Statistical Deep Learning: Project 4

Anton Holm and Vera Andersson

2022-03-08

Background

Image recognition and classification are becoming increasingly important as we become more dependent on technology in everyday situations. In this project, we use convolutional neural networks of varying architecture to classify bird images, investigating which parameters and network features are the most successful. The goal of this analysis is to find a model which is able to, with high certainty, predict as many correct classifications as possible.

Dataset

The dataset picked is the "Birds 400 - Species Image Classification" dataset made available by Kaggle at [https://www.kaggle.com/gpiosenka/100-bird-species], since we are interested in image classification. This dataset contains $224 \times 224 \times 3$ color images of 400 different bird species, which are divided into a training set of 58 388 images, a test set of 2000 images (with 5 images per species) and a validation set of the same size. We selected this dataset in particular because of the high quality of the images, as each image contains solely one bird which covers at least half of the pixels in the image, and each image is of the same dimension. Furthermore, the labels are available for each image enabling image classification.

However, since the dataset is large, we downsample it by using only four randomly selected classes to reduce the training time, resulting in 711 observations in the training set. Additionally, we merge the training, test, and validation sets into a single dataset to randomly select our own training and test sets where the test set covers 20% of our data. We do this to make sure that the partition is made completely at random, as we are unaware of how these datasets were divided by the person providing the dataset. In Figure 1 we can see the first 5 images of each class from the training dataset to get some intuition about the appearance of the images, and we note that some images have been distorted by the resizing, which could impact the classification results. It is worth to keep in mind that even though we randomly select 4 out of 400 classes, we end up with two different species of ducks which could have some similar properties.

Methods

We perform image classification using a convolutional neural network (CNN) and varying some of the parameters and the architecture. This method is suitable since the CNN extracts significant patterns in the images using filters, which we can use in a feedforward neural network (FFNN) to classify the images. The architecture of the model is chosen based on several criteria. We want to avoid downsampling the images too quickly, minimizing the risk of losing important information. While downsampling, we want to increase the number of channels, making sure that the patterns and properties found by the model is saved somewhere. In this regard, the channels work similar to features of a model. We also want to reduce the image size sufficiently in the layer prior to the FFNN. As the original images are quite large, we use a stride of size 2 in the first layer, which decreases the training time significantly. Using stride means that the kernel skips every other pixel in the convolutional layer. This gives us a reasonable downsample without having to build too deep of a network which would increase computation time drastically and could make the model more complex than it needs to be. Furthermore, max pooling is used to pick up the strongest signals and we use a 3×3 kernel throughout since it is symmetric and sufficiently small, making the computational time manageable. Since we perform classification, the softmax output function is used to provide a probability distribution of the image classification. Additionally, we use Adam as our optimizer to decay the effective learning rate, and set the loss to be `sparse_categorical_crossentropy` since we use classification. We did try smaller changes to the number of layers and layer parameters but the results did not improve. The details of the chosen architecture can be found in the model summary located in the Appendix. The model is implemented using the `keras_model_sequential` function and the packages `keras` and `tensorflow`.

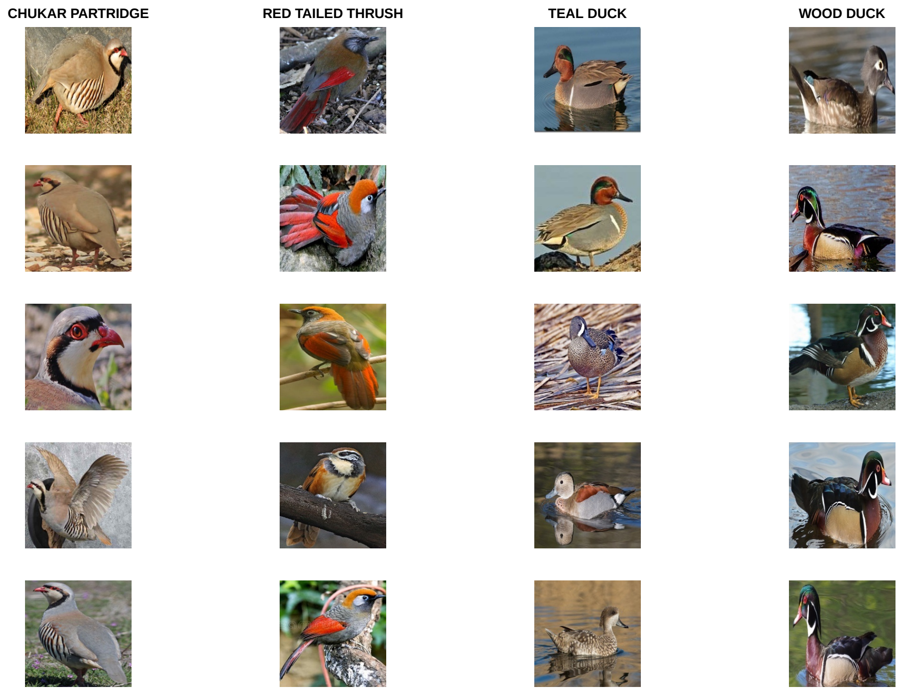

Figure 1: A sample of the first 5 images from each class in the training set.

In order to try to find as good of a model as possible, we vary the learning rates, activation functions and initializations one at a time. We compare learning rates of 0.0001, 0.001 and 0.005, the ReLU and tanh activation functions, and initializations from a Gaussian distribution with mean 0 and standard deviation 0.05, from a Glorot uniform distribution and from a LeCun uniform distributions. The weight initializations are used in the FFNN.

Furthermore, we monitor the training and validation loss as well as the accuracy during training to find the optimal models in each case based on the validation loss, and compute the test loss and error for the final chosen models for comparison. For each of the 6 combinations of hyperparameters, activation function and weight initialization, we extract 5 different model weights from different epochs that we find interesting enough to take a closer look at. We make the selection based on manual stopping where we save each model at every epoch and then use the model from the final epoch, the last epoch before the validation loss converges and three epochs where the model has low validation loss. Finally, we compute and compare the test loss for the 30 models choosing the model with the lowest loss. When training the model, we use a validation set taken from 20% of the batch in each iteration.

Moreover, we decide to use a batch size of 32 and one reason for this is that we might want to improve our model in the future. One such way is too use batch normalization and in order to do this, we would need to compute the mean and standard deviation of each batch. Choosing too small of a batch size would result in uncertain estimations of these statistics. At the same time, since we are doing classification, we are using the softmax activation function in our output layer and cross-entropy as our loss function. In this case, if we use a large batch size and end up with a 100% training accuracy, the softmax function will saturate. The update of the Adam algorithm is $\theta \leftarrow \theta + \Delta\theta$ where $\Delta\theta \propto \frac{1}{\sum_{k=1}^{a} g(t-a)}$ for some positive integer a and large enough t, the number of epochs. Here, $g(s)$ is the gradient of the loss function at epoch s which would go to zero as the softmax function saturates, resulting in the update for the Adam algorithm to blow up, making our training impossible since the value of θ would change drastically in each iteration, making it impossible to stay within some local (or global) minimum unless the flat surface is large enough.

Results and interpretations

For each of the 6 different models, we train them until we see that the validation loss has converged. We do this by continuously following the validation loss and accuracy as can be seen in Figure 2, as well as in Figure A1-A5 in the Appendix. We can see that there is quite a lot of fluctuation early in the training. This could be due to large changes in parameter estimations of the model in each epoch. There are several possible ways to mitigate this effect such as adding more data, using some regularization method such as drop-out or having a smaller learning rate. We tried using drop-out which gave us less fluctuation in the training but worse test statistics. We also tried using smaller learning rates, but in that case the model got stuck immediately. Furthermore, we notice that in several cases, the validation loss suddenly spikes either upwards or downwards. One possible explanation for this is that some combination of nodes in the network are suddenly turned off or given large weights, resulting in such a drastic shift in prediction.

