Radhika Magaji
DATA 5322
May 11, 2022

Predict bird species using Neural networks from images.

Abstract

Image classification/recognition has proven to be considered a benchmark for modern methodologies in machine learning. It is easy for us humans to recognize sounds and images merely through our senses. However, in this study, the model attempts to classify 18 bird species using images obtained from the Xeno-Canto website by training separate deep neural networks. Two models were trained for image recognition, with one small and a large dataset giving us an accuracy of 97.66% and 86.40% accuracy, respectively. This research is beneficial to raising awareness of biodiversity lost to human activity.

Intro and Overview

The goal of the report is to implement image classification models to predict bird species. ImageNet, a pre-trained network that consists of human-annotated images that help test algorithms, is used to build our image model. The supervised machine-learning algorithm's neural network is trained on human-labeled images.

Theoretical Background:

A neural network processes data in a way analogous to how a human brain does. A supervised learning method called deep learning consists of interconnected nodes in a layered structure. On a high level, a neural network takes inputs, which are then processed in hidden layers using weights that get modified during training. Then the model gives out a prediction. These weights are continuously modified to find patterns to make better predictions. The neural network begins to learn through the framework. The model is adaptive in that it learns from its mistakes and continuously improves the accuracy of the outcome. The most commonly used Neural networks are the Feed Forward Neural network and Recurrent neural network.

Feed Forward Network is one where information moves forward in only one direction from the input neurons, through the hidden ones, and to the output neurons. There are no loops in the network. Convolutional Neural Network (CNN) is a feed-forward network widely used for image classification.

On the other hand, a Recurrent Neural network considers the current and previously received inputs before giving an output. It memorizes previous outputs by giving each output as an input to the next layer within the network.

In this study, the CNN model has been implemented using the deep learning Keras library. Keras is a high-level neural networks API whose core data structures are layers and models.

Before building the model, tuning the hyperparameters are paramount as those parameters will be applied to the training data and determine how the network is trained. For example, the number of hidden layers used within the regularization techniques can drastically improve the model performance. Drop-out is one such technique that increases the generalization powers.

The activation function introduces non-linearity to the models. It helps decide whether the node should fire or not, depending on how vital the neuron's input to the network is. The most popular one used is the rectifier activation function. The learning rate parameter is another important one that tells the model how fast or slow it is supposed to learn. If it is set to a low rate, the progress is slow and learns through more minor updates to the weights.

The batch size defines the count of data samples that will be distributed through the network.

One epoch is when an entire dataset is passed forward and backward through the neural network only once to update its internal parameters. Since one epoch could be too big to feed the network at once, we divide it into several smaller batches.

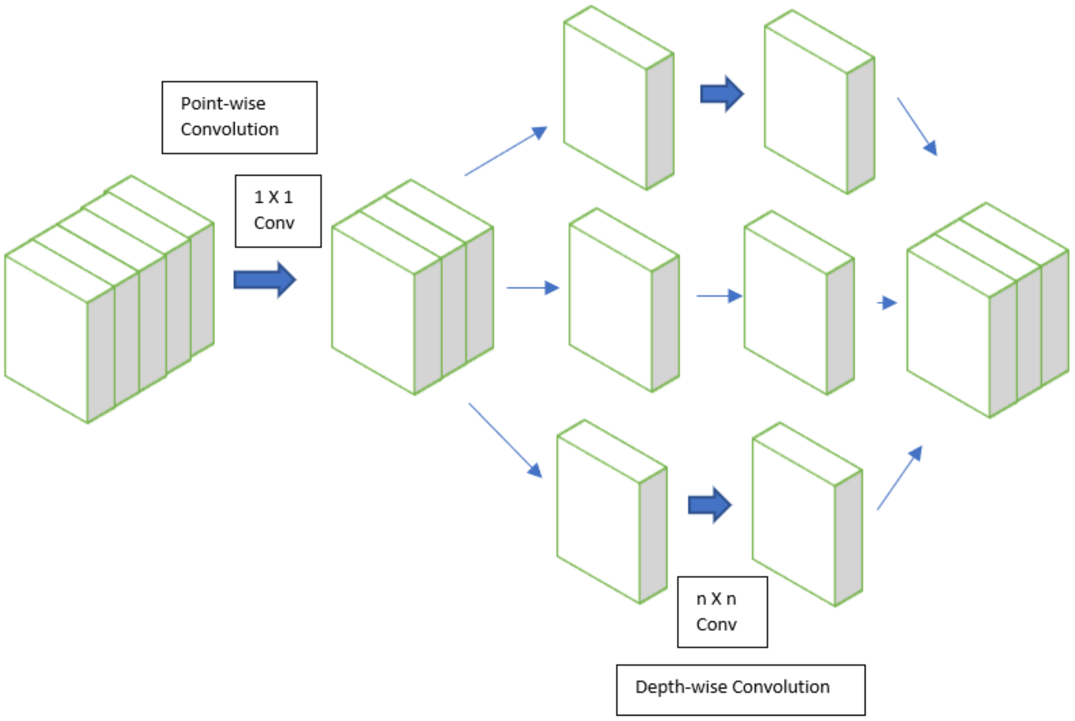
The primary hyperparameters for compiling the model are loss, optimizers, and metrics. The loss function implies how well the prediction model performs in predicting the expected outcome.

