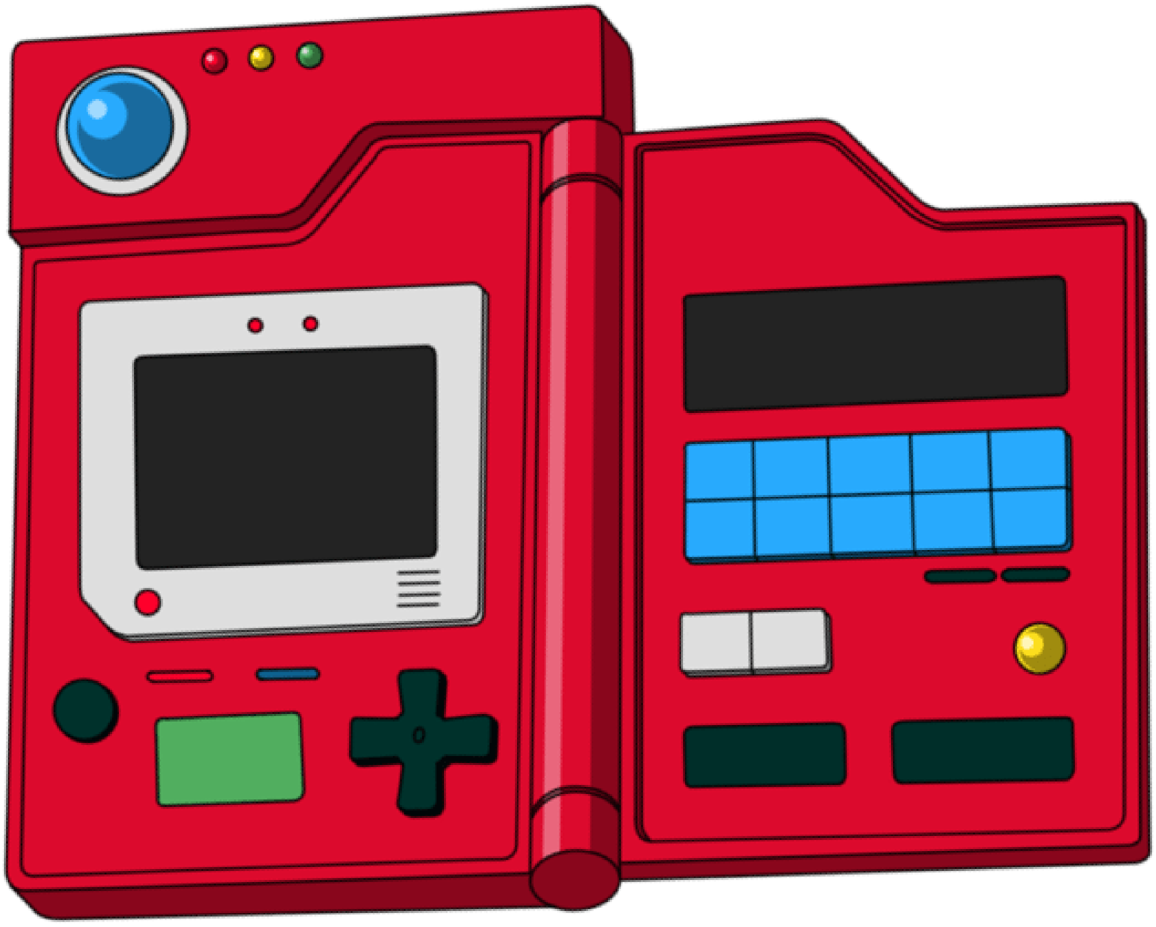


Springboard Final Capstone Project
Gotta Catch ‘Em All
Image Classification Using CNN Keras

Kris Seo

Mentor: Preetjot Singh

Introduction

1. Motivation

In today's technological society, computer vision is one of the most popular machine learning areas that enable the digital world to interact with the physical world. Computer Vision enables self-driving cars to recognize a sense of surroundings and detects objects. As I am a huge Pokemon fan, I want to build a real-life Pokedex to classify given Pokemons with using Keras and experience machine learning skills using Computer Vision.