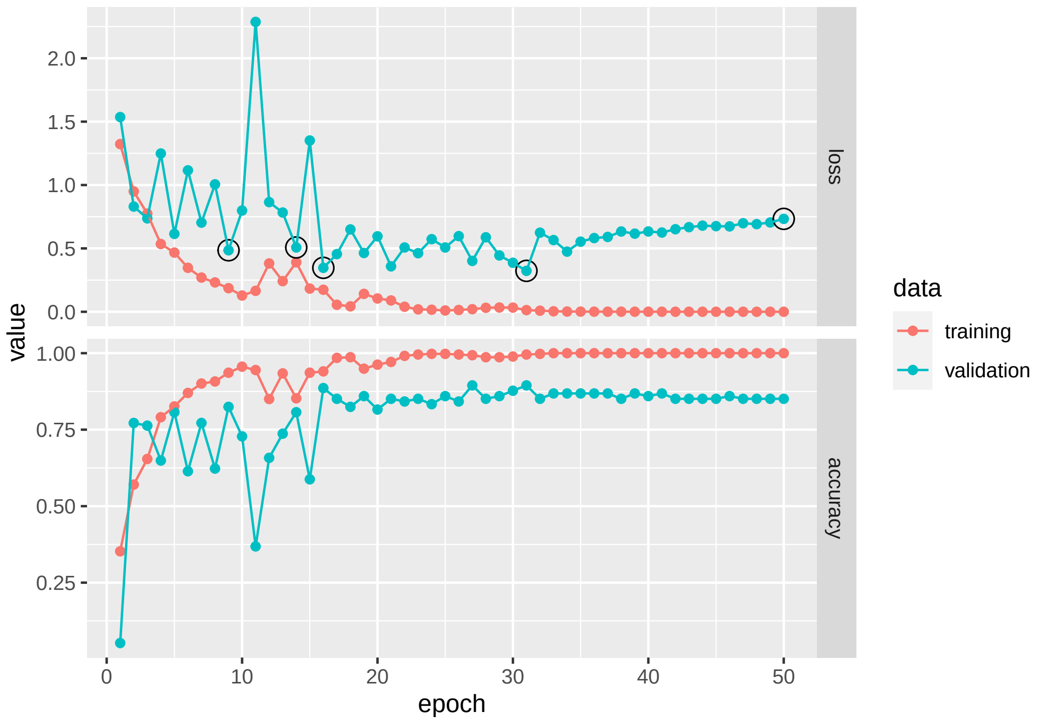

Figure 2: The early stopping curves for the model with learning rate 0.001, the ReLU activation function and Glorot uniform initialization. The circled data points correspond to the 5 models chosen for further evaluation.

Finally we pick out 5 different versions for each model, which are marked by the circles around the data points at different epochs in the above mentioned figures. Lastly, using the test data, we test our model, calculating the test loss for each model and version as can be seen in Table 1. We decide to use the test loss for model selection over the accuracy since this is more reliable. For example, in our case, if a model has low loss it also has high accuracy. However, models with high accuracy could have high loss. This is due to the soft clustering where some models could have a high accuracy, but predict correctly with low certainty and wrongly with high certainty. Based on the results in Table 1, we decide to use the model with learning rate equal to 0.001, ReLU activation function and Glorot uniform weight initialization, i.e. Version 1 from Model 6 which has the weights from the ninth training epoch.

Table 1: Table 1: Rows show validation loss for 6 different models with different parameter values while in the columns there are 5 different weights for each model chosen by manual stopping.

	Version 1	Version 2	Version 3	Version 4	Version 5
Model 1	0.6383	0.5548	0.4947	0.6366	1.0145
Model 2	0.6257	0.5459	0.4375	0.4781	0.4831
Model 3	3.0626	0.8813	0.6932	0.8443	1.8427
Model 4	0.5600	0.5257	0.5624	0.5634	0.6173
Model 5	0.6177	0.4735	0.4203	0.6442	0.9684
Model 6	0.3055	0.4656	0.3579	0.4562	0.4562

The model's accuracy on the test data is 90.2%, having 14 misclassifications out of the 143 test images. In Figure 3, we can see how the model predicts images from the different classes. We note that the model has higher success in classifying images from class 0 and 1, i.e. the Chukar Partridge and Red Tailed Thrush having only one misclassification per class. However, as mentioned when describing the data, the model seems to have more problems classifying the two duck species (Teal Duck and Wood Duck). Most of the misclassifications in these two groups happen due to the fact that our model confuses the two species while a few images are wrongly classified as class 0 or 1.

Furthermore, we notice that there are some images that every model is unable to predict correctly, which we now take a closer look into. We begin by analyzing why there is one image from class 0, the Chukar Partridge, that our model is unable to predict correctly. In for example Figure 4a, we can see the misclassified image of a Chukar Partridge. As can be noted from Figure 1, the Chukar Partridge has two very distinct attributes. It has a striped pattern on its stomach and a black curved line on its head over the eye. In fact, in every image of the Chukar Partridge except for the misclassified one, this striped pattern is clearly visible, whereas the line across the eye is visible in every image. Since the only image of this class which our model is unable to correctly classify is the one that is zoomed in on the pattern across the eye, not showing the striped pattern across the stomach, it could mean that our model is searching for this striped pattern when trying to classify the Chukar Partridge and not the line across the eye.

Moreover, the image of the Red Tailed Thrush in Figure 4b is also misclassified and a possible reason for this may be that this bird is photographed from the front and most other birds from this class are photographed from the side. Hence, it may be more difficult for the network to detect the quite distinguishable red wings and tail. Additionally, this bird does not have an orange head, which most other birds from this class possess. Furthermore, in Figure 4c, we note that the Teal Duck has a vibrant green head and a distinct white line on its face, which could be a reason why our chosen model classifies it as a Wood Duck, since most of these birds also have green and white markings. The image is also zoomed in on the head of the bird, meaning that all information about its body type is lost.

The model is also unable to classify the last image in Figure 4d as a Wood Duck, as the bird is instead confused as a Chukar Partridge, After analyzing the images of the wood duck we noticed that this one had a brown head while the other females had a grey head. After consulting with some ornithologist websites, it turns out that this is an Australian female wood duck while the others are North-American female wood ducks. This makes it hard for the model to classify this as a wood duck since it has only been training on

