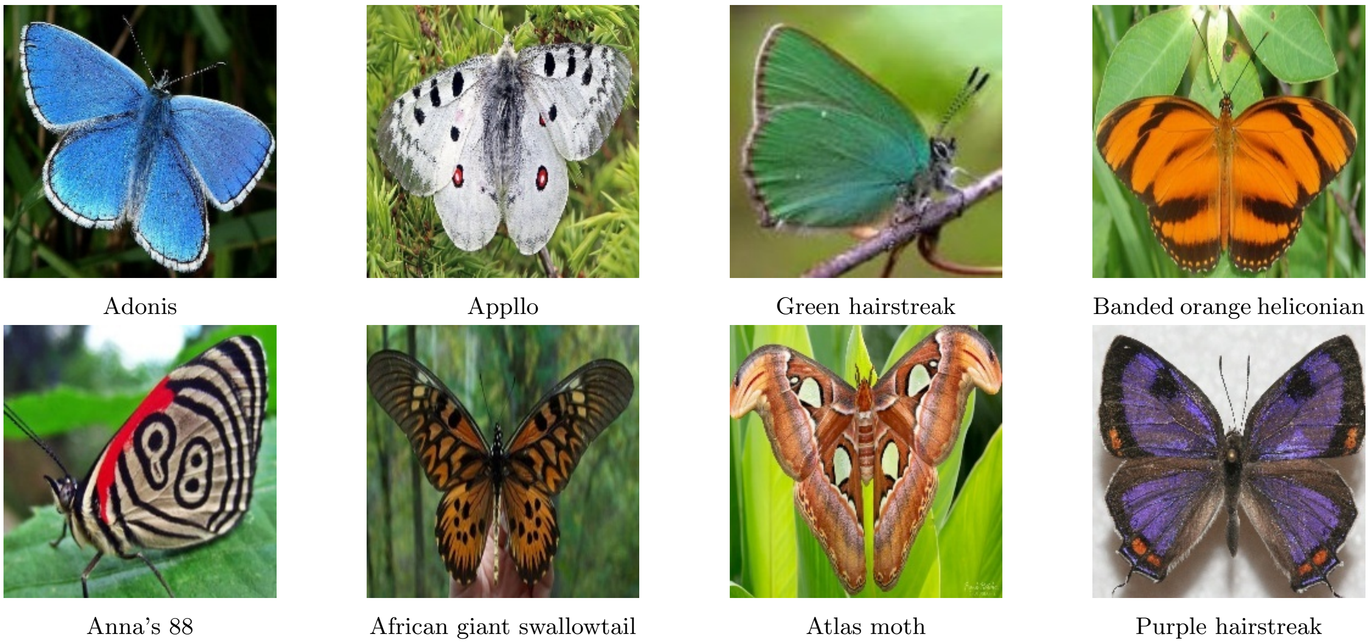

Figure 1.1: Illustrative examples of various species included in the dataset

1.2 Dataset breakdown

- **Training Set:** Contains **12,594 images** for model training and learning species characteristics.
- **Evaluation Set:** Includes **1,000 images** for tuning hyperparameters and assessing model performance.

Chapter 2

XCEPTION MODEL

2.1 Introduction

The Xception model, introduced by François Chollet in 2016, stands for "Extreme Inception." It builds upon the Inception architecture originally designed for efficient image classification tasks. The core innovation of Xception lies in its use of depthwise separable convolutions, which replace the standard Inception modules. This modification leads to a more efficient model with fewer parameters and improved performance on various image classification benchmarks.

2.2 Types of Component Layers in the Xception Model

The Xception model comprises several key layers, each serving a specific purpose in the network's architecture. It begins with convolutional layers that capture basic features from the input image by applying standard convolutions to reduce spatial dimensions and prepare the data for deeper processing. The model's core consists of depthwise separable convolution layers, which perform spatial convolutions independently across each channel (depthwise) and then combine the outputs through pointwise convolution. This approach enhances computational efficiency while maintaining high accuracy. Residual connections are integrated throughout the network to facilitate gradient flow and enable the learning of deeper representations. Pooling layers, including max pooling and global average pooling, are used to downsample feature maps, reducing their spatial dimensions while retaining essential information. Finally, the fully connected layer at the network's end consolidates all the extracted features and outputs the final classification decision. This layered structure allows Xception to balance accuracy and computational complexity.

2.2.1 Convolutional layer

A convolutional layer is a fundamental building block of Convolutional Neural Networks (CNNs). It is designed to automatically and adaptively learn spatial hierarchies of features from input images. This layer primarily involves convolution operation, followed by an optional activation function, and sometimes includes additional operations such as batch normalization and pooling.

Convolution Operation

The core operation in a convolutional layer is the convolution itself. This process involves sliding a small matrix, known as a **filter** or **kernel**, over the input data (usually an image) to produce a feature map. Here's how it works:

- **Input**: The input to a convolutional layer is usually a three-dimensional tensor of size $H \times W \times C$, where H is the height, W is the width, and C is the number of channels (e.g., 3 for an RGB image).

- **Filter/Kernel**: A filter is a small matrix of weights of size $k \times k \times C$, where k is the filter size (e.g., 3×3 or 5×5). Each filter has a corresponding set of learnable parameters.
- **Convolution**: The filter slides (or "convolves") over the input image, performing element-wise multiplication between the filter weights and the input values covered by the filter. The sum of these multiplications forms a single value in the output feature map. The filter then moves to the next position, repeating the process until the entire image has been covered.

  The convolution operation with bias can be described mathematically as follows:

  $$O_f(i,j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} \sum_{c=0}^{C-1} I(i+m, j+n, c) \cdot K_f(m,n,c) + b_f$$

  where:

  - $O_f(i,j)$ is the output value at position (i,j) in the feature map for filter f.
  - $I(i+m, j+n, c)$ is the input value at position $(i+m, j+n)$ in channel c.
  - $K_f(m,n,c)$ is the filter value at position (m,n) in channel c for filter f.
  - b_f is the bias term associated with filter f.
- **Stride and Padding**:
  - **Stride** refers to the step size with which the filter moves across the input. A stride of 1 means the filter moves one pixel at a time, while a stride of 2 means it moves two pixels at a time, reducing the spatial dimensions of the output.
  - **Padding** is the process of adding extra pixels around the border of the input, typically zeros, to control the spatial size of the output. The padding ensures the filter can cover all parts of the image, especially the edges.

