February 19, 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.

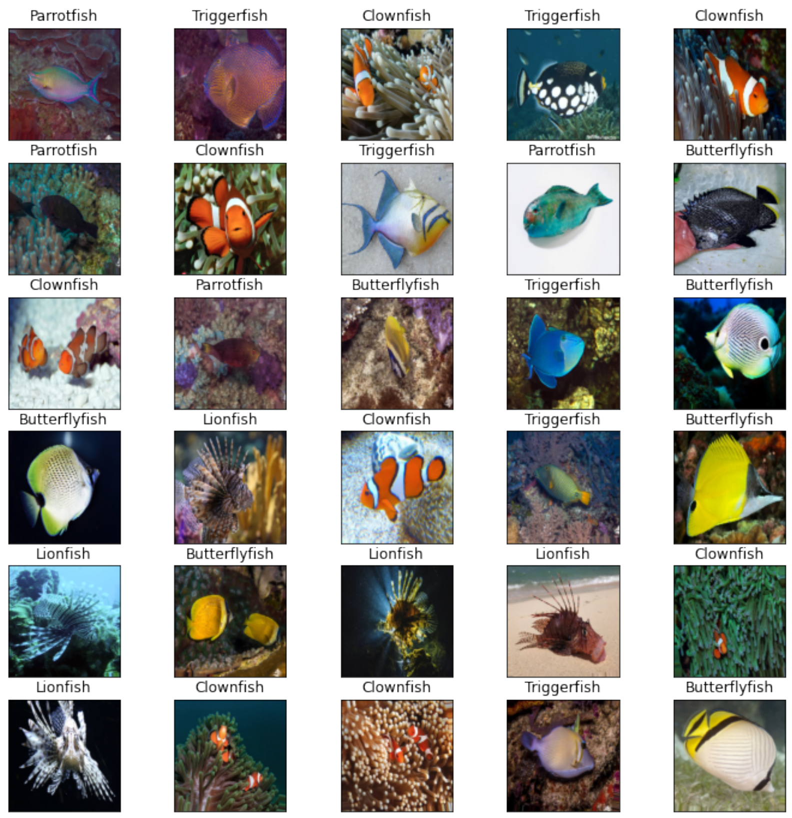


Figure 1: Dataset example

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.

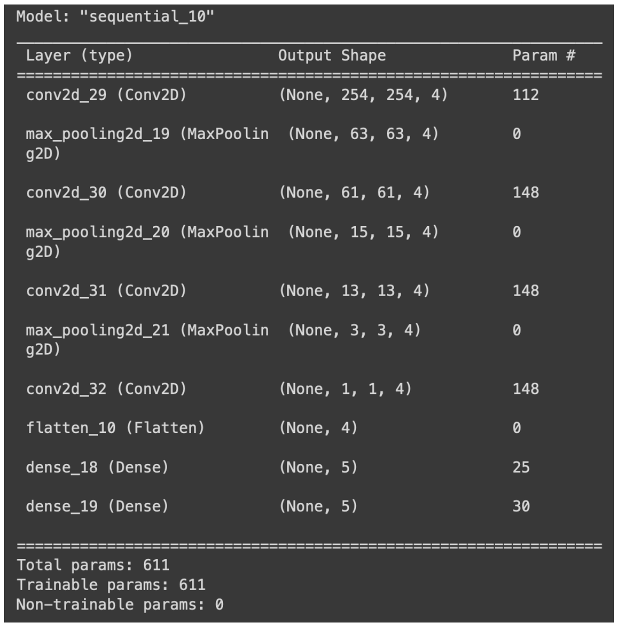
URL: https://colab.research.google.com/drive/12tNNGjWssaP43wdXAOhgY41qJ8AIDNhn

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.