2. Problem

When I watch some series of Pokemon, sometimes it is hard to classify what Pokemon it is since there are 151 different Pokemons in the 1st generation of Pokemon and some Pokemon are similar to each other if they are in the same evolution family. It is challenging to train a computer to recognize a pokemon compared to train my friend with a small number of images.

3. Clients

The toy industry or Nintendo can be beneficial from building a real-time Pokedex. In a sense of image classification, any industry can be beneficial since it is related to Computer Vision. Having image classification, kids and even adults can have more fun by identifying Pokemons through using Pokemon Go.

4. Data

I obtained pokemon image datasets from Kaggle; then, I combined two datasets, one from human_collected and the other from Bing_API to expand train and test data. I could have used Bing API to collect my own Pokemon data, but it would be tedious and inefficient to do so.

5. Approach

Before I build a real-time Pokedex using Convolutional Neural Network, I start summarizing each pokemon image with a Histogram of Oriented Gradients, which is a feature descriptor used in image preprocessing for the purpose of object detection and use Support Vector Machine to classify pokemon. Then, I will construct an architecture of Convolutional Neural Network, which is inspired by the human visual cortex and train the neural network through AWS EC2 since AWS EC2 supports GPUs to train my network faster than CPU in my local laptop. I could train my neural network with HOG feature descriptions; however, it would depreciate pure CNN. The reason behind HOG depreciating pure CNN is that deep neural networks do not require hand-crafted features because hidden layers in neural networks will take care of it. HOG is a hand-crafted feature that pools gradient magnitude and direction over each cell block in an image; it is a representation of a given image. Therefore, I built two models, CNN and HOG + SVM, and the accuracy of CNN was much better than HOG + SVM.

6. Deliverables

I will update my codes, a report and a presentation on Github

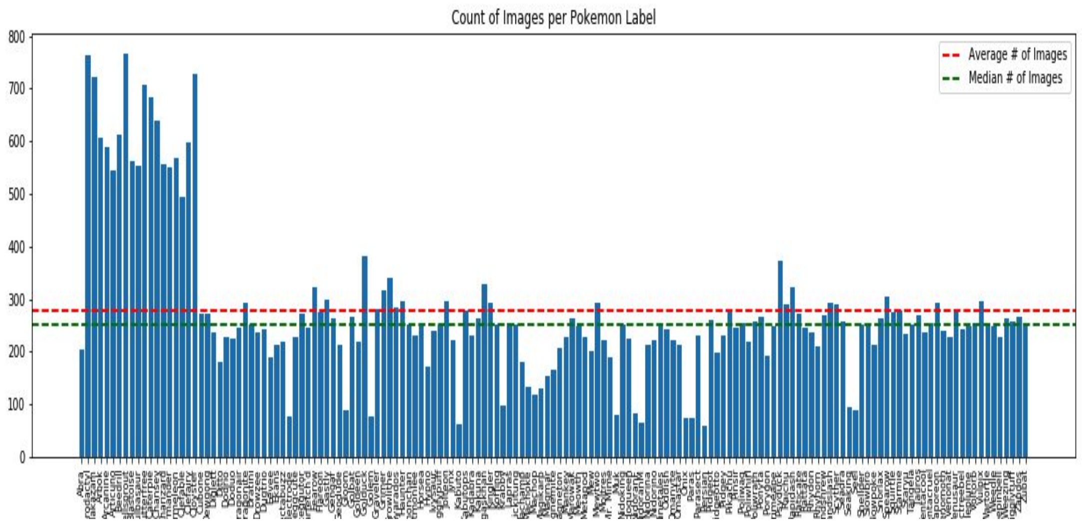
Data Wrangling/EDA

After collecting the data, I realized there were some images misclassified and not a Pokemon by looking at a preview of files. Therefore, I had to take time to manually go through pokemon folders and identify ones that should not be included to prevent my neural network from being noisy from this misclassified data. For example, Onix is a ground-type of Pokemon and also a car from Chevrolet if you search this term on Google. This type of error will fool a neural network.