Activation Function

After the convolution operation, the output feature map is typically passed through an activation function, most commonly the **ReLU (Rectified Linear Unit)**, defined as:

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

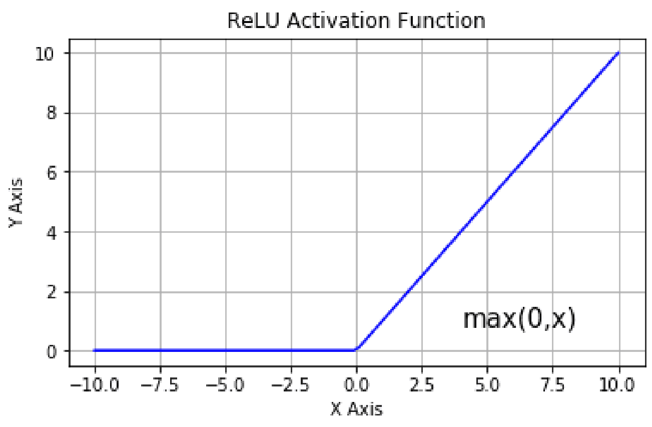

Figure 2.1: Graph of ReLu function

This activation function introduces non-linearity into the model, allowing it to learn more complex patterns. The ReLU function sets all negative values in the feature map to zero, retaining only the positive values.

Output Feature Map

The final output of the convolutional layer is a set of feature maps, one for each filter used. If the layer has F filters, the output will have dimensions $H' \times W' \times F$, where H' and W' are the height and width of the output feature map, depending on the stride and padding.

Feedforward Process

- **Input to the Layer**: The input to the convolutional layer is the output from the previous layer or the original image.
- **Filter Application**: Each filter is applied across the entire input.
- **Feature Map Generation**: The result of each convolution is a feature map, representing learned features such as edges, textures, or more complex structures in deeper layers.
- **Activation**: The feature map is passed through an activation function, adding non-linearity.
- **Output**: The processed feature maps are then passed to the next layer in the network.

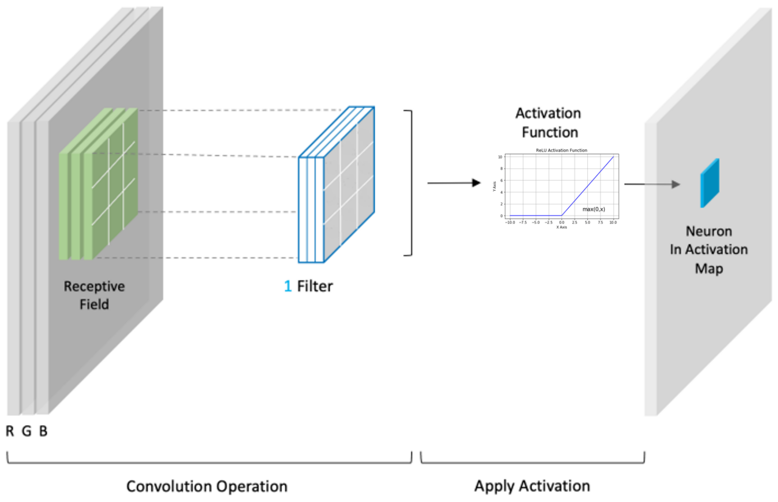

Figure 2.2: Feedforward Process of convolutional layer

By applying multiple filters across the input, the layer captures various features at different spatial locations. The convolution operation and activation functions enable the network to learn hierarchical patterns, from simple edges in early layers to complex shapes in deeper layers. This capability makes convolutional layers the cornerstone of modern computer vision applications.

2.2.2 Depthwise Separable Convolutional layer

Depthwise separable convolution is a variation of the traditional convolution operation, designed to reduce computational complexity while preserving the model's ability to capture essential features. It separates the spatial and channel-wise operations, making them more efficient without sacrificing performance.

Depthwise Convolution

The first step in separable convolution is the depthwise convolution, where a single filter is applied independently to each input channel. Unlike standard convolution, which combines all input channels to produce a single output, depthwise convolution maintains the spatial relationships within each channel. Here's how it works:

- **Input**: The input is a tensor of size $H \times W \times C$, where H is the height, W is the width, and C is the number of channels.
- **Filter/Kernel**: In depthwise convolution, each input channel has its filter of size $k \times k \times 1$, where k is the filter size. Thus, there are C filters in total.
- **Convolution**: Each filter independently convolves with its corresponding input channel, producing a feature map of size $H' \times W' \times 1$. The output of the depthwise convolution is a set of feature maps, one for each input channel, with the same number of channels as the input.

  Mathematically, for each channel c, the output is calculated as:

  $$O_c(i,j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} I_c(i+m, j+n) \cdot K_c(m,n) + b_c$$

  Where:

  - $I_c(i+m, j+n)$ is the input value at position $(i+m, j+n)$ in channel c.
  - $K_c(m,n)$ is the filter value at position (m,n) for channel c.
  - b_c is the bias term associated with the filter for channel c.

The result of this operation is a set of C feature maps, each of size $H' \times W'$.

