PokeTaipu: A ResNet-Based Web Application for Pokemon Type Classification

Yihe Guo, Melody Li, Yue Shu, Roxanne Yang, Gary Yao

Case Western Reserve University
10900 Euclid Ave, Cleveland OH 44106
{yxg352, yxl1632, yxs626, hxy298, gxy76}@case.edu

Abstract

Pokémon, probably the most well-known video game across the world, has released eight official versions since the foundation of its franchise by Satoshi Tajiri in 1995. Over two and a half decades, the game has influenced players from all generations with its game play, anime, and most importantly, various types of pokemons. In order to proivde a more intelligent approach to classify the types of all pokemons, we developed PokeTaipu, a Angular-based web application to classify pokemons from Generation I of the Pokémon franchise, as our phase one implementation. In this report, we will introduce in details about the architecture design of PokeTaipu and how the back-end model was trained. In particular, we trained the model in both Residual Neural Network and Convolutional Neural Network to compete for the best result. Based on our experimental data, we concluded that the overall performance of ResNet in this scenario surpasses that of ConvNet.

1. Introduction

In the new information age, the stress and temptation are overwhelming for all members of the society. College students, in particular, are among the most overworked individuals of all. Fortunately, the modern technologies have provided us much convenience and entertainments. One of the most popular technology topics is machine learning, and neural network are among the most used algorithms. While neural network can have many versatile uses such as auto-driving, image processing, and facial recognition. We have decided to break the tradition in the usage of neural network and apply it in a way that would help reliving stress. We decided to combine it with another topic hot in recent years - Pokemon. Ever since the creation of Pokemon by Satoshi Tajiri and Ken Sugimori in the 1980s, eights generations of Pokemons have been designed and integrated into games and anime. There are currently 890 Pokemons spread over 18 types: fire, water, grass, electric, ground, ice, flying, rock, steel, normal, fighting, ghost, dark, psychic, poison, dragon, fairy, and bug [1]. Despite the unique appearance of each individual Pokemons, there is an internal pattern underlying all designs with respect to their types. Given an image of a poekmon, classify it to one of the types. Such a task is perfect for neural network to carry out. We have developed and trained two neural network which classify the type of a pokemon image. One utilized a traditional convolutional neural network, the other is a deep network of 18 layers and trained with the residual learning framework. The trained network is then saved and ready to take input.

The appearance of each pokemon are drastically different. Many pokemon were created base on animals, and some pokemons can be quite colorful. However, those who are familiar with pokemon games can usually guess the type of a certain pokemon upon seeing them. This is because the pokemons are designed by a certain pattern. For example, grass type pokemon generally look green and have plant parts like leaves or flowers attached to them. Fire type pokemon are usually red and electric types are yellow. Besides color, facial expression also differ by types. Ghost type pokemons usually look scary or nonchalant. Fairy type pokemon, on the other hand, commonly have jewels attached to them.

The rest of the paper will be structured as follows: in Section 2 we will introduce where we crawled our training and testing data from. In Section 3 we will elaborate with further details on how we cleaned our data before training, how we evaluated the ResNet approach and ConvNet approach to train our model, as well as how we structured the architectural design of our web app implementation. In Section 4 and 5 we will compare the results we had from our training, and finally in Section 6 we will summarize what we take away from this project.

2. Data

We utilized the first generation pokemon data from Kaggle [3]. There are images of 151 pokemons, roughly 60 images for each pokemon. The images came with the pokemon name, type, and other information such as their attack and speed. The images came in a varity of formats such as jpg, png, and gif. The images are either screenshots taken from the pokemon anime or fan drawings on various backgrounds. The diversity in the art style, shade of color, and similarity to the original pokemon design in our image data allows the model to achieve a good generalization.

3. Methods

The users will be uploading a pokemon image through the web application portal. The image will then be converted to a `numpy` array of desired dimension with the help of Pillow package. This `numpy` array will be the input of our already trained network and outputting the label for this image. The output is then displayed back to the user through the web application. The network is pretrained in order to shorten the waiting time for user. As mentioned in Section 1, we trained two neural networks and compared their accuracy in order to correctly classify a pokemon image to its type. The better network was saved using the pytorch library and the save function in the `nn.Module` class, which saves a trained network by storing the state dictionary (the structure and parameter of the trained network) into a pt file. The front end was developed with the Angular Typescript framework. The server side was developed by the Python Flask framework. Although Flask is not the most common server-side development framework, we used it because it is written in Python, and our models are also implemented in Python. With a Python-based server-side code, our web application would be easily integrated with the model data [2].