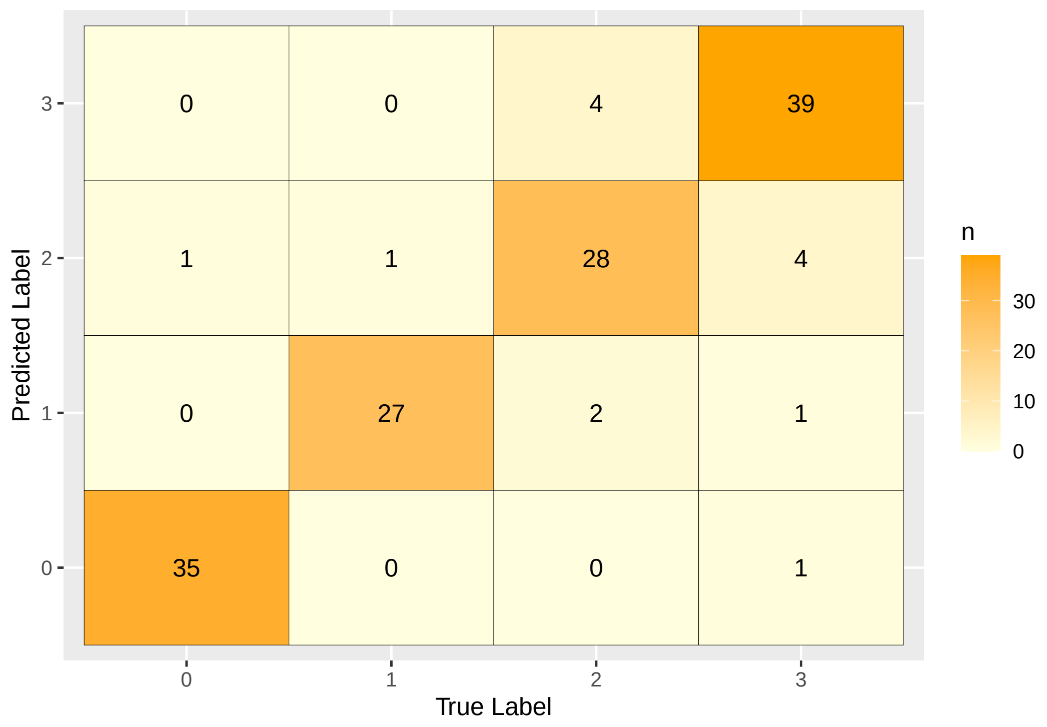

Figure 3: The confusion matrix for the final model, where the labels 0, 1, 2 and 3 correspond to the Chukar Partridge, Red Tailed Thrush, Teal Duck and Wood Duck, respectively.

North-American wood ducks. Another reason for the misclassification of this image, and others, could be that the image is rescaled to match the dimensions of the other images in the data set. This could distort some of the properties of the bird that the model is looking for. We can see that the distortion has enhanced the pattern across the stomach which could be why the model classify it as a Chukar Partridge.

Considering that we output a probability distribution, we are able to take a look at how uncertain or certain our model is when making the wrong prediction. We note that most of the misclassifications are made with a very high probability while some of the predictions lie close to the border between the wrong prediction and the correct prediction or are split across several classes. At the same time, the images which the model classifies correctly are mostly classified with over 99% certainty.

Discussion

When it comes to choosing the model architecture, hyperparameters, and activation functions, there is always room for improvement. One such improvement we could try in the future is using absolute max pooling. In our training we tried using the hyperbolic tangent as our activation function, however, since max pooling is applied, we only pick up the signals closest to 1. Since the hyperbolic tangent outputs values between -1 and 1, we could apply absolute max pooling to also pick up signals close to -1.

Another issue we came across when training different models was that using too small or large learning rates resulted in our training getting stuck immediately. Since we had quite large fluctuation in our validation loss when training, a smaller learning rate could be tested to see if this helps with smoothing out the learning curve. One way for us to be able to use a smaller learning rate could be to implement a short circuit as in the ResNet architecture, making sure we are always doing some updating in the training. Furthermore, if we optimized the model, making it faster to train, we could perform a grid search over all parameters, activation functions, weight initializations etc. in order to find a better combination for the model.

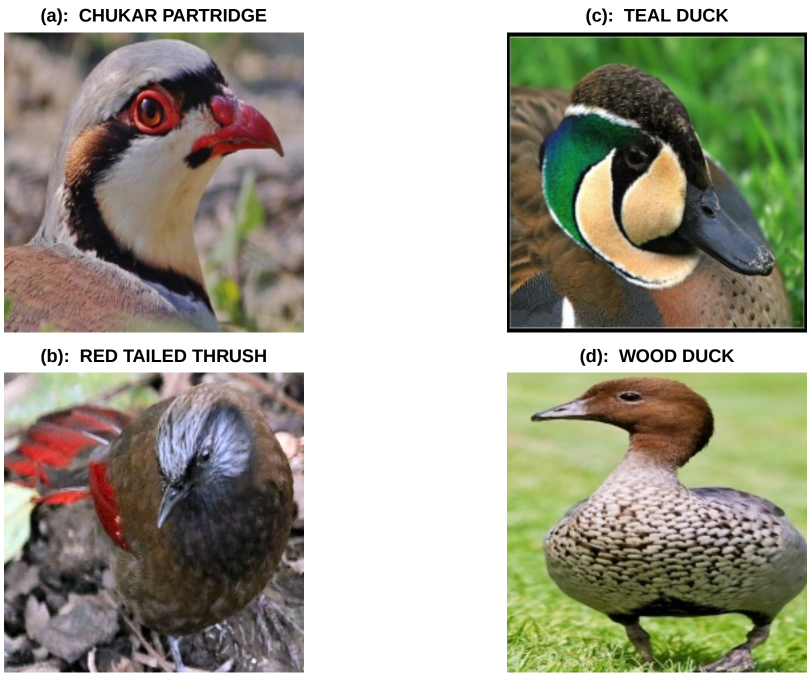

Figure 4: A sample of misclassified images from each class in the test set.

Furthermore, we noted in our results that many of the images that were misclassified were in some sense outliers. Some images were taken from another angle while some were zoomed in. There was also a problem of images being rescaled. One possible improvement to handle this could be to construct more training data by altering the images we have. For instance, we could make a copy of every image and zoom in on only the bird's head to make the model more susceptible to these types of images. We could also rescale the images by for example stretching them in one direction and then cropping them, or shrinking the image in one direction while adding zero padding in order for the model to perform better when given images that have been altered. Further improvements could be to add noise to the images and to rotate them, since birds are living things and therefore, not every image will show a bird sitting straight up. They could for example be flying downwards or upwards or be lying down.

Finally, something we should take a closer look into are certain epochs in our training. We can see in the training figures (Figure 2 and Figure A1-5) that we have large spikes, both up and down, in our validation loss in some epochs. As mentioned in the result, this could be due to certain nodes being turned on or off. We could compare the weights in these epoch with other epochs and see if we can find which nodes that had a drastic change in weights. If for example we see a large decrease in validation loss while certain nodes suddenly get turned on, we could keep these fixed with a large weight while training the model and see what happens. If we instead have a large increase in validation loss while some nodes are turned on, we could try to remove these and see how that affects our training.

Appendix

Supplementary Figures

Below is the model summary of the final model chosen above.

```
## Loaded Tensorflow version 2.7.0
```

```
## Model: "sequential_19"
## _________________________________________________________________________________
##  Layer (type)                          Output Shape                      Param #
## =================================================================================
##  conv2d_97 (Conv2D)                    (None, 111, 111, 32)              896
##
##  max_pooling2d_77 (MaxPooling2D)       (None, 55, 55, 32)                0
##
##  conv2d_96 (Conv2D)                    (None, 53, 53, 64)                18496
##
##  max_pooling2d_76 (MaxPooling2D)       (None, 26, 26, 64)                0
##
##  conv2d_95 (Conv2D)                    (None, 24, 24, 64)                36928
##
##  max_pooling2d_75 (MaxPooling2D)       (None, 12, 12, 64)                0
##
##  conv2d_94 (Conv2D)                    (None, 10, 10, 128)               73856
##
##  max_pooling2d_74 (MaxPooling2D)       (None, 5, 5, 128)                 0
##
##  conv2d_93 (Conv2D)                    (None, 3, 3, 256)                 295168
##
##  flatten_18 (Flatten)                  (None, 2304)                      0
##
##  dense_37 (Dense)                      (None, 64)                        147520
##
##  dense_36 (Dense)                      (None, 4)                         260
##
## =================================================================================
## Total params: 573,124
## Trainable params: 573,124
## Non-trainable params: 0
## _________________________________________________________________________________
```

Below are the training and validation loss and accuracy plots of the remaining models.

