Pokémon Type Classification

Joseph Laurel

What are Pokémon?

- Collectively refers to all 898 ‘species’ of fictional creatures from the franchise of the same name

What are Pokémon?

- In the video game franchise, Pokémon are captured and then subsequently used to battle other individuals (real and computer) with their own Pokémon
    - Battles are generally in teams of 6 (often times unique) Pokémon.
    - Using attacks to defeat your opponent's Pokémon

- Defeating another 'Pokémon trainer' happens when any number of your Pokémon are still standing while their Pokémon are all defeated

Pokémon Types

- Many factors go into Pokémon battle calculations, but one of the most defining factors of Pokémon battles is type advantages

- Allows your attacks to be 2x effective

Pokémon Types

Attacker \ Defender	Normal	Fire	Water	Grass	Electric	Ice	Fighting	Poison	Ground	Flying	Psychic	Bug	Rock	Ghost	Dragon	Dark	Steel	Fairy
Normal													½	0			½	
Fire		½	½	2		2						2	½		½		2	
Water		2	½	½					2				2		½			
Grass		½	2	½				½	2	½		½	2		½		½	
Electric			2	½	½				0	2					½			
Ice		½	½	2		½			2	2					2		½	
Fighting	2					2		½		½	½	½	2	0		2	2	½
Poison				2				½	½				½	½			0	2
Ground		2		½	2			2		0		½	2				2	
Flying				2	½		2					2	½				½	
Psychic							2	2			½					0	½	
Bug		½		2			½	½		½	2			½		2	½	½
Rock		2				2	½		½	2		2					½	
Ghost	0										2			2		½		
Dragon															2		½	0
Dark							½				2			2		½		½
Steel		½	½		½	2							2				½	2
Fairy		½					2	½							2	2	½	

- There are currently 18 types of Pokémon, each with their own strengths and weaknesses
- Pokémon can have up to TWO types!


GRASS
FIRE
WATER
NORMAL
ELECTRIC
PSYCHIC
FIGHTING
ROCK
GROUND
FLYING
BUG
POISON
DARK
GHOST
ICE
STEEL
DRAGON
FAIRY

Project Goal

- Can a model determine the type(s) of a Pokémon from an image?
- Can we then use this model to alert a player who has no knowledge of Pokémon when the opponent's Pokémon is a 'threat'?
- Utilize a convolutional neural network to solve this multi-label classification problem and maximize precision and recall when determining Pokémon type(s)

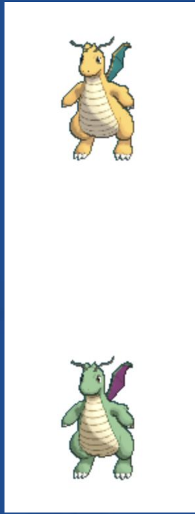
The Data

- All images hosted on https://pokemondb.net/pokedex/national
- 898 pokemon
- 8360 colored images were scraped off of the website (taken from in-game sprites) and used, along with their names and type labels to create data frame
- Most images were 128x128, a select few 'official artwork' images were larger and not square

Data Processing Pipeline (EDA)

- Images were transparent and converted into numpy arrays by utilizing Pillow (Python Imaging Library)
- Saved into folders corresponding to their appropriate types
- Labels were generated by cross-referencing image names with a dataframe containing Pokémon names and labels
- Labels were one hot encoded before modeling
- Excluded ‘shiny’ Pokémon in web scraping process as these are purposely colored differently and appear at a 1/8192 chance

	ID	Name	Primary Type	Secondary Type	Image	Additional Images
0	#001	bulbasaur	Grass	Poison	https://img.pokemondb.net/sprites/bank/normal/...	['https://img.pokemondb.net/artwork/bulbasaur....
1	#002	ivysaur	Grass	Poison	https://img.pokemondb.net/sprites/bank/normal/...	['https://img.pokemondb.net/artwork/ivysaur.jp...
2	#003	venusaur	Grass	Poison	https://img.pokemondb.net/sprites/bank/normal/...	['https://img.pokemondb.net/artwork/venusaur.j...
3	#004	charmander	Fire	NaN	https://img.pokemondb.net/sprites/bank/normal/...	['https://img.pokemondb.net/artwork/charmander...
4	#005	charmeleon	Fire	NaN	https://img.pokemondb.net/sprites/bank/normal/...	['https://img.pokemondb.net/artwork/charmeleon...



