Original Proposal

Title:

Pokémon Type Recognition

Members:

Brady Bartusiak & Ivy Truong

Problem Statement:

The goal is to determine a Pokémon's type(s), out of the 18 possible types, based off of its image. The types each Pokémon have to be classified into are Normal, Fire, Grass, Water, Ice, Flying, Fighting, Ground, Rock, Steel, Ghost, Dark, Fairy, Dragon, Psychic, Electric, Bug, or Poison. In this project, we will attempt to create a neural network that classifies all 8 generations of Pokémon.

Motivation:

This problem seemed interesting to solve because both of us have an interest in Pokémon. We wanted to see if we could design a neural network that could determine a Pokémon's type since it has become harder for humans to determine a Pokémon's type based on its physical characteristics as more generations have been created (so far there are 8 generations of Pokémon to date).

Related Works:

https://github.com/hemagso/Neuralmon: This is a github page for a project named Neuralmon. Neuralmon used a neural network to try and determine a Pokémon's type by looking at the sprite of the Pokémon. This is our project's goal as well. This github has a database of Pokémon sprites which could be useful to training our own neural network.

https://github.com/wtingda/CNN-Pokémon-Classifier : This github project is very similar to the one listed above. This one however can identify multiple types when determining the typing of the Pokémon as opposed to one type like the previous project.

https://medium.com/@dimart/pok%C3%A9mon-recognition-d3ad5cadc61e: This article uses CNN and image classification to have the system determine who the Pokémon is based on images of scenes from the TV show. This article classifies 4 different Pokémon: Pikachu, Tangela, Charizard, and Gengar. This was our initial approach to the project, but then we decided to design a neural network to determine a Pokémon's type.

Methods:

1) Decide which type of neural network to use in order to do the image classification of Pokémon. Learn more about image classification neural networks.
2) Determine if we want to classify one type or classify all types for each Pokémon, since some Pokémon have two types.
3) Gather images or sprites of Pokémon from all 8 generations to use as data. Make sure that the images are the same size. Site to get sprites: https://pokemondb.net/sprites#gen8
4) Shuffle and split the images into a group for testing and a group for training.
5) Run the neural network with training data to get the trained model.
6) Run test data on the trained model.

Time Table:

By March 13th collect all of the images that will be used and determine which type of neural network will be used. We will also learn about all the libraries needed for developing the neural network.

By April 13th get most of the code done so that in the remaining month we can train and test the model, adjust code or parameters, and test the code for possible errors.

By May 13th have the project, the testing, and the presentation done.

Mid -Term Progress

Proposed Solution:

We have decided that we will use CNN to complete this project because we are doing a form of image recognition.

So far, our team has successfully downloaded images to use for training and testing our model. We have also successfully built a basic framework of the CNN and were able to run the CNN with the dataset we have so far. To document our progress, we created a Github repository to store the progress made so far on the project, meeting goals and achievements, and problems that we encountered on the days we met for the project.

Ivy: Created Github repository and downloaded Pokemon dataset into repository. Did some research on multi-CNNs. On my own time I tested the CNN code we had and made some adjustments.

Brady: Built basic framework of the CNN and ran code with the dataset. Did some research on multi-CNNs.

Some failed solutions we encountered was that the CNN we started running ran into an overfitting problem. In each epoch, the accuracy of the model drastically increased and reached 100% accuracy in 10 epochs, while the loss function value for each epoch was close to 0. In this case we will have to add in more parameters that will help reduce the overfitting problem such as adding regularizers, adding more data, include batch normalization on the data before moving onto the next convolution layer, implementing dropout so that data is not 'overused', alter the number of filters, or alter the kernel size of the filter.

One challenge that we ran into is that for a majority of the Pokemon we need two outputs because some Pokemon have two types instead of one type. This led us to research multi-output CNNs and how we can implement them for our project. So far in our research for implementing multi-output CNNs, most projects have created separate models for each output, but it has incurred a lot of running time because the CNN would be copied 'n' number of times for each different type of output to fully classify an image with multiple characteristics/outputs. A second challenge is there are a few pokemon that change looks drastically but they are still the same type which may skew the results of the CNN. We have decided to leave in all the data, even the special cases to observe how the model would classify those images in the future. Another challenge we have encountered is that we planned on classifying Pokemon from all 8 generations, but the dataset available online that sorts the Pokemon is only available up to generation 5. If we wanted to classify all 8 generations of Pokemon, we would have to download a lot of Pokemon sprites for all Pokemon in generations 6 through 8, and sort them by their type. We have decided to stick with using Pokemon images up to generation 5 due to time constraints. Another challenge or concern we had was how the model should predict the test dataset. There were some possibilities on how to organize the test dataset in batches using the Keras library, which were using a test dataset that has already been organized by labels, and using a test dataset that does not have the labels predicted yet. To solve this challenge, we will have to do more research and decide ourselves how we want the test dataset to be

