Dara Guadalupe Carrasquillo 01/23/2024

Dylan Rodríguez Flores

IronHack: Convoluted Neural Networks Project Report

I. Description of the chosen CNN architecture

Our model has the following architecture:

Model: "sequential_18"

Layer (type)	Output Shape	Param #
conv2d_85 (Conv2D)	(None, 32, 32, 32)	896
conv2d_86 (Conv2D)	(None, 32, 32, 32)	9,248
batch_normalization_30 (BatchNormalization)	(None, 32, 32, 32)	128
max_pooling2d_46 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_87 (Conv2D)	(None, 16, 16, 64)	18,496
conv2d_88 (Conv2D)	(None, 16, 16, 64)	36,928
batch_normalization_31 (BatchNormalization)	(None, 16, 16, 64)	256
max_pooling2d_47 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_89 (Conv2D)	(None, 8, 8, 128)	73,856

conv2d_90 (Conv2D)	(None, 8, 8, 128)	147,584
batch_normalization_32 (BatchNormalization)	(None, 8, 8, 128)	512
max_pooling2d_48 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten_18 (Flatten)	(None, 2048)	0
dropout_27 (Dropout)	(None, 2048)	0
dense_28 (Dense)	(None, 128)	262,272
dropout_28 (Dropout)	(None, 128)	0
dense_29 (Dense)	(None, 10)	1,290

Total params: 551,466 (2.10 MB)

Trainable params: 551,018 (2.10 MB)

Non-trainable params: 448 (1.75 KB)

II. Explanation of preprocessing steps

For this step we loaded the data and then used the following code:

Normalize the pixel values to a range of [0, 1]

