Pokémondium: A Machine Learning Approach to Detecting Images of Pokémon (2023)

Ritwik Raj Saxena, Ibrahim Mamudu, and Eric Nieters

Abstract

Plenty of image recognition programs exist, but very few of them are dedicated solely to Pokémon (refer to the appendix to know what Pokémon is). In addition to online Pokémon directories (called the Pokédexes), many of them do not include image recognition. The goal of our research is to combine image recognition with our Pokémon directory to create a new type of catalog with machine learning. We call our research a Pokémondium, a portmanteau for 'Pokémon compendium' Our paper introduces an empirical comparative investigation into the capability of some familiar machine learning algorithms, including Convolutional Neural Networks, Transfer Learning with Pretrained Models, Support Vector Machines and K-Nearest Neighbors classification, in the context of identifying the Pokémon in a user-uploaded Pokémon image. We have developed a website where users may input images of Pokémon and view the results of our algorithms as they run on those images. Not only does our research provide the name of the Pokémon, but it also displays the statistics for the Pokémon. The performance of the four algorithms is compared over two metrics: accuracy and execution time. The accuracy of our Transfer Learning with Pretrained Models algorithm turned out to be 90.7 percent which was the best among all the models. It is worth mentioning that Convolutional Neural Networks followed close on its heels with an accuracy of 90.2 percent.

Keywords: Object detection, Image recognition, Image classification, Tensors, Reverse image search.

Introduction

Briefly Introducing Object Detection and Image Classification, while Delineating the Need for the Most Accurate Object Detection Algorithm

Object detection involves differentiating and identifying two or more objects in the same image. Image classification is a special case of object detection where an image with one object is classified. Industries like robotics and automation, the fields of medicine, biological sciences, education and learning, business and information technology, law, social media, and many others are deeply dependent on machine learning [1, 2]. With an overabundance of image datasets on the internet, the need for comparative analyses of machine learning algorithms has arisen in recent years. This is particularly important because in the present age of data flareup, before analytics is embarked upon, we must know which algorithm is the most appropriate one to be applied to one of the disparate image types and datasets. This forms the basis of our research.

Pokémon Image Recognition

Our research aims to draw comparisons between the accuracy of four machine learning algorithms to classify any of the first six generations of 721 Pokémon from their images. The algorithms we employed are Convolutional Neural Networks, Pre-Trained Models, Support Vector Machines, and K-Nearest Neighbor classification. By employing a dataset of Pokémon images as a common underlying thread for the estimation of the same metrics across all the algorithms, the respective performances of the models, which are all built upon unique architectures, have been rendered analogous to each other. We evaluated the efficiency of each model and gained insights on the internal working of the algorithms.

Our research comes with a website where a user may upload a Pokémon image, choose the algorithm of their choice, and demand to know what Pokémon the uploaded image contains. The interactive website uses the chosen algorithm, runs it over the image, and outputs the name and

attributes of the predicted Pokémon and the accuracy with which the prediction was made. Thus, it is similar to a Pokédex (an index of Pokémon data) in the sense that it stores data of the first six generations of Pokémon. An additional feature of our Pokémondium is that it is, as we write this report, the most comprehensive machine learning-based Pokémon directory available when it is deployed.

The combined number of images that were used to train the algorithms is more than ten thousand. We drew those images from various sources across the internet and ensured that all those images were open source and freely available, even if rendered by individual artists. Also, we do not claim ownership of those images. All we did was use the images to train our machine-learning models. This prevented any copyright infringement issues.

Related Work

Much work has been done in the field of object recognition using machine learning algorithms, specifically neural networks, and soft computing techniques. However, Pokemon Image Recognition seems to be a rather untouched area.

A user who goes by the handle Code AI gives a step-by-step description of how they used CNN for Pokémon image recognition [3]. They have used Vishal Subbiah's Pokémon Image Dataset from Kaggle [4]. Chirra et al, in their recent research paper have used transfer learning to classify Pokémon images [5]. They have created an index of Pokémon which they term Pokepedia. Their ResNet101 pre-trained model gave the best accuracy when measured against the rest of their models [5].

Chu *et al*., in 2023, presented their work 'Attribute Prediction in Pokémon Images using CNNs' at the second International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA) in China [6]. Their algorithm can determine the number of Pokémon belonging to a specific type (fire, water, grass, bug etc.) in their dataset consisting of images of 802 Pokémon. They utilized the images of the Pokémon obtained from a scorching game branded as Pokémon as their dataset [6]. They crafted a CNN algorithm harnessed it for type classification from the images in their dataset before they enhanced (preprocessed) their data (Pokémon images) and ran another iteration of CNN on it to improve their results [6]. However, their algorithm cannot determine any other statistic of a Pokémon, including the generation or even the name. This gives it serious limitations compared to the work that we propose.

