Andy Votava summer internship 2022

# RPS

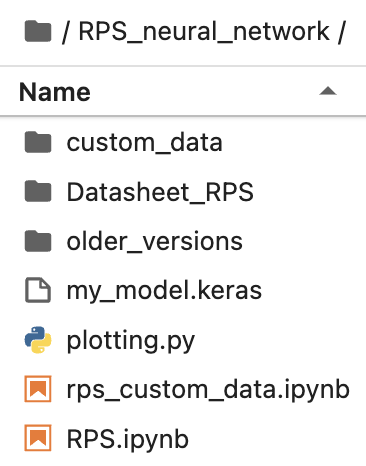
## Summary

This Is a 2d cnn model designed for demo purposes and my first neural network I have created. It is trained to distinguish between the hand signs for Rock, Paper, and Scissors to create a more engaging experience between the neural network and the user. It is 42,883 parameters, pruned to 90%, quantized to 7 bits with 1 integer bit, and refined to increase accuracy in unideal environments.

## Folder structure

There are two main folders for this project RPS\_neural\_network and rps\_refine.

**RPS\_neural\_network** was used on the EAF to define, quantize, train on a [download dataset](https://public.roboflow.com/classification/rock-paper-scissors/1), and complete most of the pruning.



In this folder you will find:

RPS.ipynb - which is the Jupiter notebook file where the model is defined, pruned, and trained.

rps\_custom\_data.ipynb - where I attempted to train the model on only the custom data set I created from Ben’s camera. This did not work well due to the small size and bad variety of the custom dataset.

Plotting.py - a script for plotting the ROC curve

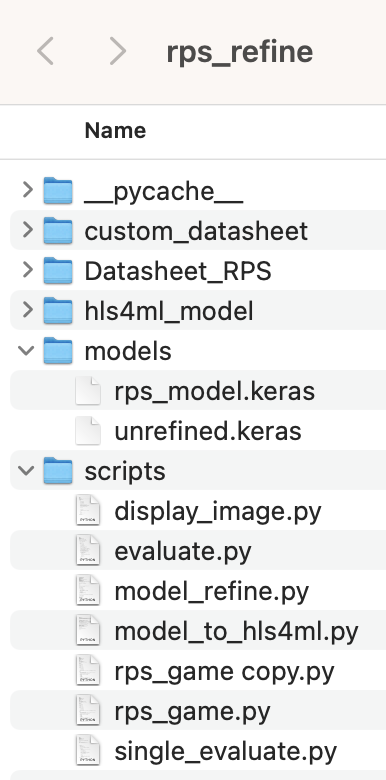
My\_model.keras - the saved unrefined model pruned to 80%

Older\_versions - Saved older models

Datasheet\_RPS - Where the downloaded dataset is stored (NOTE: I took this out of the RPS\_nueral\_network folder to conserve space however there is a copy in the rps\_refine might have to change a few path names)

Custom\_data - Where the custom dataset is stored

**Rps\_refine** was used locally on my machine to refine the model on my custom dataset and to finish pruning



This folder contains:

Custom\_datasheet - my custom dataset

Datasheet\_RPS - the downloaded dataset

Hls4ml\_model - The HLS4ML synthesis of the model (was too big to fit on FPGA)

Models - where models are stored. Unrefined is the model just trained on the downloaded dataset and rps\_model is the final model

Scripts - Where all the python scripts are stored (NOTE: Might need to change argument of cv2.VideoCapture to change camera, press q to quit out of video capture)

Display\_image.py - a function that displays the camera image through opencv

Evaluate.py - a function to continuously predict the image that comes through the camera

Model\_refine.py - a script to take the unrefined model and refine it

Model\_to\_hls4ml.py - a script to run the refined model through HLS4ML synthesis