x_train = x_train.astype('float32') / 255.0

x_test = x_test.astype('float32') / 255.0

Normalization: Scales pixel values to [0, 1] to improve model performance, stability, and convergence.

```
# Convert the labels into one-hot encoding
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

One-Hot Encoding: Converts integer labels into a vector format to avoid ordinal assumptions and ensure compatibility with the loss function.

III. Details of the training process (e.g., learning rate, batch size, number of epochs)

Details:

1. Model fit: The model seemed to stabilize around the 20 epochs so we left it at that number and configured early stopping to halt the training when the monitor variable did not improve further. Batch size started at 128 but we quickly figured that smaller sizes allowed for better learning and more robust results.
    a. Epochs: 20
    b. Batch size: 64
2. Early Stopping: Implemented to have the model halt all training procedures when the val_loss did not improve for three consecutive epochs.
    a. Monitor: val_loss
    b. Patience: 3
    c. Restore Best Weights: True
3. Learning Scheduler: Implemented to drop the learning rate by a factor of 0.5 to prevent overfitting in later epochs.
    a. Monitor: val_loss
    b. Factor: 0.5
    c. Patience: 2
    d. Min_lr: 0.0000001

IV. Results and analysis of model's performance

With the model displayed above we managed to achieve a test accuracy of 0.83 and a test loss of 0.58. Past this point, the model would need to increase in complexity to achieve higher results. Learning took a longer time than first stipulated, however, within the given time frame, it has achieved reasonable results for its low complexity. A confusion matrix was created in the testing, and we concluded that the pictures depicting animals seem to be giving the model much more trouble than any other category. Cats were most frequently incorrectly labeled as dogs and vice versa.

V. Best model

The model in this report outperformed all previous iterations, yielding a test accuracy score of 0.83 and a test loss of 0.58. The previous iterations had trouble with overfitting the data and subsequent revisions showed signs of underfitting. We knew our model would need to be somewhere in the middle and so we adjusted the layers until there was no further improvement.

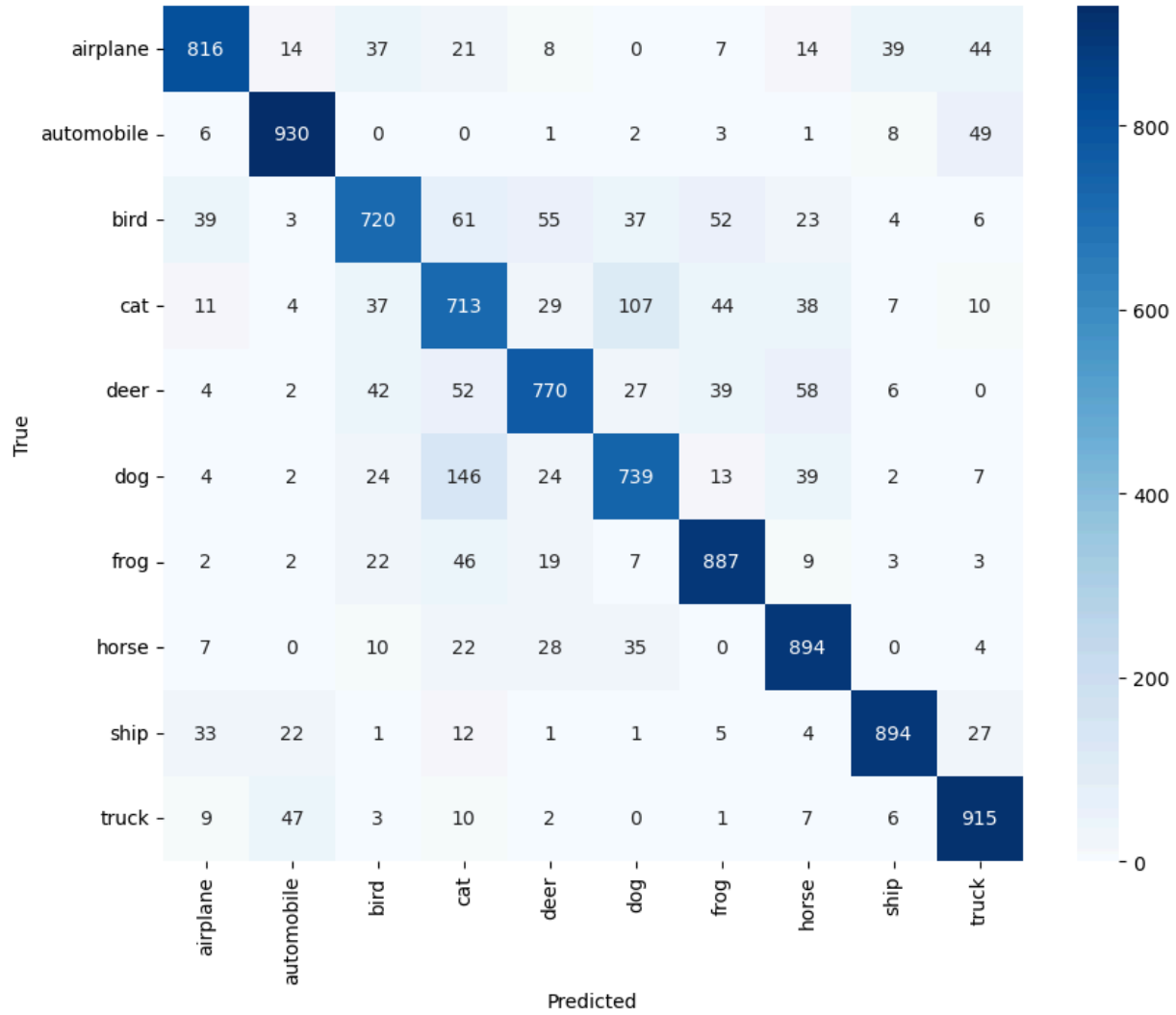
After that, we fine tuned the training details (epochs, batch size, etc.) to achieve higher results. This did improve the results further until we obtained the model displayed above.

VI. Insights gained from the experimentation process.

The most important insight gained from the experimentation was the importance of the architecture of the model and how the layers interact. Removing or adding a single element to one of the layers can yield dramatically different results on the testing. Furthermore, there are many ways in which the models can be fine tuned and optimized for better performance. Sometimes the most complex model is not necessarily the best one and it was very interesting to see that pacing the model and having it learn slowly yielded better results overall.

VII. Visualizations and diagrams

Confusion matrix of the test results:



VIII. Description of the chosen Transfer Learning Model:

Why ResNet?

ResNet (Residual Network) is often used for CIFAR-10 because of its efficiency, simplicity, and proven success in solving image classification problems.

Why ResNet18?

Faster model with reasonable accuracy.

Works well when there are computational or time constraints.

Great for people who are just starting with deep learning and want to experiment quickly.

Our ResNet18 model has the following architecture:

Model: Adjusted ResNet18

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 16, 16]           9,408
       BatchNorm2d-2          [-1, 64, 16, 16]             128
              ReLU-3          [-1, 64, 16, 16]               0
         MaxPool2d-4            [-1, 64, 8, 8]               0
            Conv2d-5            [-1, 64, 8, 8]          36,864
       BatchNorm2d-6            [-1, 64, 8, 8]             128
              ReLU-7            [-1, 64, 8, 8]               0
            Conv2d-8            [-1, 64, 8, 8]          36,864
```

