Project Report

Gotta Plot 'Em All | PokeNet

Reid, Brandon
Vangala, Sravani
Buchireddy, Shreya Reddy

06/25/2020

Participation

Reid, Brandon | Team Lead | Machine Learning - spent the project specializing and studying machine learning. Built a CNN for image classification of Pokemon.

Vangala, Sravani | Data Analysis - worked on data analysis, feature extraction, and data visualization, also scribed/voiced the project presentation.

Buchireddy, Shreya Reddy | Data Analysis - worked on data analysis, feature extraction, and data visualization, also scribed/voiced the project presentation.

Image Dataset

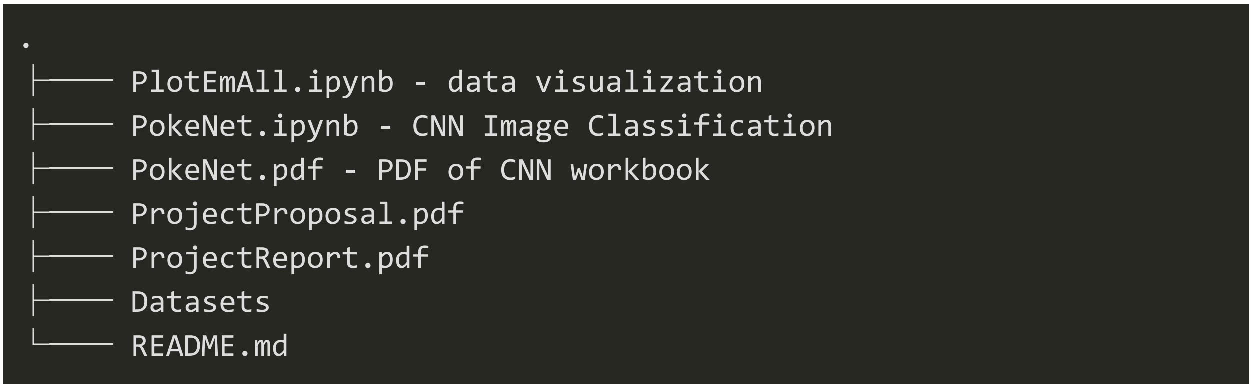
pokemon.zip

The Complete Pokemon Dataset

Workflow & Collaboration

The collaboration of this project was done via slack communication, as well as email. The shared codebase was managed through github. Image datasets were too large to store on github so we shared them through google drive via zip file. The github repository included two notebooks, one for data visualization, and one for our machine learning model. For the machine learning portion of this project we imported Tensorflow into our Jupyter Workbook using Python.

The project directory is as followed:



```
.
├──── PlotEmAll.ipynb - data visualization
├──── PokeNet.ipynb - CNN Image Classification
├──── PokeNet.pdf - PDF of CNN workbook
├──── ProjectProposal.pdf
├──── ProjectReport.pdf
├──── Datasets
└──── README.md
```

Abstract

Pokemon is a global icon for children and adults everywhere. It is a TV series that has expanded into video games, card games, movies, merchandise, and everything in-between. The motivation behind this project is to further understand the dynamics of the pokemon universe through data, while also having fun and learning in the process. Given how popular pokemon is, there is an extremely large amount of data to use for analysis and visualization. For the machine learning portion of this project we would easily be able to scrub the web for images of pokemon very easily.

The desired outcome of this project was to incorporate different data science concepts using an accumulation of pokemon data we found through various sources like kaggle and google images. The main goal of this project was split into two desired outcomes, provide statistical analysis and data visualization, and utilize pokemon data for machine learning purposes.

Our goal with the machine learning portion of the project was to do something both fun and challenging. A very well known tool in the world of Pokemon is a “Pokedex” where a pokemon trainer can use it to discover and analyze pokemon as they come across them in the wild. This provided the project inspiration to attempt to create the building blocks of a real world machine learning “Pokedex” that could essentially take an image of a pokemon and predict its name using a standard Convolutional Neural Network. As more libraries like TensorFlow become more robust, projects like this show how easy it can be to get a small and simple Convolutional Neural Network running.