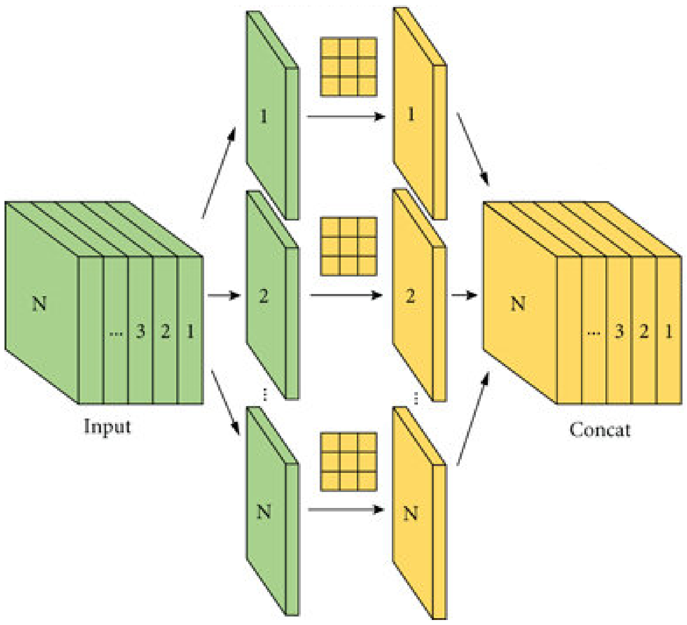

Figure 2.3: Depthwise convolution

Pointwise Convolution

Following the depthwise convolution, a pointwise convolution is performed. This operation uses a 1×1 filter to combine the feature maps produced by the depthwise convolution across channels:

- **Filter/Kernel**: The pointwise filter has a size of $1 \times 1 \times C$, where C is the number of input channels. The filter applies to the entire depth (all channels) at each spatial location.
- **Convolution**: For each spatial position (i, j), the pointwise convolution combines the values from all input channels by performing a weighted sum:

$$O(i,j) = \sum_{c=0}^{C-1} F_c(i,j) \cdot W_c + b$$

  Where:

  - $F_c(i,j)$ is the feature map value at position (i, j) from channel c.
  - W_c is the weight associated with channel c in the 1×1 filter.
  - b is the bias term added after the weighted sum.
- **Output**: The result is a new set of feature maps, typically with a different number of channels, F, determined by the number of filters used in the pointwise convolution.

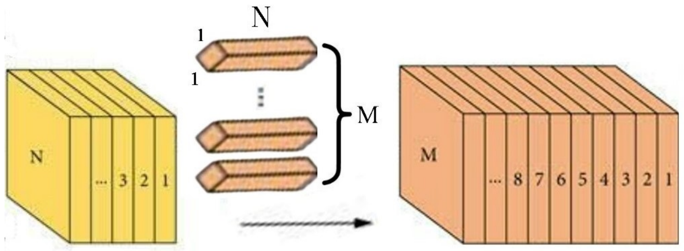

Figure 2.4: Pointwise convolution

Efficiency and Effectiveness

The combination of depthwise and pointwise convolutions makes separable convolution highly efficient. The model significantly reduces the number of parameters and computational costs by decomposing the standard convolution into these two operations. For instance, in a standard convolution with an input of size $H \times W \times C$ and output of size $H' \times W' \times F$, the number of multiplications is $k^2 \times C \times F \times H' \times W'$. In separable convolution, the operations are split into $k^2 \times C \times H' \times W'$ for the depthwise convolution and $C \times F \times H' \times W'$ for the pointwise convolution, leading to a substantial reduction in complexity.

Feedforward Process

- **Input to the Layer**: The input is the feature maps from the previous layer.
- **Depthwise Convolution**: Each filter convolves with its respective input channel independently, preserving spatial features within the channel.
- **Pointwise Convolution**: A 1×1 convolution integrates the output from the depthwise convolution across channels, combining them into a new set of feature maps.
- **Activation**: The output feature maps typically pass through an activation function for non-linearity.
- **Output**: The resulting feature maps are forwarded to the next layer, which can be another separable convolution or a different type of layer.

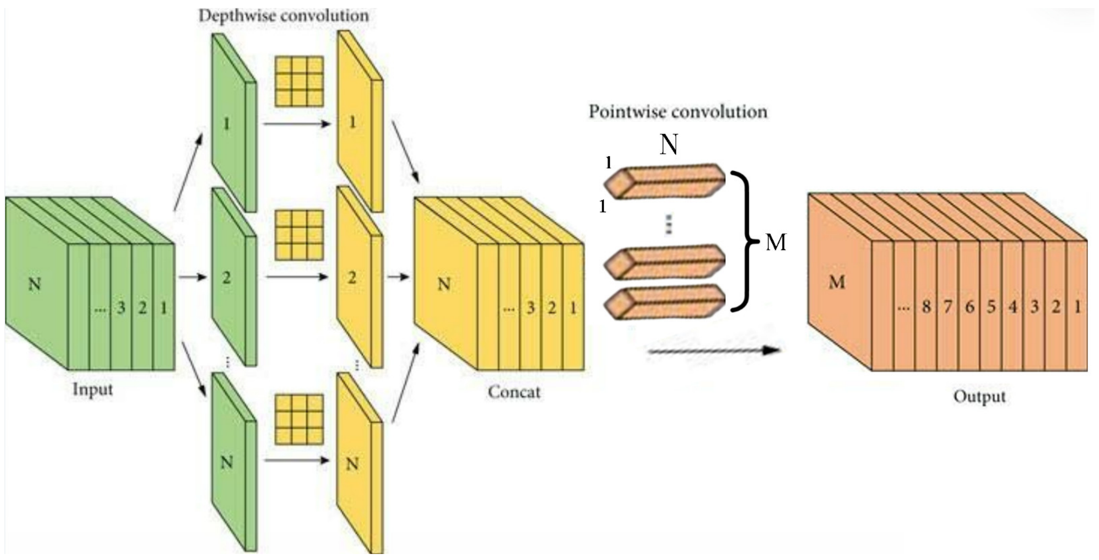

Figure 2.5: Feedforward Process of separable convolutional layer

Separable convolution's ability to efficiently capture and combine spatial and channel-wise information makes it a powerful tool in deep learning, particularly in models like Xception that require both high performance and computational efficiency.

2.2.3 Pooling Layer

The pooling layer is a crucial component of Convolutional Neural Networks (CNNs), designed to reduce the spatial dimensions (height and width) of the input feature maps while preserving important information. This reduction helps decrease the computational load, control overfitting, and extract dominant features that are invariant to small transformations in the input.

Types of Pooling

There are several types of pooling operations, with the most common being Max Pooling and Average Pooling:

Max Pooling:

Max pooling is the most widely used pooling operation. It works by sliding a window over the input feature map and recording the maximum value within the window. The result is a downsampled feature map where only the most significant features are retained.