batched up before being predicted using the model. One more challenge we found is with bringing in our images. When we output our images the coloration of them is off. We're not sure what is causing this issue and if this is causing an issue for the CNN or not.

Evaluation Strategy:

So far, we have not completed any tasks regarding the evaluation of our CNN model for the project. To measure the performance of our project, we plan on measuring the accuracy, mean squared error, precision, recall, and f1 score of each epoch when running the model. These measures of performance may change throughout the project. Another way we may plan on evaluating our model is to use the cross validation method, where we utilize different ratios of training to test datasets to train and predict our model in order to see if our model improves based on accuracy or mean squared error.

Over the next six weeks we will learn more about multi-output CNNs and continue writing our code. To be more specific, first we plan on optimizing our CNN model to work with one output. Currently the data is overfitted so we plan on using dropout layers and batch normalization layers to prevent the model from overfitting. First, we will add one dropout layer after each max pooling layer with a dropout rate of 0.2. If the model does not improve, we may increase the dropout rate by 0.1, where the max rate we will go is 0.5. For the batch normalization layer we will add one batch normalization layer after each Convolution layer. After that we will modify the number of filters and the kernel size in our Conv2D layers to refine our model. We currently use 32 and 64 for filters for our two convolution layers respectively. We will keep doubling the number of filters, and testing after each change, until they reach 256 and 512 filters respectively. We will compare the results of each and see which of them works the best. For determining the kernel size, we will start with a kernel size of 3x3, and increase the kernel size to 5x5 if we want more generalized features, or decrease the kernel size to a 2x2 if we want more specific features. Another parameter we may alter in our Conv2D layers is the stride parameter, which is how many spaces the pooling filter moves to condense the output of the previous layer. For now, we plan on sticking with a stride of 1 so that the model does not miss any features when doing the pooling operation. After finding the best CNN model we think fits with our data, if the accuracy and mean squared error improves as more epochs are processed, we may increase the number of epochs to train our model longer to see where it converges. Overall, we will do a combination of parameter tuning for each layer of our model, mainly focusing on implementing dropout layers and batch normalization layers to reduce the overfitting problem. Finally, we will work on converting our model to have multiple outputs instead of one so that we can properly categorize the pokemon. To do this we will need to conduct more research on how multi output CNNs work.

We are currently behind our proposed schedule. The complications we ran into is causing our coding to take longer than we had expected. To readjust we will have to extend the time we predicted to finish the code. Instead of finishing most of the code by April 13th we will give ourselves an extra week and a half and have most of the code done by April 25th. During the period starting from today up to April 25th, we will primarily focus on tuning the parameters of our Conv2D layers and max pooling layers, and adding a few more layers if needed to find a good fitting model that trains our Pokemon dataset. This will result in having less time for finishing touches and creating our presentation due March 13th.

Final Report

Methodology/Model Architecture(s):

To develop our model, we developed it using Keras library and ran the model in Jupyter notebook.

The dataset we used for our project came from another github project:
https://github.com/wtingda/CNN-Pokemon-Classifier

Before preprocessing our data, we noticed that there was a huge unequal distribution of images for each type, where 'Normal' types were the majority and 'Dark' types were the minority (as an example). To resolve this issue, we oversampled the data by duplicating images in the minority classes to match the number of image files stored in the majority classes. This would make sure that the necessary patterns are recognized for each class, and the minority classes are not underrepresented when classifying image data.

To preprocess our data, we used an image data generator from the Keras library. The preprocessing function, tf.keras.applications.vgg16.preprocess_input, was used for image processing. The image data generator class was mainly used to create batch datasets from our train and test data image datasets we obtained before. We also have an image data generator that inputs data augmentation while creating the batches for our datasets, such as modifying the zoom range, shift range, and flipping images to make the model train harder, but we decided not to include it until we see that the model is overfitting.