With this, we could expand outside of this project in the future, to create a fun mobile application that could be used to take a picture of a pokemon with your phone, and it could provide you a prediction with the name, stats, etc. of the pokemon. This is just one example of what could be done, since this project is using a standard Convolutional Neural Network, we could expand it to do image recognition on all types of datasets.

Design

As stated above, this project uses Python for management, analysis, and visualization of our datasets, and also uses Python with TensorFlow for our Convolutional Neural Network. The code for this project was managed in two Jupyter Workbooks, one for data visualization and one for our CNN. The following sections will attempt to describe what a CNN is, it's architecture, etc. This report makes the assumption that the user has a basic understanding of neural networks.

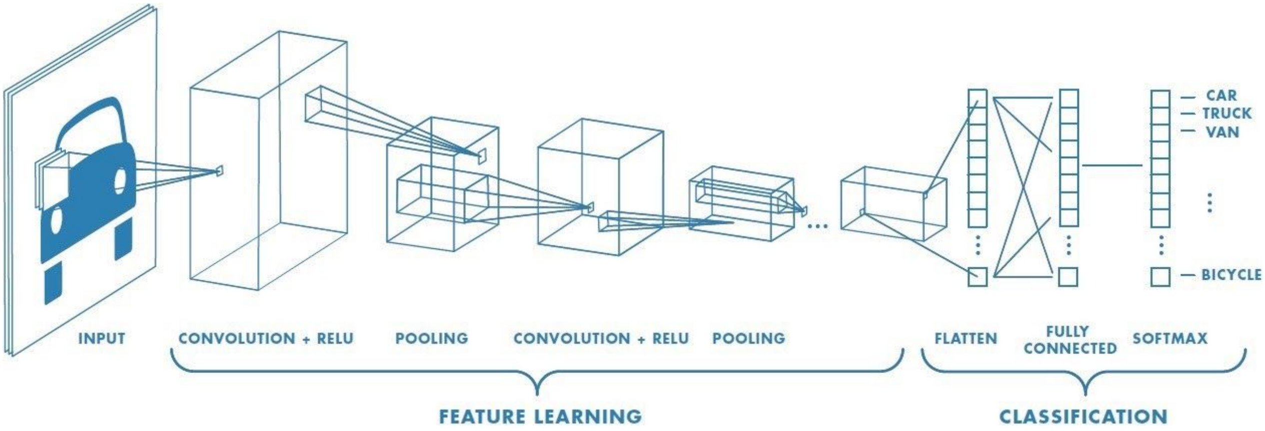
The goal of this project was not to go extremely in depth with the building blocks of a CNN, but provide an example of how simple and quick it can be to utilizing libraries like TensorFlow to get around some of the grunt work of building a neural network while also having fun in the process.

Convolutional Neural Networks (CNN)

Essentially, similar to neural networks, CNN's are made up of layers of neurons that have specific weights and biases that are learned in an unsupervised approach. The CNN will take an input image, assign those weights and biases to various features of the image to attempt to find patterns and distinctions. CNNs are useful in reducing larger images down to a form which is easy to process, while also maintaining critical features for good predicted outcomes.

A simple CNN consists of the following layers:

- Input layer
- Convo layer (Convo + ReLU)
- Pooling layer
- Fully connected(FC) layer
- Softmax/logistic layer



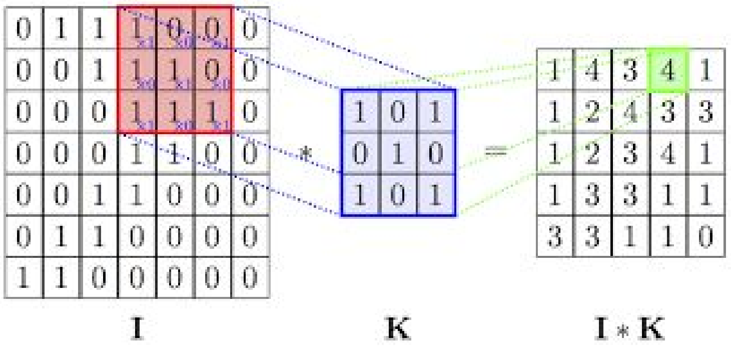
Input Layer

This is the layer that houses image data. In order to prepare image data for the input layer, images must be in the form of an array, represented in a three dimensional matrix. Similar to one of our class assignments where we get a three dimensional array of RGB color percentages of an image,. Images must be reshaped into a single dimension as well.