- **Input**: The input to a max pooling layer is a feature map of size $H \times W \times C$, where H is the height, W is the width, and C is the number of channels.
- **Pooling Window**: The pooling operation uses a window of size $p \times p$, where p is typically 2 or 3. The window slides across the feature map with a certain stride, selecting the maximum value within the window.
- **Stride**: The stride determines the step size of the window as it moves across the feature map. A stride of 2, for instance, halves the dimensions of the output feature map.
- **Output**: The output of max pooling is a downsampled feature map of size $H' \times W' \times C$, where:

$$H' = \frac{H - p}{\text{stride}} + 1, \quad W' = \frac{W - p}{\text{stride}} + 1$$

Mathematically, for a given region in the input feature map, the output value at position (i, j) is calculated as:

$$O(i, j) = \max(\{I(i + m, j + n) \mid 0 \leq m, n < p\})$$

Where:

– $I(i + m, j + n)$ are the values in the pooling window.

– $O(i, j)$ is the output value at position (i, j).

Average Pooling:

Average pooling works similarly to max pooling, but instead of taking the maximum value, it calculates the average of all the values within the pooling window. This operation is less aggressive than max pooling and can be useful for tasks that require more nuanced information retention.

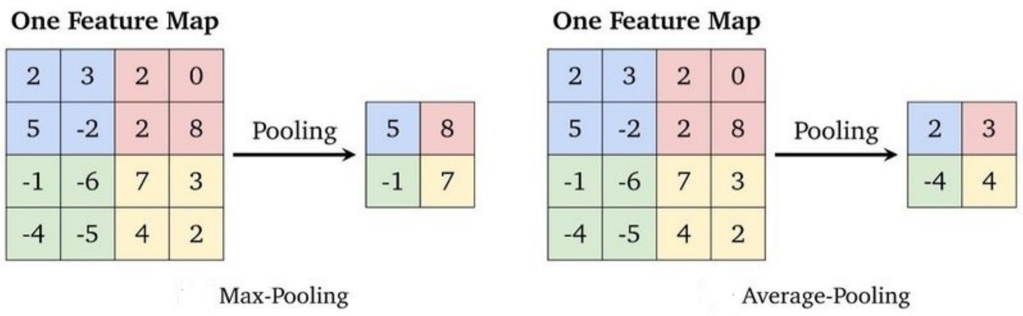

Figure 2.6: Examples of max pooling and average pooling

Benefits of Pooling Layers

Pooling layers offer several key benefits:

- **Dimensionality Reduction**: By reducing the size of the feature map, pooling layers lower the computational requirements for subsequent layers.
- **Translation Invariance**: Pooling introduces a degree of invariance to small translations in the input, making the model more robust to shifts and distortions.
- **Overfitting Control**: Pooling reduces the number of parameters in the network, helping to prevent overfitting, especially when training on small datasets.

2.2.4 Fully Connected Layer

The fully connected layer, also known as a dense layer, is a crucial component in neural networks. Unlike convolutional layers, where connections are local, each neuron in a fully connected layer is connected to every neuron in the previous layer. This enables the model to combine features learned from all previous layers and make final predictions.

- **Input**: The input to the fully connected layer is a flattened vector derived from the output of the preceding layer. If the previous layer outputs a multidimensional feature map, it is flattened into a one-dimensional vector.
- **Weights and Biases**: Each neuron in the fully connected layer has its own set of weights and a bias term:

$$z_j = \sum_{i=1}^{n} w_{ij} \cdot x_i + b_j$$

where:

- x_i is the i-th input value.
- w_{ij} is the weight connecting the i-th input to the j-th neuron.
- b_j is the bias term for the j-th neuron.
- z_j is the output of the j-th neuron before applying the activation function.

- **Activation Function**: After computing the weighted sum and adding the bias, an activation function is applied to introduce non-linearity. Common activation functions include ReLU, sigmoid, and softmax, depending on the task.
- **Output**: The output of the fully connected layer is a vector where each element represents a score, probability, or feature, depending on the number of neurons in the layer.

Dense Connections

The fully connected layer features dense connections, meaning each neuron receives input from all neurons in the previous layer. This allows the model to integrate all features and make decisions based on the entire input.

- **Feature Aggregation**: The fully connected layer aggregates features learned across the network, allowing the model to capture complex relationships in the data.
- **Final Decision**: In classification tasks, the final fully connected layer typically outputs class scores, which are then converted to probabilities via a softmax function.

Feedforward Process

- **Input to the Layer**: The input to the fully connected layer is the flattened output from the preceding layer, often a convolutional or pooling layer.
- **Weight Application**: The input vector is multiplied by the layer's weights, and the bias is added to compute the pre-activation output for each neuron.
- **Activation and Output**: The pre-activation output is passed through an activation function, producing the final output of the layer, which may be fed into another fully connected layer or serve as the model's final prediction.

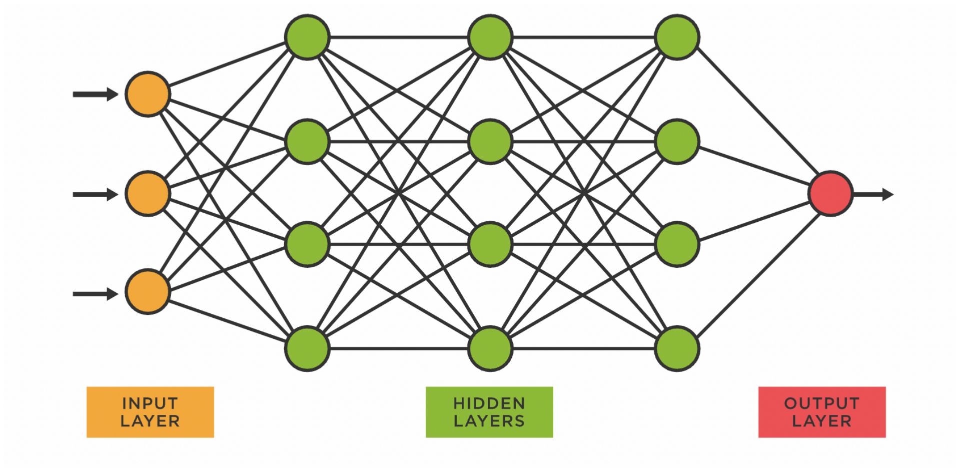

Figure 2.7: Feedforward Process of Fully Connected Layer

The fully connected layer plays a vital role in transforming the learned features into actionable predictions. Its dense connections make it capable of integrating all features and producing the final output, whether that is a class label, a probability distribution, or a regression value.