Some of my data files were jpeg types. Since the png file is a lossless compression algorithm, all jpeg files are converted to png files before I train my neural network. Then, I converted image files into Numpy ndarray objects with 64 x 64-pixel values with 3 RGB values and normalized by dividing 255, which is the maximum number that can be stored. The purpose of normalization is to train my CNN faster, which converges faster in [0,1] scale than in [0,255]. Hence the shape of each ndarray was (64,64,3).

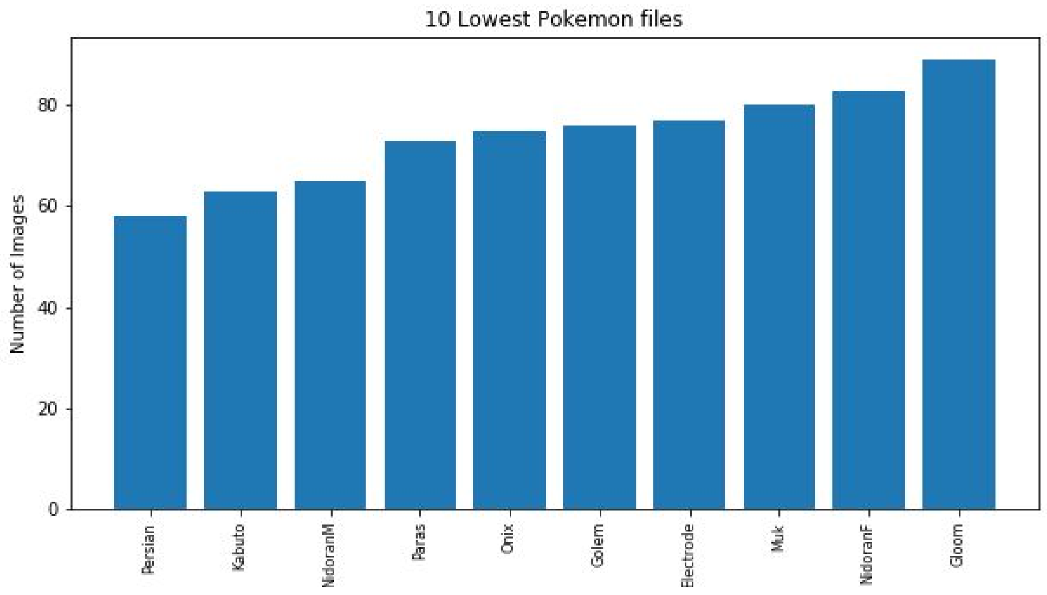
In parallel, I labeled each pokemon corresponding to its folder. The dataset had about ~42,022 pokemon files with 151 classes and the average number of each class image file was about 280. Even though I thoroughly cleaned the dataset, some images contained more than one pokemon.

Figure 1:



Here are the ten lowest counts of images per Pokemon. These pokemon have less than 100 image files, so I cannot guarantee that the classifier will correctly classify these pokemon. My model will not work great with these pokemon.

Figure 2:



Building Support Vector Machine using H.O.G

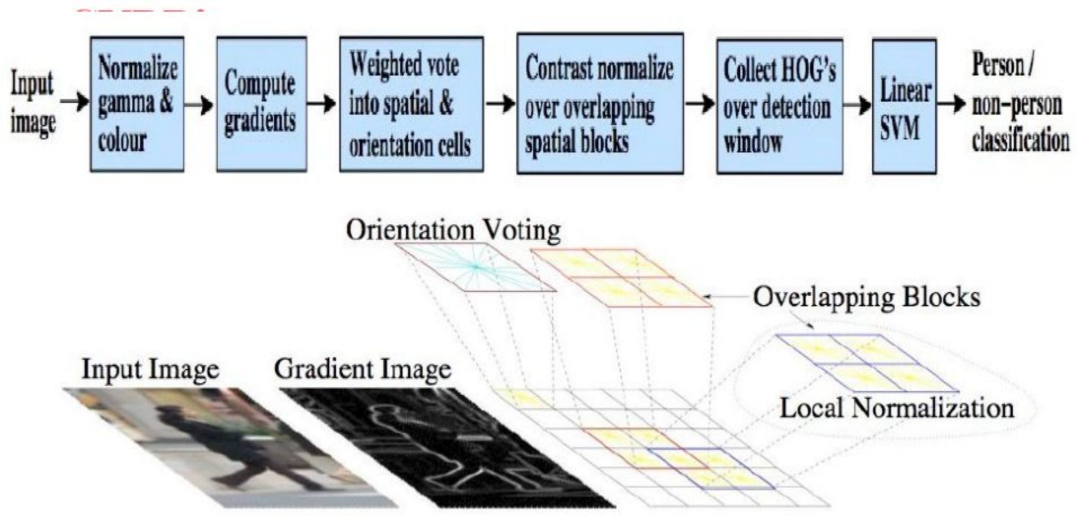
As I mentioned above, the Histogram of Oriented Gradients is a feature descriptor, which is a representation of an image that simplifies the image by extracting useful information. HOG has demonstrated to be much lower complexity in terms of computing time when compared to other feature descriptors, which allowed it to be used in many areas of image processing as real-time applications. Moreover, cell rotations and translations do not affect HOG values.