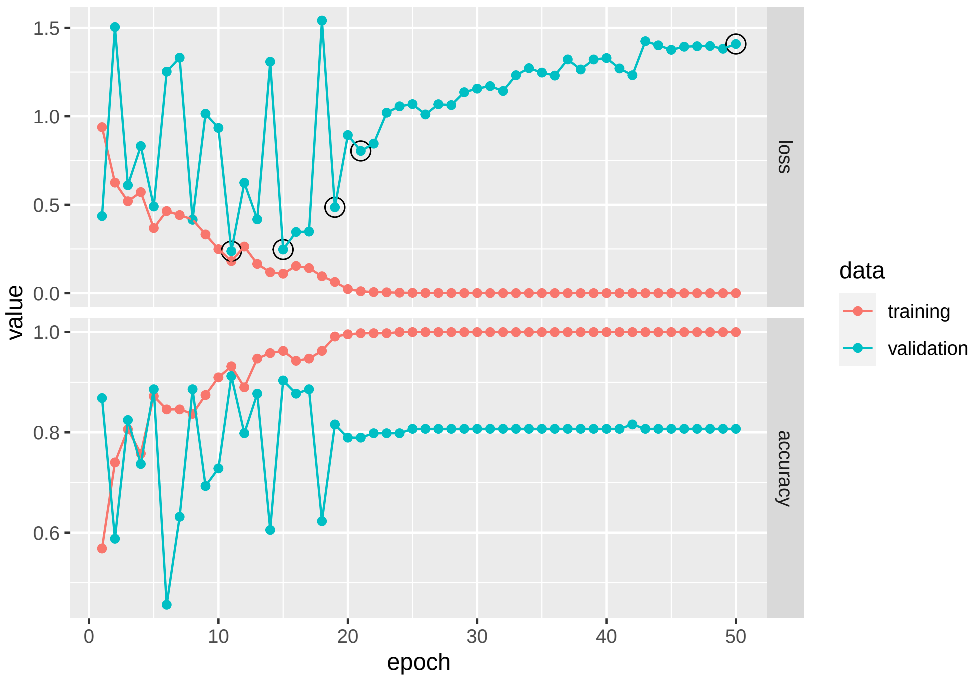

Figure A1: The early stopping curves for the model with learning rate 0.001, the ReLU activation function and Gaussian initialization with mean 0 and standard deviation 0.05. The circled data points correspond to the 5 models chosen for further evaluation.

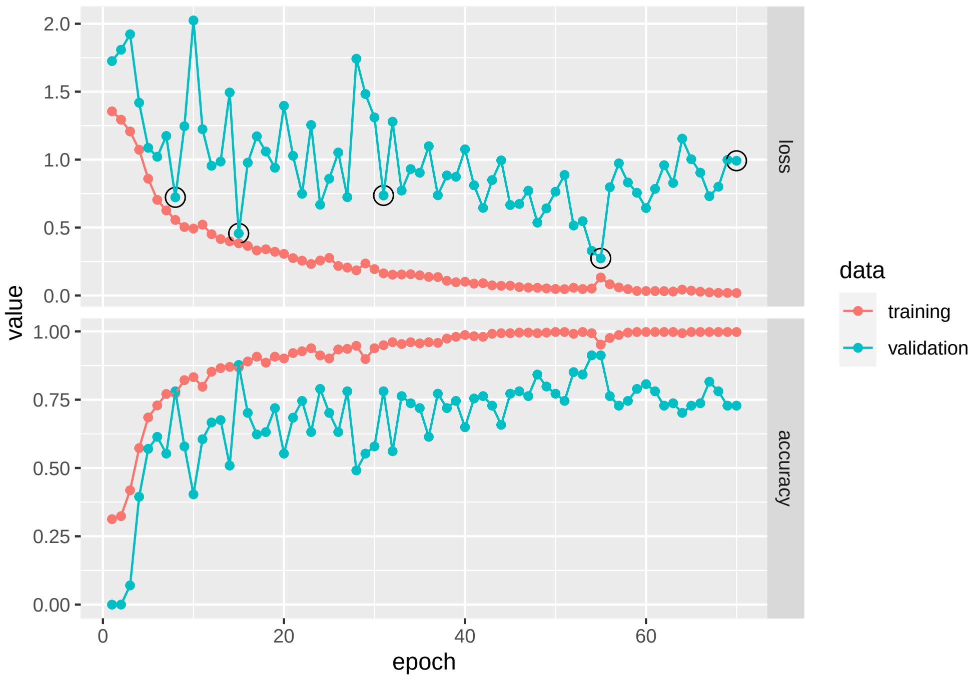

Figure A2: The early stopping curves for the model with learning rate 0.0001, the ReLU activation function and Gaussian initialization with mean 0 and standard deviation 0.05. The circled data points correspond to the 5 models chosen for further evaluation.

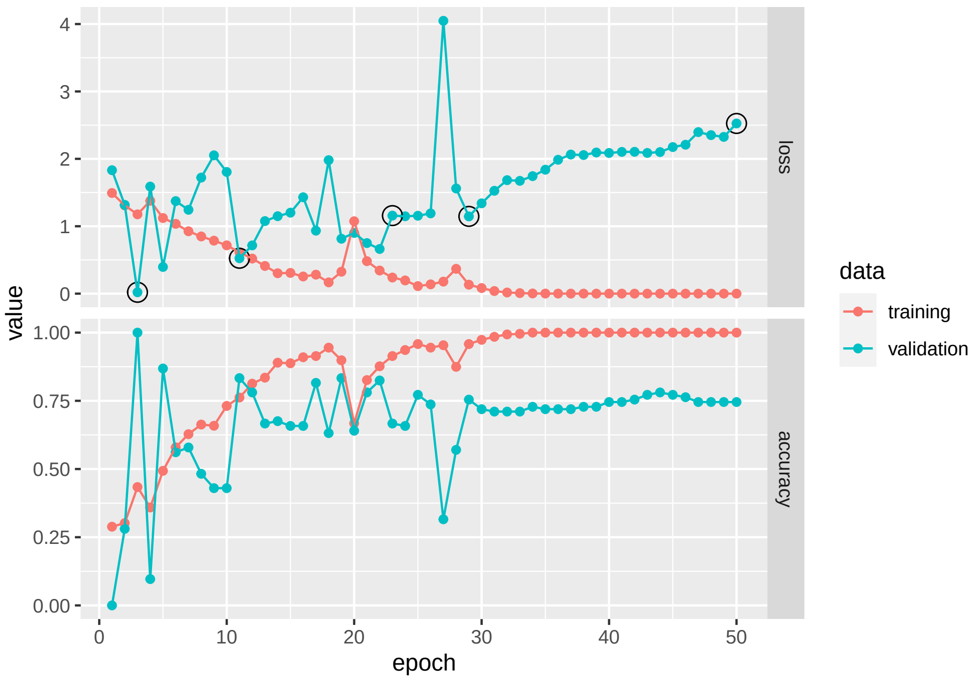

Figure A3: The early stopping curves for the model with learning rate 0.005, the ReLU activation function and Gaussian initialization with mean 0 and standard deviation 0.05. The circled data points correspond to the 5 models chosen for further evaluation.

