What’s That Pokemon?

Tariq Zahroof
Stanford University
tzahroof@stanford.edu

Abstract

This paper investigates Pokemon type prediction given a sprite input. I developed two neural network architectures- a convolutional neural network (CNN) and a fully-connected network (dubbed SNN) with a 1-hot input vector derived from the sprite’s colors - to observe the effectiveness of color-informed and shape-informed decision-making on a small training set. Models were evaluated by traditional metrics (F1 score, recall, and precision). Although both models performed similarly and decently, they ultimately suffered from correlation problems between the train and test set and the vague visual design evidenced by my approximation of Bayes error by human analog.

1 Motivation

As of June 2018, Nintendo’s Pokemon franchise was the highest grossing franchise of all-time, earning almost $60 billion since its creation. Pokemon has permeated into all areas of social life, including the mobile Pokemon Go! craze of 2016, the recent $150 million `Detective Pikachu` movie, and the pervasiveness of the yellow, lovable mascot, Pikachu. Furthermore, with the announcement of the new Pokemon Switch games, Sword and Shield, Nintendo’s juggernaut shows no signs of slowing down its cash cow. But, is the behemoth actually as strong as it portrays itself, or is it slowly splitting at the seams?

Pokemon are animals in the Pokemon universe, and each Pokemon has 1 or 2 affiliated types to indicate its strengths and weaknesses. Effective visual communication of Pokemon typing is key, as players of all ages battle (i.e. dog-fight) 807 unique Pokemon at the time writing. As such, I propose to create a neural network to predict the typing of a Pokemon, based on image inputs, to evaluate past and present Pokemon design clarity.

1.1 Neural Network Introduction

Inspired by the Pokedex of the in-game universe, I sought to create the best neural network architecture to identify the type combination of a Pokemon. Given an input of an RGB (3-channel) 64x64 Pokemon sprite image, the neural network identifies an 18-dimensional vector detailing the probability of that Pokemon belonging to a specific type via the output of a sigmoid layer. Afterwards, a prediction algorithm runs to determine 1.) which types and 2.) how many types to assign a Pokemon. In the spirit of the games, the latest generation (Generation 7) served as the test set, modeling how players interpret new Pokemon based on their knowledge of previous generations.

To this end, I tried two different neural network architectures. One architecture, the **convolutional neural network (CNN)**, directly passes the 3x64x64 image through a series of convolutional and pooling layers before arriving at the fully-connected sigmoid layer, thus using all image information for type determination. The other architecture, dubbed the **simple neural network (SNN)**, converts the image into a 15-dimensional 1-hot vector indicating the 5 most prominent colors, and then uses a traditional fully-connected model architecture for results. The SNN tailors the model to focus on colors specifically and disregard form entirely. Ultimately, both models had very similar results, suggesting that color is the key component for type identification given training data of only images from previous generations’ Pokemon.

2 Related work

Previous work on neural networks for Pokemon type classification was done by Soares (1) for the unofficial *Journal of Geek Studies*. Soares used a small convolutional neural network (2 (convolution

-> max pool) layers -> 2 fully connected layers) with a softmax output for single type identification. Notably, his use of data augmentation to artificially increase the training set yielded promising results. I sought to improve upon his work with multi-label classification, a significantly bigger, custom dataset, and a more flexible network. For this purpose, Tsuomakas, Grigorios, et al. (2), suggested the use of cross-entropy loss for multi-label classification.

A large concern for the model's effectiveness was the small dataset of only 807 unique Pokemon (although multiple incarnations exist for each, depending on the generation of introduction). In their research on networks with small datasets, Srinistava, Hinton, et al. (3) suggested the use of dropout after each activation function layer to avoid overfitting. Further work by Yuan and Fine (4) recommended reducing the inputs' dimensions via human insight to reduce noise in the training data. On the other hand, Liapis noted that convincing fake Pokemon could be made by evolutionary algorithms using color palettes as inputs (5). In conjunction with Yao, Zweig, et al's work (6) in natural processing to represent words as 1-hot embedding vectors to carry pertinent information, I was inspired to develop the 1-hot color vector input for the SNN network.