Currently, our dataset consists of Pokemon images from the first 5 generations of Pokemon and are sorted by primary and secondary types. This means that there are duplicate images for each Pokemon, since they have to be sorted by their first type and then sorted again into another folder for their second type. The type 1 folder and type 2 folders are then split into a train and test dataset for each type. After oversampling the training data, we had 15300 Pokemon images for type 1 training data and over 70000 images for type 2 training data. We decided not to oversample the test data for each type at this point, so there are about 2000 images for the test dataset of type 1 and about 2000 images for the test dataset of type 2. This will be used to train our first model, which will be discussed below.

For the model architecture, we first decided to use two CNN models to classify the Pokemon's primary and secondary type. To start off, we referenced the Alexnet architecture, where they have a 1-1-3 Conv2D layers pattern with max pooling layers in between. For our model, we decided to follow a similar pattern and have 2-2-3 Conv2D layers with max pooling layers in between. We also included a dropout layer with a rate of 0.25 after each max pooling layer to prevent the model from overfitting. For the first model, the last layer consisted of a softmax activation function with an output unit of 18 classes to represent the 18 different types a Pokemon may possess. For the second model, the last layer consisted of a softmax activation function with an output unit of 19 classes, where the last class is a 'None' class that represents a Pokemon that does not have a secondary type. We decided to separate the multi label classification problem into 2 multi class classification problems because it was easier to understand how softmax activation function works and the basic classification concept with our current knowledge. We used softmax as the activation function in the last layer because similar to sigmoid, it predicts the probabilities of multiple classes (more than 2 classes compared to sigmoid). Different from sigmoid, all the

probabilities predicted for each class must add up to 1. After obtaining the output for each image, we chose the class with the highest probability as the class that most likely fits the input image. The loss function used was categorical cross entropy to output a probability out of the 18 or 19 classes for each image trained in each of our models.

Another model we were considering to solve the multi label classification problem was to use one CNN model to classify the top 2 classes with the highest probability to represent the Pokemon's primary and secondary type. In this case, we were thinking that the highest probability would represent the primary type and the second highest probability would represent their secondary type. The model architecture would be very similar to the first model architecture explained above, but the last layer would output 19 units and the activation function would be sigmoid with a binary cross entropy loss function. Based on the research done so far, this would train the model to output a probability for each class independent of each other. This means that for each class/label, the model would output a probability between 0 and 1 if the image fits that label. Compared to the last layer specifications listed in the last architecture, the probabilities of all the classes do not add to 1, since they are independent of each other. This architecture would be better than the first architecture because we would only have to run a single model instead of 2 separate models. Unfortunately, we did not have time to test this architecture.

To run the second model, we would have to modify how our current dataset is stored. Currently, the datasets are separated into type 1 and type 2 folders, with images sorted by type in each folder. Running the second model consists of combining all of the image data in both type 1 and type 2 folders into one huge folder, and have the images separated by both types. The single folder will have 19 subfolders to represent the possible primary and secondary types, and there will be duplicates of each Pokemon so that they can be classified for both of their types.

Here are the links below that go more in depth about the loss functions and last layer activation functions to determine how we designed our models.
https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy
https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/binary-crossentropy

Evaluation and Results:

To evaluate the accuracy of our models we first stored the predicted results of the models in two separate arrays. Then, we built a function that took the arrays and converted them from the softmax outputs into the corresponding type of the pokemon. Then we took these predicted arrays and compared them to the test labels to see if they matched. We had a counter to keep track of how many were correct and divided the number of correct predictions by the total number of predictions to obtain the accuracy of the model. Using this function, we ran multiple tests changing the dropout rate, the kernel size, the pool size, and the number of strides to see which model would work best with our dataset and have the highest accuracy. We then created a table in Excel to record the accuracies of the models. Since the images for the testing type 1 and type 2 weren't the same between type 1 and type 2 we couldn't directly measure the accuracy of guessing both type 1 and type 2 for one image. Therefore, we used statistics to calculate the accuracy of

guessing both types correct for one pokemon with the equation $Combined\ Accuracy = (\frac{Accuracy\ of\ Type1}{100} * \frac{Accuracy\ of\ Type2}{100}) * 100$. Below is the table containing the results of our tests.