Calculate HOG descriptors

HOG calculates a gradient magnitude between the horizontal gradient (from left to right pixels) and vertical (from up to down) gradient of the corresponding pixel's intensity value and the angle between them by arctan(grad_y/ grad_x) in each window of an image. Afterward, I divide the image into many 2x2 pixel cells. In each cell, the magnitude values of these 64 cells are binned and cumulatively put into 9 bins of unsigned direction with no sign; so 0-180 degrees rather than 0-360 degrees are considered and negative values are ignored.

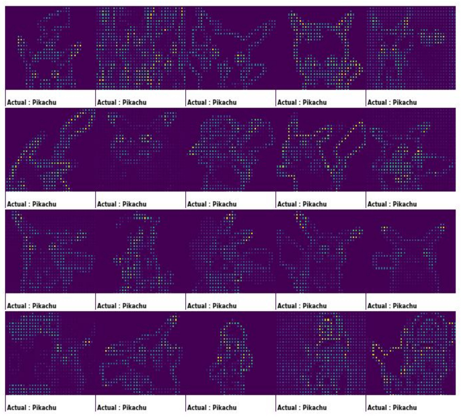
In order to make histogram invariant to illumination and contrast, normalization is applied by grouping cells and overlapping/ striding. Grouping cells will ensure that low contrast areas are stretched and overlapping will ensure consistency of magnitude and direction. Therefore, HOG detects edges in an image.

Figure 3: Example of Workflow Using HOG from Standford CS231A


Input image
Normalize gamma & colour
Compute gradients
Weighted vote into spatial & orientation cells
Contrast normalize over overlapping spatial blocks
Collect HOG's over detection window
Linear SVM
Person / non-person classification
Orientation Voting
Overlapping Blocks
Input Image
Gradient Image
Local Normalization

To calculate stable and simple values of the gradient magnitude and direction, all images are converted to greyscale. Color information won't matter for finding a gradient direction and magnitude; however, edge information will. Here are some examples of pokemon images using HOG.

