Classification of Pokemon Sprites

Ryan Conley
conley24@msu.edu

Esteban Echeverri
echeve11@msu.edu

Joshua Erno
ernojosh@msu.edu

Jotham Teshome
teshomej@msu.edu

Michigan State University
426 Auditorium Rd, East Lansing, MI 48824

Abstract

In this project, we aim to classify different Pokemon sprites utilizing the power of a convolutional neural network. The model is trained on various types of data, including screenshots collected from Pokemon battles and Pokemon images found in publically available datasets. We also make use of the OpenCV library to process our images, as well as Pytorch and Tensorflow for the creation of our neural network.

1. Introduction

Pokemon has become a household name over the past few decades, and with the franchise nearing its 30th year, there are now over 1000 Pokemon in the game. Due to the franchise's long history, many players that began playing in the earlier generations and have since stopped may have trouble recognizing newer Pokemon sprites. At the same time, it can even be difficult for consistent players to keep up with the ever-growing library of Pokemon there are to choose from, which is only made harder given the pixelated nature of the in-game sprites.

In this project, we aim to train a neural network that can easily and accurately classify Pokemon sprites found within a set of images. With this, we hope to make the identification of Pokemon less of a chore, and something that can be done quickly through a machine learning model.

To do this, we implement a neural network, utilizing the nature of this type of model to handle the complexity that can be seen in classifying multiple types of images into Pokemon. This model can be given a screenshot of an image, and through the use of an image processing library, can isolate the sprite of the rival Pokemon appearing on the screen. From here, the model will be fed the isolated sprite image, where it will finally be able to classify and return the name of the Pokemon that corresponds to the given sprite.

2. Methodology

2.1. Data Collection

To obtain the data necessary for training our classification model, we are taking two approaches. The first method of data collection is to use real battle images from Pokemon games to collect the sprites of Pokemon in a live setting. To do so, we plan to take screenshots of the Pokemon battles with the use of emulators. We will need to make sure the pokemon in each image is clearly visible. We also would like to include images taken from various orientations, so that our model can account for this when classifying images. To make these images suitable for our purposes, we will also need to preprocess them, which we will cover in the following section.

The second method for data collection that we plan to use is data found across multiple sources containing images of Pokemon and their sprites. Using the various sources of images we found, we will combine them to create one large dataset that we can use to train and test our model performance. One of the datasets we are using is Abdullah [1]. This dataset contains over 1000 high-quality images of Pokemon sprites, which will be useful to us as it provides many sprites that we can use when classifying Pokemon. Another dataset we are using is Zhang [2]. While this dataset does not contain purely sprite images of Pokemon, we still believe it will

provide us with useful images when classifying Pokemon.

2.2. Model Creation

2.2.1. Data preprocessing

The preprocessing of our data is done in several steps. First, it is important to note that many of the source images are named as "x.png" where x is a unique id assigned to each pokemon. With that being said, we needed to establish an association between each pokemon's ID and their name. A data set called "pokedex" solved this problem with a csv file. The row number of the file matched the pokemon's id and included other information such as the pokemon's name and type. We extracted this file's information as a pandas dataframe and utilized it to put source images into the correctly named pokemon sub folder.

For each pokemon ID within the range [0, 50), we search the source directories for images named "[ID].png". If a match is found, it is placed in a subdirectory of the pokemon's name along with all the other matched images. In addition to placing the original source images, we also created augmented samples. These samples were either rotated by a random amount, zoomed in slightly, or gaussian noise was applied. The augmented images were placed in the same named subdirectory as the original source images.

2.2.2. Model Architecture

For our convolutional neural network, we used PyTorch to design and create the architecture. We decided to take a simple approach for our initial model and plan on making the model more complex based on our initial results. Our initial model consists of two 2D convolutional layers, with kernel size (3, 3), that are each followed by a Rectified Linear Unit (ReLU) layer. The ReLU layers in our network introduce non-linearity to our model and help it learn complex patterns in the data. Across these 2D convolutional layers, we increased the number of channels from 3 to 64, then from 64 to 128. By increasing the number of channels, it allowed for our model to learn and capture a more diverse set of features from the input sprite images the model was trained on. Next, we used a 2D max pooling layer to downsample the spatial dimensions of the input data while retaining the most important information. Finally, we flattened the features and used a series of two linear layers and a ReLU layer to map the features into the number of classes we are trying to predict from.