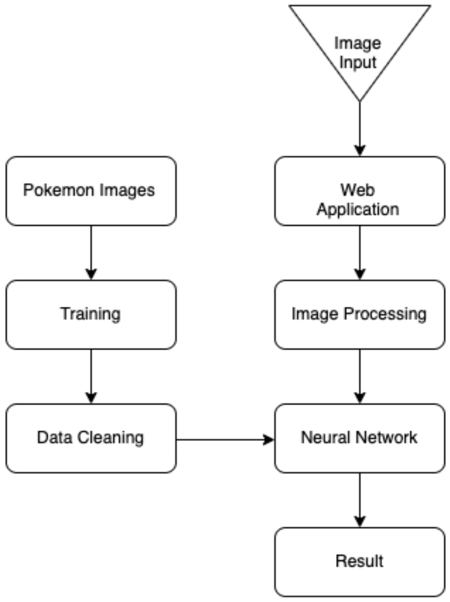


Figure 1. PokeTaipu Application Workflow

3.1. Data Cleaning

The back-end of this project is split into two parts: data cleaning and model training. The data training part is crucial for the project as it prepares the data our models train on. In the data preparation part, the images go through several steps in its transformation to usable data for the model. Originally images were stored in folders named after respective pokemons. We mapped every pokemon's type to the pokemon name of that type.

1. Label Processing

   First the types of different pokemons are transform to one hot encoding and outputted to a separate cvs file in `typeProcessing.py`

2. Matrix Transforming

   Then the python script `labelImages.py` iterate through the images in different pokemon folders and transforming images to 224*224*3 numpy arrays using random cropping from the Pillow package

3. Matrix Merging

   The above python script also loads the cvs file that contains the one hot encoding for types with respect to each pokemon and concatenated all data matrices and label matrixes into large matrixes of sizes sample*224*224*3 and sample *18 respectively.

4. Data Usage

   After the desired numpy arrays are created, we use `loader.py` to load the numpy arrays in both `model.py` and `torchTestTrain.py` before we train the neural network models.

For our models, we require out `numpy` array to be of size 3 x 255 x 255 as our input images are expected to be in `RGB` format. If the size of the image is larger than the required size, we simply use the `ImageOps.fit` method from the `Pillow` array, which has builtin random cropping to ensure the validity of the remaining image. However, when the size of the input image is smaller than the required size, we need to perform some auto fitting on its size by replicating the image three times over a white background.

3.2. Models

In order to produce the best classification result, we compared the classification result of two neural networks. The

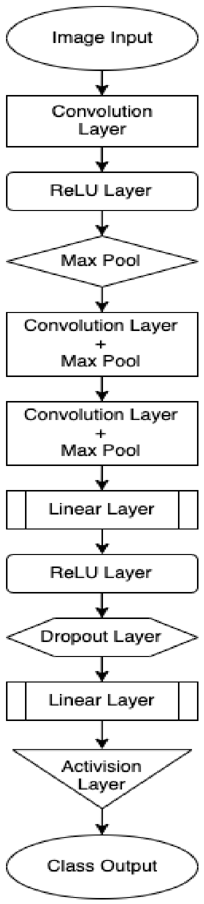


Figure 2. PokeTaipu Convolutional Network Model Representation

first one is a convolutional neural network with max pooling and drop out. The second one is an eighteen-layer deep network that utilize deep residual learning for image recognition. model.py contains the convolutional network and tourchTestTrain.py contains the deep network.

Both python scripts contains a $classify$ method which is left as an interface for the front end. This method takes an input string for the path of an image, and output the type label for this image.

The data processing step would give us readily usable numpy arrays. We call loader.py in both models' python script in order to load all of the image arrays for later training.