There are a few different implementations of image detection that are currently available to be used by the public. The most common app that's used is Google Images. Google Images website (Google Lens) takes a user's image and rapidly comes up with the identification of the object that the user is attempting to identify. Many other search engines, such as Bing and Yahoo, have created similar algorithms.

What makes this research different is that it's directly related to identifying Pokémon. A search engine will identify the Pokémon, but all that the search engine will do is link someone who queries to websites that contain the information about the identified Pokémon. This research not only identifies the Pokémon but provides the statistics of the Pokémon. It's intended to be more user-friendly and that users don't have to rely on other websites for the information.

Another related work are online Pokémon directories (commonly referred to as Pokédexes). What makes Pokémondium different from a Pokédex is that our creation can give all information about a Pokémon based solely on the image of the Pokémon uploaded to it. It will detect the Pokémon in the image and provide information about it.

Proposed Work

The fact that no well-known machine learning based model exists for Pokémon image classification formed our motivation. In this regard our hypothesis was that machine learning

algorithms will help us classify and predict Pokémon based on image input. We also hypothesized that CNNs would have the best accuracy among all the models.

The main objectives of our project include the identification of the first six generations of Pokémon (containing 721 Pokémon) with image recognition and to detect from more than one different "style" of Pokémon images, including screenshots from video games, from artwork of individual artists and studios, from TV shows and from card portraits. Our models are trained only on 721 Pokémon because we needed a certain minimum number of images for each Pokémon so that our algorithms can train adequately, and there was a limited availability of images of Pokémon beyond the sixth generation (refer to the appendix).

Therefore, our project is not intended to be an exact or better alternative to a modern Pokédex, which contains data of many more Pokémon than ours. Rather, it is meant to be a tool that is used for people who may not recognize a Pokémon, or who might want to look up the stats of a Pokémon based on an image. That said, as we write the report, our Pokémon directory is the largest machine learning-based Pokémon index or Pokémon compendium. It combines the features of reverse image search (for example, using Google Lens) and a Pokédex.

A variety of Pokémon image datasets are available to evaluate the performance of different algorithms. Some orthodox Pokémon image datasets for assessing the performance of machine learning algorithms are Kvpratama's Pokémon Images Dataset containing 819 Pokémon images with a compressed total size of size 74 MB [7] and Rohan Asokan's The Complete Pokémon Images Dataset, which is a collection of 898 images of all the Pokémons taken from the Pokédex database with a compressed total size of 117 MB [8]. Both the datasets are accessible on Kaggle. However, since our aim was to create a more comprehensive catalogue of the first six generations of Pokémon, we created our own dataset for this study consisting of 10,073 images.

The information that's provided as the output once the algorithm is provided with the image includes the name of the Pokémon that it identifies as well as its stats, type, and generation. The only prompt that it takes is an image, so the web interface is simple and user-friendly. The stats include the ID index, the gender, the height, the weight, and other attributes of the Pokémon. If the user finds a Pokémon image on the internet and wishes to know more about it, the user will only have to click a picture of the image using his or her smartphone and upload it to our program using the UI. They may also download the picture and upload it directly to the appropriate input field in the UI we propose to create and have almost completed. The user may also input the path of the Pokémon in his or her device or storage as a string into the algorithm instead of directly uploading an image. We have trained our models and tested them using datasets they have not encountered before. Our algorithms are also able to show the accuracy of testing and training. The accuracy of the training and testing sets can be viewed from the back end of the website. This is how the analytics for each algorithm were computed.

Convolutional Neural Networks (CNNs)

We used Convolutional Neural Networks for image classification for the reasons listed as under:

1. Pattern identification: They were created specifically for image classification; they efficiently recognize patterns from the images.
2. Color recognition: They focus on efficacious color recognition and different color planes.
3. Critical feature isolation: They minutely track extremely critical features, like edges, on each image to produce an immensely precise prediction.

We have used the InceptionResNetV2 CNN algorithm, which is built in the Keras package within the TensorFlow library of Python.

InceptionResNetV2 or Inception-ResNet-V2 is a convolutional neural architecture which uses the Inception family of architectures integrating Inception with residual connections. It is described as a very deep architecture that combines elements from both Inception architecture, created by Google, and Residual Networks, shortened to ResNet. This architecture was developed with an aim to enhance algorithmic performance by leveraging residual connections in conjunction with the Inception modules [9].