Epochs	Dropout	Kernel_Siz	Pool_Si	Strides	Accuracy1	Accuracy2	Combined Accuracy
50	0.2	3,3	2,2	2	57.8	70.79	40.91662
50	0.2	3,3	3,3	2	62.17	58	36.0586
50	0.2	3,3	3,3	3	57.22	63	36.0486
50	0.2	4,4	2,2	2	58.17	71.3	41.47521
50	0.2	4,4	3,3	2	57.76	60.53	34.962128
50	0.2	4,4	3,3	3	50.51	70.19	35.452969
50	0.35	3,3	2,2	2	57.97	56.34	32.660298
50	0.35	3,3	3,3	2	52.47	56.85	29.829195
50	0.35	3,3	3,3	3	53.07	56.5	29.98455
50	0.35	4,4	2,2	2	54.44	70.1	38.16244
50	0.35	4,4	3,3	2	49.69	68.4	33.98796
50	0.35	4,4	3,3	3	37.9	47.81	18.11999
70	0.5	3,3	2,2	2	55.59	54.89	30.513351
70	0.5	3,3	3,3	2	51.32	43.28	22.211296
70	0.5	3,3	3,3	3	40	20.7	8.28
70	0.5	4,4	2,2	2	59.66	65.96	39.351736
70	0.5	4,4	3,3	2	57.89	53.19	30.791691
70	0.5	4,4	3,3	3	50.92	48.46	24.675832
					Best 1	Best 2	Combined Best
					62.17	71.3	44.32721

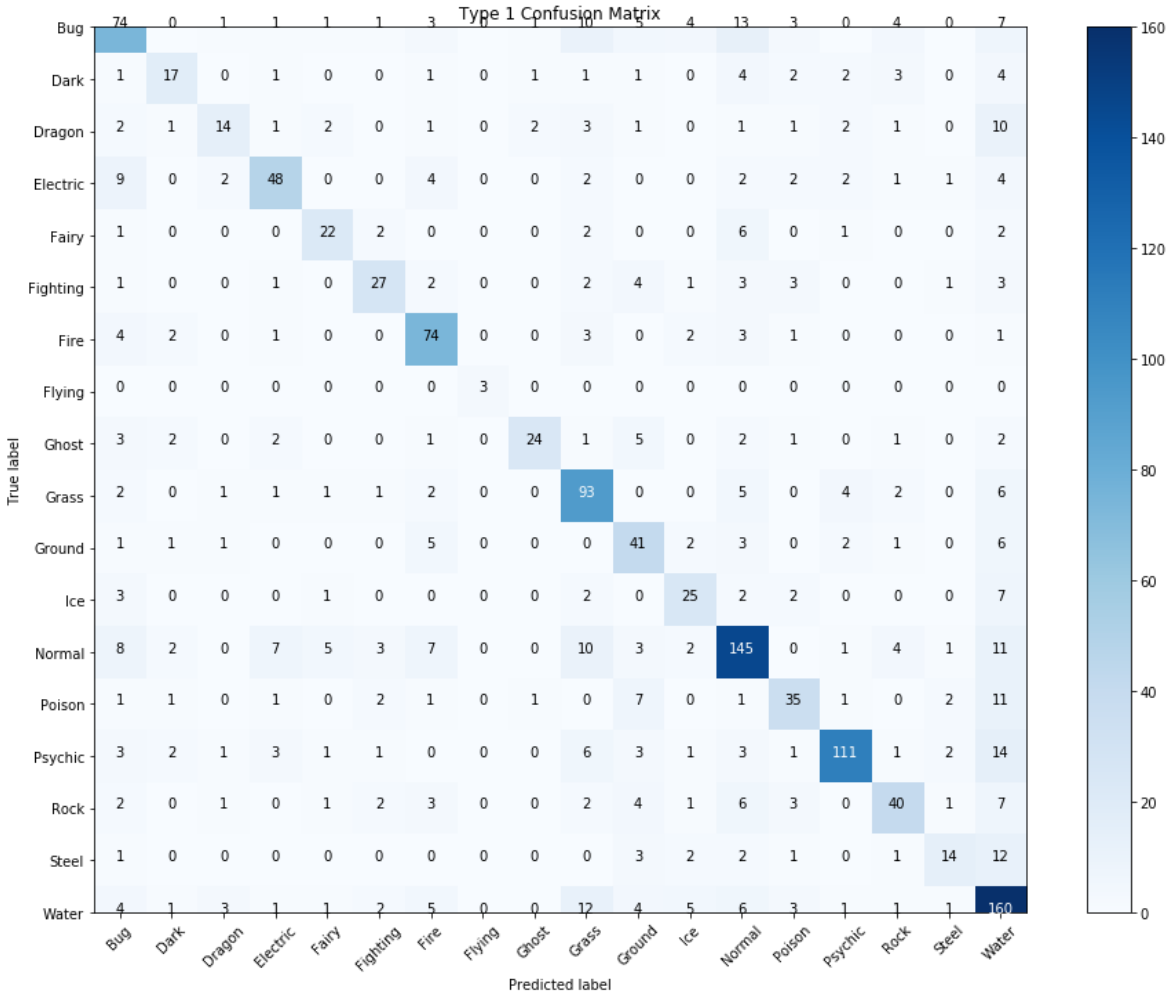
After all the tests we chose the models with the highest accuracy and ran them again but with 600 epochs each. The values we obtained for our final test are in the table below. After obtaining the results we then wanted to visualize our results so we could better understand how well our model performed. To do this we made two confusion matrices from our final test, also shown below. Using a confusion matrix we can see how well the model performed with each type and determine if there were any types having more trouble being predicted than others. The confusion matrices have the true type of the pokemon on the left and what our model predicted on the bottom. For example in model 1 if you look at the grass row 93 of the 118 pokemon were predicted as grass which means we got 93 correct, 2 of the grass pokemon were predicted as bug, 0 were predicted as dark, and so on. This means that when you observe a confusion matrix when you look at the diagonal, from top left to bottom right, there should be a blue line which both of our matrices have. The trend in Model 2 is harder to see because the none type has so many more pokemon compared to the other types but the trend is still there. We also noticed in model 2 there a lot of pokemon misscategorized as normal. We aren't entirely sure why this is the case but we believe it is