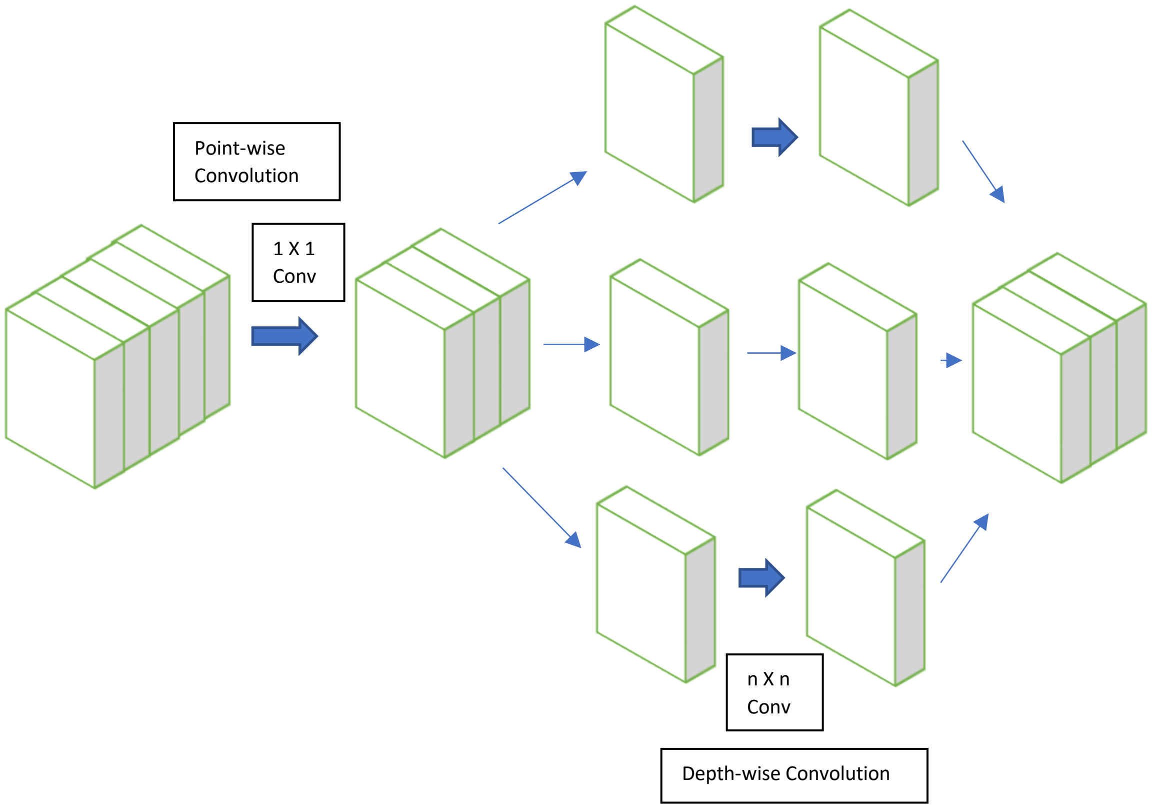
Optimizers are used to reduce training time. It is an algorithm that adjusts the weights and learning rate throughout training. Thus, it helps in reducing the overall loss and improving the accuracy. There are different optimizers, namely Gradient Descent, Stochastic Gradient Descent, Adam optimizer, and more.

To make results more interpretable, the accuracy metric is used to evaluate the performance of train and test data.