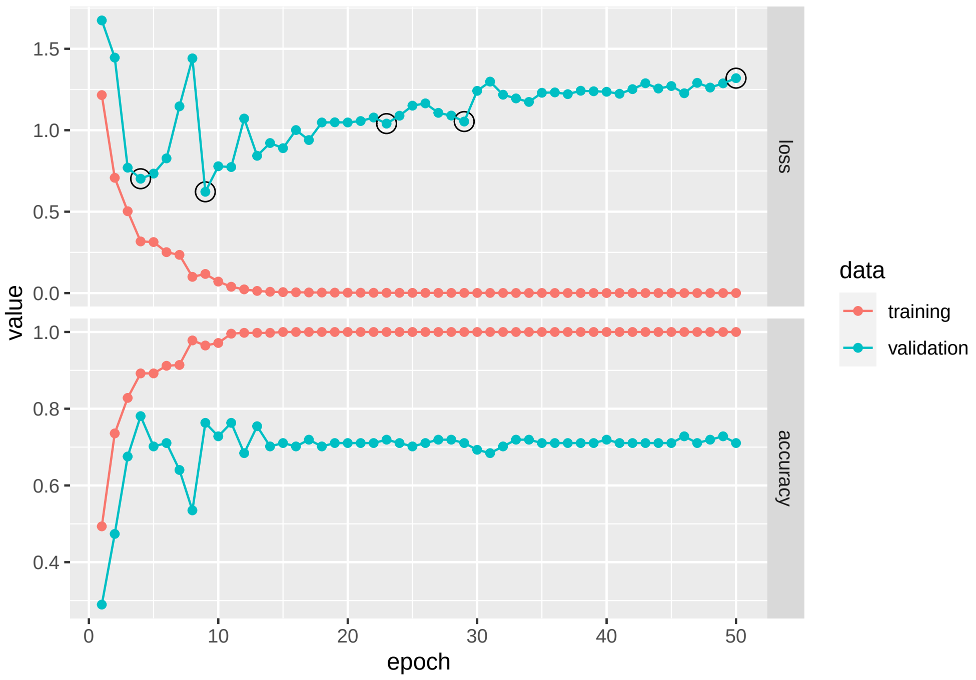

Figure A4: The early stopping curves for the model with learning rate 0.001, the tanh activation function and Gaussian initialization with mean 0 and standard deviation 0.05. The circled data points correspond to the 5 models chosen for further evaluation.

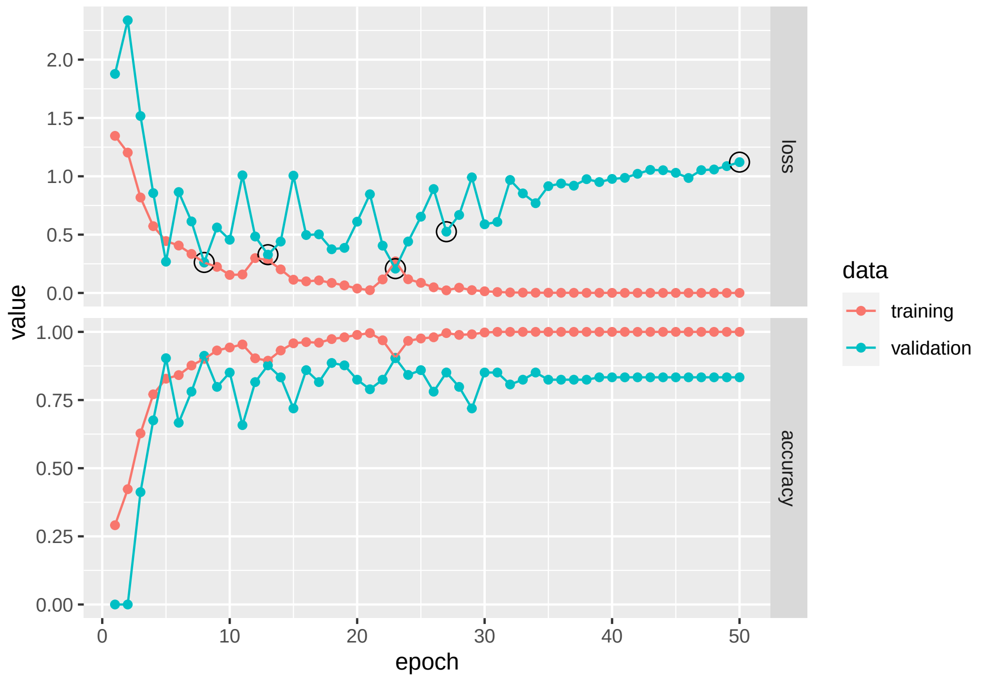

Figure A5: The early stopping curves for the model with learning rate 0.001, the ReLU activation function and LeCun initialization. The circled data points correspond to the 5 models chosen for further evaluation.

Code

This code loads our data and divides it into train and test sets.

```
set.seed(931031)
#### Loading the data set and dividing it into training and test sets ####
#-1 since we have classes from 0-399 and sample.int only takes integers larger than 0.
random_classes <- sample.int(400, 4)-1

# Randomize which birds to take
df <- read_csv("birds.csv") %>%
  filter(`class index` %in% random_classes)

saveRDS(df, file = "df.rdata")

# Loading data and constructing training and test datasets.
train <- df %>%
  mutate(`class index` = as.factor(`class index`)) %>%
  group_by(`class index`) %>%
  sample_frac(0.8)
levels(train$`class index`) <- c(0, 1, 2, 3)

test <- df %>%
  filter(!(filepaths %in% train$filepaths)) %>%
  mutate(`class index` = as.factor(`class index`))
levels(test$`class index`) <- c(0, 1, 2, 3)

saveRDS(test, "Test_df.rdata")
```

```r
# Loading images
train_image <- lapply(train$filepaths, load.image)
test_image <- lapply(test$filepaths, load.image)

train_label <- train$`class index`
test_label <- test$`class index`


image_array_train <- array(dim = c(length(train_image), 224, 224, 3))
for (i in 1:length(train_image)){
  image_array_train[i, , , ] <- load.image(train$filepaths[i])[, , , ]
}

image_array_test <- array(dim = c(length(test_image), 224, 224, 3))
for (i in 1:length(test_image)){
  image_array_test[i, , , ] <- load.image(test$filepaths[i])[, , , ]
}
```

Code to show our images in data section.

```r
#### Plotting the first 5 images of each class in the training set ####
# Extracting the first 5 images from each class in training set

sub_df <- readRDS("df.rdata")
subset <- sub_df %>%
  group_by(`class index`) %>%
  slice(1:5)

# Loading the images
subset_images <- lapply(subset$filepaths, load.image)

# Creating labels for each column for the plot
names <- c("CHUKAR PARTRIDGE", rep(NA, 4),
           "RED TAILED THRUSH", rep(NA, 4),
           "TEAL DUCK", rep(NA, 4),
           "WOOD DUCK", rep(NA, 4))

# Creating the plot
par(mfcol = c(5,4), mar = c(0.2, 0.2, 1.3, 0.2), oma = rep(1, 4))
subset_images %>%
  purrr::set_names(names) %>%
  purrr::map(as.raster, max = 255) %>%
  purrr::iwalk(~{plot(.x); title(.y, cex.main = 0.8)})
```

