March 16, 2022

Fish Family Classification using Deep Learning

Data analysis and preparation

Sam Tihen
CMPSCI5390
University of Missouri - St. Louis

https://www.overleaf.com/read/sqkwpscdbrsk

1 Introduction

Deep Learning can be used to classify what type of fish is pictured in a photograph. This project's focus will be on classifying five different Linnaean families of fish. There are multiple species within each family, and there are significant variations in size, shape, and coloration between the species.

1.1 Project Motivation

I created a website called MyScubaDives.com in 2011, and the primary purpose of the website was to create a database of all dive sites and dive operators around the world, and allow people to log their dives and share images of what they saw. Around this time, image ”tagging” and facial recognition was being popularized by Facebook, and I thought it would be interesting to attempt to use AI to help people identify what fish where in their photos. The site ultimately failed to gain popularity, and I lost interest in the project as usage and revenue never really grew. This class project seems like a good opportunity to explore what I failed to do a decade ago.

2 Data

2.1 Data Collection

To create the dataset, online image searches were used to find and download more than 1000 images, split across five different Linnaean families of fish.

The photographs are of multiple species within each family, and are taken from various angles and with various lighting conditions.

Each image is limited to one family of fish, but multiple individuals within the same family may be present.

2.2 Data Preprocessing

The images were imported into Apple Photos and exported as JPEG High Quality with a maximum height and width of 500px to reduce the overall dataset file size and limit any file format issues.

2.3 Data Distribution

1. Clownfish — Item Count: 201
2. Lionfish — Item Count: 215
3. Parrotfish — Item Count: 210
4. Triggerfish — Item Count: 215
5. Butterflyfish — Item Count: 210

There are 1051 images total.

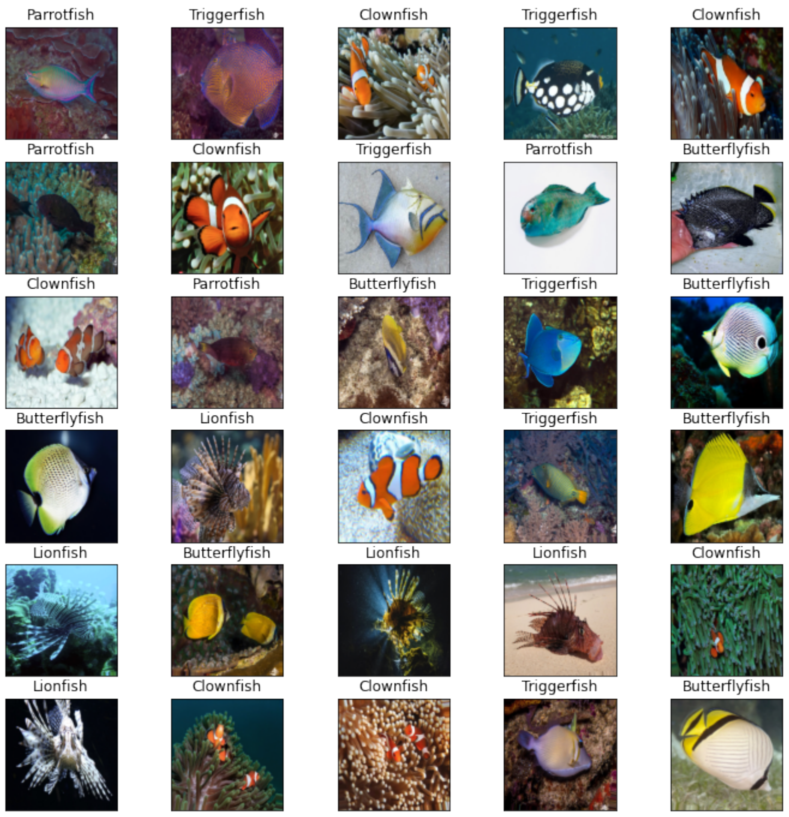


Figure 1: Dataset example

2.4 Data Normalization

The ImageDataGenerator included with Keras will be used to re-scale each image with a 1/255 ratio. I believe the color will be relevant to effective categorization, and will keep all color channels.

2.5 Project Data Processing Code

The data for this project was manipulated using Google Colab and Python.