BatchNorm2d-9	[-1, 64, 8, 8]	128
ReLU-10	[-1, 64, 8, 8]	0
BasicBlock-11	[-1, 64, 8, 8]	0
Conv2d-12	[-1, 64, 8, 8]	36,864
BatchNorm2d-13	[-1, 64, 8, 8]	128
ReLU-14	[-1, 64, 8, 8]	0
Conv2d-15	[-1, 64, 8, 8]	36,864
BatchNorm2d-16	[-1, 64, 8, 8]	128
ReLU-17	[-1, 64, 8, 8]	0
BasicBlock-18	[-1, 64, 8, 8]	0
Conv2d-19	[-1, 128, 4, 4]	73,728
BatchNorm2d-20	[-1, 128, 4, 4]	256
ReLU-21	[-1, 128, 4, 4]	0
Conv2d-22	[-1, 128, 4, 4]	147,456
BatchNorm2d-23	[-1, 128, 4, 4]	256
Conv2d-24	[-1, 128, 4, 4]	8,192
BatchNorm2d-25	[-1, 128, 4, 4]	256
ReLU-26	[-1, 128, 4, 4]	0
BasicBlock-27	[-1, 128, 4, 4]	0
Conv2d-28	[-1, 128, 4, 4]	147,456
BatchNorm2d-29	[-1, 128, 4, 4]	256
ReLU-30	[-1, 128, 4, 4]	0
Conv2d-31	[-1, 128, 4, 4]	147,456
BatchNorm2d-32	[-1, 128, 4, 4]	256
ReLU-33	[-1, 128, 4, 4]	0
BasicBlock-34	[-1, 128, 4, 4]	0
Conv2d-35	[-1, 256, 2, 2]	294,912
BatchNorm2d-36	[-1, 256, 2, 2]	512
ReLU-37	[-1, 256, 2, 2]	0
Conv2d-38	[-1, 256, 2, 2]	589,824
BatchNorm2d-39	[-1, 256, 2, 2]	512
Conv2d-40	[-1, 256, 2, 2]	32,768
BatchNorm2d-41	[-1, 256, 2, 2]	512
ReLU-42	[-1, 256, 2, 2]	0
BasicBlock-43	[-1, 256, 2, 2]	0
Conv2d-44	[-1, 256, 2, 2]	589,824

```
      BatchNorm2d-45           [-1, 256, 2, 2]             512
             ReLU-46           [-1, 256, 2, 2]               0
          Conv2d-47            [-1, 256, 2, 2]         589,824
      BatchNorm2d-48           [-1, 256, 2, 2]             512
             ReLU-49           [-1, 256, 2, 2]               0
       BasicBlock-50           [-1, 256, 2, 2]               0
          Conv2d-51           [-1, 512, 1, 1]       1,179,648
      BatchNorm2d-52           [-1, 512, 1, 1]           1,024
             ReLU-53           [-1, 512, 1, 1]               0
          Conv2d-54           [-1, 512, 1, 1]       2,359,296
      BatchNorm2d-55           [-1, 512, 1, 1]           1,024
          Conv2d-56           [-1, 512, 1, 1]         131,072
      BatchNorm2d-57           [-1, 512, 1, 1]           1,024
             ReLU-58           [-1, 512, 1, 1]               0
       BasicBlock-59           [-1, 512, 1, 1]               0
          Conv2d-60           [-1, 512, 1, 1]       2,359,296
      BatchNorm2d-61           [-1, 512, 1, 1]           1,024
             ReLU-62           [-1, 512, 1, 1]               0
          Conv2d-63           [-1, 512, 1, 1]       2,359,296
      BatchNorm2d-64           [-1, 512, 1, 1]           1,024
             ReLU-65           [-1, 512, 1, 1]               0
       BasicBlock-66           [-1, 512, 1, 1]               0
AdaptiveAvgPool2d-67           [-1, 512, 1, 1]               0
          Linear-68                 [-1, 10]           5,130
================================================================
Total params: 11,181,642
Trainable params: 11,181,642
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 1.29
Params size (MB): 42.65
Estimated Total Size (MB): 43.95
----------------------------------------------------------------
```