The key components of InceptionResNetV2 include:

1. Inception modules, which consist of multiple parallel convolutional pathways of different kernel sizes (1x1, 3x3, 5x5, etc.) and pooling operations. These modules enable the model to learn the features of images at multiple scales.
2. Residual connections: Borrowing the best features of the ResNet architecture, InceptionResNetV2 includes residual connections which are also referred to as skip connections. These connections allow the flow of information through the network by adding shortcuts that bypass some layers, which in turn helps in relieving the vanishing gradient problem, facilitating the training of deep neural networks.

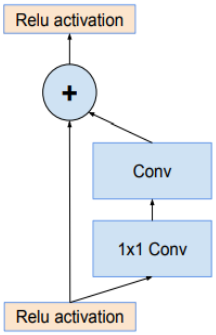


Fig 1. Simplified version of a ResNet connection [9, 10]

3. Stem layers: The beginning layers of the network, which are named stem layers, are involved in processing input images, and extracting basic features. These layers consist of convolutional, pooling, and normalization operations. They are common to all CNNs, including InceptionResNetV2.
4. Reduction blocks: Interspersed within the architecture are reduction blocks that reduce the spatial dimensions of the feature maps, helping to decrease computational burden while retaining important information. They use pooling.

Conv2D (Convolutional 2D) is a specific type of layer used for processing 2-dimensional spatial data, most prominently, images. Conv2D layers apply convolutional operations to input data to extract various features through learned filters or kernels. They are the stem layers in a CNN. InceptionResNetV2 incorporates various types of layers, including Conv2D layers.

Pretrained Models with Transfer Learning

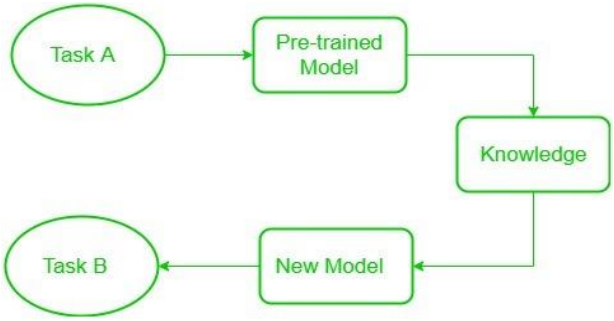


Fig 2 [11]: Transfer Learning block diagram

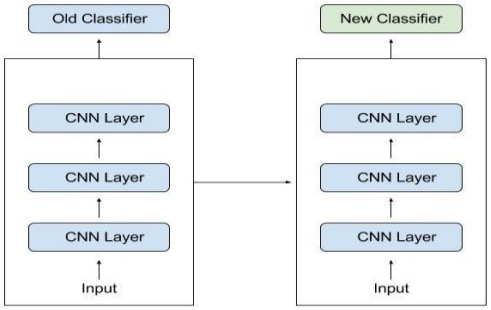


Fig 3 [12]: Transfer learning where the pretrained model as well as the new classifier are based on CNNs

Transfer Learning can be done in two ways:

1. Extending the pretrained model: This is used by our InceptionResNetV2 model. Extending a pretrained model involves adding new layers on the top of the existing pre-trained model, keeping the pre-trained layers frozen (not trainable). The added layers are used for task-specific classification. The pre-trained layers act as feature extractors, capturing generic patterns and high-level features from the original dataset they were trained on (e.g., ImageNet, on which InceptionResNetV2 is trained). The additional layers learn to interpret the high-level features extracted by the pre-trained layers and adapt them for the specific nuances of the new dataset. These new layers are trained from scratch.
2. Adjusting the pretrained model (Finetuning): This is also termed finetuning. It involves unfreezing and modification of the topmost layers of the pretrained model, also termed the old classifier. Some existing layers in the pre-trained model are unfrozen, allowing them to be further trained (finetuned) on the new dataset while also adding new layers on top. Finetuning is defined as updation of weights of both the newly added layers and selected layers from the pretrained model. This allows the network to adapt not only to the new task but also to adjust some of its previously learned representations to better suit the new data.

Using a pre-trained model for our work proved to be a very beneficial and efficient way for us to achieve our goal of image detection. A number of pretrained models for image detection exist, and the model that we opted for is called SentenceTransformers, a Python library that is like word2vec. This library was used by us because it combines text embeddings and image embeddings into a large multi-dimension coordinate plane. After this, it computes similarities using an approach like K-Nearest Neighbor algorithm. It can easily be finetuned and we finetuned it for our dataset using neural network layers.