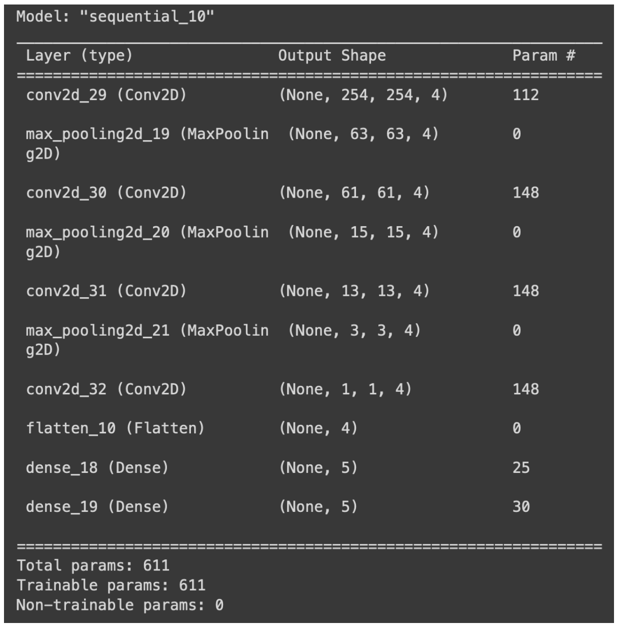
URL: https://colab.research.google.com/drive/12tNNGjWssaP43wdXAOhgY41qJ8AIDNhn

3 Model Architecture Design

3.1 Building an Overfitting Model

To create a good baseline model, I tested and evaluated a few different architectures and plotted their learning curves. The following subsections represent the attempts that were at the cusp of overfitting while keeping total parameter count as low as possible. I found that using maxpooling between layers helped keep the parameters lower, and used it between layers in all architectures.

3.1.1 4x4x4x4 Architecture



```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_29 (Conv2D)          (None, 254, 254, 4)       112

 max_pooling2d_19 (MaxPoolin  (None, 63, 63, 4)        0
 g2D)

 conv2d_30 (Conv2D)          (None, 61, 61, 4)         148

 max_pooling2d_20 (MaxPoolin  (None, 15, 15, 4)        0
 g2D)

 conv2d_31 (Conv2D)          (None, 13, 13, 4)         148

 max_pooling2d_21 (MaxPoolin  (None, 3, 3, 4)          0
 g2D)

 conv2d_32 (Conv2D)          (None, 1, 1, 4)           148

 flatten_10 (Flatten)        (None, 4)                 0

 dense_18 (Dense)            (None, 5)                 25

 dense_19 (Dense)            (None, 5)                 30

=================================================================
Total params: 611
Trainable params: 611
Non-trainable params: 0
```

Figure 2: 4x4x4x4 Model Summary

This model failed to train past approximately 70% accuracy. I believe the number of parameters was insufficient to train.

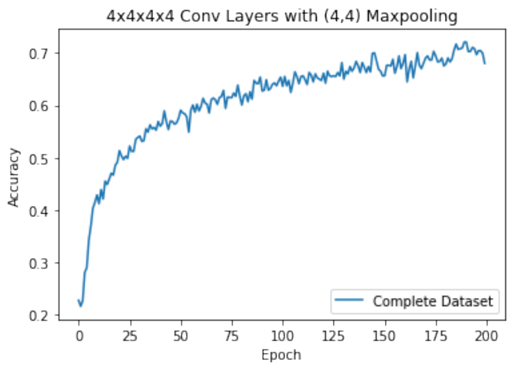
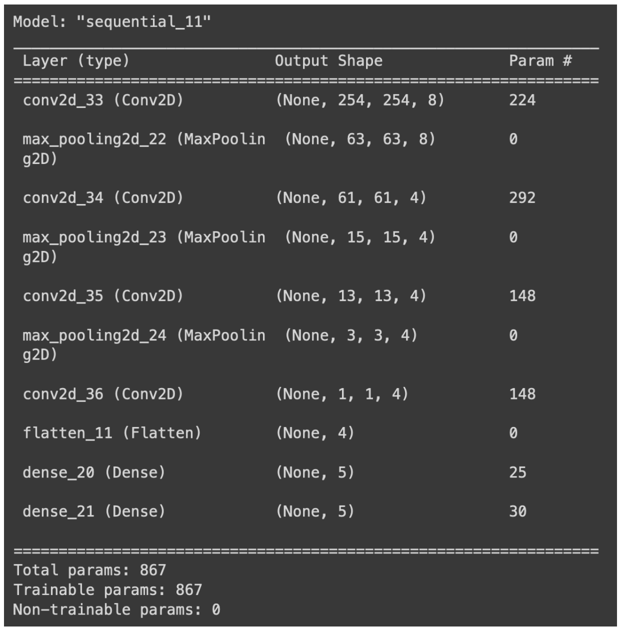