Figure 4: Pikachu Extracted From HOG vs Given Pikachu



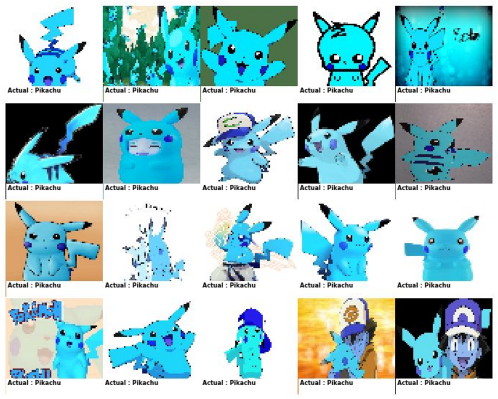
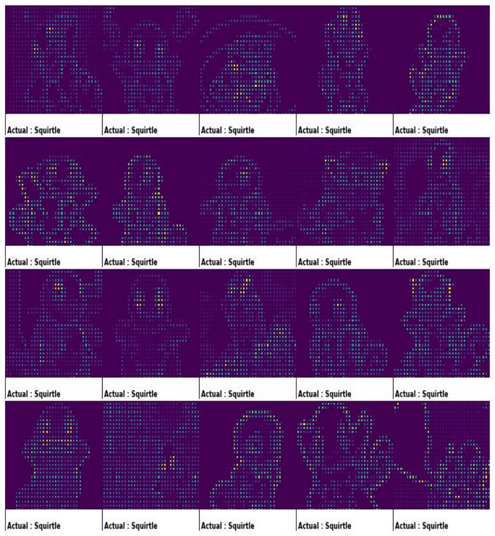
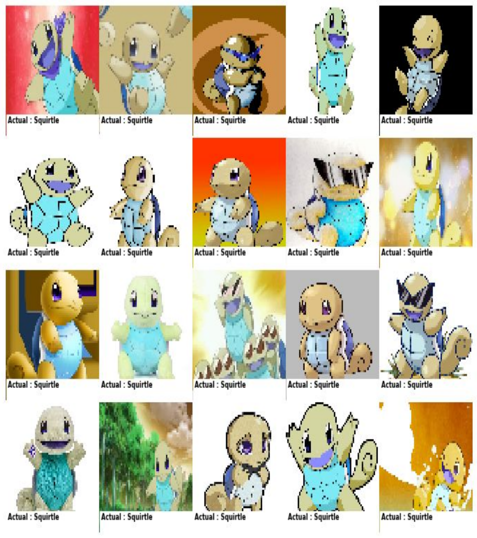


Figure 5: Squirtle Extracted From HOG vs Given Squirtle





Training Support Vector Machine

The normalization performed by scikit-image is not the one recommended by the paper, so I extract HOG feature descriptions from OpenCV. I used Linear-SVM, which divides spaces using different HOG feature descriptions of Pokemon. I was not sure if the data is linearly separable, so I also used a Radial Basis Function with Gaussian Kernel. Basically, I used Kernel Trick to transform non-linearly separable data into linearly separable data. In the end, I compared two models, one using a RBF and the other using a linear function, but could not find the difference in a sense of accuracy. In SVM, C value is a parameter to the trade-off of misclassification. I set C value as 2.67 to set my model to be Soft SVC. Higher C value will cause lower bias and higher variance since we penalize the cost of misclassification.

Evaluation of Model

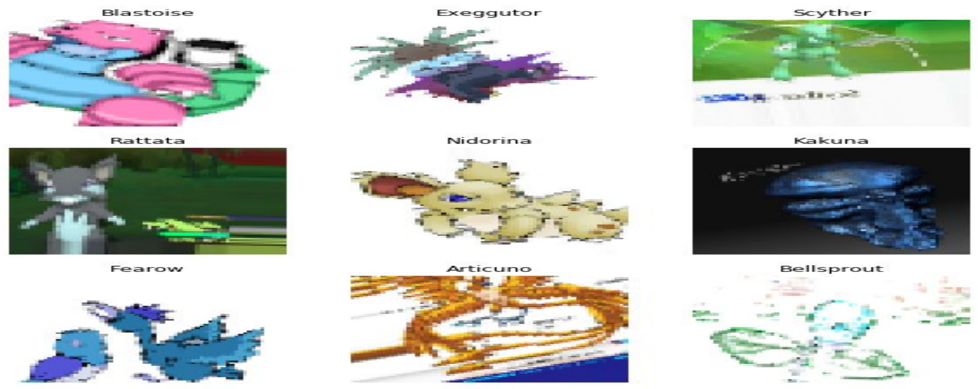
After training the model to classify Pokémon, I evaluated how the model works well with the test data. However, the accuracy with the test data was not high enough even though accuracy with the training data was pretty decent.

From the confusion matrix, I see there were lots of false-positives with a specific Pokemon called Bellsprout. I could not figure out exactly why HOG feature descriptors provided false-positive related to Bellsprout; however, it might be the reason that Bellsprout was the largest data, containing 766 image files. So, the model was trained in bias toward Bellsprout due to its largest amount of data.

Building Convolutional Neural Network

Since my dataset was not big enough (people usually consider dataset is big enough if each class has 1,000 examples), data augmentation was applied in the training set to extend the variability of images by randomly flipping or shifting vertically/horizontally and changing the brightness of an image. By applying data augmentation to the dataset, I could prevent overfitting from a model and increase the generalizability of the model.