2.3 Layers of the Xception model

The Xception model comprises several layers organized sequentially, designed to capture complex patterns in images. Below is a general description of these layers:

1. **Entry Flow**

   - **Convolution Layers**: The model begins with a series of standard convolution layers (with ReLU activation and batch normalization). These layers extract low-level features such as edges and textures from the input images.
   - **First Block of Depthwise Separable Convolutions**: The entry flow contains three blocks of depthwise separable convolutions. Each block consists of a depthwise convolution followed by a pointwise convolution. These blocks are responsible for capturing increasingly complex features as the network goes deeper.
   - **Max Pooling**: After each block of depthwise separable convolutions, a max pooling layer is applied. This reduces the spatial dimensions of the feature maps, allowing the model to focus on the most relevant features while reducing computational complexity.
   - **Residual Connections**: Each block also includes residual connections that add the input of the block to its output. These connections help in maintaining gradient flow and avoid vanishing gradients during training.

2. **Middle Flow**

   - **Eight Blocks of Depthwise Separable Convolutions**: The middle flow consists of eight identical blocks of depthwise separable convolutions, each followed by batch normalization and ReLU activation. Unlike the entry flow, there are no pooling layers here. The purpose of the middle flow is to deeply refine and enhance the feature representations extracted in the entry flow. The identical blocks allow the model to capture more complex patterns over multiple layers without changing the spatial dimensions.
   - **Residual Connections**: Similar to the entry flow, each block in the middle flow also includes residual connections, helping the model to learn more effectively.

3. **Exit Flow**

   - **Two Blocks of Depthwise Separable Convolutions**: The exit flow begins with two blocks of depthwise separable convolutions, again followed by batch normalization and ReLU activation. These layers further refine the features extracted by the previous layers.
   - **Max Pooling**: A max pooling layer is applied after the depthwise separable convolutions, further reducing the spatial dimensions and preparing the feature maps for the final classification stage.
   - **Final Convolution Layer**: A final convolution layer is used to reduce the number of channels to match the number of classes in the dataset. This layer is followed by a global average pooling

layer, which reduces each feature map to a single value, resulting in a vector of predictions.

- **Fully Connected Layer**: The final fully connected (dense) layer produces the output, typically followed by a softmax activation function to yield the class probabilities.

4. Global Average Pooling

Instead of fully connected layers with many parameters, Xception uses global average pooling to reduce each feature map to a single value. This greatly reduces the number of parameters and helps prevent overfitting.

5. Softmax Layer

The final output layer is a softmax layer, which converts the output into probabilities for each class. The class with the highest probability is chosen as the prediction in a classification task.

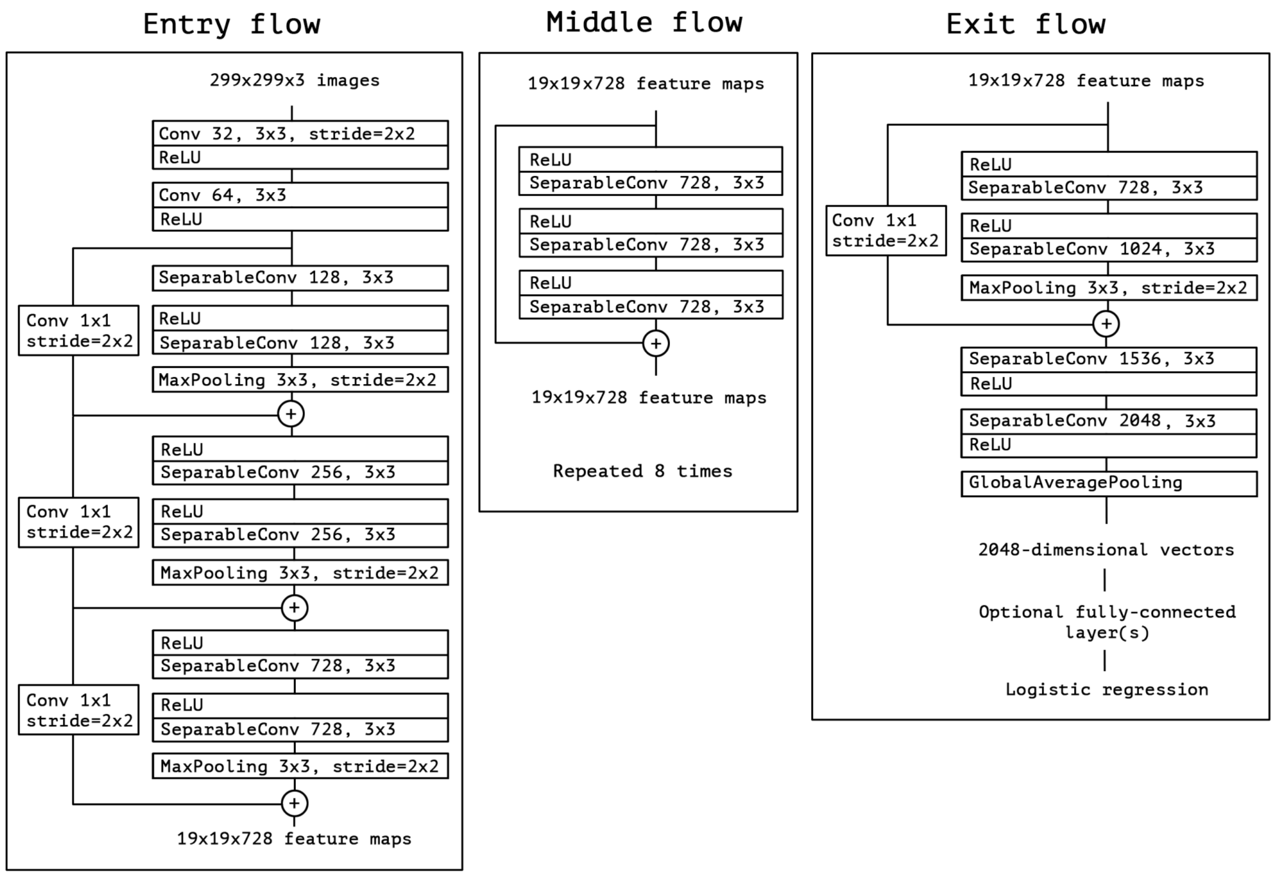

Figure 2.8: Xception model architecture

Summary

The Xception model's architecture is characterized by its use of depthwise separable convolutions, which significantly reduce the number of parameters and computational costs compared to traditional convolutional networks. The model is structured straightforwardly and modularly, with the entry, middle, and exit flows capturing progressively more abstract and high-level features from the input images. Residual connections throughout the model ensure efficient learning, making Xception a powerful and efficient architecture for image classification tasks.