Rps\_game - script to play RPS with neural network

Single\_evaluate.py - Runs through all custom data images and predicts on them

## Pruning

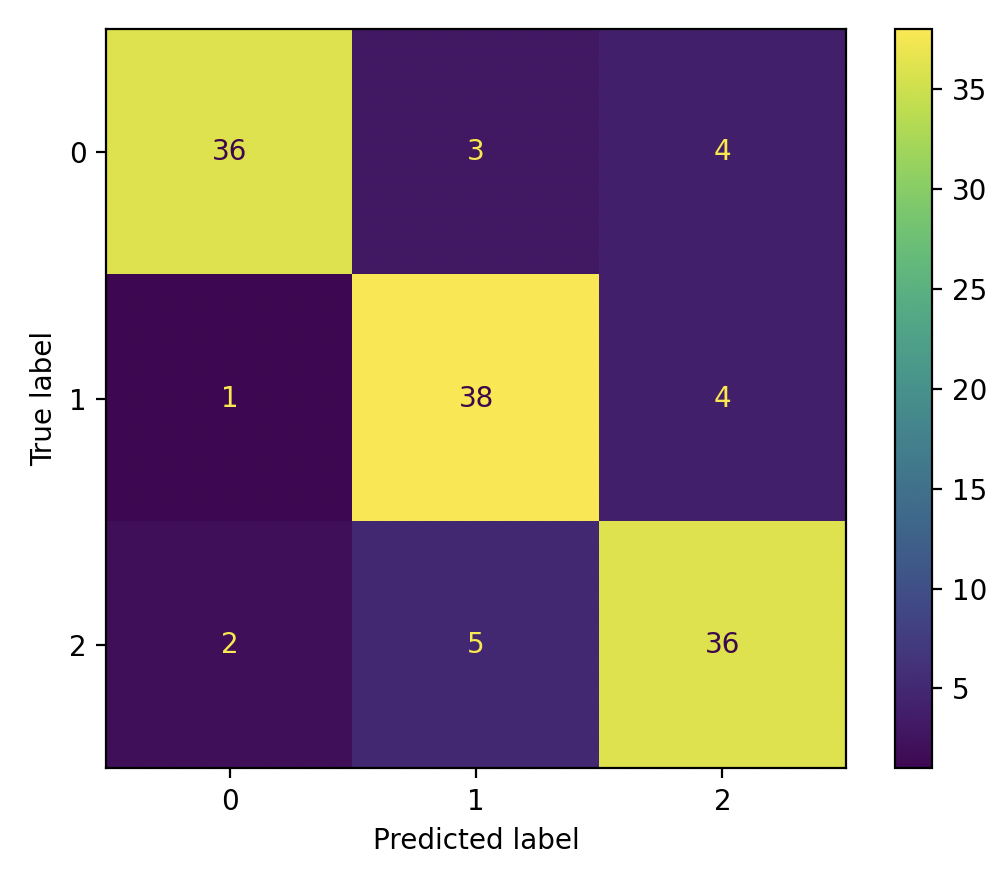
Through the pruning process I experimented with many different parameters and found that pruning 80% on the downloaded dataset and then while refininging, pruning the last 10% yielded the best results on the custom dataset. I also tested soly pruning the model on the download dataset, the custom dataset, and with different pruning ratios between the datasets and all returned inferior results.