Figure 3: 4x4x4x4 Accuracy Learning Curve

3.1.2 8x4x4x4 Architecture



```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_33 (Conv2D)          (None, 254, 254, 8)       224       
                                                                 
 max_pooling2d_22 (MaxPoolin  (None, 63, 63, 8)        0         
 g2D)                                                            
                                                                 
 conv2d_34 (Conv2D)          (None, 61, 61, 4)         292       
                                                                 
 max_pooling2d_23 (MaxPoolin  (None, 15, 15, 4)        0         
 g2D)                                                            
                                                                 
 conv2d_35 (Conv2D)          (None, 13, 13, 4)         148       
                                                                 
 max_pooling2d_24 (MaxPoolin  (None, 3, 3, 4)          0         
 g2D)                                                            
                                                                 
 conv2d_36 (Conv2D)          (None, 1, 1, 4)           148       
                                                                 
 flatten_11 (Flatten)        (None, 4)                 0         
                                                                 
 dense_20 (Dense)            (None, 5)                 25        
                                                                 
 dense_21 (Dense)            (None, 5)                 30        
                                                                 
=================================================================
Total params: 867
Trainable params: 867
Non-trainable params: 0
```

Figure 4: 8x4x4x4 Model Summary

This model failed to train past approximately 80% accuracy. Again, I believe the number of parameters was insufficient to train.

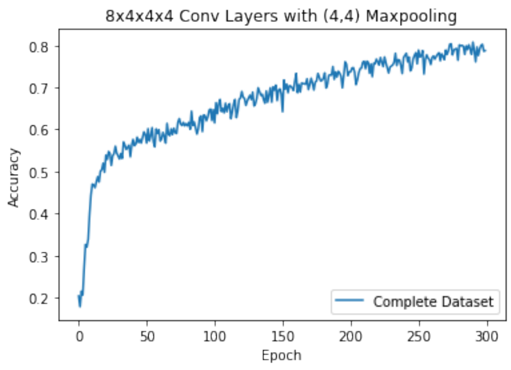
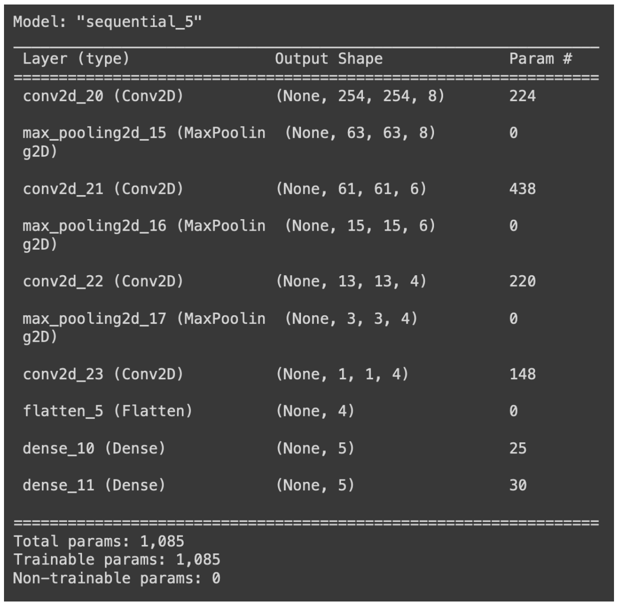


Figure 5: 8x4x4x4 Accuracy Learning Curve

3.1.3 8x6x4x4 Architecture


```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_20 (Conv2D)          (None, 254, 254, 8)       224

 max_pooling2d_15 (MaxPoolin  (None, 63, 63, 8)        0
 g2D)

 conv2d_21 (Conv2D)          (None, 61, 61, 6)         438

 max_pooling2d_16 (MaxPoolin  (None, 15, 15, 6)        0
 g2D)

 conv2d_22 (Conv2D)          (None, 13, 13, 4)         220

 max_pooling2d_17 (MaxPoolin  (None, 3, 3, 4)          0
 g2D)

 conv2d_23 (Conv2D)          (None, 1, 1, 4)           148

 flatten_5 (Flatten)         (None, 4)                 0

 dense_10 (Dense)            (None, 5)                 25

 dense_11 (Dense)            (None, 5)                 30

=================================================================
Total params: 1,085
Trainable params: 1,085
Non-trainable params: 0
```

Figure 6: 8x6x4x4 Model Summary

This was the first model that I was able to train to approximately 100%.