Figure 6: Examples of Augmented Pokemon Images



What we want the computer to do is to differentiate between all the images and figure out each unique feature from pokemon. When we see Pikachu, we recognize its color is yellow, its cheek is red and its tail is brown. The computer will figure out each edge and curves and pass through a neural network similar to the visual cortex. As a result, we want the Neural Network to understand the high-level features of an image.

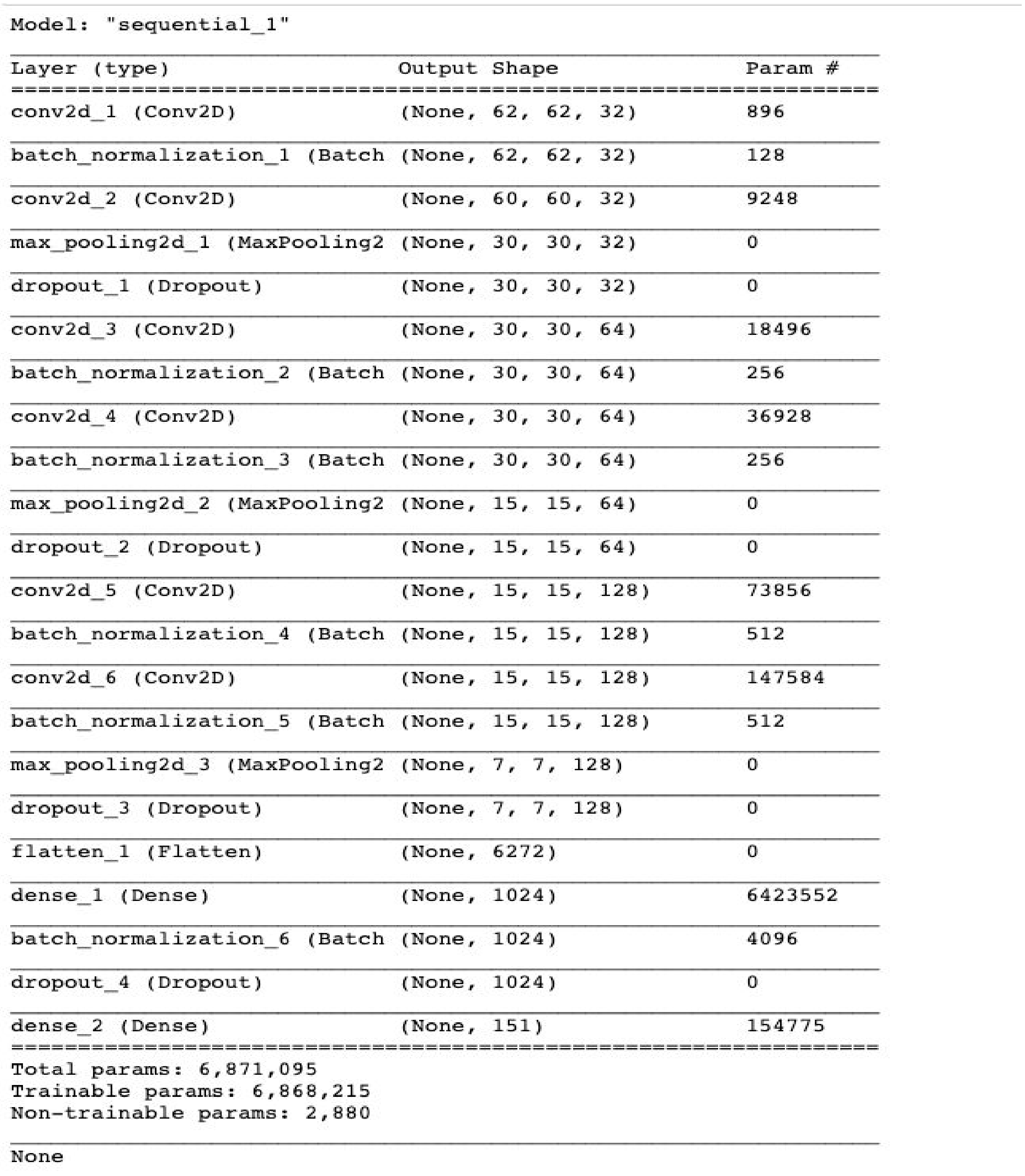
I used a smaller and similar architecture of the Neural Network from VGGNet and six convolutional layers for feature extraction with regularization using Dropout. Dropout can be applied separately to both input layers and the hidden layers and masks the outputs of a percent of units from a layer by setting the output to zero (in our case, it is 30% of the units in our dense layers). The flattened layer is used to flatten out 128 of the 7 x 7 feature maps that we get as output from the sixth convolution layer followed by Batch Normalization, MaxPooling, and Dropout.