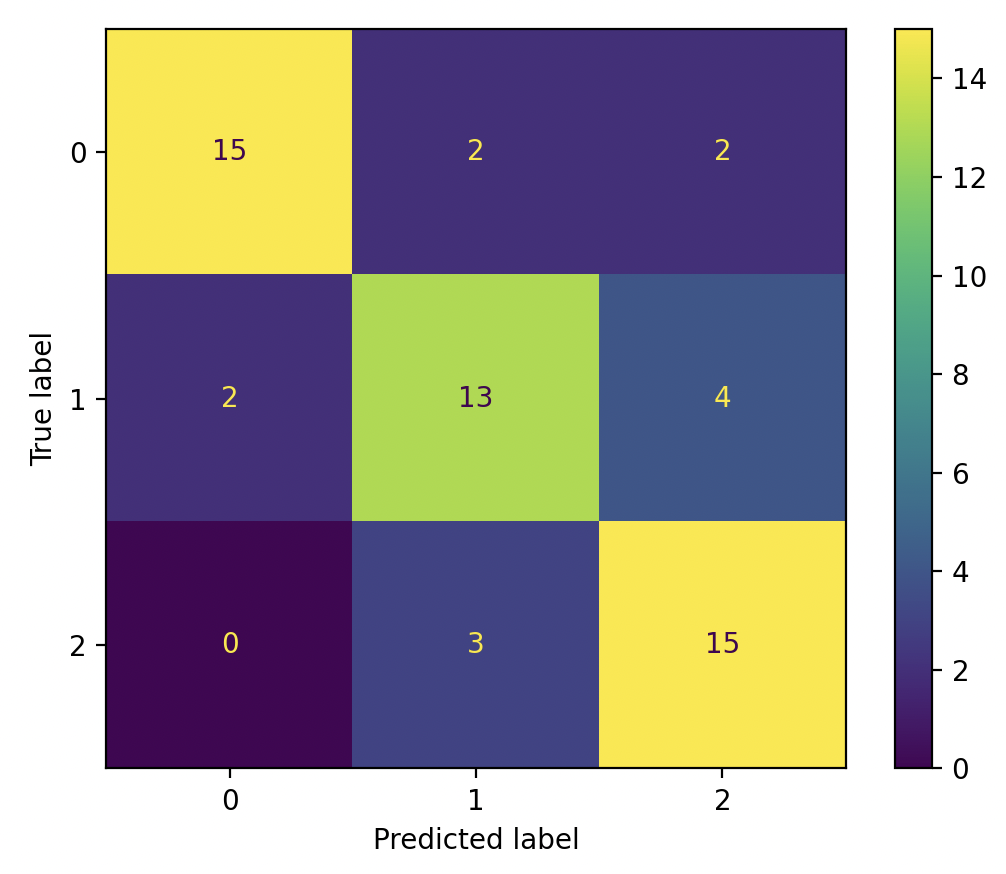
## Quantization

To figure out the optimal quantization I started with <10, 1> and worked my way down until the model accuracy had a heavy drop off around 5-6 bits. I went back one bit to 7 before the accuracy dropped off and settled there.

## Results

The complete model is 48,223 Parameter pruned 90% and <7, 1> quantized. It possesses a 83% training accuracy and a 72% validation accuracy on the custom dataset with confusion matrices shown below.

Training Validation



## Improvements

Even though this model performs pretty well, there are some improvements that I believe will push the model to ~90% accuracy. The first one being a more diverse custom dataset to refine the model on. As of writing this, the dataset consists of 185 images of both Ryan’s and my hand. We created this dataset in Ben’s room over the course of a few hours and there are certain shortcomings within this dataset. Since Ryan and I are of similar ethnicity the model refinse on one skin tone and may not produce similar results for different skintones. To fix this I would include images of hands with many different skintones to make the model more robust. The other issue with this dataset is that we used the same camera and lighting to create the dataset. This could cause problems when moving to a different location with a different lighting. Solving this issue is as simple as varying the lighting and the camera focus throughout the dataset, molding the model to be more robust.

The smaller improvement that could be made is a learning rate scheduler. After using it on my YOLO model, it is clear a learning rate scheduler allows the loss to settle deeper into a minimum. I am confident that implementing these changes onto the RPS cnn would improve the accuracy.

## HLS4ML synthesis

I did run it through HLS4ML synthesis but it predicted that the model was too big to fit onto an FPGA. I was more focusing on the YOLO model at this time so no further sythasiss were completed.

# 

# 

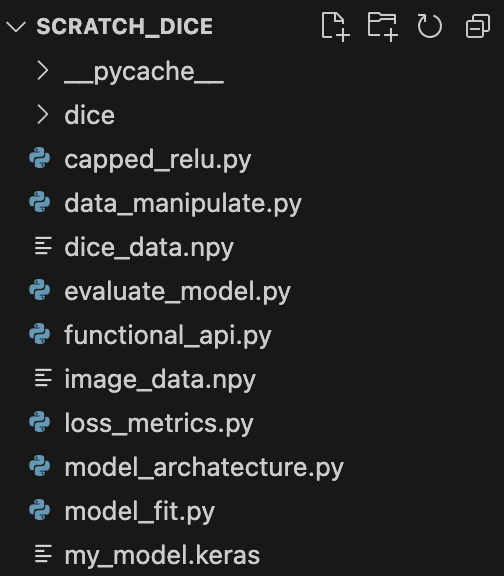
# YOLO model

## Summary

This is a custom YOLO model I have been working on with the goal of being pruned, quantized and uploaded to a FPGA. This model mimics the architecture of the tinissimo YOLO model but slightly upscaled to match our dataset better. While I was not able to get it fully functional, I learned a ton and it is not far off from operation.