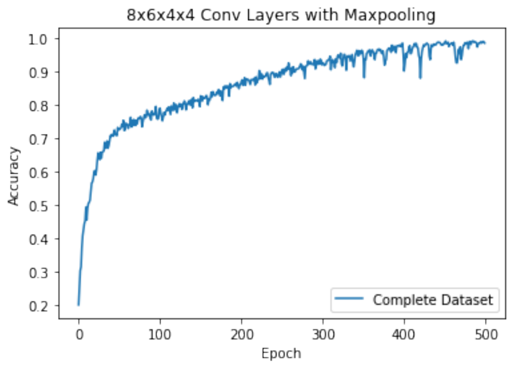


Figure 7: 8x6x4x4 Accuracy Learning Curve

3.1.4 Architecture Comparisons

When I compared the architectures I tried, I saw that the models with an insufficient number of trainable parameters simply stalled out before they were able to overfit. The 8x6x4x4 architecture has just enough parameters available to overfit, and will be a good baseline to use when attempting to build a model that can generalize and do well on validation and test data.

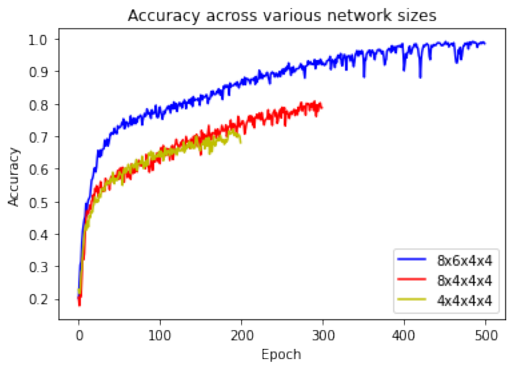


Figure 8: Accuracy Comparison

3.1.5 Output as Input

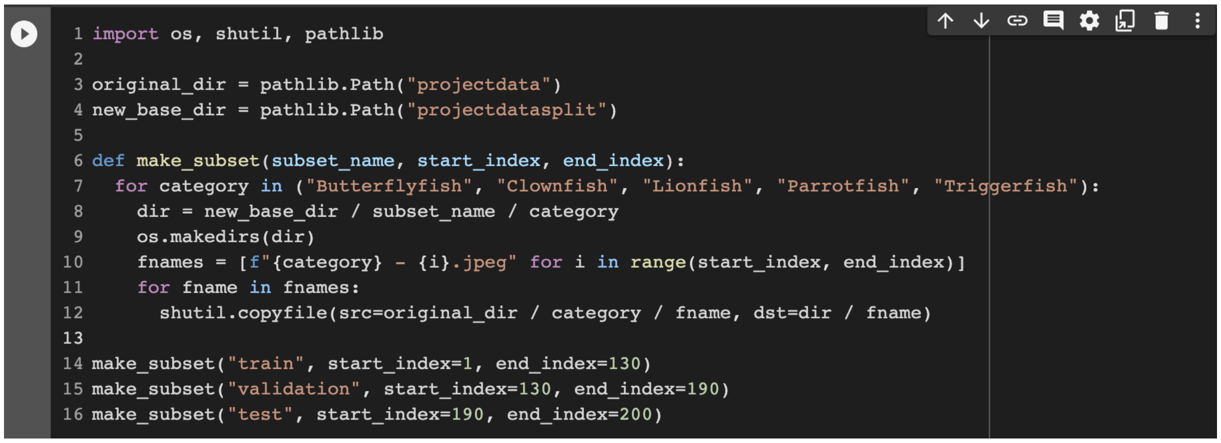
When I added the output as input channels, I was able to use a much smaller network to train to 100%.