3.2.1 Convolutional Neural Network

Convolutional neural networks work particularly well with images and if commonly used for image classification, so we decided to try it out for our pokemon type classificatoin. We created a model with three convolutional layers, paired with max pooling of size 2 and a drop out rate of 0.25. We used several activation such as Rectified Linear Units and log softmax function. This is a fairly standard design of convolutional neural networks, and judging by our experiment results, it performs better than other configurations. We have made several tuning attempts in order to achieve the end result we have. The details about the experiments are listed below in the Section 4 [5].

This network uses the number of correctly classified images as the accuracy measure, and uses the Adagrad optimizer to impove results.

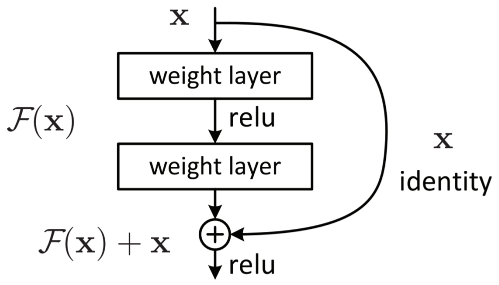

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 3. Building Block for the Residual Network [4]

3.2.2 Deep Residual Learning

As researchers tackled classification problems with deep convolution neural networks, one main challenge was effectively passing back gradients. The tendency of precision loss and other issues in neural nets of 100+ levels counter intuitively produced models of worse train and validation loss. This issue was elegantly solved by the development of the Residual Networks using jump connection between convolution layers. This allowed smoother gradient decent in ultra deep models leading to recording break developments in deep learning.

As a demonstration of the underlying theory, a shortcut connection which perform identity mapping is shown below. Let's call the desired underlying mapping as $H(x)$, and define $F(x) := H(x) - x$ for the mapping created by the nonlinear layers. Then the original mapping naturally transform to $F(x) + x$. This can be realized by a feed forward network with jumping connects.

Now let's consider $H(x)$ as an underlying mapping of a few stacked layers (might not be the entire net). We hypothesize that these layers approximate the residual function $F(x) := H(x)x$, and the original function then becomes $F(x) + x$. We consider such approximation done by stacked layers are the building block of the deep network. To formalize the syntax, the we consider the building block as $y = F(x, W_i) + x$ where x and y are the input and output for the layers in consideration. If we adopt the residual learning to every few layers, using the building block shown

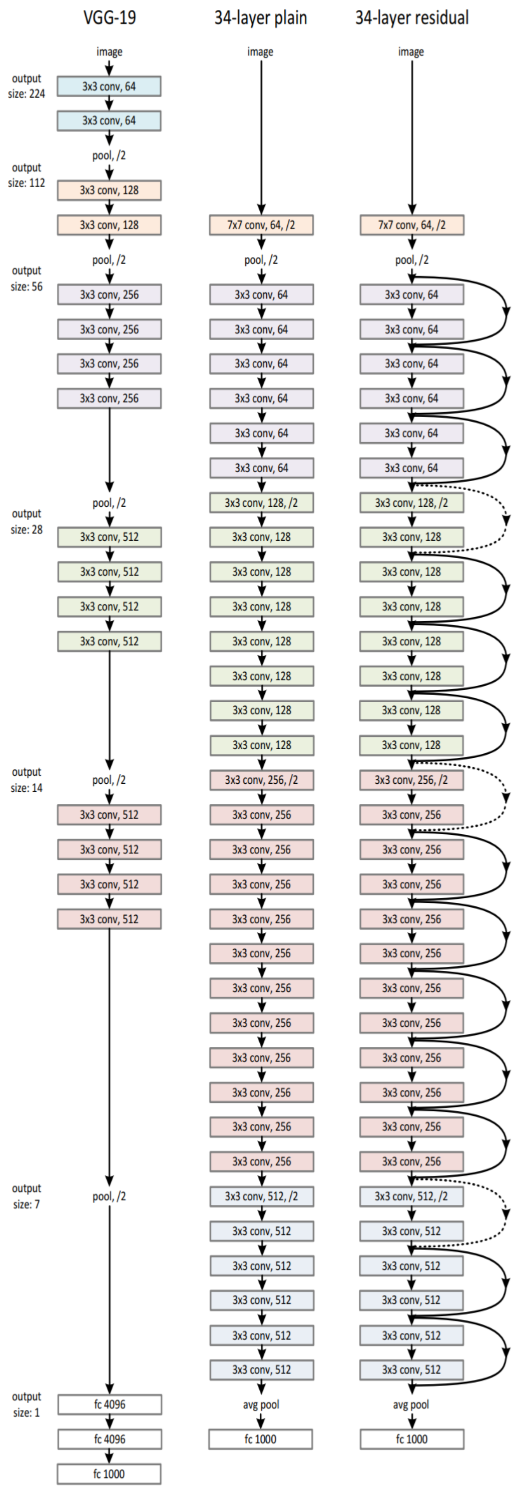
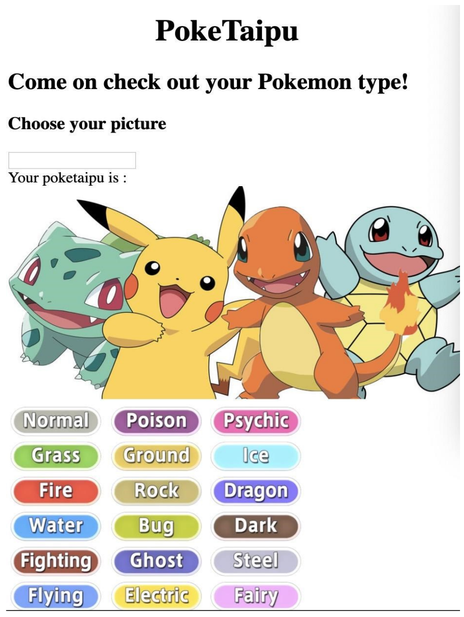


