Implementation:

For this project we decided to use a custom built CNN, with a bottleneck fire module that helps to reduce noise from the data. We used this module repeatedly in our CNN structure. We used dense, pooling and 2d convolution layers.

Results:

We tried a bunch of different optimizers, but ultimately the adam optimizer performed the best. We tried to use some of Keras's prebuilt CNN structures, including VGG19, and ResNet50. We tried transfer learning with these models using the Imagenet initial weights, but in the end we could only get them up to a 67% accuracy on the validation set, so we decided to focus on building our own CNN from scratch. Our own model worked the best and we got it to predict on the test set with an 88% accuracy.

Challenges:

One of the hardest parts was just finding the best hyperparameters to make the model better. Sometimes we sort of just had to guess and then wait a while for the data to train to see if the change actually improved the model. Another challenge was overfitting the model. As we trained the model the accuracy did improve, but when we trained for 20 epochs the difference between the train and validation accuracy started to grow, which indicated that we were starting to overfit.

Conclusions:

We developed a much deeper understanding of CNNs and how they work. We were quite disappointed that the transfer learning did not produce better results, because from what we read online, the imagenet pretrained weights are supposed to be quite effective for these sort of image recognitions sorts of problems.

Individual Contributions:

Zoe:

I pasted our 2dimensional images on top of each other to simulate rgb images so that we could use our train data in Keras's prebuilt models that take in rgb data. I spent a lot of time trying to get out transfer learning models to work. I focused mostly on the vgg19 model, because I was familiar with it from my own final project. I played around with optimizers and learning rates. I used the prebuilt model as a base and added an average pooling, dense and dropout layer to improve accuracy. I also tried freezing different sections of the CNN and training the parts individually and together, but in the end the models were not as accurate as the custom model

that Jacob had designed. I helped Jacob, finetune his fire module, by adding more layers and adjusting increasing the filter size to further reduce noise and improve accuracy.

Jacob:

I set up the notebook for importing the competition datasets into Colab using the Kaggle API. I started with a proof-of-concept TFLearn DNN that achieved 75% accuracy. Shortly after, I rewrote the notebook to use the better-supported TensorFlow.Keras library instead. I batched the dataset and applied the TPU strategy with reference to Tensorflow documentation. That sped up the training process by a factor of about 200. Training a neural net with about a million parameters for fifteen epochs now takes only around two minutes. That allowed us to experiment with a large number of hyperparameters and optimize our model.