3 Dataset and Features

3.1 Data Collection

The dataset was custom-harvested from the fan-made Pokemon Showdown battling client. By analyzing the code, I located the client's online web repository of sprites for every generation of Pokemon games (including front, back, and shiny portraits), used masks to magnify and center the sprites, and then applied an RGBA-to-RGB converter to locally save the images as 64x64x3 images. Finally, I altered the client's Javascript code to store a JSON file detailing the label vectors (i.e. types) for each Pokemon sprite. A sample of the images and the types is shown in figures 1 and 2, respectively.

Figure 1: A sample of random Pokemon sprites

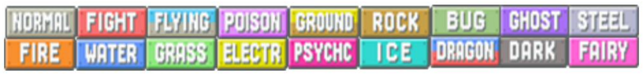


Figure 2: The 18 types of the label vector

3.2 Data Input

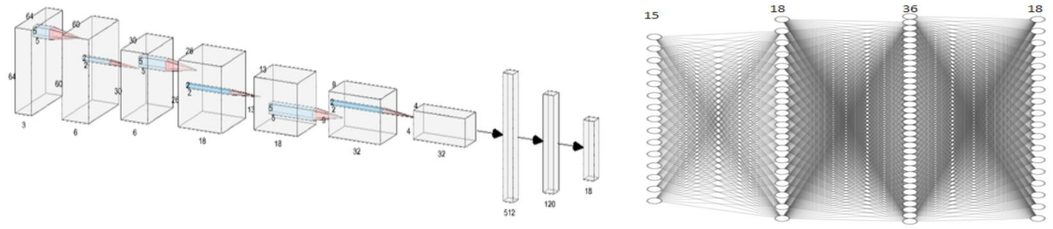
3.2.1 CNN

Initially, I tested the models by inputting images without data augmentation. However, although the training error reduced to 0, the network performed poorly on the test distribution due to training set memorization. As such, I performed data augmentation to generalize the network to any sprite orientation and focus the network primarily towards color for type identification. Data augmentation randomly altered the orientation of the image, and included rotation (up to 40 degrees), translation (up to 0.2x in x and y), scaling (from 1x to 1.3), and horizontal flipping (with 0.5 probability).

Data augmentation prevented the network from memorizing the training set, but also stagnated the training error to around 0.250 within a reasonable training time. These results implied that color and shapes were insufficient for predicting types, given the dataset size.

3.2.2 SNN

To create a 1-hot input vector to detail the embeddings of the images, I used the Python `colorgram` package to extract the top 5 colors (RGB components), and then stacked the colors in order of greatest



(a) CNN (b) SNN

Figure 3: Model architectures

proportionality. The largest color, the background color, was excluded from the vector as noise. The form of the vector is shown below:

$$x = [\text{red}_{\text{color 1}} \quad \text{blue}_{\text{color 1}} \quad \text{green}_{\text{color 1}} \quad \dots \quad \text{red}_{\text{color 5}} \quad \text{blue}_{\text{color 5}} \quad \text{green}_{\text{color 5}}] \tag{1}$$

No data augmentation was performed, as the network failed to reduce the training error to 0 with the tested small model structures within a reasonable time. Additionally, for this network, the first generation of Pokemon games (Pokemon Red, Blue, and Yellow) was omitted due to the grayscale sprites due to hardware limitations at the time. In contrast, the aforementioned games' sprites was kept for the CNN to inform shape-based classification decisions.

3.3 Data Split

The data was split into two sets: train and test. Test data was selected from the latest generation, representing a player's encounter with never-before-seen Pokemon in a new Pokemon game. More importantly, this split ensured the greatest amount of training data for the models. Although there are 7 generations of Pokemon games, each game only has sprites from the current and previous generations. Thus, relegating Pokemon from earlier generations to the test set would deprive network of up to 7 unique sprites, as adding the Pokemon's other sprites to the train set would result in overly optimistic prediction errors.