3.2 Using a Validation Dataset

After finding that smallest architecture most capable of overiftting was an 8x6x4x4 model, I had a good baseline to start validating based on a seperate dataset.

3.2.1 Dataset splitting

To create a validation dataset, I chose to use the python function shown in our textbook, which allowed me to keep my initial dataset unchanged and. The function simply creates new directories and moves numerically identified pictures to a new directory. This allows me to change the percentages of the datasets more easily if I encounter an issue, but does not affect the reproducibility.



```python
import os, shutil, pathlib

original_dir = pathlib.Path("projectdata")
new_base_dir = pathlib.Path("projectdatasplit")

def make_subset(subset_name, start_index, end_index):
  for category in ("Butterflyfish", "Clownfish", "Lionfish", "Parrotfish", "Triggerfish"):
    dir = new_base_dir / subset_name / category
    os.makedirs(dir)
    fnames = [f"{category} - {i}.jpeg" for i in range(start_index, end_index)]
    for fname in fnames:
      shutil.copyfile(src=original_dir / category / fname, dst=dir / fname)

make_subset("train", start_index=1, end_index=130)
make_subset("validation", start_index=130, end_index=190)
make_subset("test", start_index=190, end_index=200)
```

Figure 9: Data Splitting

3.2.2 Validation Architecture Research

Once the data was split, I tested a number of different architectures to evaluate validation accuracy and loss.

I implemented early stopping if there was no improvement of validation accuracy for 20 epochs. I

implemented checkpointing to keep the best model based on validation accuracy.

3.2.3 Model Sizes with Validation Dataset

This section includes experimental results around the network sizes based on validation accuracy.

The overall result is that larger network sizes did not seem to improve validation accuracy, with the exception of a 64x64x32x4 network which ended up with approximately the same results as the 8x6x4x4 network.