Changes made to ResNet18:

1. Replaced the First Convolution Layer

The original ResNet-18 was built to work with large pictures, so it starts by looking at the image with a 7x7 kernel and skips steps (stride of 2), which makes the image much smaller right away.

For CIFAR-10, we replaced this kernel with a 3x3 kernel and made it look at every step (stride of 1), so the image stays the same size (32x32).

Why is this important?

CIFAR-10 images are tiny (32x32 pixels). If we shrink them too quickly, the model loses important details.

Using a smaller "window" and not skipping steps helps the model keep more information from the images, making it better at recognizing things like cats, dogs, and airplanes.

2. Removed the MaxPooling Layer After the First Convolution

In the original ResNet-18, there's a MaxPooling layer after the first convolution. This layer downsamples the image by taking the largest value in a region, reducing the image size.

Why is this important?

CIFAR-10 images are already very small (32x32). Using MaxPooling right after the first layer would shrink the image too quickly and lose critical information.

By removing the MaxPooling layer, the model retains more details from the image, improving its ability to classify the small CIFAR-10 images accurately.

3. Replaced the Fully Connected Layer

The original fully connected layer in ResNet-18 outputs predictions for 1,000 classes. For CIFAR-10, this was replaced with a fully connected layer that outputs predictions for 10 classes.

Why is this important?

CIFAR-10 has only 10 categories (like cats, dogs, airplanes, etc.), so the model needs to output probabilities for just these 10 classes.

IX. Explanation of preprocessing steps

For this step we loaded the data and then used the following code:

```
# Normalize the pixel values to a range of [0, 1]
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
```

