Image Classification of Species of Birds

Nitharsan Sivakanthan

5/11/22

Abstract:

Convolution Neural Networks have been used frequently to classify images into categories. Using this deep learning algorithm paired with computational power, models have been trained on different datasets capable of achieving amazing results in classification. We train a model to classify images of birds into one of 18 species common to the western Washington State area. After tuning for parameters, the best model was able to achieve accuracy on the testing data of roughly 50%. With more computational power, we expect to be able to reach higher accuracy.

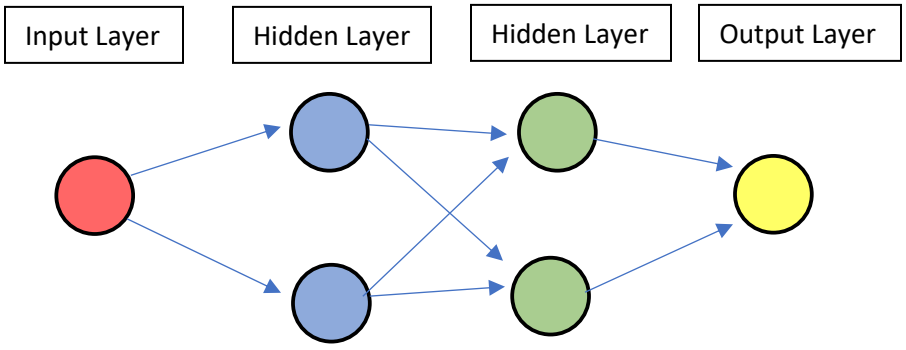
Problem Statement:

We want to classify images of birds common in western Washington State by their species.

Background:

Neural Networks –

Neural networks take on an input layer, apply hidden layers to this input layer, and obtain an output layer. These hidden layers allow us to perform various linear and nonlinear transformations to each layer's input ultimately providing us with an output. The more inputs in our input layer and the more hidden layers we filter through, the more complex the neural network gets.



Each layer consists of nodes that establish linear connections between nodes of other layer(s). The hidden layers have nonlinear activation functions that compute the linear transformations

used to connect nodes between layers. There are various activation functions that are convenient to use in these hidden layers because of their computational ease, particularly regarding their differentiability.

Convolution Neural Networks (CNN) –

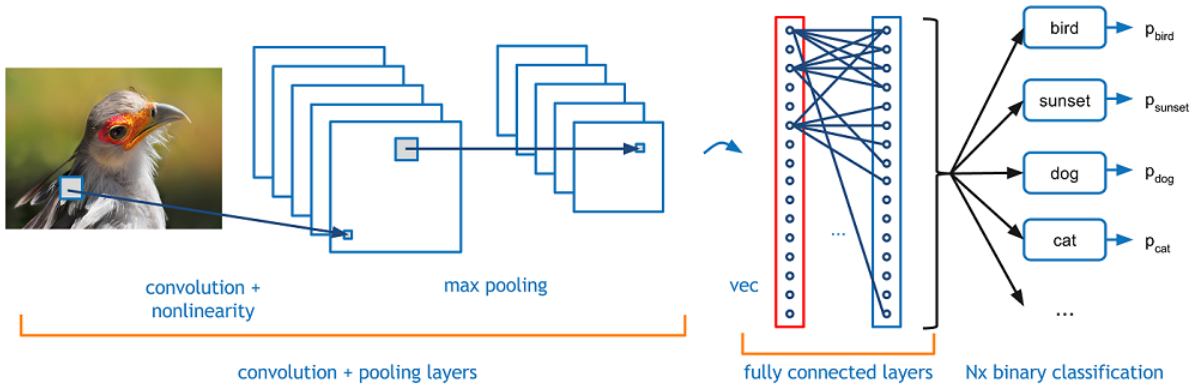


Fig. 1. "A Beginner's Guide to Understanding Convolutional Neural Networks Part 1." KDnuggets, https://www.kdnuggets.com/2016/09/beginners-guide-understanding-convolutional-neural-networks-part-1.html.