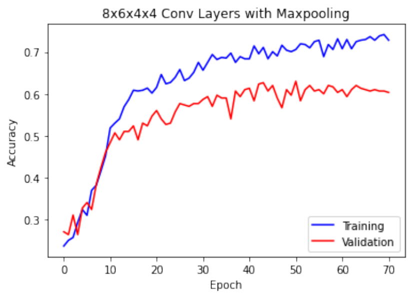


Figure 10: 8x6x4x4 Training vs Validation

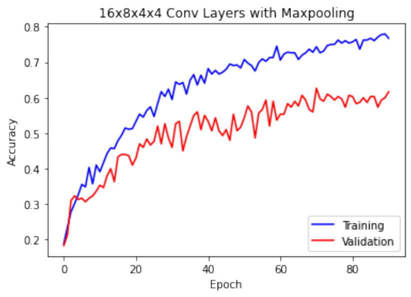


Figure 11: 16x8x4x4 Training vs Validation

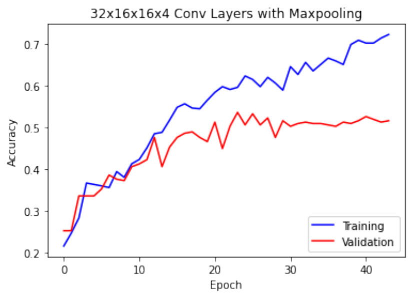


Figure 12: 32x16x16x4 Training vs Validation

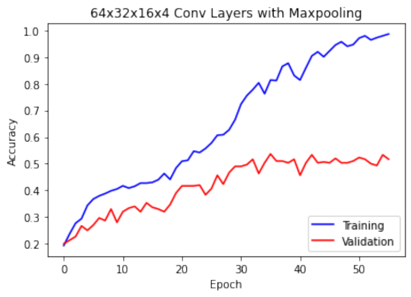


Figure 13: 64x32x16x4 Training vs Validation

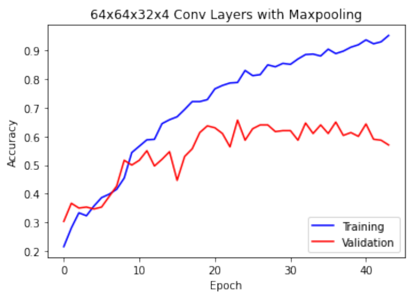


Figure 14: 64x64x32x4 Training vs Validation

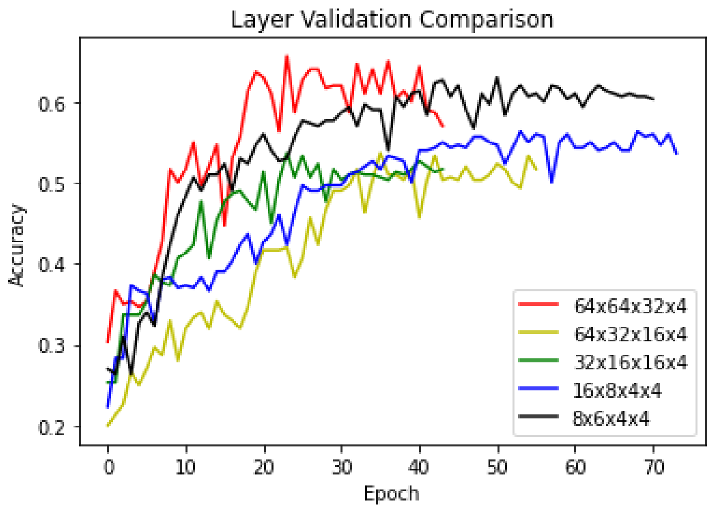


Figure 15: Validation Comparison

3.2.4 Results based on Test Dataset

Evaluating against the test dataset resulted in similar findings. Most of the models resulted in 50% accuracy. This is bigger than the baseline accuracy of 20%. The 64x32x16x4 model was the winner at 66% accuracy, but the validation accuracy was lower and I do not think that it is an accurate representation of it's predictive ability.

3.3 Image Augmentation

To further improve my model, I explored a number of different image augmentation possibilities. I compared flipping, shearing, shifting, zooming, and a combined augmentation.

3.3.1 Horizontal Flipping

This type of augmentation did not result in significant improvement of my model.

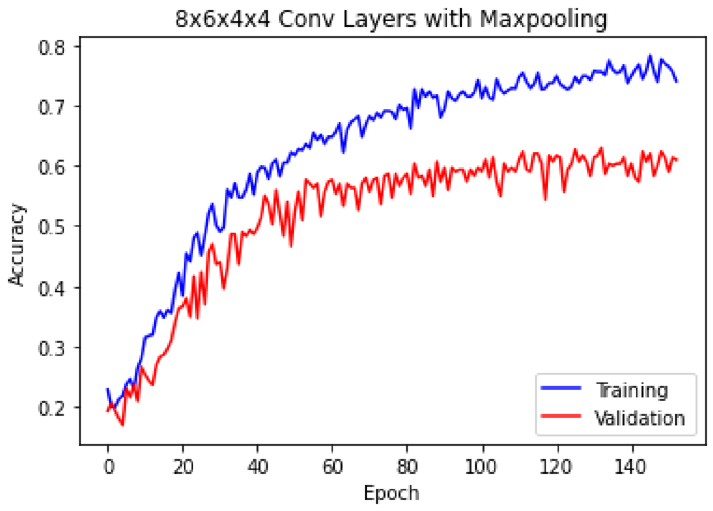


Figure 16: Horizontal Flipping

3.3.2 Zooming

This type of augmentation did not result in significant improvement of my model, and it actually seems to have made it very slightly worse. I allowed a zoom amount of 0.2.

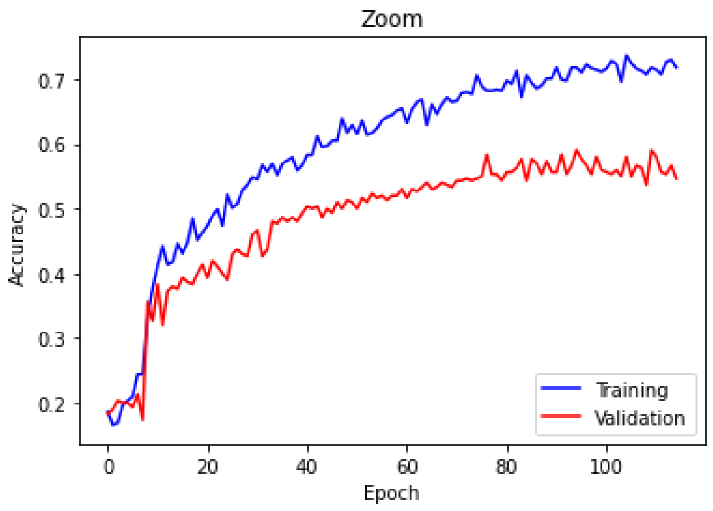


Figure 17: Zooming

3.3.3 Shearing

I used a shear amount of 0.2. This made my model noticeably worse.