Chapter 3

APPLYING THE XCEPTION MODEL FOR BUTTERFLY AND MOTH CLASSIFICATION

3.1 Model Implementation Overview

The implemented model utilizes the Xception architecture for classifying butterfly and moth images. The key components of the model are as follows:

- **Input Shape**: The model accepts images of size $(224, 224, 3)$, representing 224x224 RGB images.
- **Base Model**: Xception is used as the base model with pre-trained ImageNet weights and the top classification layer excluded (`include_top=False`).
- **Feature Extraction**: A `GlobalAveragePooling2D` layer is applied to the output of Xception, reducing the spatial dimensions to a single vector per feature map.
- **Fully Connected Layers**:
  - `Dense (1024 units, ReLU activation)`: A fully connected layer with 1024 units and ReLU activation.
  - `Dropout (rate=0.5)`: A dropout layer with a rate of 0.5 to reduce overfitting.
- **Output Layer**: A `Dense` layer with `NUM_CLASSES` units and softmax activation for class probability output.

3.2 Training the Model

Training the model involves optimizing its parameters to accurately classify images. The parameters that are learned during training include the weights and biases of the convolutional layers, the depthwise separable convolutions, and the fully connected layers. The training process is conducted in three main steps: defining the loss function, selecting an optimizer, and updating the model's weights through backpropagation.

3.2.1 Loss Function

The loss function quantifies the difference between the model's predictions and the actual labels. For classification tasks, the **categorical cross-entropy loss** is widely used. It is defined as follows:

$$L = -\sum_{i=1}^{N} y_i \cdot \log(\hat{y}_i)$$

where:

- N is the number of classes.
- y_i is the true label for class i (1 if the class is correct, 0 otherwise).
- $\hat{y}_i$ is the predicted probability for class i.

The cross-entropy loss penalizes incorrect predictions, especially when the predicted probability for the correct class is low. Minimizing this loss function guides the model toward more accurate predictions.

3.2.2 Optimizer

The **Adam optimizer** is commonly used for training deep learning models due to its efficiency and effectiveness. Adam stands for Adaptive Moment Estimation and combines the advantages of two other popular optimizers: AdaGrad and RMSProp. The Adam optimizer adapts the learning rate for each parameter individually by computing the first moment (mean) and the second moment (uncentered variance) of the gradients.

The key equations for updating the model parameters using Adam are:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

where:

- m_t and v_t are the first and second moment estimates, respectively.
- β_1 and β_2 are exponential decay rates for the moment estimates (typically 0.9 and 0.999, respectively).
- g_t is the gradient of the loss with respect to the parameters at time step t.
- α is the learning rate.
- ϵ is a small constant to prevent division by zero.

Adam's adaptive learning rates and moment estimates make it well-suited for complex models, where the learning dynamics can vary across different layers.

3.2.3 Backpropagation and Weight Update

The model's parameters include the weights and biases of the convolutional layers, the depthwise separable convolutions, and the fully connected layers. During training, the model's predictions are compared against the true labels using the cross-entropy loss function. The gradients of this loss concerning the model's parameters are calculated using backpropagation. The Adam optimizer then updates the weights based on these gradients, reducing the loss over time. This iterative process is repeated over multiple epochs until the model converges to a set of parameters that minimizes the loss function.

3.2.4 Feedforward and Feedback Loop

- **Feedforward Phase**: The input images are passed through the network, generating predictions.
- **Loss Computation**: The predictions are compared with the true labels to compute the cross-entropy loss.
- **Backpropagation Phase**: The gradients of the loss are propagated backward through the network.
- **Weight Update**: The Adam optimizer adjusts the model's parameters based on the computed gradients.

This loop continues until the loss converges, indicating that the model has learned the mapping from input images to their corresponding labels.

3.3 Results

The accuracy and loss graphs illustrate how the model's performance evolved during training. They provide insight into the learning dynamics and the effectiveness of the training process.

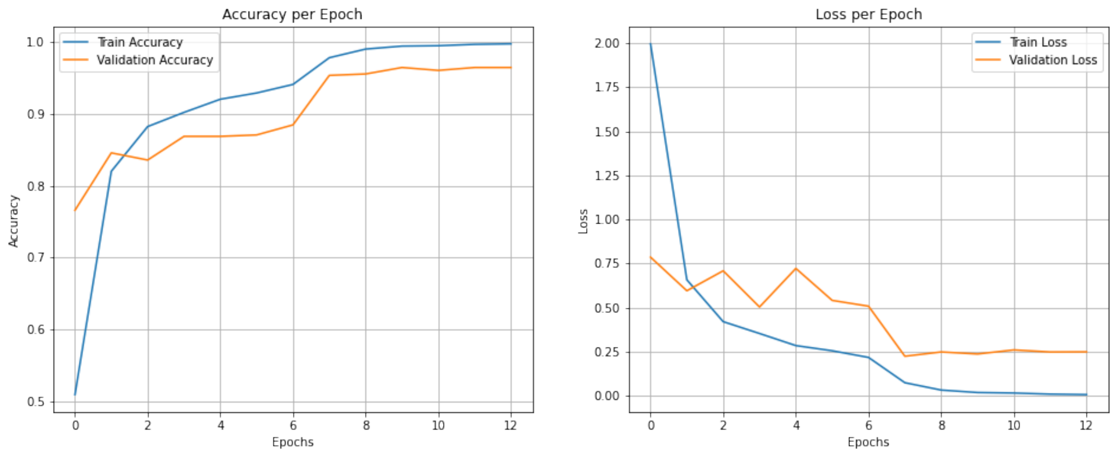

Figure 3.1: Accuracy and Loss Function Value per Epoch during Training

Key Observations:

- **Convergence**: Both the accuracy and loss graphs indicate proper convergence, with accuracy increasing and loss decreasing consistently at the last epochs.
- **Overfitting/Underfitting**: The absence of overfitting is evident from the validation accuracy remaining high and stable, without a significant gap between training and validation performance. There are no signs of underfitting, as both accuracy and loss show favorable trends.
- **Epochs to Convergence**: The model achieved optimal performance within the observed number of epochs, demonstrating efficient learning.