Below follows 6 similar code chunks. Each code chunk produces a model and evaluates it at 5 different epochs.

```r
#### Model 1 ####
#### lr = 0.001, activation = relu, init = Gaussian
set_random_seed(931031)

model_conv1 <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "relu",
                input_shape = c(224,224,3),
                strides = c(2,2)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 256, kernel_size = c(3,3), activation = "relu")

# Random initiation on weights
model_conv1 %>%
  layer_flatten() %>%
  layer_dense(units = 64, activation = "relu", initializer_random_normal(mean = 0, stddev = 0.05)) %>%
  layer_dense(units = 4, activation = "softmax", initializer_random_normal(mean = 0, stddev = 0.05))

summary(model_conv1)

model_conv1 %>% compile(
  optimizer = optimizer_adam(learning_rate = 0.001),
  # Use one vector with factors 1-50 for classes instead of one-hot vectors
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)

history <- model_conv1 %>%
  fit(
    x = image_array_train, y = as.matrix(as.numeric(as.character(train_label))),
    batch_size = 32,
    epochs = 50,
    validation_split = 0.2,
    verbose = 2, # Only show loss/acc for the last batch in the epoch
    callbacks = callback_model_checkpoint(
      monitor = "val_loss",
      filepath = 'Models\\Model1\\saved-model.{epoch:02d}-{val_loss:.2f}.hdf5',
      mode = "min",
      save_best_only = FALSE)
    )

# Saving the history
History_df <- history %>% as_tibble()
saveRDS(History_df, file = "Hist\\History.Rda")
hist_df <- readRDS("Hist\\History.Rda")

plot1 <- hist_df %>%
  ggplot(aes(x = epoch, y = value, col = data)) +
  geom_point() +
  geom_point(data = hist_df %>%
               filter(epoch %in% c(11, 15, 19, 21, 50) & metric == "loss" & data == "validation"),
             pch = 21, size = 4, colour = "black") +
  geom_line() +
  facet_grid(rows = vars(metric), scales = "free")
saveRDS(plot1, file = "Figures\\Plot1.rdata")

# Saving the 5 chosen models for learning rate = 0.001
mod1 <- load_model_hdf5(filepath = "Models\\Model1\\saved-model.11-0.24.hdf5",
                        custom_objects = list("RandomNormal" = initializer_random_normal))
mod2 <- load_model_hdf5(filepath = "Models\\Model1\\saved-model.15-0.25.hdf5",
                        custom_objects = list("RandomNormal" = initializer_random_normal))
mod3 <- load_model_hdf5(filepath = "Models\\Model1\\saved-model.19-0.49.hdf5",
                        custom_objects = list("RandomNormal" = initializer_random_normal))
mod4 <- load_model_hdf5(filepath = "Models\\Model1\\saved-model.21-0.80.hdf5",
                        custom_objects = list("RandomNormal" = initializer_random_normal))
mod5 <- load_model_hdf5(filepath = "Models\\Model1\\saved-model.50-1.41.hdf5",
                        custom_objects = list("RandomNormal" = initializer_random_normal))

# Extracting the test losses
test_score1 <- evaluate(mod1, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[1]
test_score2 <- evaluate(mod2, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[1]
test_score3 <- evaluate(mod3, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[1]
test_score4 <- evaluate(mod4, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[1]
test_score5 <- evaluate(mod5, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[1]

eval1 <- cbind(test_score1, test_score2, test_score3, test_score4, test_score5)
```

```r
set_random_seed(931031)
#### Model 2 ####
### lr = 0.0001, activation = relu, init = Gaussian
model_conv2 <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "relu",
                input_shape = c(224,224,3),
                strides = c(2,2)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 256, kernel_size = c(3,3), activation = "relu")

# Random initiation on weights
model_conv2 %>%
  layer_flatten() %>%
  layer_dense(units = 64, activation = "relu", initializer_random_normal(mean = 0, stddev = 0.05)) %>%
  layer_dense(units = 4, activation = "softmax", initializer_random_normal(mean = 0, stddev = 0.05))

model_conv2 %>% compile(
  optimizer = optimizer_adam(learning_rate = 0.0001),
  # Use one vector with factors 1-50 for classes instead of one-hot vectors
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)
```

```r
history2 <- model_conv2 %>%
  fit(
    x = image_array_train, y = as.matrix(as.numeric(as.character(train_label))),
    batch_size = 32,
    epochs = 50,
    validation_split = 0.2,
    verbose = 2, # Only show loss/acc for the last batch in the epoch
    callbacks = callback_model_checkpoint(
      monitor = "val_loss",
      filepath = 'Models\\Model2\\saved-model.{epoch:02d}-{val_loss:.2f}.hdf5',
      mode = "min",
      save_best_only = FALSE)
    )

# Validation loss had not converged. More epochs was needed
history2a <- model_conv2 %>%
  fit(
    x = image_array_train, y = as.matrix(as.numeric(as.character(train_label))),
    batch_size = 32,
    epochs = 20,
    validation_split = 0.2,
    verbose = 2, # Only show loss/acc for the last batch in the epoch
    callbacks = callback_model_checkpoint(
      monitor = "val_loss",
      filepath = 'Models\\Model2\\saved-model.{epoch:02d}-{val_loss:.2f}.hdf5',
      mode = "min",
      save_best_only = FALSE)
    )

# Saving the history
history2a_df <- history2a %>%
  as_tibble() %>%
  mutate(epoch = epoch + 50)

History2_df <- history2 %>%
  as_tibble() %>%
  rbind(history2a_df)
saveRDS(History2_df, file = "Hist\\History2.Rda")
hist_df2 <- readRDS("Hist\\History2.Rda")

plot2 <- hist_df2 %>%
  ggplot(aes(x = epoch, y = value, col = data)) +
  geom_point() +
  geom_point(data = hist_df2 %>%
               filter(epoch %in% c(08, 15, 31, 55, 70) & metric == "loss" & data == "validation"),
             pch = 21, size = 4, colour = "black") +
  geom_line() +
  facet_grid(rows = vars(metric), scales = "free")
saveRDS(plot2, file = "Figures\\Plot2.rdata")

# Saving the 5 chosen models
mod1_2 <- load_model_hdf5(filepath = "Models\\Model2\\saved-model.08-0.72.hdf5",
                          custom_objects = list("RandomNormal" = initializer_random_normal))
```

```r
mod2_2 <- load_model_hdf5(filepath = "Models\\Model2\\saved-model.15-0.46.hdf5",
                          custom_objects = list("RandomNormal" = initializer_random_normal))
mod3_2 <- load_model_hdf5(filepath = "Models\\Model2\\saved-model.31-0.74.hdf5",
                          custom_objects = list("RandomNormal" = initializer_random_normal))
mod4_2 <- load_model_hdf5(filepath = "Models\\Model2\\saved-model.55-0.27.hdf5",
                          custom_objects = list("RandomNormal" = initializer_random_normal))
mod5_2 <- load_model_hdf5(filepath = "Models\\Model2\\saved-model.70-0.99.hdf5",
                          custom_objects = list("RandomNormal" = initializer_random_normal))

# Extracting the test losses
test_score1_2 <- evaluate(mod1_2, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[
test_score2_2 <- evaluate(mod2_2, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[
test_score3_2 <- evaluate(mod3_2, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[
test_score4_2 <- evaluate(mod4_2, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[
test_score5_2 <- evaluate(mod5_2, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[

eval2 <- cbind(test_score1_2, test_score2_2, test_score3_2, test_score4_2, test_score5_2)
```

```r
#### Model 3 ####
### lr = 0.005, activation = relu, init = Gaussian
set_random_seed(931031)

model_conv3 <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "relu",
              input_shape = c(224,224,3),
              strides = c(2,2)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 256, kernel_size = c(3,3), activation = "relu")

# Random initiation on weights
model_conv3 %>%
  layer_flatten() %>%
  layer_dense(units = 64, activation = "relu", initializer_random_normal(mean = 0, stddev = 0.05)) %>%
  layer_dense(units = 4, activation = "softmax", initializer_random_normal(mean = 0, stddev = 0.05))

model_conv3 %>% compile(
  optimizer = optimizer_adam(learning_rate = 0.005),
  # Use one vector with factors 1-50 for classes instead of one-hot vectors
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)

history3 <- model_conv3 %>%
  fit(
    x = image_array_train, y = as.matrix(as.numeric(as.character(train_label))),
    batch_size = 32,
    epochs = 50,
    validation_split = 0.2,
    verbose = 2, #Only show loss/acc for the last batch in the epoch
    callbacks = callback_model_checkpoint(
      monitor = "val_loss",
      filepath = 'Models\\Model3\\saved-model.{epoch:02d}-{val_loss:.2f}.hdf5',
      mode = "min",
      save_best_only = FALSE)
    )

# Saving the history
History3_df <- history3 %>%
  as_tibble()
saveRDS(History3_df, file = "Hist\\History3.Rda")
hist_df3 <- readRDS("Hist\\History3.Rda")

plot3 <- hist_df3 %>%
  ggplot(aes(x = epoch, y = value, col = data)) +
  geom_point() +
  geom_point(data = hist_df3 %>%
               filter(epoch %in% c(03, 11, 23, 29, 50) & metric == "loss" & data == "validation"),
             pch = 21, size = 4, colour = "black") +
  geom_line() +
  facet_grid(rows = vars(metric), scales = "free")
saveRDS(plot3, file = "Figures\\Plot3.rdata")

# Saving the 5 chosen models
mod1_3 <- load_model_hdf5(filepath = "Models\\Model3\\saved-model.03-0.02.hdf5",
                          custom_objects = list("RandomNormal" = initializer_random_normal))
mod2_3 <- load_model_hdf5(filepath = "Models\\Model3\\saved-model.11-0.52.hdf5",
                          custom_objects = list("RandomNormal" = initializer_random_normal))
mod3_3 <- load_model_hdf5(filepath = "Models\\Model3\\saved-model.23-1.16.hdf5",
                          custom_objects = list("RandomNormal" = initializer_random_normal))
mod4_3 <- load_model_hdf5(filepath = "Models\\Model3\\saved-model.29-1.15.hdf5",
                          custom_objects = list("RandomNormal" = initializer_random_normal))
mod5_3 <- load_model_hdf5(filepath = "Models\\Model3\\saved-model.50-2.52.hdf5",
                          custom_objects = list("RandomNormal" = initializer_random_normal))

# Extracting the test losses
test_score1_3 <- evaluate(mod1_3, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[