Initially, random images from the training set were saved to the train-dev set for model tuning. However, this caused two major problems: 1.) the dev error was optimistic, since the model had been exposed to the Pokemon before (albeit in different forms), and 2.) the dev error was not representative of test set performance, since the test distribution was later observed to be different from that of the training set, which in turn caused the train-dev set to further bias the model away from the test distribution.

I took advantage of both the front and back portraits available of each Pokemon to increase the training set size. However, shiny Pokemon (alternative recolorings of Pokemon) were not incorporated into the datasets, as the novelty color palettes would likely confuse the neural networks, as they do not make much sense to humans either. Ultimately, the training set was 5160 and 4858 for the CNN and SNN, respectively, while the test set was 366 images. Due to the small test set size (only 7% of the data), a dev set was not created with the test data. As such, the test set was effectively treated as the dev set for the unreleased Generation 8 of Pokemon (while preserving the limited train set size).

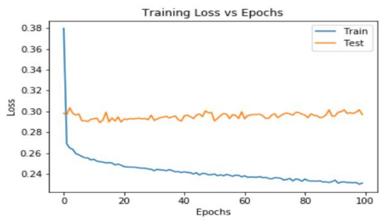
4 Methods

All models were implemented in PyTorch, due to in-built Python scripting support and ease of rapidly testing different model structures. Model training was performed on an AWS EC2 Deep Learning instance.

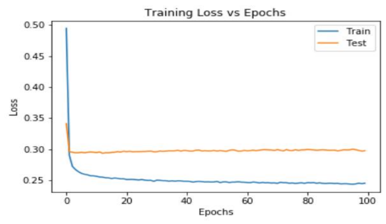
4.1 Loss Function and Prediction

Both networks used traditional cross-entropy loss on an 18-length vector, effectively comparing the prediction's type probabilities to the actual type combination. Thus, the model attempted to minimize the following loss function:

$$L = \frac{-1}{N} \sum_{i=1}^{N} y_i * \log(\hat{y_i}) + (1 - y_i) * \log(1 - \hat{y_i}) \tag{2}$$



(a) CNN



(b) SNN

Figure 4: Train/test loss graphs

After the algorithm output the 18-dimensional predicted label, the prediction algorithm computed the actual predicted type combination. The algorithm would choose the up to two of the largest types with probabilities above a certain threshold; if no type satisfied this condition, then the type with the largest probability would be selected. Initially, the threshold was set at 0.5. However, due to the small training set and the relatively large number of types, the type probabilities were fairly low; as such, I adjusted the threshold to 0.2 for improved multi-type guessing performance.

4.2 CNN

A network of 3 (convolution -> max pool layers) -> 3 fully connected (FC) layers (3Con3FC) was selected, as shown in Figure 3. Batch normalization and dropout were applied after the activation functions of each layer to speed up training and avoid overfitting the small dataset, respectively. Due to dataset size, I focused on testing small networks to prevent overfitting, and used a filter size of 5 in the convolutional layers to gather more information about color than shape. Three fully connected layers helped reduce the large input from the convolutional layer to the 18-dimensional output. The model train/test loss can be found in 4.

Initially, I tried a 2Con3FC model after data augmentation (similar to Soares' approach), but noticed that the training error would stagnate around 0.25. As such, I tested 3Con3FC and 5Con3FC models to see if the model was not flexible enough. Although these models reduced the training error, the test error did not reduce, and quickly stagnated or began to increase as epochs passed. I then implemented dropout to further reduce over-fitting, but all models still produced relatively the same error. Ultimately, the 3Con3FC model was chosen for having the lowest cross-entropy loss and F1 score by a small margin.