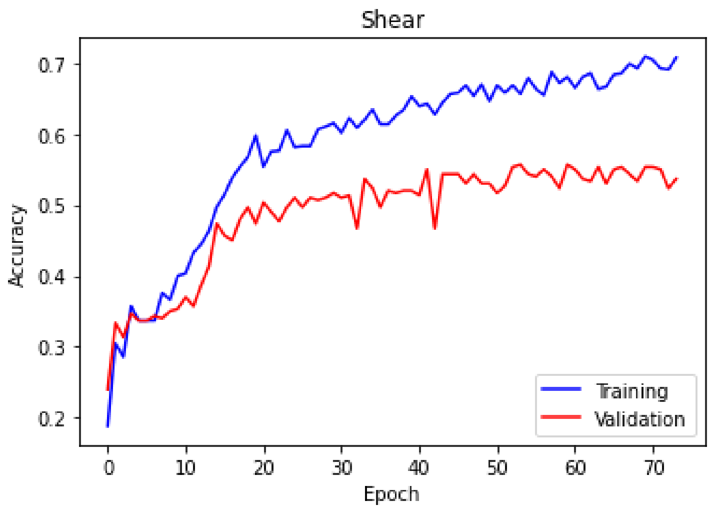


Figure 18: Shearing

3.3.4 Shifting

I allowed a height and width shift of 0.2 each. This type of augmentation resulted in significantly worse outcomes.

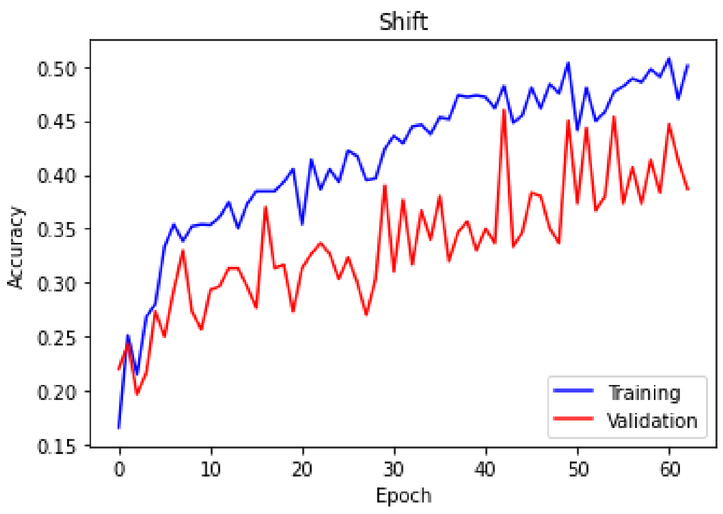


Figure 19: Shifting

3.3.5 Rotation

I allowed a rotation range of 0.4. This type of augmentation did not result in significant improvement of my model.

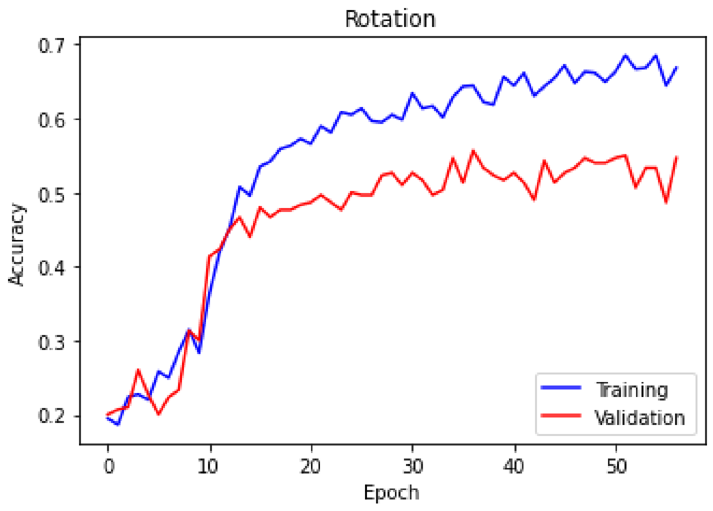


Figure 20: Rotation

3.3.6 Combined Augmentation

I attempted to combine all previous augmentations. This ultimately did about as good as te model with no augmentation.