Convolution Neural Networks are useful for image classification problems. These networks use a convolution layer to discover patterns or features in the image inputs. For each image, the algorithm will compare groups of pixels of that image and compute a filtered value. These filtered values are then generalized into a pooling layer which creates a new general image of the feature found in the convolution layer. The algorithm can repeat this process multiple times. We can use these connects made between images and features to predict what is in a new image based on the images used to train the algorithm.

Model Parameters -

Neural network algorithms have many parameters: number of units, activation function, input shape, dropout rate, number of epochs, batch size, and validation split. In addition, for convolution neural networks, we must provide the algorithm with a kernel size, pool size, and number of filters.

Below is a table with a description of each of these parameters.

Parameter:	Description:
Units	The number of nodes in each layer.

Activation Function	The activation function used in the hidden layer.
Input Shape	The shape of the initial inputs into the input layer.
Dropout Rate	Used to reduce the complexity of the model by dropping out nodes during the training of the model. This can affect overfitting or underfitting of the training data.
Epochs	Number of times the data is used to train a model.
Batch Size	The number of training inputs put through the neural network at once.
Validation Split	The training/validation split of the data when training the model.
Kernel Size	The dimensions of the convolution image.
Pool Size	The dimensions of the image after pooling.
Filters	The number of features being filtered in the convolution layer.

Methodology:

Data –

The images were sourced from the Birds 400 dataset on Kaggle. The dataset consists of over 60,000 images across 400 species. For this paper, we use up to 60 images for each of 18 species of birds common to the western Washington State area. All the images are 224 x 224 x 3 dimensions, in color, and in the jpg format.

Process –

After preparing the data for the CNN, we first process the algorithm on a subset of our data to make sure the CNN is working properly. Once this model is complete, we train a model now utilizing all the data to establish a baseline model. We train more models tuning for parameters and compare the testing accuracy with the baseline model to choose the best model.

Results:

For each of our trained models, there are three convolution and pooling layers in this model. Each pair of convolution and pooling layer is accompanied by a layer dropout. The input shape is the same for each model. Relu and softmax activation functions are used for all the models. And the categorical cross-entropy is used to calculate loss.

Model with subset of data-

Using only 10 images of each of the 18 species of birds in the training data, a CNN model is trained to ensure the algorithm works properly on the data.

This model achieves 33% accuracy on the testing data.

Baseline model-

Now our models will use 30 images for the training data and 30 images for the testing data for sake of computational ease. The pseudocode provided in the Appendix is for the baseline model. All the parameters chosen for this baseline model can be found there.

This model achieves 49.44% accuracy on the testing data.

Parameter Tuning-

Next, we train models that change certain parameters to see its effect on the testing accuracy compared to the testing accuracy of the baseline model. The table below summarizes this process.

Model:	**Description:**	**Accuracy:**
Baseline	Baseline comparison model, all parameters in appendix	49.44%
Increased Dropout Rate	Increased the dropout rate to .65-.75 compared to .25-.35 in the baseline model	36.48%
Reduced Kernel and Pool Size	Reduced sizes of Kernel Pool to 2 X 2 compared to 4 X 4 in the baseline model	36.30%
Increased Batch Size	Increased Batch Size to 80 compared to 40 in the baseline model	47.96%
Increase Epochs	Increased Epochs to 60 compared to 30 in the baseline model	50.93%
Decreased Dropout Rate & Batch Size	Decreased the drop out rate to .15-.25 and decreased the Batch Size to 20 compared to .25-.35 and 40 respectively in the baseline model	45%

The parameters chosen in the baseline model seem to be close to optimal. With more Epochs we can slightly increase the accuracy of the model on the testing data.

Conclusion:

Using deep learning, we developed a model that can classify an image of a bird into one of 18 species of birds common to western Washington State area with roughly 50% accuracy. There is potential to increase the level of accuracy of the model with more computational power and better tuning.