Normalization: Scales pixel values to [0, 1] to improve model performance, stability, and convergence.

```
# Convert the labels into one-hot encoding
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

One-Hot Encoding: Converts integer labels into a vector format to avoid ordinal assumptions and ensure compatibility with the loss function.

X. Details of the training process (e.g., learning rate, batch size, number of epochs)

Details:

-Model fit: The model seemed to stabilize around the 10 epochs so we left it at that number and configured early stopping to halt the training when the monitor variable did not improve further. Batch size stayed at 64 given the results we had gotten when changing batch size in ResNet50.
-Epochs: 10
-Batch size: 64
-Early Stopping:
  Monitor: val_loss
  Patience: 3
  Restore Best Weights: True
-Learning Scheduler:
  Monitor: val_loss
  Factor: 0.001

XI. Results and analysis of model's performance

With ResNet, we experimented on the 18 and the 50. We previously chose 50 because it had the ability of getting high accuracy for CIFAR-10, but we realized it was too complex for our time and computer system. So we researched further and decided to use ResNet18. On the first try, ResNet18 outperformed our model.

XII. Best model

Of the 2 models we used, ResNet18 had better results.

XIII. Insights gained from the experimentation process.

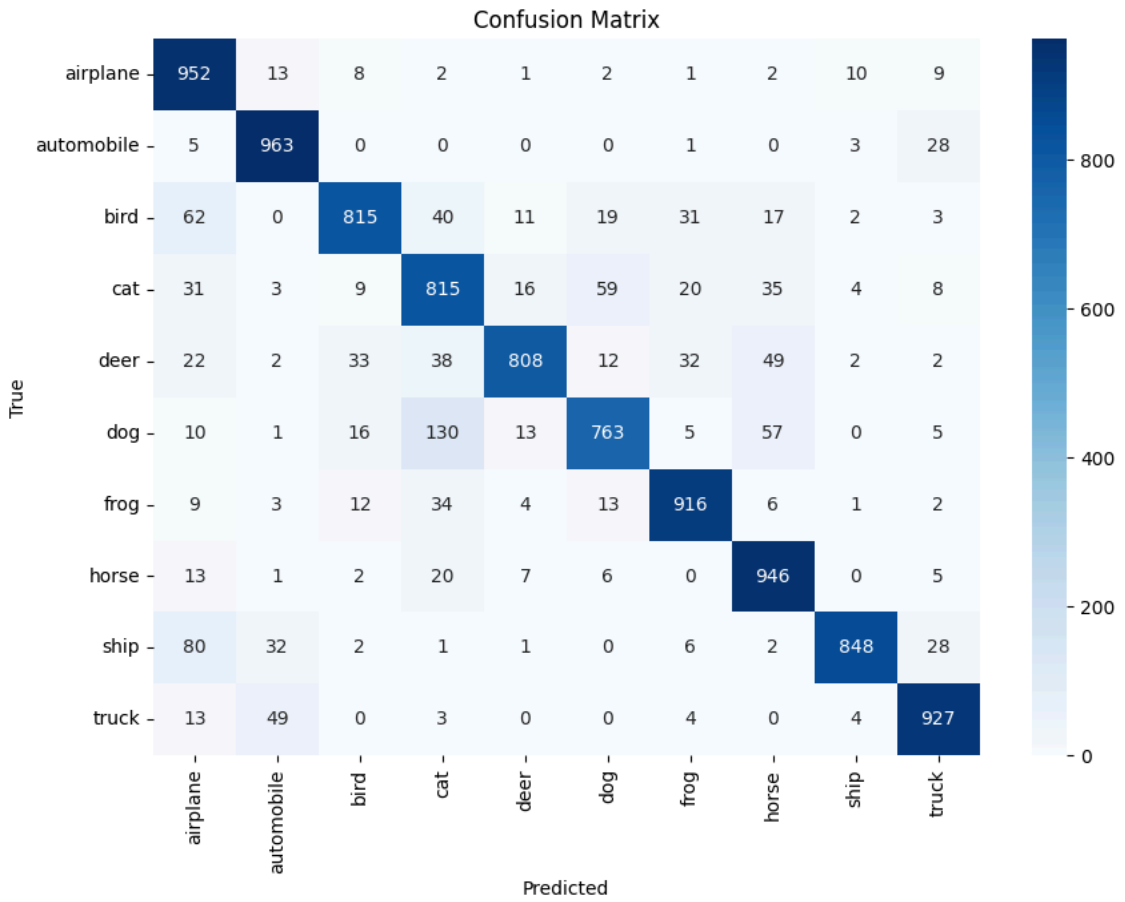
Probably the most important insight we gained from this project is that pretrained models can be helpful to get better test results in less time than training your own model.

Another important insight gained from the experimentation was the importance of the architecture of the model and how the layers interact. Removing or adding a single element to one of the layers can yield dramatically different results on the testing.

Furthermore, there are many ways in which the models can be fine tuned and optimized for better performance. Sometimes the most complex model is not necessarily the best one, this showed in both our models.

XIV. Visualizations and diagrams

Confusion matrix of the test results of ResNet18:


Confusion Matrix
airplane 952 13 8 2 1 2 1 2 10 9
automobile 5 963 0 0 0 0 1 0 3 28
bird 62 0 815 40 11 19 31 17 2 3
cat 31 3 9 815 16 59 20 35 4 8
deer 22 2 33 38 808 12 32 49 2 2
dog 10 1 16 130 13 763 5 57 0 5
frog 9 3 12 34 4 13 916 6 1 2
horse 13 1 2 20 7 6 0 946 0 5
ship 80 32 2 1 1 0 6 2 848 28
truck 13 49 0 3 0 0 4 0 4 927
True
airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck
Predicted
800
600
400
200
0