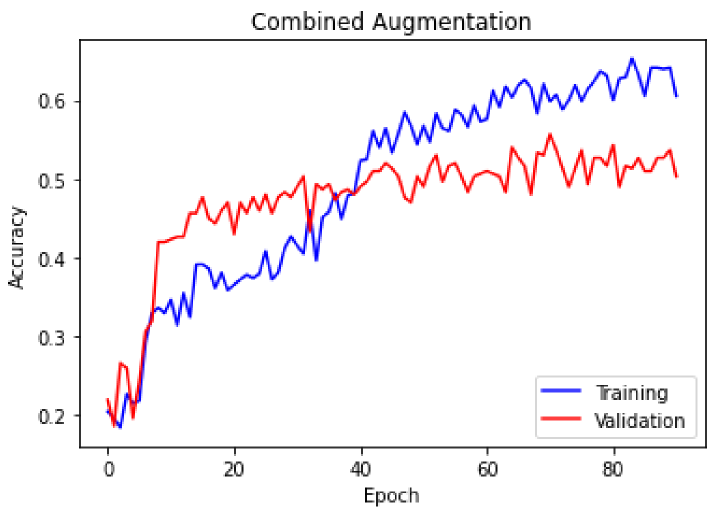


Figure 21: Combined

3.3.7 Flip and Rotate Augmentation

I next attempted to combine only the best performing augmentations: flip and rotate.

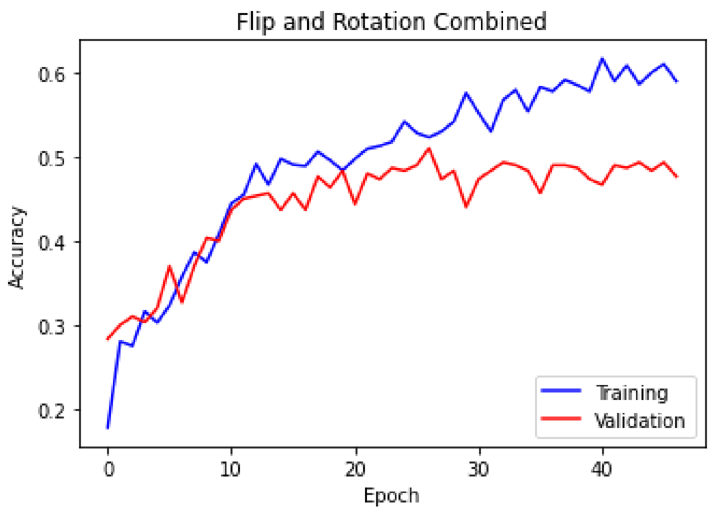


Figure 22: Combined

3.3.8 Augmentation Type Comparison

I also compared these result to each other. Horizontal Flipping was the overall best augmentation method.

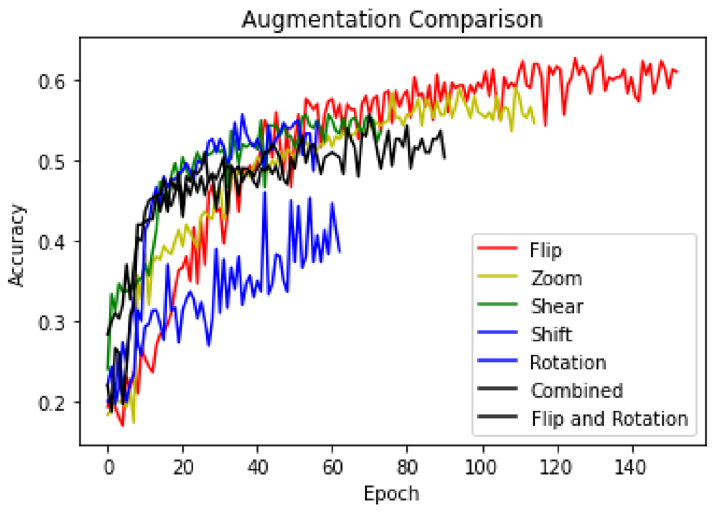


Figure 23: Augmentation Comparison