test_score2_3 <- evaluate(mod2_3, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[
test_score3_3 <- evaluate(mod3_3, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[
test_score4_3 <- evaluate(mod4_3, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[
test_score5_3 <- evaluate(mod5_3, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[

eval3 <- cbind(test_score1_3, test_score2_3, test_score3_3, test_score4_3, test_score5_3)
```

```
#### Model 4 ####
### lr = 0.001, activation = tanh, init = Gaussian
set_random_seed(931031)

model_conv4 <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "tanh",
                input_shape = c(224,224,3),
```

```r
                 strides = c(2,2)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "tanh") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "tanh") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3,3), activation = "tanh") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 256, kernel_size = c(3,3), activation = "tanh")

# Random initiation on weights
model_conv4 %>%
  layer_flatten() %>%
  layer_dense(units = 64, activation = "tanh", initializer_random_normal(mean = 0, stddev = 0.05)) %>%
  layer_dense(units = 4, activation = "softmax", initializer_random_normal(mean = 0, stddev = 0.05))

model_conv4 %>% compile(
  optimizer = optimizer_adam(learning_rate = 0.001),
  # Use one vector with factors 1-50 for classes instead of one-hot vectors
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)

history4 <- model_conv4 %>%
  fit(
    x = image_array_train, y = as.matrix(as.numeric(as.character(train_label))),
    batch_size = 32,
    epochs = 50,
    validation_split = 0.2,
    verbose = 2, # Only show loss/acc for the last batch in the epoch
        callbacks = callback_model_checkpoint(
          monitor = "val_loss",
          filepath = 'Models\\Model4\\saved-model.{epoch:02d}-{val_loss:.2f}.hdf5',
          mode = "min",
          save_best_only = FALSE)
    )

# Saving the history
History4_df <- history4 %>%
  as_tibble()
saveRDS(History4_df, file = "Hist\\History4.Rda")
hist_df4 <- readRDS("Hist\\History4.Rda")

plot4 <- hist_df4 %>%
  ggplot(aes(x = epoch, y = value, col = data)) +
  geom_point() +
  geom_point(data = hist_df4 %>%
               filter(epoch %in% c(04, 09, 23, 29, 50) & metric == "loss" & data == "validation"),
             pch = 21, size = 4, colour = "black") +
  geom_line() +
  facet_grid(rows = vars(metric), scales = "free")
saveRDS(plot4, file = "Figures\\Plot4.rdata")
```

```r
# Saving the 5 chosen models
mod1_4 <- load_model_hdf5(filepath = "Models\\Model4\\saved-model.04-0.70.hdf5",
                          custom_objects = list("RandomNormal" = initializer_random_normal))
mod2_4 <- load_model_hdf5(filepath = "Models\\Model4\\saved-model.09-0.62.hdf5",
                          custom_objects = list("RandomNormal" = initializer_random_normal))
mod3_4 <- load_model_hdf5(filepath = "Models\\Model4\\saved-model.23-1.04.hdf5",
                          custom_objects = list("RandomNormal" = initializer_random_normal))
mod4_4 <- load_model_hdf5(filepath = "Models\\Model4\\saved-model.29-1.05.hdf5",
                          custom_objects = list("RandomNormal" = initializer_random_normal))
mod5_4 <- load_model_hdf5(filepath = "Models\\Model4\\saved-model.50-1.32.hdf5",
                          custom_objects = list("RandomNormal" = initializer_random_normal))

# Extracting the test losses
test_score1_4 <- evaluate(mod1_4, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[
test_score2_4 <- evaluate(mod2_4, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[
test_score3_4 <- evaluate(mod3_4, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[
test_score4_4 <- evaluate(mod4_4, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[
test_score5_4 <- evaluate(mod5_4, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[

eval4 <- cbind(test_score1_4, test_score2_4, test_score3_4, test_score4_4, test_score5_4)
```

```r
#### Model 5 ####
### lr = 0.001, activation = relu , init = lecun uniform
set_random_seed(931031)

model_conv5 <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "relu",
                input_shape = c(224,224,3),
                strides = c(2,2)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 256, kernel_size = c(3,3), activation = "relu")

# Random initiation on weights
model_conv5 %>%
  layer_flatten() %>%
  layer_dense(units = 64, activation = "relu", initializer_lecun_uniform()) %>%
  layer_dense(units = 4, activation = "softmax", initializer_lecun_uniform())

model_conv5 %>% compile(
  optimizer = optimizer_adam(learning_rate = 0.001),
  # Use one vector with factors 1-50 for classes instead of one-hot vectors
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)

history5 <- model_conv5 %>%
  fit(
    x = image_array_train, y = as.matrix(as.numeric(as.character(train_label))),
    batch_size = 32,
    epochs = 50,
    validation_split = 0.2,
    verbose = 2, # Only show loss/acc for the last batch in the epoch
    callbacks = callback_model_checkpoint(
      monitor = "val_loss",
      filepath = 'Models\\Model5\\saved-model.{epoch:02d}-{val_loss:.2f}.hdf5',
      mode = "min",
      save_best_only = FALSE)
    )

# Saving the history
History5_df <- history5 %>%
  as_tibble()
saveRDS(History5_df, file = "Hist\\History5.Rda")
hist_df5 <- readRDS("Hist\\History5.Rda")

plot5 <- hist_df5 %>%
  ggplot(aes(x = epoch, y = value, col = data)) +
  geom_point() +
  geom_point(data = hist_df5 %>%
               filter(epoch %in% c(08, 13, 23, 27, 50) & metric == "loss" & data == "validation"),
             pch = 21, size = 4, colour = "black") +
  geom_line() +
  facet_grid(rows = vars(metric), scales = "free")
saveRDS(plot5, file = "Figures\\Plot5.rdata")

# Saving the 5 chosen models
mod1_5 <- load_model_hdf5(filepath = "Models\\Model5\\saved-model.08-0.26.hdf5",
                          custom_objects = list("LecunUniform" = initializer_lecun_uniform))
mod2_5 <- load_model_hdf5(filepath = "Models\\Model5\\saved-model.13-0.33.hdf5",
                          custom_objects = list("LecunUniform" = initializer_lecun_uniform))
mod3_5 <- load_model_hdf5(filepath = "Models\\Model5\\saved-model.23-0.21.hdf5",
                          custom_objects = list("LecunUniform" = initializer_lecun_uniform))
mod4_5 <- load_model_hdf5(filepath = "Models\\Model5\\saved-model.27-0.52.hdf5",
                          custom_objects = list("LecunUniform" = initializer_lecun_uniform))
mod5_5 <- load_model_hdf5(filepath = "Models\\Model5\\saved-model.50-1.12.hdf5",
                          custom_objects = list("LecunUniform" = initializer_lecun_uniform))

# Extracting the test losses
test_score1_5 <- evaluate(mod1_5, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[
test_score2_5 <- evaluate(mod2_5, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[
test_score3_5 <- evaluate(mod3_5, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[
test_score4_5 <- evaluate(mod4_5, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[
test_score5_5 <- evaluate(mod5_5, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[

eval5 <- cbind(test_score1_5, test_score2_5, test_score3_5, test_score4_5, test_score5_5)
```

```r
#### Model 6 ####
### lr = 0.001, activation = relu , init = Glorot uniform
set_random_seed(931031)
```

```r
model_conv6 <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "relu",
                input_shape = c(224,224,3),
                strides = c(2,2)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3,3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = 256, kernel_size = c(3,3), activation = "relu")

# Random initiation on weights
model_conv6 %>%
  layer_flatten() %>%
  layer_dense(units = 64, activation = "relu", initializer_glorot_uniform()) %>%
  layer_dense(units = 4, activation = "softmax", initializer_glorot_uniform())

model_conv6 %>% compile(
  optimizer = optimizer_adam(learning_rate = 0.001),
  # Use one vector with factors 1-50 for classes instead of one-hot vectors
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)

history6 <- model_conv6 %>%
  fit(
    x = image_array_train, y = as.matrix(as.numeric(as.character(train_label))),
    batch_size = 32,
    epochs = 50,
    validation_split = 0.2,
    verbose = 2, #Only show loss/acc for the last batch in the epoch
    callback_model_checkpoint(
      monitor = "val_loss",
      filepath = 'Models\\Model6\\saved-model.{epoch:02d}-{val_loss:.2f}.hdf5',
      mode = "min",
      save_best_only = FALSE)
    )

# Saving the history
History6_df <-  history6 %>%
  as_tibble()
saveRDS(History6_df, file = "Hist\\History6.Rda")
hist_df6 <- readRDS("Hist\\History6.Rda")

plot6 <- hist_df6 %>%
  ggplot(aes(x = epoch, y = value, col = data)) +
  geom_point() +
  geom_point(data = hist_df6 %>%
               filter(epoch %in% c(09, 14, 16, 31, 50) & metric == "loss" & data == "validation"),
             pch = 21, size = 4, colour = "black") +
  geom_line() +
  facet_grid(rows = vars(metric), scales = "free")
saveRDS(plot6, file = "Figures\\Plot6.rdata")

# Saving the 5 chosen models
mod1_6 <- load_model_hdf5(filepath = "Models\\Model6\\saved-model.09-0.49.hdf5", , custom_objects=list(
mod2_6 <- load_model_hdf5(filepath = "Models\\Model6\\saved-model.14-0.51.hdf5", , custom_objects=list(
mod3_6 <- load_model_hdf5(filepath = "Models\\Model6\\saved-model.16-0.35.hdf5", , custom_objects=list(
mod4_6 <- load_model_hdf5(filepath = "Models\\Model6\\saved-model.31-0.32.hdf5", , custom_objects=list(
mod5_6 <- load_model_hdf5(filepath = "Models\\Model6\\saved-model.50-0.73.hdf5", , custom_objects=list(

# Extracting the test losses
test_score1_6 <- evaluate(mod1_6, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[
test_score2_6 <- evaluate(mod2_6, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[
test_score3_6 <- evaluate(mod3_6, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[
test_score4_6 <- evaluate(mod4_6, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[
test_score5_6 <- evaluate(mod4_6, image_array_test, as.numeric(as.character(test_label)), verbose = 0)[

eval6 <- cbind(test_score1_6, test_score2_6, test_score3_6, test_score4_6, test_score5_6)
```