In order to properly prepare images for the Conv Layer, they must also be shuffled, and resized to standard image sizes. For this project we resize images down to 96 x 96.

Convolution Layer

This layer can be thought of as the main feature extraction layer, where it will attempt to gather features using edge detection. All images are made of dark and light horizontal and vertical edges. Convolution works to separate an image into many distinct features based on these edges. You can think of a face, and how the nose has distinct edges. Without going into too much detail, convolution uses dot product (matrix multiplication) to filter out these edge features. Stepping over a grid of pixels a certain number of times until the whole image is covered. Below is an example of convolution, you can think of K as a small feature (edge) of an image, and the red space is a section of a bigger image, the 1’s are edges. You can see how weights are then calculated using the dot product.



In the diagram above you’ll see Convo + ReLU, ReLU is the process of changing negative values during this process to 0.

Pooling Layer

A layer used between two convolution layers, used to reduce spatial complexity of an image after it’s been convoluted. Without this step it would be too costly on computation when we start training on our dataset.

Fully Connected Layer

This layer takes the output of the previous layers, flattens them and turns them into a single vector that can be an input for the next stage. Then takes the inputs from the feature analysis and applies weights to make a prediction.

Milestones

The following milestones will provide details for direction in building the CNN, and what was accomplished. For further code details please refer to the codebase.

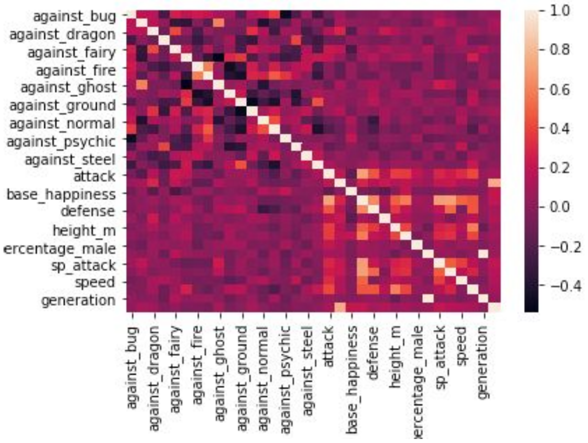
Part 1. Data Analysis | Data Visualization

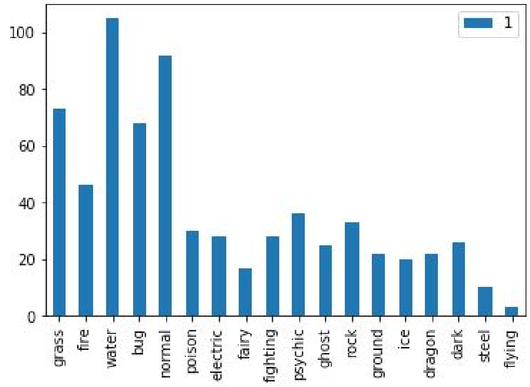
✓ *Distribution of Pokemon Types*

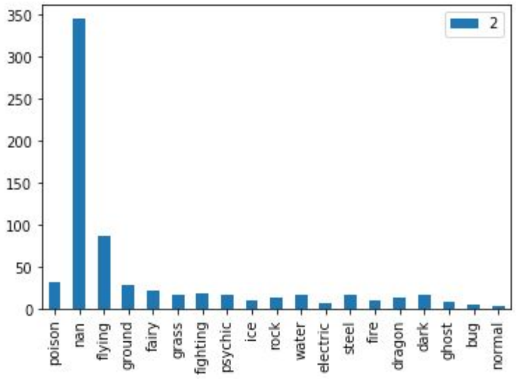
✓ *Single vs dual types using attack and special attack attributes*