The last layer in the model is loaded with Xception-network that is pre-trained on the ImageNet database. Xception module is an extreme version of the Inception module. The main goal of an Inception module is to extract complex features by computing 1*1,3*3, and 5*5 convolutions. Then the output of this layer is fed into the next layer. These convolutions are then used to compress the input, and later filters are applied to each depth space. Xception, on the other hand, reverses this process. It applies the filters first on each depth map and then compresses the input space. And hence this method performs depth-wise separable convolution. The figure below is illustrative of the Xception process.

More information on ImageNet is found in the appendix.


Point-wise Convolution
1 X 1 Conv
n X n Conv
Depth-wise Convolution


Point-wise Convolution
1 X 1 Conv
n X n Conv
Depth-wise Convolution

Methodology:

The data consists of images of birds from Washington obtained from the Xeno-Canto website. A hundred images of each of 18 species were used in this study. For image classification of birds, two models were built, one with smaller samples of 8 species and another one with all 18 species. Train and test folders with subfolders for each species were created. A simple feed-forward network is built and class ids and class names were generated based on the human-labeled folder names of species.

There are a few essential steps from building to evaluating the model. The first step is defining the architecture of the model itself. The study implements a simple feed-forward network. It is a network that is unidirectional in that it does not form a cycle, unlike Recurrent Neural Networks.

The model is sequentially built, consisting of a linear stack of layers, with the first layer being the input layer, the last the output, and a series of one or more hidden layers in between.

Tuning hyperparameters for Image classification:

Keras Layers	**Hyperparameters**	**Value**
Input Dense layer	Units	Equal to the number of classes
Dropout layer	Dropout	20-40%
Activation layer	Activation	Relu
Final output layer	Activation	Softmax

The dense layer, which is the first input layer, is provided with units equal to the number of classes. A small drop-out value of 20% was used in between the layers. Changing them to values between 20-40% did not improve the model significantly. The activation function used between dense layers is ReLU – Rectified Linear Units, followed by another drop-out layer. The last output layer uses the "softmax" activation function as it is most commonly used for a multi-class prediction.

More Model parameters:

Hyperparameter	**Value**
Learning rate	0.001
Batch size	32-64
Epochs	6
Loss function	Categorical_crossentropy
Optimizer	optimizer_adam
Metric	Accuracy

Model compilation hyperparameters:

A lower learning rate of 0.001 was used, which makes sure to converge the training process smoothly though slowly. An optimal batch size of 32 and 64 were used in both models. The number of epochs used was six, as it was enough to reach a good accuracy with a low validation loss.

Model Fit hyperparameters:

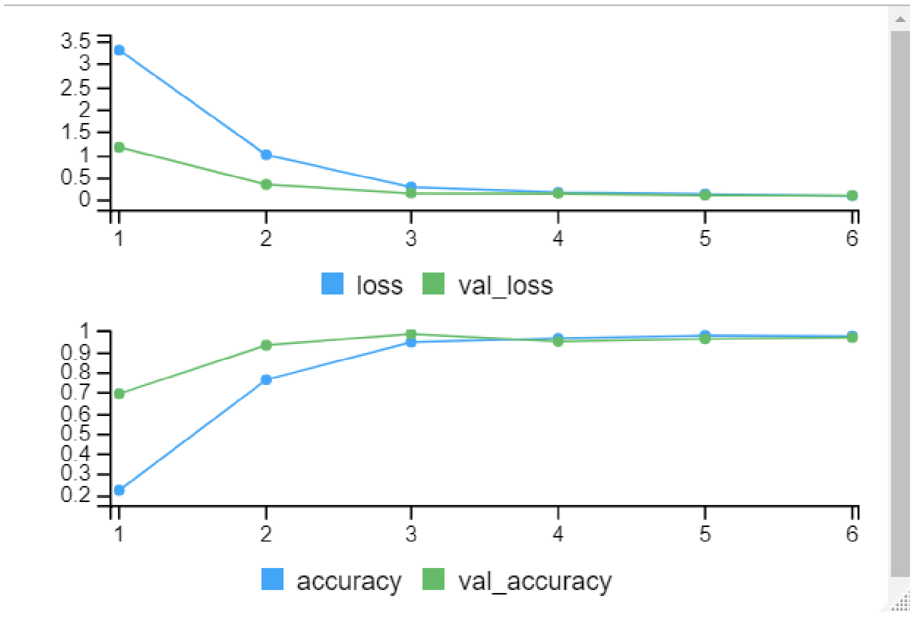
Categorical cross-entropy was used for loss function as the problem is a multi-class classification model with two or more output labels. The response is assigned a one-hot category encoding value in the form of 0s and 1. The response is converted into categorical encoding using Keras.utils to_categorical method. Adam Optimizer was used to optimize the model.