BatchNormalization normalizes each batch by both mean and variance reference in the training process to prevent internal covariate shift.

Moreover, BatchNormalization will train the neural network to converge faster due to not adding a bias term to the network. Max pooling is used to reduce the dimensionality of input representation by revealing the maximum value from a collection of elements. The final output layer was 151 nodes representing each Pokemon using the softmax activation function, which is the standard for multi-class classification and outputs probabilities of each class.

Nonlinearities and preservation of dimension that help to improve the robustness of the network and control overfitting while other activate functions have a problem of vanishing gradients. Therefore, I had to use relu as my activation function. Furthermore, I compiled my model using RMSProp which update parameters separately.

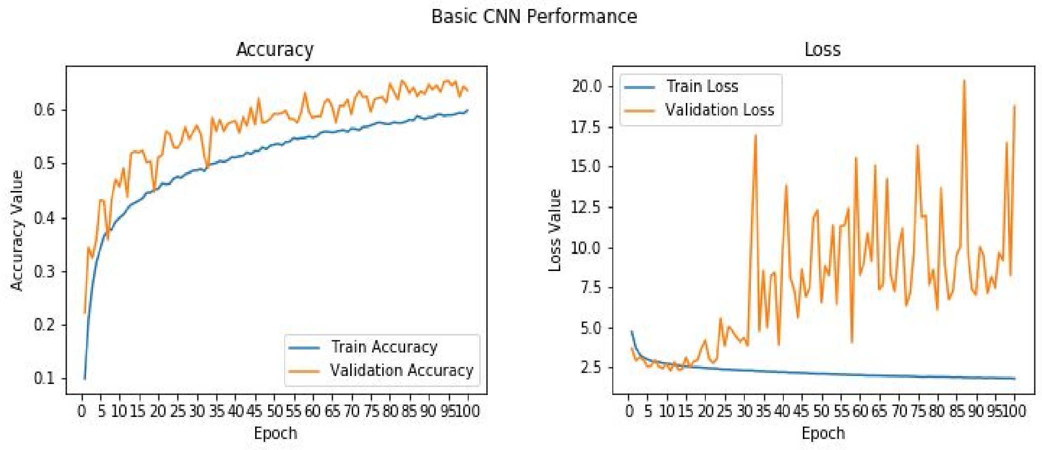
Figure 7: Summary of CNN Architecture



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 62, 62, 32)        896       
_________________________________________________________________
batch_normalization_1 (Batch (None, 62, 62, 32)        128       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 60, 60, 32)        9248      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 32)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 30, 30, 32)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 30, 30, 64)        18496     
_________________________________________________________________
batch_normalization_2 (Batch (None, 30, 30, 64)        256       
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 30, 30, 64)        36928     
_________________________________________________________________
batch_normalization_3 (Batch (None, 30, 30, 64)        256       
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 15, 15, 64)        0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 15, 15, 64)        0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 15, 15, 128)       73856     
_________________________________________________________________
batch_normalization_4 (Batch (None, 15, 15, 128)       512       
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 15, 15, 128)       147584    
_________________________________________________________________
batch_normalization_5 (Batch (None, 15, 15, 128)       512       
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 7, 7, 128)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              6423552   
_________________________________________________________________
batch_normalization_6 (Batch (None, 1024)              4096      
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 151)               154775    
=================================================================
Total params: 6,871,095
Trainable params: 6,868,215
Non-trainable params: 2,880
_________________________________________________________________
None
```

Evaluation of Model

Figure 8: Performance of CNN



Our CNN model’s accuracy value increases with a total of 100 epochs. An epoch is when the complete dataset has passed through CNN once. I see there is no overfitting for the model; however, our validation loss function fluctuates. As the final result, I could achieve an accuracy of 73%, which is a lot better from the accuracy of 35.3% with HOG + SVM.