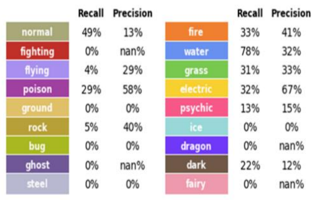
4.3 SNN

A network of 3 FC layers was ultimately chosen, as shown in 3. Again, batch normalization and dropout were applied after each FC layer for faster training and to avoid overfitting, respectively. Additionally, the network was kept chosen to match the small predictor space to the small output space and the small training set. I also tested a 2 FC layer and 5 FC layer network as well, and chose the 3 FC layer marginally outperformed the other 2. The same problems from the CNN- the test error stagnating very quickly- affected the model, suggesting an inherent irreducible error between the training distribution and the test distribution. The train/test loss of the model can be found in 4.

5 Results and Discussion

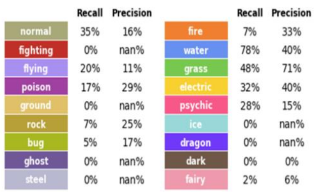
Both models used a 0.01 learning rate for fast training times while still maintaining accuracy, after having tried 0.1, 0.001, and 0.0001 as well. Adam Optimization, with its effective default parameters per paper recommendation, was used over Gradient Descent to further reduce the training error once it stagnated. ReLU served as the activation function for hidden layers. The minibatch size was set to 32 to balance the time per iteration and the noise introduced by the small batch size (as compared to 16 and 64). Finally, deeper layers of both networks increased in the number of channels towards the middle of the networks, essentially gathering more information from the inputs before summarizing it as the 18-length vector. Working in multiples of 2 compared to the previous layer's nodes provided the most consistent results.

To give a human-analog to the Bayes approximation, I (a veteran of Pokemon generations 1-6) predicted the types of the test set (Generation 7) and added my results as a baseline comparison. The following Figures 1, 5 show the precision, recall, and F1 score for the models. Frequently, the model



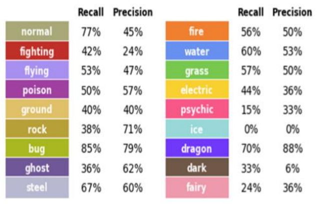
	Recall	Precision		Recall	Precision
normal	49%	13%	fire	33%	41%
fighting	0%	nan%	water	78%	32%
flying	4%	29%	grass	31%	33%
poison	29%	58%	electric	32%	67%
ground	0%	0%	psychic	13%	15%
rock	5%	40%	ice	0%	0%
bug	0%	0%	dragon	0%	nan%
ghost	0%	nan%	dark	22%	12%
steel	0%	0%	fairy	0%	nan%

(a) CNN



	Recall	Precision		Recall	Precision
normal	35%	16%	fire	7%	33%
fighting	0%	nan%	water	78%	40%
flying	20%	11%	grass	48%	71%
poison	17%	29%	electric	32%	40%
ground	0%	nan%	psychic	28%	15%
rock	7%	25%	ice	0%	nan%
bug	5%	17%	dragon	0%	nan%
ghost	0%	nan%	dark	0%	0%
steel	0%	nan%	fairy	2%	6%

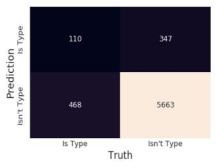
(b) SNN



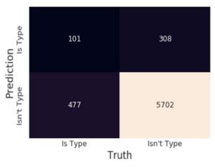
	Recall	Precision		Recall	Precision
normal	77%	45%	fire	56%	50%
fighting	42%	24%	water	60%	53%
flying	53%	47%	grass	57%	50%
poison	50%	57%	electric	44%	36%
ground	40%	40%	psychic	15%	33%
rock	38%	71%	ice	0%	0%
bug	85%	79%	dragon	70%	88%
ghost	36%	62%	dark	33%	6%
steel	67%	60%	fairy	24%	36%

(c) Tariq

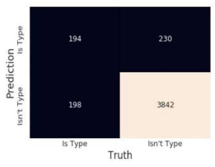
Figure 5: Type-based recall and precision statistics



(a) CNN



(b) SNN

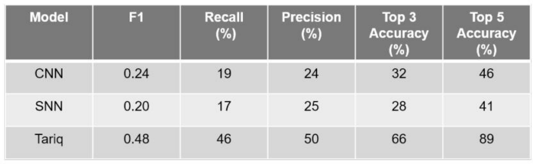


(c) Tariq

Figure 6: Confusion matrices

would have 0 true positive and 0 false positive for shape-based classes, resulting in "nan" scores for precision. Additionally, note that the confusion matrices (Figure 6) can be misleading, as there were a minimum of 14 true negatives per Pokemon, versus a max of 2 false positives and false negatives.