✓ *Heatmap showing correlation between pokemon base stats*

✓ *Plot legendary vs non-legendary Pokemon*



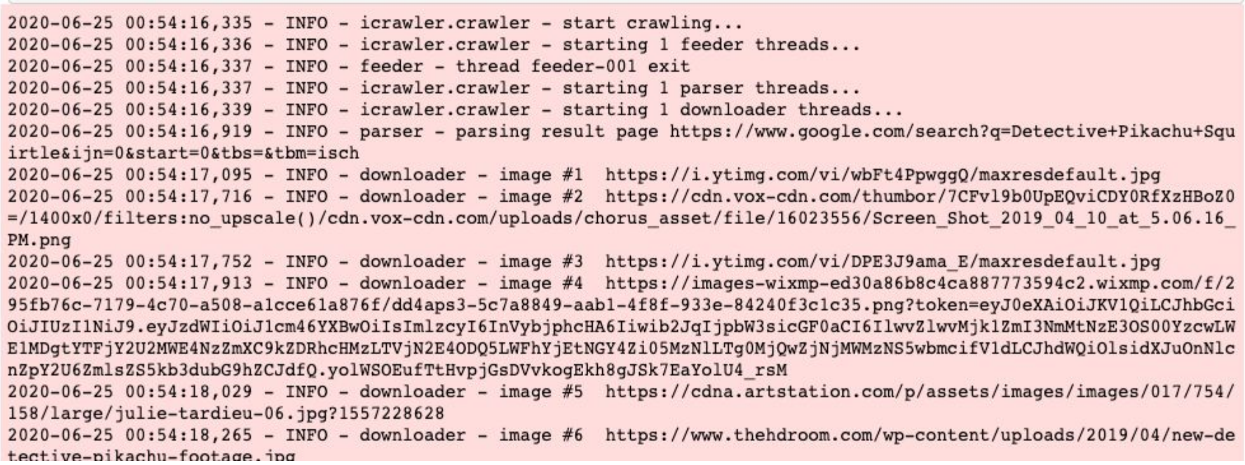




Part 2. Machine learning | Convolutional Neural Network

✔ *Crawl Web for Images of Pokemon for dataset*

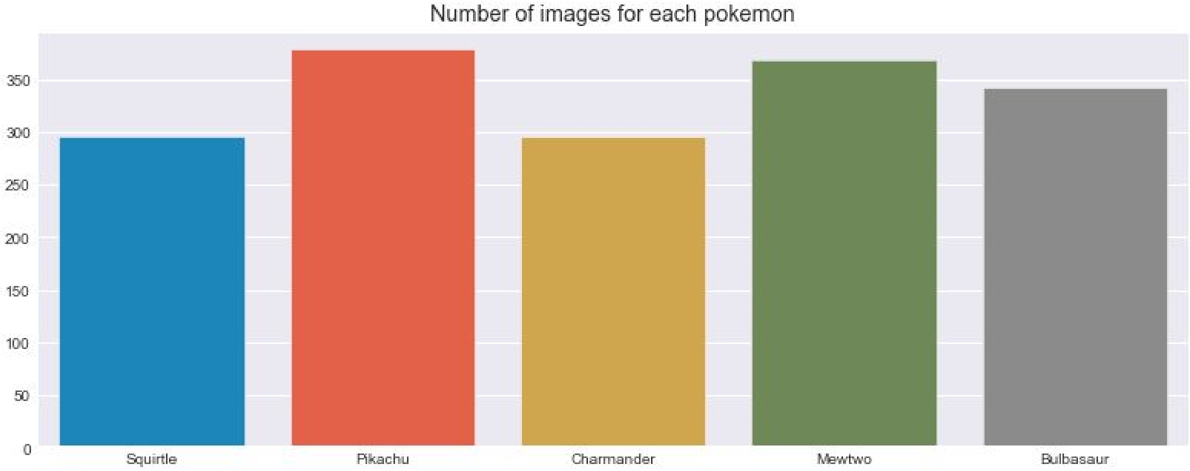
Library used: `GoogleImageCrawler`



```
2020-06-25 00:54:16,335 - INFO - icrawler.crawler - start crawling...
2020-06-25 00:54:16,336 - INFO - icrawler.crawler - starting 1 feeder threads...
2020-06-25 00:54:16,337 - INFO - feeder - thread feeder-001 exit
2020-06-25 00:54:16,337 - INFO - icrawler.crawler - starting 1 parser threads...
2020-06-25 00:54:16,339 - INFO - icrawler.crawler - starting 1 downloader threads...
2020-06-25 00:54:16,919 - INFO - parser - parsing result page https://www.google.com/search?q=Detective+Pikachu+Squ
irtle&ijn=0&start=0&tbs=&tbm=isch
2020-06-25 00:54:17,095 - INFO - downloader - image #1  https://i.ytimg.com/vi/wbFt4PpwggQ/maxresdefault.jpg
2020-06-25 00:54:17,716 - INFO - downloader - image #2  https://cdn.vox-cdn.com/thumbor/7CFvl9b0UpEQviCDY0RfXzHBoZ0
=/1400x0/filters:no_upscale()/cdn.vox-cdn.com/uploads/chorus_asset/file/16023556/Screen_Shot_2019_04_10_at_5.06.16_
PM.png
2020-06-25 00:54:17,752 - INFO - downloader - image #3  https://i.ytimg.com/vi/DPE3J9ama_E/maxresdefault.jpg