This code produces the table showing the results of different models using different parameters, activation functions and weight initializations.

```
# Creating a table of the test losses for each model
evaluations <- rbind(eval1, eval2) %>%
  rbind(eval3) %>%
  rbind(eval4) %>%
  rbind(eval5) %>%
  rbind(eval6) %>%
  round(4)
colnames(evaluations) <- (c("Version 1", "Version 2", "Version 3", "Version 4", "Version 5"))
rownames(evaluations) <- c("Model 1", "Model 2","Model 3", "Model 4","Model 5", "Model 6")

saveRDS(evaluations, file = "eval_matrix.Rda")

# This shows our results from the 6 models
eval_matrix <- readRDS("eval_matrix.Rda")

table1 <- kable(eval_matrix, booktabs = T, align = "l", longtable = TRUE, caption = "Table 1: Rows show
  kable_styling(latex_options = "striped")

table1
```

Code for confusion matrix.

```
# Confusion Matrix
# 90.2% accuracy
# 14 of 143 wrong
pred_probs <- predict(mod1_6, image_array_test)
‘Predicted Label‘ <- pred_probs %>%
  max.col()-1
accuracy <- mean(pred_onehot == test_label)
‘True Label‘ = test_label
```

```r
conf_mat <- table(`Predicted Label`, `True Label`) %>%
  as_tibble()

conf_plot <- conf_mat %>%
  ggplot(aes(x = `True Label`, y = `Predicted Label`)) +
  geom_tile(aes(fill = n), colour = "black") +
  geom_text(aes(label = n)) +
  scale_fill_gradient(low = "light yellow", high = "orange")

saveRDS(conf_plot, file = "Figures\\conf_plot.png")
```

Code for producing the plot of a sample of the images that are always misclassified (Figure 4).

```r
#### Plotting a sample of the images that were always misclassified ####
# Extracting the images that were always classified incorrectly
test_set <- readRDS(file = "test_df.rdata") %>%
  rowid_to_column()
test_fp <- test_set$filepaths
test_labs <- test_set$labels

# Loading the images
wrong_test <- lapply(test_fp[c(1, 41, 65, 110)], load.image)

# Creating labels for each image for the plot
names <- c(paste("(a): ", test_labs[1]),
           paste("(b): ", test_labs[41]),
           paste("(c): ", test_labs[65]),
           paste("(d): ", test_labs[110]))

# Creating the plot
par(mfcol = c(2, 2), mar = c(0.2, 0.2, 1.3, 0.2), oma = rep(1, 4))
wrong_test %>%
  purrr::set_names(names) %>%
  purrr::map(as.raster, max = 255) %>%
  purrr::iwalk(~{plot(.x); title(.y, cex.main = 0.8)})

# 1 (classified as teal duck), 41 (classified as teal duck),
# 65 (classified as wood duck), 75 (classified as red thrush),
# 95 (classified as teal duck), 110 (classified as chukar)

# 1, 41, 65, 75, 95, 110 were misclassified
```