Summary of Computational Results:

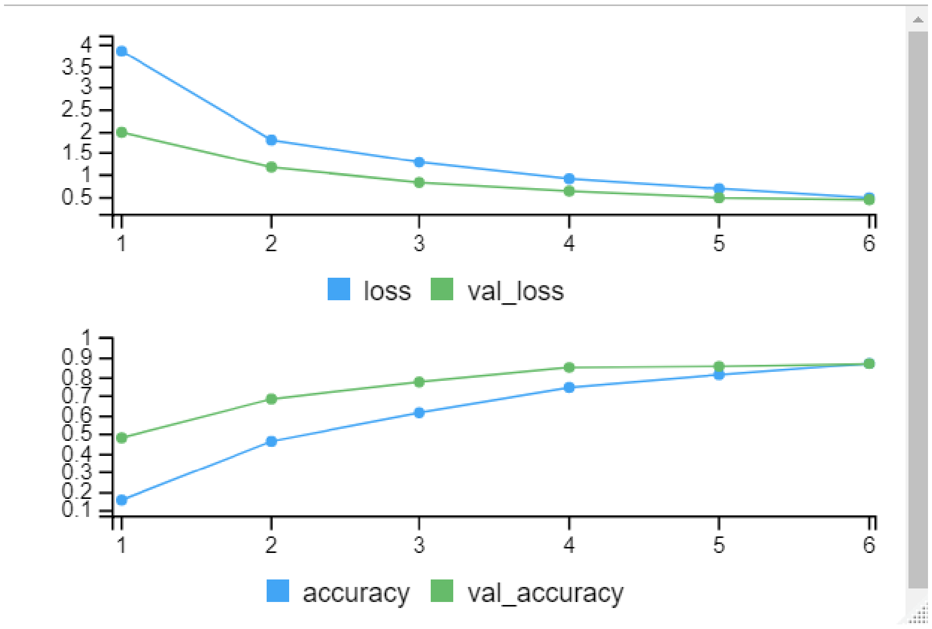
Image Classification: The model for both smaller 8 species and the entire dataset of 18 species performed well. The models were trained on ImageNet pre-trained CNN network and then given to the output layer. Below are the computational plots.

Data	**Accuracy**	**Epochs**	**Time taken**
Sample of 18 classes	0.8674	6	5101.60s
Sample of 8 classes	0.9766	6	2282.92

Sample of 8 classes



Sample of 18 classes



Citations

1. https://www.kaggle.com/datasets/gpiosenka/100-bird-species
2. https://www.kaggle.com/datasets/rohanrao/xeno-canto-bird-recordings-extended-a-m
3. https://iq.opengenus.org/xception-model/#:~:text=Xception%20is%20a%20deep%20convolutional,version%20of%20an%20Inception%20module.
4. https://www.r-bloggers.com/2021/03/how-to-build-your-own-image-recognition-app-with-r-part-1/

Appendix:

Packages/functions used for Image classification:

- Load all packages
  1. Tidyverse for data cleaning and processing
  2. Keras for deep learning tools
  3. Tensorflow
  4. Reticulate
  5. readR
  6. tuner
- ImageNet: A large dataset of marked up photographs intended for machine learning and computer vision research.

Psuedocode:

- Read the training folders, assign the class label and read images from the directory
- Divide into train and test and validation split of 20%
- One hot encodes the output
- Scale the images by dividing by 255 pixels
- Build CNN model
- Load xception network the last layer could be applied
- Specify hyperparameters
- Compile model
- Start training the data with the given batch size 32 and over 6 epochs
- Evaluate the model

Audio Classification:

- Load data
- readmp3() function to load the mp3 files
- Data Preparation
  - dir_ls() to list all directories

    - downsample() to convert the sampling rate
    - use @left to read amplitude
    - @bit to read bit size
- Build feedforward models
- keras_model_sequential() to create sequential model
- compile() to compile the model
- predict() to predict the data