2020-06-25 00:54:17,913 - INFO - downloader - image #4  https://images-wixmp-ed30a86b8c4ca887773594c2.wixmp.com/f/2
95fb76c-7179-4c70-a508-a1cce61a876f/dd4aps3-5c7a8849-aab1-4f8f-933e-84240f3c1c35.png?token=eyJ0eXAiOiJKV1QiLCJhbGci
OiJIUzI1NiJ9.eyJzdWIiOiJ1cm46YXBwOiIsImlzcyI6InVybjphcHA6Iiwib2JqIjpbW3sicGF0aCI6IlwvZlwvMjk1ZmI3NmMtNzE3OS00YzcwLW
E1MDgtYTFjY2U2MWE4NzZmXC9kZDRhcHMzLTVjN2E4ODQ5LWFhYjEtNGY4Zi05MzNlLTg0MjQwZjNjMWMzNS5wbmcifV1dLCJhdWQiOlsidXJuOnNlc
nZpY2U6ZmlsZS5kb3dubG9hZCJdfQ.yolWSOEufTtHvpjGsDVvkogEkh8gJSk7EaYolU4_rsM
2020-06-25 00:54:18,029 - INFO - downloader - image #5  https://cdna.artstation.com/p/assets/images/images/017/754/
158/large/julie-tardieu-06.jpg?1557228628
2020-06-25 00:54:18,265 - INFO - downloader - image #6  https://www.thehdroom.com/wp-content/uploads/2019/04/new-de
tective-pikachu-footage.jpg
```

✔ *Get Top 5 Pokemon with 200+ images*

```
Total number of pokemon: 5
Total number of images: 1679
```



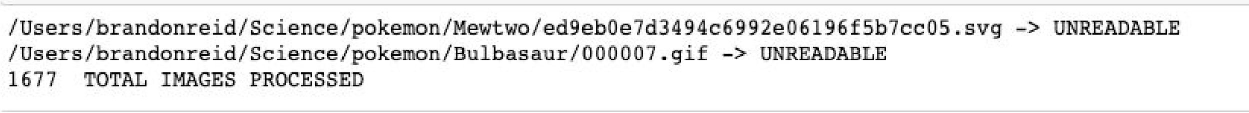
✔ *Prepare images for input layer - Shuffle, Resize, Reshape, and Scale*

Each of the above pokemon have their own directories that we loop over, shuffle the image paths and then resize each image into an array with their associated labels (pokemon). This is in preparation for the input layer.