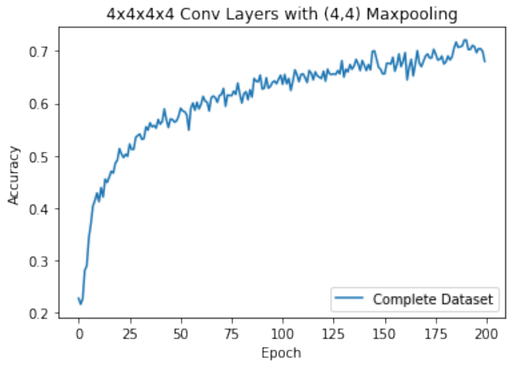
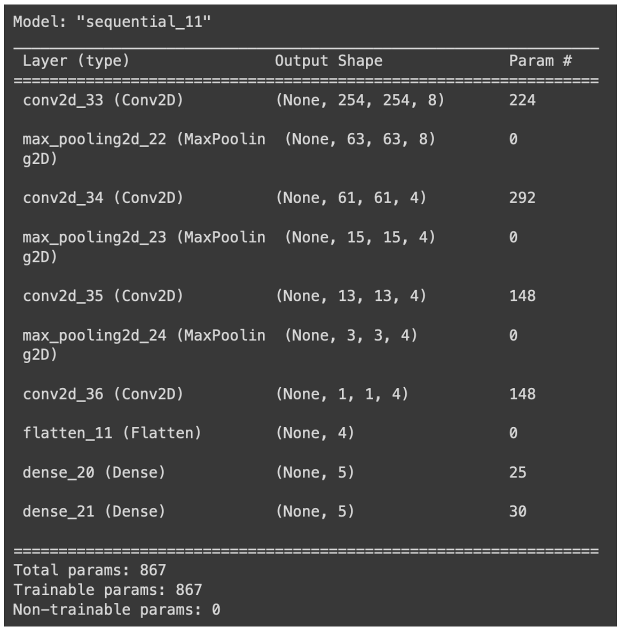


Figure 3: 4x4x4x4 Accuracy Learning Curve

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.

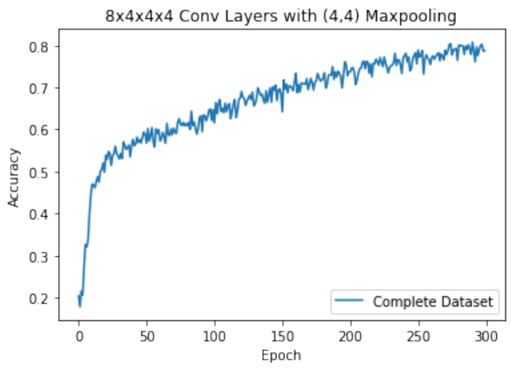
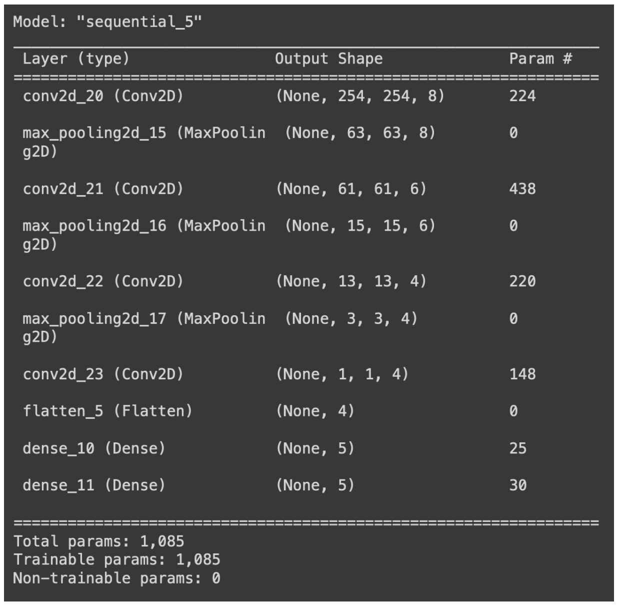


Figure 5: 8x4x4x4 Accuracy Learning Curve

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%.

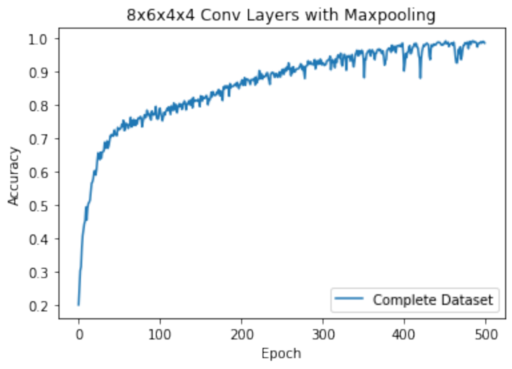


Figure 7: 8x6x4x4 Accuracy Learning Curve

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.

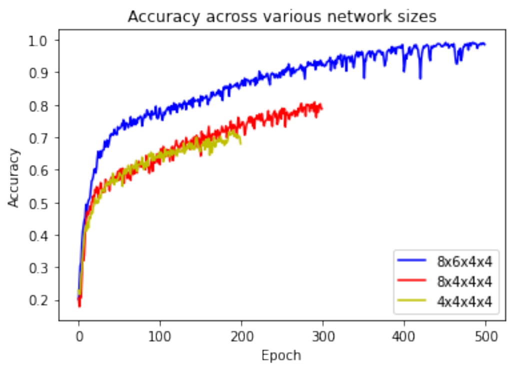


Figure 8: Accuracy Comparison

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%.