3.3.1 Confusion Matrix

The confusion matrix for the classification of 100 butterfly species shows strong performance. Most of the diagonal elements are 10, indicating perfect classification for those species. The largest off-diagonal value is 3, which means that in the worst case, only 3 images were misclassified into another species. The smallest value on the diagonal is 7, suggesting that even the least accurately classified species had 7 out of 10 images correctly identified. Overall, this matrix demonstrates high accuracy with minimal misclassification.

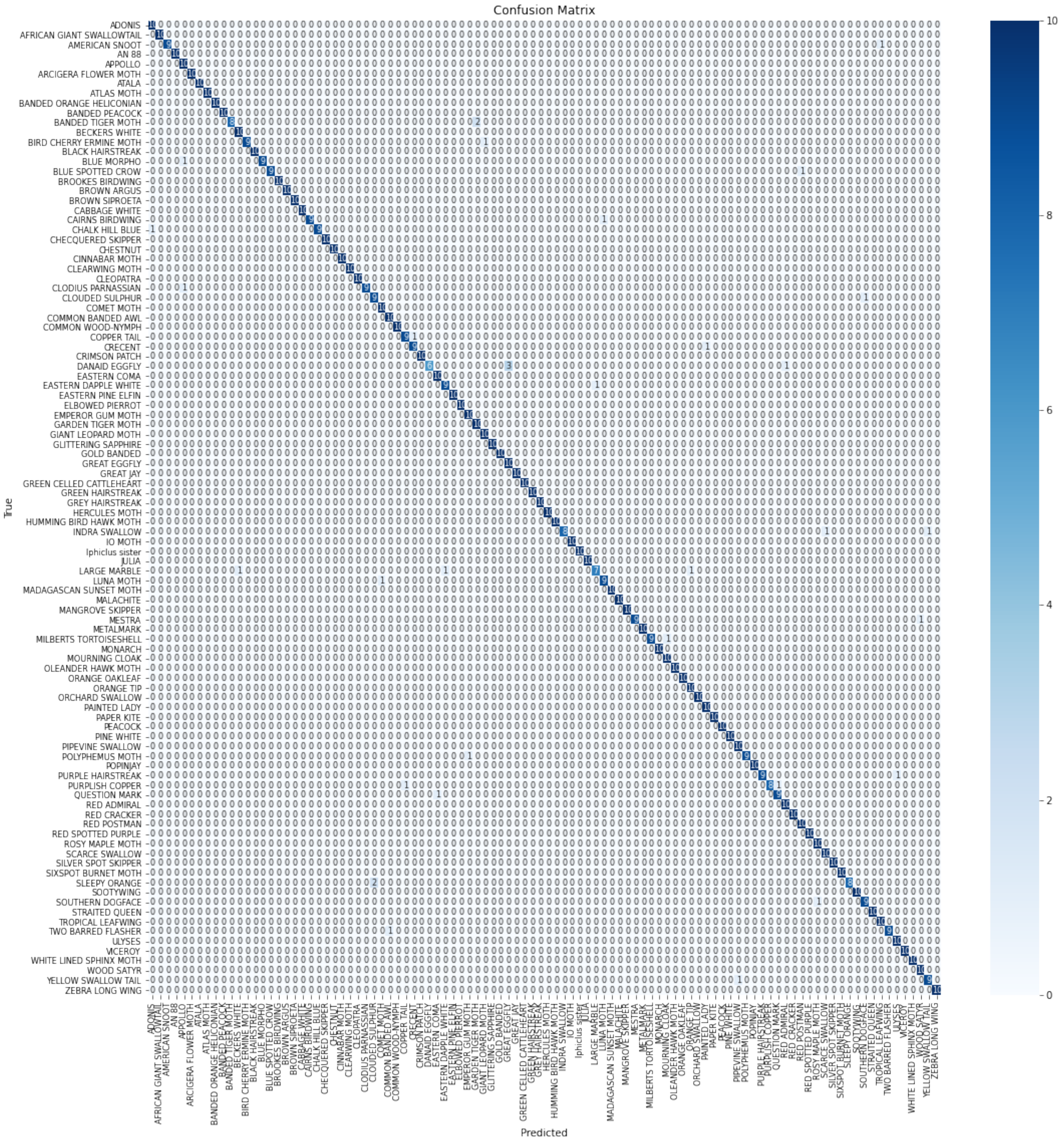

Figure 3.2: Confusion Matrix on validation set

3.3.2 Precision Table

The precision table shows that the classification model performs exceptionally well, with most precision values close to 1, indicating that the model is highly accurate in identifying the correct species. The lowest precision value is 0.77, which suggests that while the model is still reasonably accurate, there is some room for improvement in distinguishing a few specific species. Overall, the model demonstrates strong precision across the majority of the species.