From the confusion matrix, it is easily shown that our classifiers could classify the test data robustly compared to HOG + SVM since its diagonal represents true positives. However, ['Golem', 'Krabby', 'Muk', 'Onix', 'Paras', 'Persian'] were not classified correctly due to the relatively small amount of image data. Furthermore, I also figured out the false negatives of these Pokémon to figure out what features our CNN could not capture. We figured that our neural network cannot distinguish thoroughly when it gets similar pokemon. In this project, similar Pokémon can be thought of as Pokémon within its evolution chain. For example, Onix and Graveler are rock Pokémon and have similar features, so our false-negative pokemon were rock pokemon.

Figure 9: False Negatives for 0 True Positive Pokemon

```
Golem ['Graveler', 'Geodude', 'Raticate', 'Kangaskhan', 'Weedle', 'Dodrio', 'Primeape', 'Tangela', 'Fearow', 'Cloyste
r']
--------------------------------------------------------------------------
Krabby ['Kingler', 'Seel', 'Snorlax', 'Jolteon', 'Charmeleon']
--------------------------------------------------------------------------
Muk ['Grimer', 'Ekans', 'Cloyster']
--------------------------------------------------------------------------
Onix ['Graveler', 'Geodude', 'Rhydon', 'Ninetales', 'Gastly', 'Sandslash', 'Blastoise', 'Cloyster']
--------------------------------------------------------------------------
Paras ['Parasect', 'Onix', 'Kingler', 'Goldeen', 'Weedle', 'Vulpix', 'Diglett', 'Voltorb', 'Aerodactyl', 'Beedrill',
'Cloyster']
--------------------------------------------------------------------------
Persian ['Meowth', 'Geodude', 'Mankey', 'Rhyhorn', 'Doduo', 'Ninetales', 'Kangaskhan', 'Growlithe', 'Arcanine']
--------------------------------------------------------------------------
```

EVOLUTION CHAIN
No.098
No.099
Lv.28
KRABBY
KINGLER
© SEREBII.NET 2007
EVOLUTION CHAIN
No.088
No.089
Lv.38
GRIMER
MUK
© SEREBII.NET 2007
EVOLUTION CHAIN
No052
No053
Lv.28
MEOWTH
PERSIAN
© SEREBII.NET 2004
EVOLUTION CHAIN
No.074
No.075
No.076
Lv.25
GEODUDE
GRAVELER
GOLEM
© SEREBII.NET 2007

Future Work/Conclusion

To train the image classifier to achieve near or above human-level accuracy, we'll need a massive amount of data, large computation power, and lots of time on our hands. Since we don't have these three, transfer learning will be a key to use pre-trained weights, which already knows how to classify different images from other datasets. The neural network for image classification will learn to detach from low-level features such as lines and shapes to higher-level features as the network goes deeper.

Furthermore, I could gather more images of Pokémon which were not classified well to train our model to be more robust and stable. I wanted to try out Pokemon detection using a video clip from Pokemon, but it did not work out well with our classifier due to window-sliding problem. If I could annotate bounding boxes and train my neural network with Mask-RCNN or Faster RCNN, the neural network will be improved dramatically.

Lastly, I could deploy my Deep Learning model to IOS application to classify Pokemon so that anyone can use my neural network model by getting the app. However, the problem will occur when there are multiple Pokémons because I only train my model to classify one Pokémon out of the given image. The model will work properly if the input image only contains one Pokémon and will result in a minor error if the input contains several Pokémons. To detect several Pokémon, as stated above, I will have to annotate bounding boxes for all training image files and retrain the model through AWS. However, it will be very expensive to do so even though it is relatively cheap to detect a single Pokémon. If there were GPUs and annotated Pokémon image files, it would have been much cheaper to build a Pokémon detector.