You can see we shuffle our images, so we get an unbiased dataset. Resize the images, to 96 x 96, and then like we discussed in the design section we must reshape our image array down to one column, and a good practice is to scale the images.

`to_categorical` is a Keras/TensorFlow tool to prepare our labels/classes for our model.

```
    IMG_SIZE = 96
    random.seed(SEED)
    random.shuffle(image_paths)
    …
    data.append(cv.resize(image, (IMG_SIZE, IMG_SIZE)))
    labels.append(pokemon.index(poke))

    X = np.array(data).reshape(-1, IMG_SIZE, IMG_SIZE, 3) / 255.0
    y = to_categorical(labels, num_classes=len(pokemon))
```



```
/Users/brandonreid/Science/pokemon/Mewtwo/ed9eb0e7d3494c6992e06196f5b7cc05.svg -> UNREADABLE
/Users/brandonreid/Science/pokemon/Bulbasaur/000007.gif -> UNREADABLE
1677  TOTAL IMAGES PROCESSED
```

✔ *Implement Cross Validation*

Here we create our test cases from our dataset, a very popular cross validation method is `test_train_split`

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size = 0.2, random_state = SEED)
```

❌ *Image Data Augmentation*

After some research/studying we came across data augmentation methods in order to help better train our CNN. Data Augmentation essentially takes images from a dataset and creates many different variations of each image, like rotation, transformations, hue and color changes, etc. Data Augmentation can greatly improve a training model.

However attempting data augmentation with TensorFlow tools was found extremely slow, and CPU intensive. For now we decided against this approach.

Issues found here: https://github.com/keras-team/keras/issues/12683

✔️ *Build the model*

While pruning through the image data we crawled for, we noticed many of the images were higher resolution, or of larger sizes, and we also had over a thousand of them. After further research into CNNs it was clear that a VGGNet Architecture is most beneficial for Large-Scale image Recognition.

```
model = Sequential() # set up the modal for sequential layers

# SEQUENCE LAYER: CONV => RELU => POOL
model.add(Conv2D(32, (3, 3), padding="same", input_shape=IMAGE_DIMS))
#CONV
model.add(Activation("relu")) #RELU
model.add(BatchNormalization(axis=-1))
model.add(MaxPooling2D(pool_size=(3, 3))) # POOL
model.add(Dropout(0.25)) #Dropout to next sequence

# SEQUENCE LAYER: (CONV => RELU) * 2 => POOL
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=-1))

model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=1))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25)) # Dropout to next sequence

# SEQUENCE LAYER: (CONV => RELU) * 2 => POOL
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=-1))

model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=-1))

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25)) # Dropout to next sequence

# SEQUENCE LAYER: first (and only) set of FC => RELU layers
model.add(Flatten())
model.add(Dense(1024))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5)) # Dropout to next sequence