Appendix:

#import libraries

library(keras)

library(tensorflow)

library(tidyverse)

library(e1071)

library(EBImage)

#create dataframes for each species from labeled folders and resize images

setwd("your input path")

img.bird<- sample(dir());

birds<-list(NULL);

for(i in 1:length(img.bird))

{ birds[[i]]<- readImage(img.bird[i])

birds[[i]]<- resize(birds[[i]], 100, 100)}

american_crow<- birds

#combine dataframes and set testing and training data

train_pool<-c(american_crow[1:30], american_goldfinch[1:30], american_robin[1:30], annas_hummingbird[1:30], barn_swallow[1:30], belted_kingfisher[1:30], blackcapped_chickadee[1:30], blackthroated_sparrow[1:30], cedar_waxwing[1:30], chipping_sparrow[1:30], darkeyed_junco[1:30], downy_woodpecker[1:30], house_finch[1:30], house_sparrow[1:30], northern_flicker[1:30], redheaded_woodpecker[1:30], redwinged_blackbird[1:30], violetgreen_swallow[1:30])

train<-aperm(combine(train_pool),c(4,1,2,3))

test_pool<-c(american_crow[31:60], american_goldfinch[31:60], american_robin[31:60], annas_hummingbird[31:60], barn_swallow[31:60], belted_kingfisher[31:60], blackcapped_chickadee[31:60], blackthroated_sparrow[31:60], cedar_waxwing[31:60], chipping_sparrow[31:60], darkeyed_junco[31:60], downy_woodpecker[31:60], house_finch[31:60], house_sparrow[31:60], northern_flicker[31:60], redheaded_woodpecker[31:60], redwinged_blackbird[31:60], violetgreen_swallow[31:60])

test<-aperm(combine(test_pool),c(4,1,2,3))

#set up one hot encoding

```
train_y<-
c(rep(0,30),rep(1,30),rep(2,30),rep(3,30),rep(4,30),rep(5,30),rep(6,30),rep(7,30),rep(8,30),rep(9,30),rep(10,30),rep(11,30),rep(12,30),rep(13,30),rep(14,30),rep(15,30),rep(16,30),rep(17,30))

test_y<-
c(rep(0,30),rep(1,30),rep(2,30),rep(3,30),rep(4,30),rep(5,30),rep(6,30),rep(7,30),rep(8,30),rep(9,30),rep(10,30),rep(11,30),rep(12,30),rep(13,30),rep(14,30),rep(15,30),rep(16,30),rep(17,30))

train_lab<-to_categorical(train_y)

test_lab<-to_categorical(test_y)


#Build the model

model<- keras_model_sequential()

model %>%

layer_conv_2d(filters = 40,

 kernel_size = c(4,4),

 activation = 'relu',

 input_shape = c(100,100,3)) %>%

layer_conv_2d(filters = 40,

 kernel_size = c(4,4),

 activation = 'relu') %>%

layer_max_pooling_2d(pool_size = c(4,4) )%>%

layer_dropout(rate = 0.25) %>%

layer_conv_2d(filters = 80,

 kernel_size = c(4,4),

 activation = 'relu') %>%

  layer_max_pooling_2d(pool_size = c(4,4)) %>%

layer_dropout(rate = 0.35) %>%

layer_flatten()%>%

layer_dense(units = 256, activation = 'relu')%>%

layer_dropout(rate= 0.25)%>%

  layer_dense(units = 18, activation = 'softmax')%>%
```

```
compile(loss = 'categorical_crossentropy',
        optimizer = optimizer_adam(),
        metrics = c("acc"))

#Fit the model
history<- model %>%
 fit(train,
 train_lab,
 epochs = 30,
 batch_size = 40,
 validation_split = 0.2)

#Obtain accuracy metrics
accuracy <- function(pred, truth) {
   mean(drop(as.numeric(pred)) == drop(truth)) }
model %>% predict(train) %>% k_argmax() %>% accuracy(train_y)
model %>% predict(test) %>% k_argmax() %>% accuracy(test_y)
```