We will be taking steps to improve our model's performance on test data. Some of these steps might include: adding more convolutional layers to features part of the network, adding more fully connected layers to the classifier part of the network, increasing the input and output dimensions of various layers in the network, adding dropout regularization to the network to help prevent overfitting, adding batch normalization layers to the network, and experimenting with different optimizer functions. After experimenting with these steps, we believe we will be able to improve the performance of our model and make it more generalizable, allowing it to classify the correct Pokemon at a higher rate.

2.2.3. Model Results

To test our initial model, we use a subset of the sprites that we have preprocessed. This subset of images was generated and set aside prior to the training of the model, so these specific images have not been seen by the model before. Once the images are prepared in a way that our model will accept, we feed these images into the model to evaluate our results. Currently, we are seeing an accuracy of 0.737, which we believe is a good start for our model considering the number of classes we are assigning images to as well as the simple architecture we are using for our current model. We are also testing for metrics such as precision, recall, and f1-score. For these, we are also seeing good results. To test for these values, we use sklearn's precision_score, recall_score, and f1_score functions, using these on the 'weighted' mode to account for a potential imbalance due to the labels we have. Using these, we are able to see that our model is achieving a precision of 0.758, a recall of 0.737, and an f1-score of 0.737. Overall, these are good values to be seeing on our initial model, and we believe we will see an even greater improvement on these metrics as we continue to improve our model.

3. Battle image processing

When it comes to finding objects on images there are multiple ways to approach this problem, it could be using the colors of the pixels as references, processing the image to get the edges from it and try to find a desired shape, and so on. We decided to go with the approach of finding a shape within the image

to get the rival pokemon from a screenshot of the game.

3.1. Finding shapes on Images

When it comes to pokemon sprites is really hard to specify a single shape that can identify any pokemon, specially if we consider the full picture, because we are on the scenario of a pokemon battle there will always be at least 2 pokemon on the image, so to reduce the complexity of this task we are only going to consider 1vs1 battle and limit the search on the right side of the image.

To be able to complete this task we are using OpenCV to cut the image and keep only the right side, then using the function GaussianBlur(...) we smooth the image, to finally use the Canny(...) and findContours(...) functions to find the edges of the image and the shapes that these are part of. Figure 1 is the original image from a pokemon battle, and Figure 2 represents how these transformations look like on the previous image.

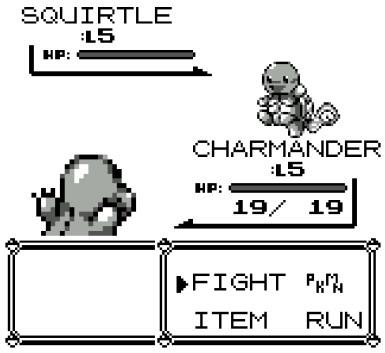

Figure 1. Original image

Figure 2. Edges from a pokemon battle image

3.2. Selecting a shape

As said before selecting a single shape that represents all the pokemon is really hard if not impossible, so the approach that we decided to take to overcome this issue is to take advantage of the simple interface that the pokemon games use, having most of its components being rectangular we can estimate that if we take the most complex shape from the image it is most likely to be the pokemon or at least a segment of the image where this is contained or close to. So following this line of thought we find the most complex shape on the image by listing all the shapes using the OpenCV function findContours(...) and then find which shape is the one with the most edges connected to it, then by knowing the position of the shape we isolated at least one third of the image using its position as the center. Figure 3 represents the final result after applying this process.

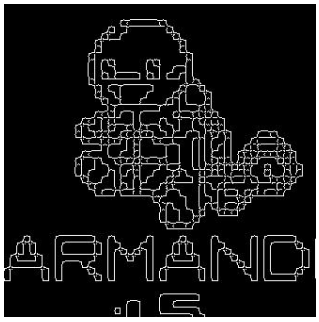

Figure 3. Isolated image of the most complex shape

3.3. Results

After locating the part of the image where the rival pokemon is, what is left to do is just isolate the portion of the original image in the same position as the edges representation. Figure 4 shows the final result.

Figure 4. Isolated portion of the original image

References

[1] Wasiq Abdullah. PokeVision 1010, 2023.
[2] Lance Zhang. 7000 Labeled Pokemon, 2019.