# softmax classifier
model.add(Dense(len(pokemon))) # Dense is number of classes to
predict
model.add(Activation("softmax"))
```

✔ *Train the Model*

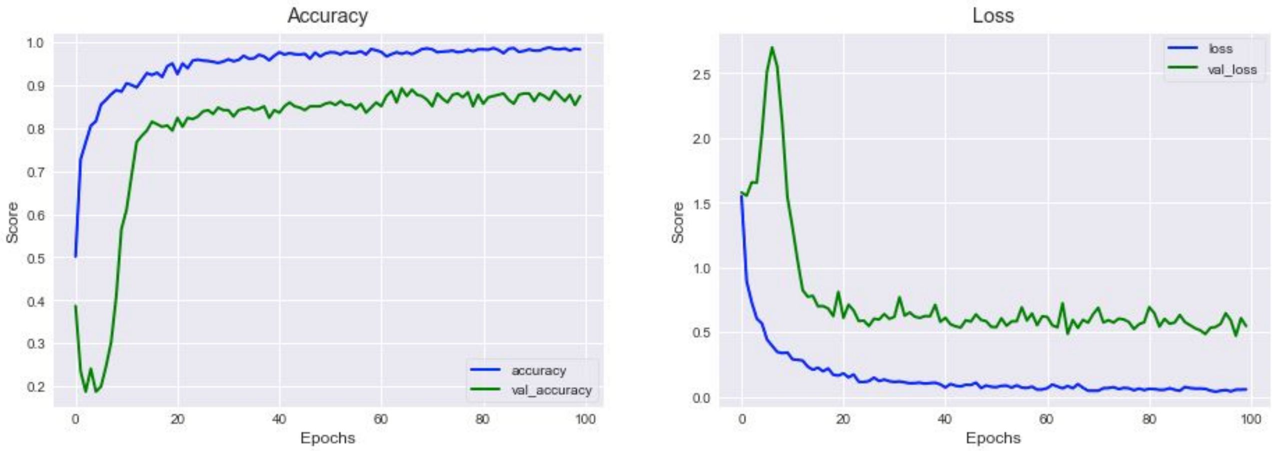
Here is where the magic happens. We fit our model with our cross validated datasets, and provide a number of epochs to train on. From what we've researched, the more epochs the better usually, but for time sake, we went with 100. This means the model will learn and train itself 100 times.

```
model.fit(X_train, y_train, batch_size=BS,
                    epochs=EPOCHS, verbose=1, validation_data=(X_test,
y_test))
```

```
Epoch 97/100
1341/1341 [==============================] - 46s 35ms/step - loss:
0.0402 - accuracy: 0.9858 - val_loss: 0.5903 - val_accuracy: 0.8631
Epoch 98/100
1341/1341 [==============================] - 46s 34ms/step - loss:
0.0544 - accuracy: 0.9806 - val_loss: 0.4704 - val_accuracy: 0.8780
Epoch 99/100
1341/1341 [==============================] - 47s 35ms/step - loss:
0.0546 - accuracy: 0.9851 - val_loss: 0.6073 - val_accuracy: 0.8542
Epoch 100/100
1341/1341 [==============================] - 46s 34ms/step - loss:
0.0561 - accuracy: 0.9836 - val_loss: 0.5463 - val_accuracy: 0.8750
```

✔ *Analyze Accuracy and Loss on Training Model*

You can see from the plots below, we start to get a pretty great accuracy percentage even after 20 epochs. Ideally, we would like to get that green val_accuracy line closer to the blue accuracy line. There's definitely room for improvement in our model, but what a great outcome for a first build CNN.



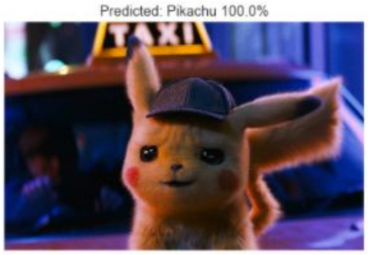
✔ *Make Predictions*

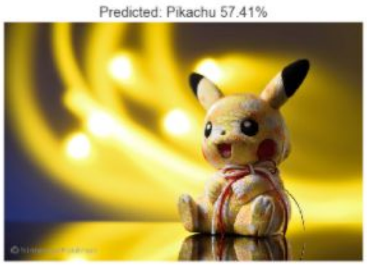
Before we can make a prediction we need to process our images that we want to predict, just like we do for our images before they go into the image layer. So we will take our images and resize, reshape and scale them.

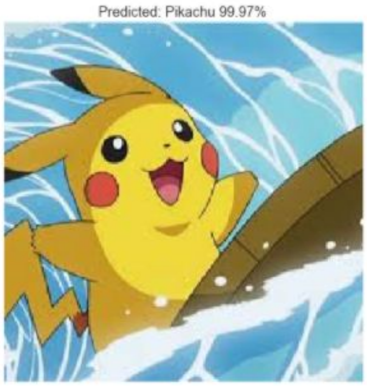
```
image = cv.resize(image, (IMG_SIZE, IMG_SIZE))
image = image.reshape(-1, IMG_SIZE, IMG_SIZE, 3) / 255.0

