Convolutional neural network

We previously attempted a regular back-propagation neural net using the average (average intensity of Red, Blue and Green pixel values) R-G-B values of each training Pokémon image. We also analyzed the coloration of different types of Pokémon through PCA and quick google searches of "X type Pokémon". We saw certain trends, a lot of fire type Pokémon had a lot of Red and Orange hues, a large number of them had flames on their bodies, Thunder type Pokémon often had cackling lightning or a distinct yellow body with black stripes. Thus, we decided to attempt a Convolutional Neural network. Just average color wasn't a great measure, since this did not capture features like flames or wings (for winged Pokémon). Flying Pokémon came in various colors, Bug Pokémon while often Green and yellow, clashed with the Leaf type Pokémon that were also frequently green. We needed our algorithm to "see" more features than simply color intensities. The positioning and shapes of those colors also mattered.

We also put the images (data points) through a fair amount of pre-processing to achieve the best results. In order to first even implement our model, we had to one-hot-encode the Pokémon types. This was relatively simple, we encoded the 18 Pokémon types using the categorical labels function on the type column in the training data. This changed our y matrix from a 601x1 matrix of values between 1 and 18 to a 601x19 matrix, with values of 0 and 1. If the y value before encoding was 1 then the 1st (technically second since the encoding started with a 0th column) column had a 1, and the rest were all 0.

We then augmented all our images. The reason we augmented images, was to be able to capture "features" more than just coloration and overall shape. Through augmentation our Neural network would be better able to identify features like "tails" or "wings" rather than just using color and overall shape (Even still the net relied primarily on coloration, as we will see in the intermediate layer activations in further below)

The augmentation phase included random rotations, small width and height shifts, zooms and shears and horizontal flips.

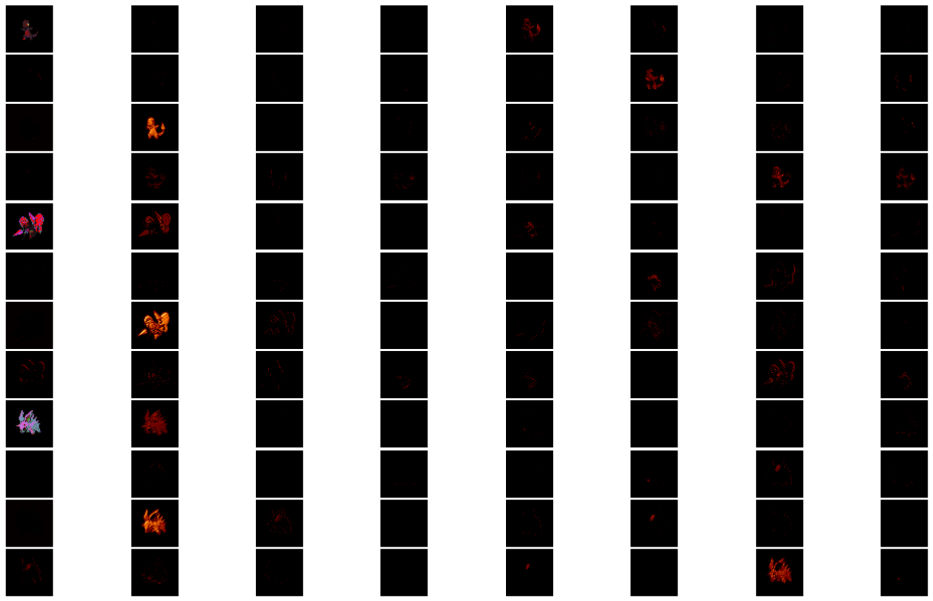
Thus, we implemented the convolutional neural network. Our net architecture had two CNN phases before final classification. The structure is shown below:

- Convolution Layer: 48 5x5 filters
- Max Pooling Layer: 2x2, with a (2,2) stride
- Convolution Layer: 96 5x5 filters
- Max Pooling Layer: 2x2, with a (2,2) stride
- Flatten Layer: Getting out data ready for the fully connected layers
- Dense Layer: 96 neurons
- Output Layer: 19 neurons (since our one hot encoding made 19 classes instead of 18) with a softmax activation function

Our inputs were the 96x96x3 matrices made up of the R, G and B values respectively of each pixel. These were values between 0 and 1, and so were already normalized.

We ran our model with a batch size of 64 for 30 epochs. As expected each epoch gave us a reduced loss, and usually increased accuracy as our model got trained further. We did attempt to use larger batch sizes, but it seemed that we did not gain any more accuracy (through better gradient estimation), and just lost speed. 64 was the sweet spot between higher accuracy and diminishing returns. 30 epochs also were the sweet spot between overfitting our model. I attempted a 20 and 60 epoch run, and the 20-epoch model was not accurate enough, while the 60-epoch tried to fit the data too much to the training model, while I achieved a 58% score when testing it against only the training data, the cross-validation score was considerable lower, at 24%. When I submitted it to Kaggle, I achieved a lower score of 25% in comparison to my ~30% score with 30 epochs.

Additionally, while observing the predictions on testing and layer activation images, I found that it seemed certain data-points in our training data were creating "false positives". For some of the data points I plotted the layer activations in sub-plot form, as seen below. (Note: this is a randomly selected small sample of the layer activations.)



If we look at the sub-plot, Charmander for example clearly fires up the neurons looking at the Red and Orange parts in the (3,2) image. This led to our fire type classification. Additionally, the (2,6) and (3,6) images show our model getting fired up about the orange and red in the tail. Beedrill seems to fire up the model at the wings and yellows aspects. This lead to our bug-type classification.

Looking at these I found that the Psychic type Pokémon did not really show a discernable pattern over the aspects that fired up our model. In fact, some of the training data, like the Abra, Kadabra, Alakazam line was overpowering the yellow from the Lightning type Pokémon. So, I removed those data points for better classification and improved the accuracy by about 2%.