## Folder Structure

There is only one folder for this project named scratch\_dice

****

This folder contains:

Capped\_relu.py - Custom activation layer

Data\_manipulate.py - gets raw data into label and image arrays

Dice\_data.npy - the numpy array of the labels

Evaluate\_model.py - prints model prediction on image

Functional\_api.py - a Keras functional api of the model. It will be useful when the YOLO model starts working and you have to pass the output layer through different activation functions

Image\_data.npy - numpy array of images

Loss\_metrics.py - Defines loss functions and metrics for training

Model\_archatecture.py - Defines model architecture

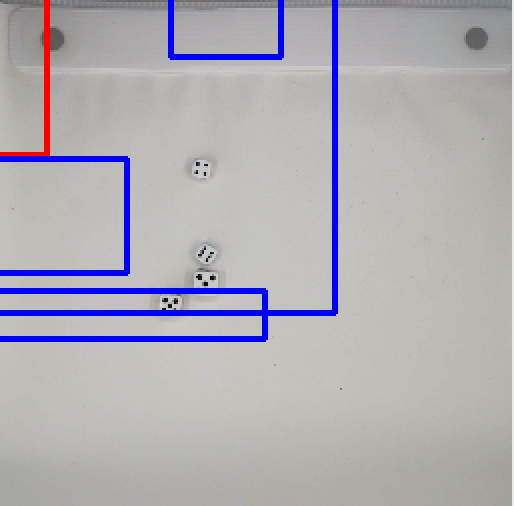
Model\_fit.py - file for model training

My\_model.keras - final saved model

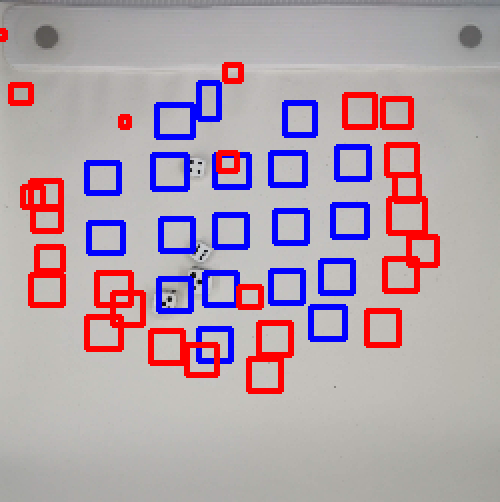
This model mimics the architecture of the Tinyissimo YOLO model but slightly upscaled to match our dataset better. I started

## Results/ Issues

The first issue I ran into was during training the loss function would not converge enough to predict sensible bounding boxes. The model would end up making predictions that look chaotic (below).



However after implementation of a learning rate scheduler and the YOLOv1 loss function. Javi and I got it predicting much more reasonable bounding box predictions.



From this image we can see that the bounding boxes are reasonably dice sized. The blue boxes indicate a higher confidence score. The issue with this image is the model is tailoring itself to the dataset. It is predicting higher confidence scores closer to the center and very low confidence scores at the edges. It also is averaging the dice's position over all the images in the dataset to minimize the loss function and It predicts the same bounding box positions for every image. It is as if it can not make a connection between the image data and the label data so it ends up just averaging the label data to minimize the loss function. I have spent days digging through the code to try to fix this error and despite my efforts I was not able to come up with a solution to this problem before I have to leave.

## Improvements

Once the YOLO model makes good predictions one Improvement to make is comparing label bounding boxes to see which one to append to a certain grid cell. When I create the labels for each grid cell I take a label bounding box and see if it falls within certain grid cells and append the label to those grid cells. If a grid cell contains two dice, the grid cell label will contain whichever dice was iterated through first. It would be a good idea to compare the bounding box’s of said dice to see which one has more overlap in the grid cell and append that one. This will make the model more robust at processing dice that are close to each other.

Another way to improve accuracy of dice that are close together is having each grid cell predict multiple bounding boxes. This would allow multiple dice in a single bounding box to be detected.

# Internship timeline (roughly)

June 20th - June 26th, Setting up everything and learning about basic neural networks and terminology

June 26th - July 10th, learning python on codecademy, going through HLS4ML tutorial, and learning more advanced neural network features and implantation in python through pytorch and keras.

July 10th - July 24th creating RPS cnn to learn about neural networks and to serve demo purposes

July 24th - Aug 11 create YOLO model, lots of debugging

# Thank you!

I want to say thank you to Ben, Javi, and Maira for helping me through the plethora of questions I had about coding, neural networks, and the lab. Also thank you to Lorenzo and Nhan for giving me this opportunity to go and learn about something that was brand new to me. I came into the lab barely knowing python and I am leaving with so many applicable skills and knowledge in machine learning. Thank you for your time and I look forward to working with all of you in the future.