Figure 4. The network architecture of the whole ImageNet [4]

in Figure 3, we can construct the architecture of the lager network in Figure 4.



PokeTaipu

Come on check out your Pokemon type!

Choose your picture

Your poketaipu is :

Figure 5. PokeTaipu Web Application User Experience

In our classification problem, the input dimension and difficulty of classification suggested the need for complex models. Therefore, we sought to train Resnet models on our Pokemon dataset to establish a base line of comparison against the models We had built.

3.3. Web Application

We built a web application to make our training model more impactful in an entertaining way. Our goal is to let users choose their own picture and use our model to tell them what PokeTaipu it is for their own pictures. The user may input any kind of pictures, like their selfies or a Pokemon picture that they would like to know more about.

We used Angular Typescript as the framework to build this application client-side. As for the server-side, since we will be integrating with the model which is written in Python, we used the Python Flask framework in order to have a smooth integration with our source code. Right now, we have a fully functioning web application that can be run on a local machine. It is able to classify any input pictures into one of the 18 PokeType.

We initially designed a user interface where the user can use a file chooser component (like File Explorer in Windows and Finder in Mac OS) to upload a file and send the local path to the server to further execute the uploaded image. However, we realized during development that the Angular framework has the security setting, so that browsers can never read the real local path data. Instead, the client-side can only get the uploaded file as raw data and send it to the server-side. The attribute path in the event after a file upload is actually a fake path. In this way, we asked the user to provide the local path to send to the server-side.

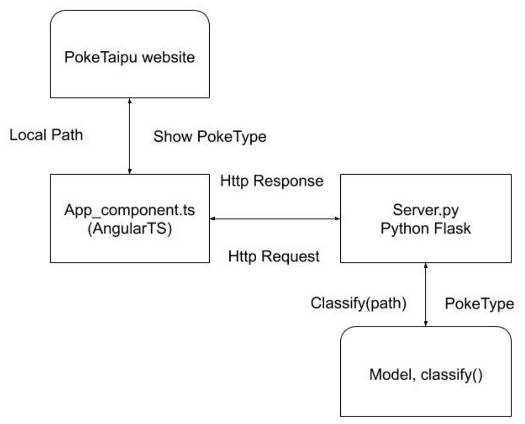


Figure 6. Current Workflow of PokeTaipu Web Application

To run the web application on local, the user needs to run the Flask server and Angular client. The server-side code is in server.py and should be executed by the command line python server.py. The client-side code can be run in the project by running "ng serve". The web application will be running locally and has access to the local path.