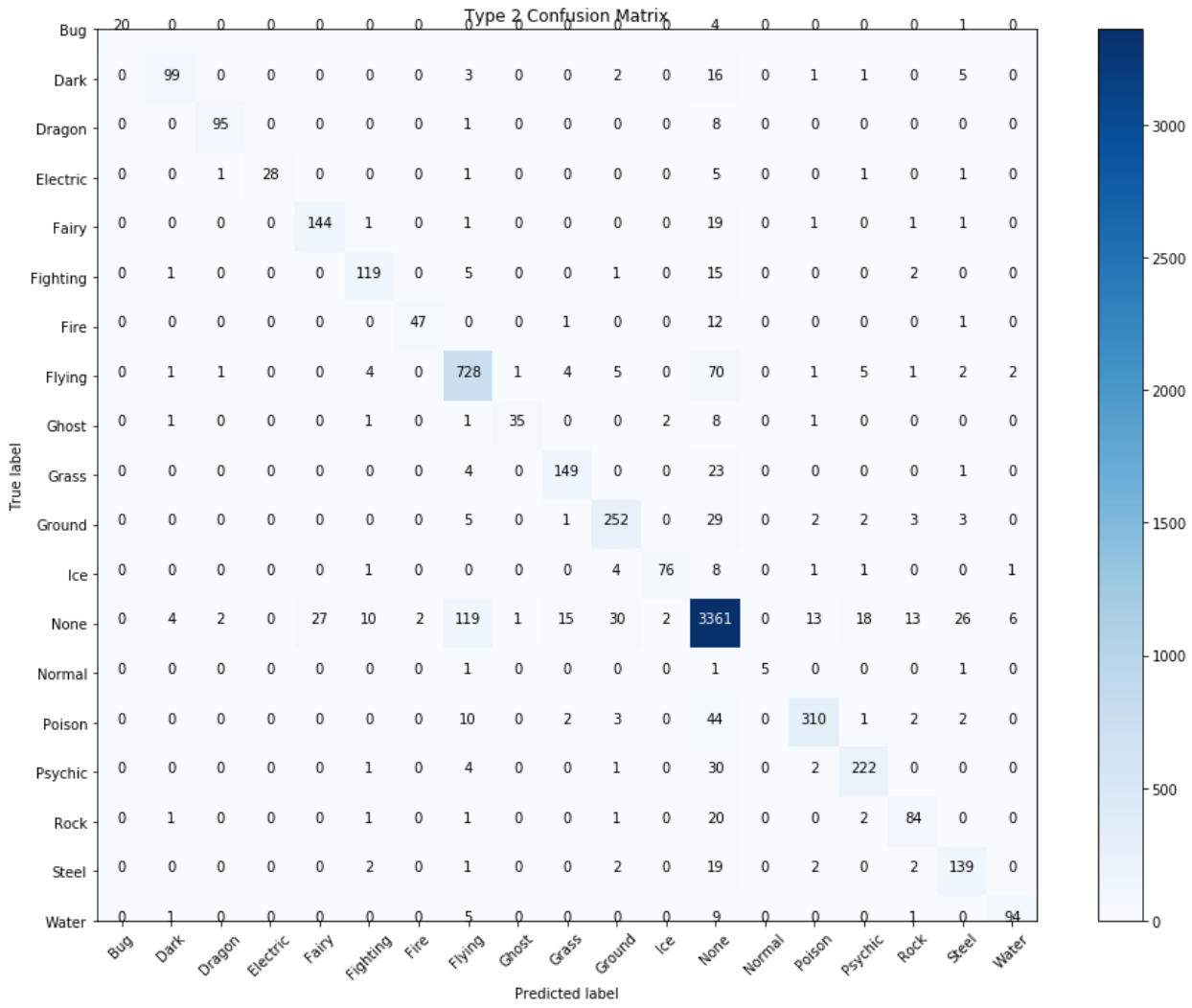
because the none category is so broad. The none category can have features of any of the other classes so it may be hard for the model to determine if there is a second type or not.