preds = model.predict(image) # run prediction
```

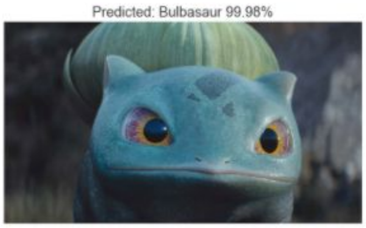
Perfect predictions with pikachu and bulbasaur across the board! You'll see we even get a perfect prediction with detective pikachu. The first time we ran our model, detective pikachu would fail to predict correctly. Our assumptions were that the dataset of images did not consist of images of detective pikachu. Our model must not have gathered features of pikachu ever being "fuzzy".

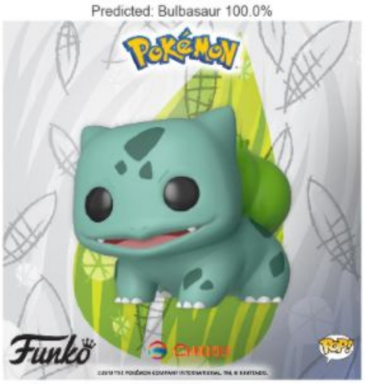
Once we retrained our model with some images from the detective pikachu movie, you can see we get perfect predictions.

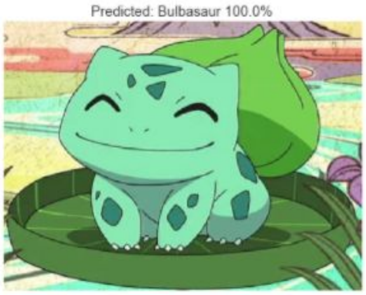










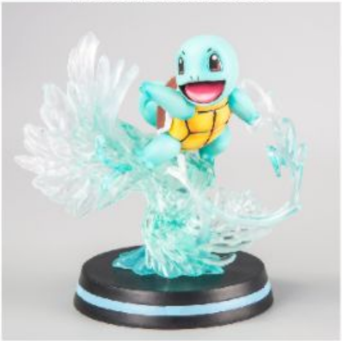


❌ *Prediction Failures*

We can see below that our model is not perfect. The toy version of squirtle fails to predict accurately. Our assumption is that the edges throw off the model with the waves of water.

Predicted: Squirtle 100.0%

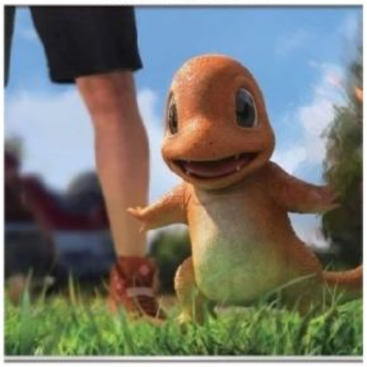
Predicted: Bulbasaur 89.24%



Predicted: Squirtle 100.0%

Below we can see that the charmander from the detective pikachu movie fails to predict accurately. Charmander was the only pokemon we didn't crawl for from the detective pikachu movie. This shows that our model is most likely making features based on the edges and smoothness of the character.

Predicted: Pikachu 99.38%



Predicted: Charmander 99.96%

Predicted: Charmander 99.97%

Reference Material

https://medium.com/@kelfun5354/building-a-simple-pokemon-convolutional-neural-net-cc724a8fb47d

https://www.pyimagesearch.com/2018/04/16/keras-and-convolutional-neural-networks-cnns/

https://www.youtube.com/watch?v=FmpDIaiMIeA

https://www.geeksforgeeks.org/image-classifier-using-cnn/

https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/

https://missinglink.ai/guides/convolutional-neural-networks/fully-connected-layers-convolutional-neural-networks-complete-guide/

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c