Currently, this web application can be run on local and has not been deployed due to some limitation. In our model source code, it takes an image file in the form of the local path and converts that into a numpy array. In the future, in order to have better user experience with file uploading component, the model source code should consider another way of converting image to numpy array. We should look into how to convert image raw data into a matrix, instead of reading from local path. Due to security protection reasons, the local path cannot be read by the browser from any web development framework [2].

In the future when there will be a need for deploying this website, or to develop PokeTaipu mobile applications, there will be a need to change the read local path to read raw data as mentioned above.

4. Experiments

As we mentioned above in the model section for convolutional neural network, we have attempted several hyperparameters in order to achieve the best accuracy rate. The results with respect to different hyperparameters are list below:

We also ran several tests for the residual network. After setting the learning rate to 0.005, weight decay to 0.01, the average accuracy achieved its highest at 43 percent.

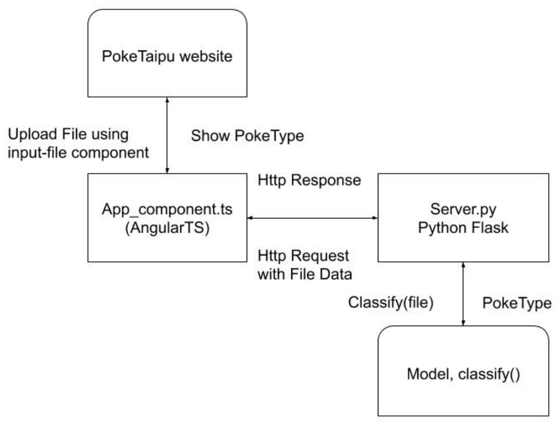


Figure 7. Future Workflow of PokeTaipu Web Application

Experiment description	Avg Loss	Avg Accuracy
3 layers, 2 dropouts	2.3828	23%
Using the cross entropy loss function	2.5761	25%
With only the 2nd dropout	2.2605	32%
With only the 2nd dropout p = 0.25	2.3267	33%
No dropout	3.0342	34%
With only the 1st dropout	2.4478	30%
Added relu layer after 2nd conv	2.8696	31%
With 2 pooling layers	2.4582	29%
512 hidden fully connected units	2.6762	33%

Table 1. Experiment ??

Iteration	Learning Rate	Weight Decay	Average Accuracy
1	0.01	None	40.4
2	0.005	0.01	43.5
3	0.005	0.01	22.5
4	0.005	0.01	29.9
5	0.005	0.01	37.5
6	0.005	0.01	30.4
7	0.005	0.01	32.1
8	0.005	0.01	35.5
9	0.005	0.01	34.2
10	0.005	0.01	40.8
11	0.005	0.01	39.1

5. Results

We can see from the accuracy output comparison of two networks that despite the fewer iterations the residual network has been trained for, it still achieved a higher accuracy rate of 43 percent. Such result is within our expectation. Compared to the three layer convolutional neural network, the eighteen layers residual network is much more versatile and powerful in capturing the minor details separating the pokemon types apart. We had concerns of type classification being too easy of a task as models became more complex, leading to overfitting. It appears from results that overfitting did not occur. While it is certainly true that the deep network is harder to train, the good news is that we do not need to train it for too many iterations for it to surpass the performance of the convolutional network. The residual network also showed much better potential in being a highly accurate model given enough training. Due to technical difficulties, we were not able to utilize the High Performance Computing Cluster at Case Western Reserve. But with proper training in the future, the residual network can not doubt perform to the best of its capacity.

6. Conclusions

Overall, we think the residual network is more appropriate for our purpose of this program. The residual network not only provides better testing performance but also pertains better maintenance. Its algorithm is supported by Microsoft Research, which guarantees that this algorithm will be updated and improved. As the functionality of this application expands, we will be removing the limitation of the input image being a pokemon image. The users will be able to input any picture they want of anything. The network should also be able to classify whether the image is a valid pokemon. If the network is trained well enough, it can even help to contribute to the success of the new generations of the pokemon family.

References

[1] Pokémon, Dec 2019.

[2] B. Chavan. Angular python flask - full stack demo, May 2018.

[3] HarshitDwivedi. Pokemon generation one, Aug 2018.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Dec 2015.

[5] Pytorch. Pytorch examples.