The model results were less than desirable, but understandable upon inspection. The first cause is the discrepancy between the train and test distributions. To excite players, Nintendo actively creates new Pokemon different from previous generations to demonstrate that the franchise is not stagnating, and thus the test set would likely be different from the training set. Instead, the new Pokemon are inspired by real-life objects and creatures not already incorporated into the franchise. The second aspect stems from the lack of unique Pokemon in the dataset- with only 807 unique shapes and 18 types, shape-based type identification (e.g. dragon) is difficult. In contrast, the models had relatively high precision and recall statistics for color-based types (e.g. fire, water, etc.) The third cause is the issue of other Pokemon information. Consider Vikavolt, the bug, electric Pokemon. The electric typing is not obvious from the imagery; any combination of bug, steel, flying, dragon, and water would be more reasonable. However, its name "Vikavolt" underlines its electric typing, while its unique Pokemon ability, levitate, endows it with the properties of a flying type. As such, image-based identification provides only a partial view of the Pokemon's typing. Finally, Nintendo's type categorizations do not always make sense, as evidenced by my F1 score of 0.48, and the seemingly random Pokemon typings (for example, Altaria, a cotton candy bird, is a dragon/flying typing due to a Japanese pun).



Model	F1	Recall (%)	Precision (%)	Top 3 Accuracy (%)	Top 5 Accuracy (%)
CNN	0.24	19	24	32	46
SNN	0.20	17	25	28	41
Tariq	0.48	46	50	66	89

Table 1: F1, recall, precision, and top X accuracy metrics of the models

6 Conclusion and Future Work

In conclusion, the CNN and SNN model architectures provided similar, somewhat weak results for identifying Pokemon typing from image inputs. The CNN used the full image input, while the SNN only used the top 5 colors. However, both small architectures primarily leveraged color information to guess typing, but suffered from a non-representative training set for the test distribution, resulting in a stagnating test error. Two ways to improve the models would be to train the model on real-world creates and objects, hand-labeled to the types (particularly for shape-based type identification). Secondly, by including names into the model and determining the type-affiliations to the roots of the words. Ultimately, though, the Pokemon universe is not easily decipherable, and an inherent error will always be present, as demonstrated by my approximation of Bayes error.

7 Code and Acknowledgements

My entire codebase is available at `https://github.com/tzahroof/PokemonTypeNN`. `Training.ipynb` and `Color_Training.ipynb` contain the Jupyter notebook with the code for the Neural Network and prediction algorithms. The code uses the PyTorch, MechanicalSoup, NumPy, colorgram, and PIL libraries, and uses a portion of image cleaning and display code by Henrique Soares (1). Additionally, I would like to thank Ashwin and Steven for their help and insight into the project.

References

[1] H. Soares, "Who is that neural network?" in *Journal of Geek Studies*, 2017.

[2] G. Tsoumakas, I. Katakis, and I. Vlahavas, "A review of multi-label classification methods," in *ADBIS workshop on data mining and knowledge discovery*, 2006.

[3] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[4] Jen-Lun Yuan and T. L. Fine, "Neural-network design for small training sets of high dimension," *IEEE Transactions on Neural Networks*, vol. 9, no. 2, pp. 266–280, 1998.

[5] A. Liapis, "Recomposing the pokémon color palette," in *Applications of Evolutionary Computation*, 2018.

[6] K. Yao, G. Zweig, M.-Y. Hwang, Y. Shi, and D. Yu, "Recurrent neural networks for language understanding," in *INTERSPEECH*, 2013.