Model	Epochs	Dropout	Kernel_Size	Pool_Size	Strides	Accuracy
Model1	600	0.2	3,3	3,3	2	67.18
Model2	600	0.5	4,4	2,2	2	88.55

Combined accuracy

59.48789





Limitations and Future Works:

One limitation we had was time and resource constraints. Our model took a long time to run, even when adjusting the learning rates of the models. We realized that the model was heavily relying on CPU power to run the model (used 75% of CPU), but were also unable to figure out how to utilize the GPU to run the model faster. One future work for this project is to do more research and figure out how to use the GPU to run our model and overcome the hardware constraints.

Another limitation we encountered was having a limited dataset. In our initial proposal, we wanted to work with Pokemon images from all 8 generations. After lots of research, we were only able to find sorted Pokemon images from the first 5 generations of Pokemon. Due to time constraints, we were not able to obtain and sort the Pokemon images from generations 6-8, and started working with what we had to train our model. Another future work for this project is to create our own dataset that also contains Pokemon images from generations 6-8 to test and improve the accuracy of our model.

As stated in the evaluation section, we couldn't directly measure the accuracy of guessing both type 1 and type 2 for a single image, since the test datasets from type 1 and type 2 do not have a 1 to 1 correlation. One future work to resolve this issue is to recombine the images and create our own training and test data set split function so we get the 1 to 1 mapping when training both our models for type 1 and type 2 to calculate the accuracy accordingly.

We also want to try more CNN models and other algorithms to try and improve our accuracy. Since we were constrained by time we decided to follow the AlexNet model and made a few adjustments by adding dropout layers and more convolution layers. We would like to continue to improve our model by adding additional layers. We want to try Average Pooling layers instead of maxPooling layers because it is possible that where the features are located in each image (localization) may have an impact on the accuracy of the model. Also, we want to try adding dense layers to see if that will improve the accuracy. One algorithm we are interested in testing is ML-KNN.

Another goal we have for the future of this project is to test out the second model architecture we discussed above and observe how the model trains the image dataset. It would be best to run and try out different architectures to compare their accuracies. It would also be worthwhile to research and apply other CNN variant architectures (VGG-16, ResNet, LeNet, etc.) and compare those architectures to see which ones can classify multi labeled Pokemon images the best.

Lastly, we would want to continue tuning hyperparameters in our models to see how our model can improve and classify multiple labels for a single image. Our goal would be to try and overfit the model as much as we can, and then adjust the model to train it harder to find patterns in the image data. Some hyperparameters we would adjust and add are batch normalization layers, dropout rate, kernel size, pooling size, number of filters, type of pooling layer (max pooling versus average pooling), the number of fully connected layers, etc.

Github Link:

https://github.com/ivyT26/CSCI4931_DLFinal