Type Distribution

- Water types were most prevalent, while Ice types were least common

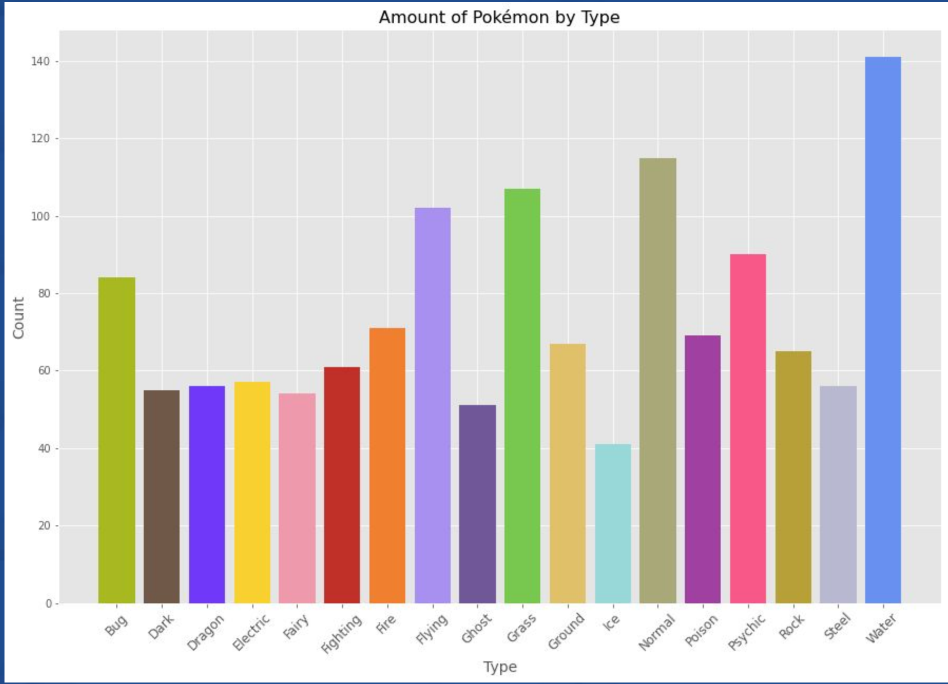
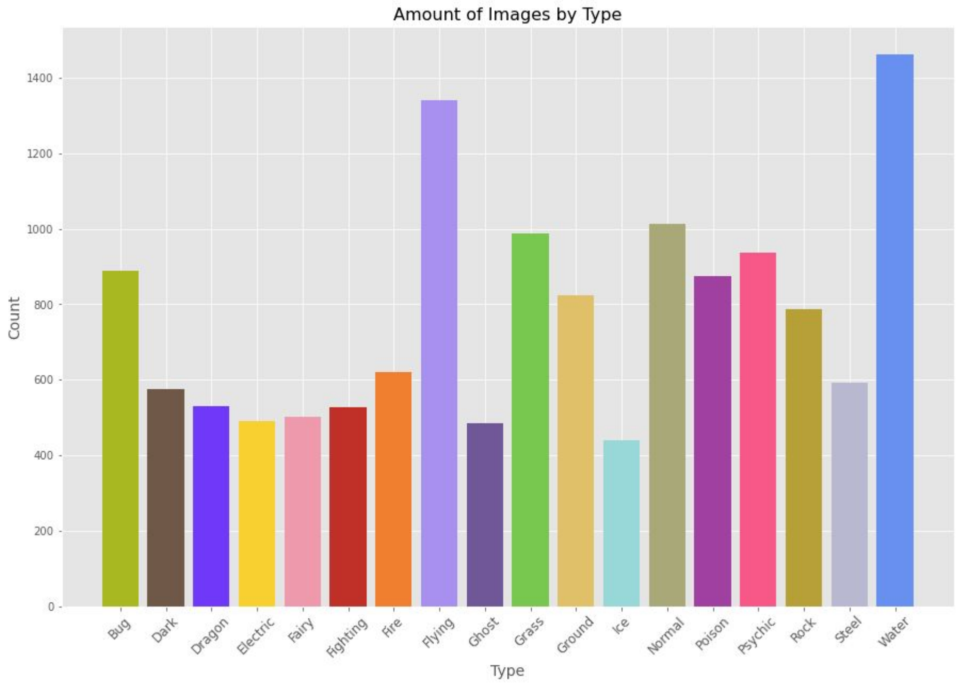


Image Distribution

- Water types were still most prevalent, while Ice types were still least common



Data Processing Pipeline (EDA)

- Utilized train-test split to split into training, testing and validation set:
  - Training Set: 5350 images
  - Testing Set: 1338 images
  - Validation Set: 1672 images

- These were split and stratified with the same random state, 20% split, and class weights were considered when training