Species name	Precision	Species name	Precision
ADONIS	0.91	HERCULES MOTH	1
AFRICAN GIANT SWALLOWTAIL	1	HUMMING BIRD HAWK MOTH	1
AMERICAN SNOOT	1	INDRA SWALLOW	1
AN 88	1	IO MOTH	1
APPOLLO	0.83	Iphiclus sister	1
ARCIGERA FLOWER MOTH	1	JULIA	1
ATALA	1	LARGE MARBLE	0.88
ATLAS MOTH	1	LUNA MOTH	0.9
BANDED ORANGE HELICONIAN	1	MADAGASCAN SUNSET MOTH	1
BANDED PEACOCK	1	MALACHITE	1
BANDED TIGER MOTH	1	MANGROVE SKIPPER	1
BECKERS WHITE	0.91	MESTRA	1
BIRD CHERRY ERMINE MOTH	1	METALMARK	1
BLACK HAIRSTREAK	1	MILBERTS TORTOISESHELL	1
BLUE MORPHO	1	MONARCH	1
BLUE SPOTTED CROW	1	MOURNING CLOAK	0.91
BROOKES BIRDWING	1	OLEANDER HAWK MOTH	1
BROWN ARGUS	1	ORANGE OAKLEAF	1
BROWN SIPROETA	1	ORANGE TIP	0.91
CABBAGE WHITE	1	ORCHARD SWALLOW	1
CAIRNS BIRDWING	1	PAINTED LADY	0.91
CHALK HILL BLUE	1	PAPER KITE	1
CHECQUERED SKIPPER	1	PEACOCK	1
CHESTNUT	1	PINE WHITE	1
CINNABAR MOTH	1	PIPEVINE SWALLOW	0.91
CLEARWING MOTH	1	POLYPHEMUS MOTH	1
CLEOPATRA	1	POPINJAY	1
CLODIUS PARNASSIAN	1	PURPLE HAIRSTREAK	1
CLOUDED SULPHUR	0.82	PURPLISH COPPER	1
COMET MOTH	0.91	QUESTION MARK	0.9
COMMON BANDED AWL	0.91	RED ADMIRAL	0.91
COMMON WOOD-NYMPH	1	RED CRACKER	1
COPPER TAIL	0.9	RED POSTMAN	0.91
CRECENT	0.9	RED SPOTTED PURPLE	1
CRIMSON PATCH	1	ROSY MAPLE MOTH	0.91
DANAID EGGFLY	1	SCARCE SWALLOW	0.91
EASTERN COMA	0.91	SILVER SPOT SKIPPER	1
EASTERN DAPPLE WHITE	0.9	SIXSPOT BURNET MOTH	1
EASTERN PINE ELFIN	1	SLEEPY ORANGE	1
ELBOWED PIERROT	1	SOOTYWING	1
EMPEROR GUM MOTH	0.91	SOUTHERN DOGFACE	0.9
GARDEN TIGER MOTH	0.83	STRAITED QUEEN	1
GIANT LEOPARD MOTH	0.91	TROPICAL LEAFWING	0,91
GLITTERING SAPPHIRE	1	TWO BARRED FLASHER	1
GOLD BANDED	1	ULYSES	0.91
GREAT EGGFLY	0.77	VICEROY	1
GREAT JAY	1	WHITE LINED SPHINX MOTH	1
GREEN CELLED CATTLEHEART	1	WOOD SATYR	0.91
GREEN HAIRSTREAK	1	YELLOW SWALLOW TAIL	0.9
GREY HAIRSTREAK	1	ZEBRA LONG WING	1

3.3.3 Performance of models

Model	F_1 Macro	Recall	Accuracy
Xception	**97%**	**97%**	**97.5%**
ResNet50	94%	94%	94%
VGG16	91%	91%	91%
MobileNetV2	94.6%	94.6%	94.6%

Table 3.2: Performance of models

3.3.4 Summary

The results indicate that the Xception model performs exceptionally well on the classification task, with a validation accuracy of 97.5% and no signs of overfitting. The confusion matrix highlights the model's detailed classification performance, while the accuracy and loss graphs confirm effective learning and convergence. These findings underscore the model's robustness and generalization capability, suggesting that further refinements may focus on addressing specific class misclassifications or exploring additional improvements.

Chapter 4

CONCLUSION

In this report, we have demonstrated the application of the Xception model for classifying a dataset of butterfly and moth images, achieving an impressive accuracy of 97.5% on the validation set. The training process involved running 15 epochs, each taking 25-30 minutes, without encountering issues of overfitting or underfitting. Importantly, there was no evidence of data bias across any of the classes.

The Xception model has exhibited several advantages over other powerful Convolutional Neural Networks (CNNs) such as VGG16 and ResNet20. These advantages include:

- **Efficient Depthwise Separable Convolutions**: Xception utilizes depthwise separable convolutions, significantly reducing computational complexity and parameter count compared to traditional convolutions while maintaining high accuracy.
- **Improved Performance**: The model's design allows it to capture intricate patterns and features with high precision, resulting in superior performance compared to other CNN architectures.
- **Robustness and Generalization**: Xception's architecture enhances its ability to generalize well across diverse datasets, contributing to its robustness in various classification tasks.
- **High Accuracy with Reduced Training Time**: The model achieves high classification accuracy with relatively shorter training times, making it efficient and practical for large-scale image classification problems.
- **No Overfitting or Underfitting**: The model's architecture effectively prevents overfitting and underfitting, ensuring that it learns relevant features without being biased towards specific classes.

Overall, the Xception model's advanced architectural design and efficient computational approach make it a compelling choice for complex image classification tasks, outperforming other leading CNN models and demonstrating its effectiveness in real-world applications.

REFERENCES

[**1**] François Chollet. "Xception: Deep Learning with Depthwise Separable Convolutions." Conference on Computer Vision and Pattern Recognition, 2018.

[**2**] Aston Zhang, Zachary C. Lipton, Mu Li, Alexander J. Smola. "Dive into Deep Learning.", 2023.

[**3**] Trevor Hastie, Robert Tibshirani, và Jerome Friedman. "The Elements of Statistical Learning: Data Mining, Inference, and Prediction." Springer, 2009

[**4**] Ian Goodfellow, Yoshua Bengio, và Aaron Courville. "Deep Learning." MIT Press, 2016.

[**5**] François Chollet. "Deep Learning with Python." Manning Publications, 2017.

[**6**] Tianyu Song, Linh Thi Hoai Nguyen, Ton Viet Ta. "MPSA-DenseNet: A novel deep learning model for English accent classification." Computer Speech & Language, 2023.

[**7**] Tianyu Song, Ton Viet Ta. "Advancing Bird Classification: Harnessing PSA-DenseNet for Call-Based Recognition" The Proceedings of the Fifth Workshop on Interdisciplinary Sciences, 2023.