Metric Justification

- Why precision and recall?
  - Goal is to successfully alert a user and advise switching Pokémon when the model recognizes a Pokémon that poses a threat
  - High <u>precision</u> will help determine bad switches, or falsely switching Pokémon when there is no threat
  - High <u>recall</u> will help determine bad 'stay-ins' or failing to identify a threat that most likely results in your Pokémon being defeated
    - Recall is VITAL in this model as reviving Pokémon can cost real life currency to a Pokémon GO player

CNN Hyperparameters

- Input Shape: (256,256,3)
- Kernel Size: (3,3)
- Pool Size: (2,2)
- Dropout: 0.25/0.50
- Activation Layer: ReLu
- Final Activation Layer: Sigmoid
- L2 Regularizer: 0.01
- Batch Size: 16
- Class Weights: Calculated based on ratios

- Optimizer: Adam
- Loss: Binary Cross-entropy
- Metrics: (Accuracy), Precision, Recall (Threshold = 0.4)

CNN Structure

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 254, 254, 16)      448
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 127, 127, 16)      0
_________________________________________________________________
dropout_6 (Dropout)          (None, 127, 127, 16)      0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 125, 125, 32)      4640
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 62, 62, 32)        0
_________________________________________________________________
dropout_7 (Dropout)          (None, 62, 62, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 60, 60, 64)        18496
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
dropout_8 (Dropout)          (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 64)        36928
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
dropout_9 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 12544)             0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               1605760
_________________________________________________________________
dropout_10 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_11 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_5 (Dense)              (None, 18)                1170
=================================================================
Total params: 1,675,698
Trainable params: 1,675,698
Non-trainable params: 0
_________________________________________________________________
```

- Our model structure is shown to the left:
  - 4 Convolutional Layers
    - Pooling
    - Dropout = 0.25
    - ReLu
  - 3 Dense Layers
    - Dropout = 0.5
    - ReLu and Sigmoid

Model Results

- Baseline predictor
  - Dummy classifier using stratified strategy
- Results:

Model	Precision	Recall
Baseline	11.72%	11.85%
Our CNN Model	83.90%	48.63%

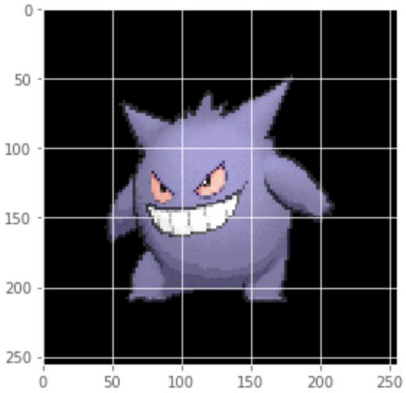
Model Results

- Model Performance after 200 epochs:

Data	Precision	Recall
Training Data	99.18%	85.40%
Validation Data	83.50%	45.59%
Testing Data	83.90%	48.63%

The ‘Good’

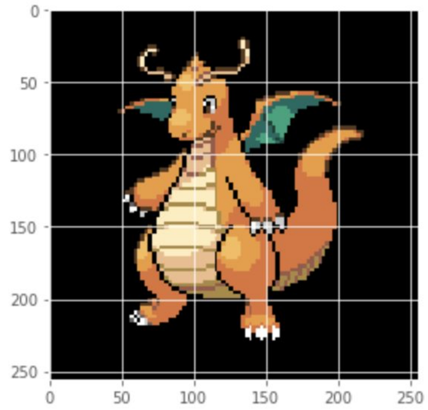
Ghost 88.2%
Poison 86.9%
Dark 8.46%
Dragon 2.26%
Grass 1.47%



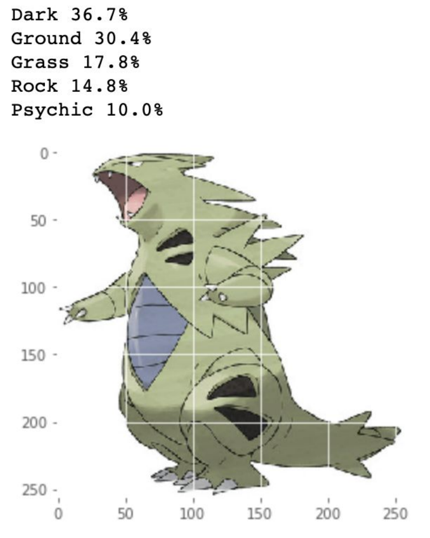
Ghost Poison

Dragon Flying

Dragon 95.9%
Flying 95.1%
Dark 0.615%
Ground 0.387%
Water 0.362%

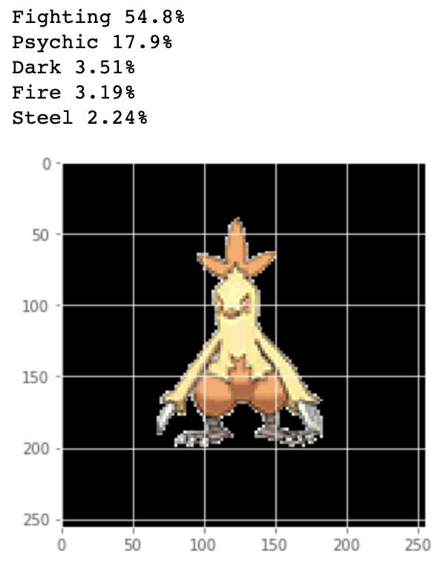


The ‘Bad’

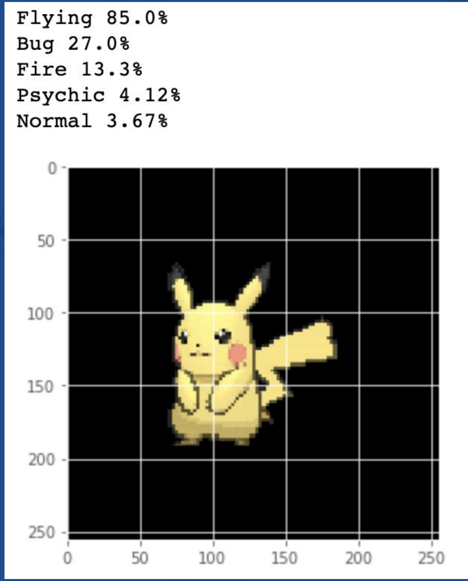


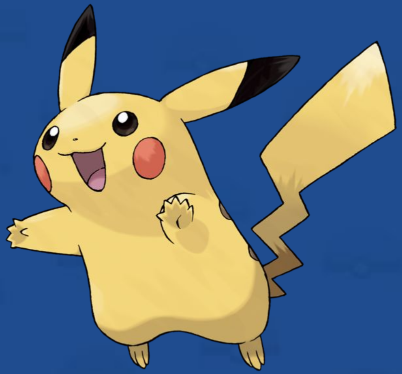
Rock Dark

Fire Fighting

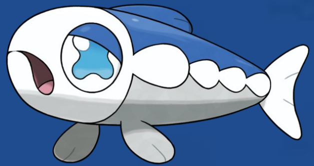


The 'Ugly'

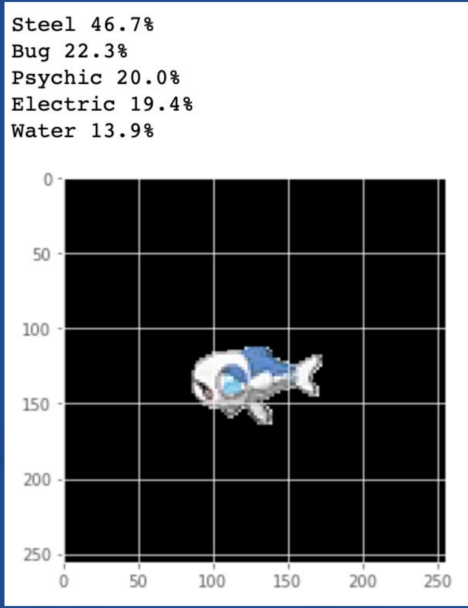




Electric



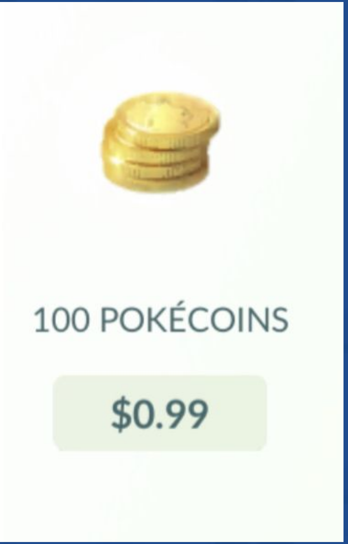
Water



Use Case?

- Pokemon GO application
- Roughly $0.30 to revive a Pokémon
    - Certain amount of revives are awarded for playing

- If our model can identify threats at 36.78% higher recall than the baseline, we could potentially save a player up to $0.11 when the opposing Pokémon poses a threat to the user



Additional Steps

- IMPROVE THE MODEL
- Repeat with more images of Pokémon:
  - More images without ‘clean’ backgrounds
  - Different angles, fan-made drawings

- Look into creating an ROC curve by adjusting thresholds
- Find best/worst performing individual labels
- Apply same modeling pipeline to many other image classification problems

- For Detailed Analysis: https://github.com/jtlaurel/Pokemon-Type-Classifier

Any Questions?

Github: github.com/jtlaurel
Email: jtlaurel46@gmail.com

Pokémon