We used this approach for one of our algorithms as it has quick training speeds and has a text-to-image embedding feature. This is an incredibly useful asset for our project, as we are attempting to identify images (Pokémon) with labels (their names). (Refer to the appendix for some more information on this.)

Support Vector Machines (SVMs)

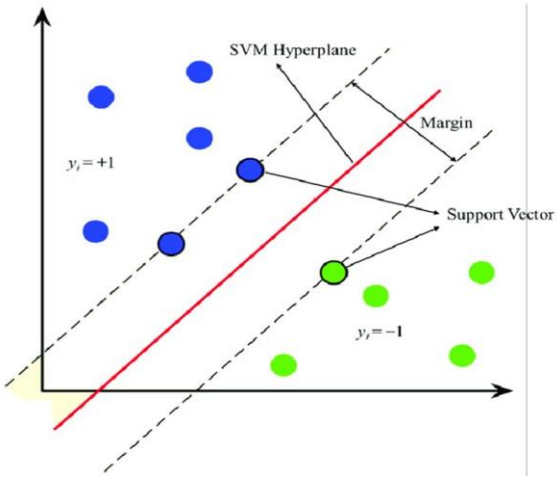


Fig 4 [13]: The diagrammatic representation of a linear SVM Classifier that separates the two classes.

SVMs aim to find the optimal hyperplane to separate different classes while maximizing the margin between the closest data points (support vectors).

K-Nearest Neighbors

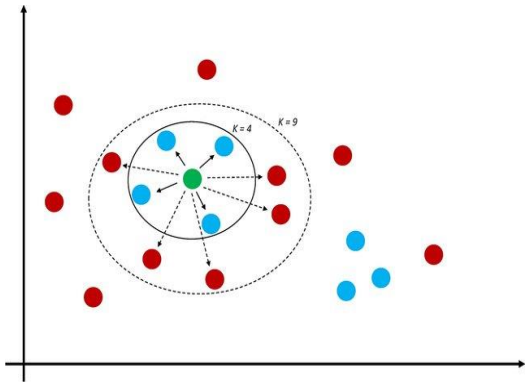


Fig 5 [14] A mathematical illustration of K-NN

An instance of the k-nearest neighbor machine is shown in the above figure [14].

1. The classes are catalogued based on their feature values.
2. The classification of a member belonging to one of the classes is determined by a majority vote in the member's neighborhood.
3. The number k is the number of neighbors that will be used to classify the member. They are the ones that most closely resemble the member to be classified.
4. The choice of k is crucial to classification. For example, if it is needed to classify the green member based on k = 3, the member would be placed alongside the blue members. If k = 9 is chosen, the member would be placed alongside the red members.

Experimental Evaluation

We have an active, working implementation of all the four algorithms. We have evaluated the performance of all the algorithms, including Convolutional Neural Networks, Transfer Learning using Pre-Trained Models, K-Nearest Neighbors, and Support Vector Machines.

Hardware

We have coded as well as run our programs on three different machines, each respectively belonging to the three of us. The statistics were calculated while the programs were executed on Eric Nieter's computer. Depicted below is a table describing the hardware that has been used to conduct this research:

	Eric	Ibrahim	Raj
GPU	NVIDIA GeForce RTX 4070 Mobile	8-Core Integrated GPU	Intel HD Graphics 620
CPU	i7-13620H (16Ghz)	Apple M1 Chip	Intel Core i7-7600U
Cores	6 Cores	8 Cores	Dual Core
RAM	64GB	8GB	8GB
Storage	3TB SSD	256GB SSD	256GB SSD

Table 1. Hardware specifications

Dataset

The dataset for this research consists of 10,073 images of 721 Pokémon. Originally upon training these models, there were about 5,400 images for the 721 Pokémon. There were many accuracy errors, especially with the CNN algorithm, where Pokémon were being inaccurately predicted.

The original collection of data consisted of using a library called "bing-image-downloader", which takes a string query and a set number of images to download. At first, the original parameters were the name of the Pokémon, then 10 images of each of them. This collected roughly 7,210 images, which were then sifted by hand to ensure that some image data, which was deemed poor for machine learning purposes, was removed. Some examples of such data are: images of Pokémon that contain props, clothing, watermarks/words, complex backgrounds, or images that were not of the Pokémon that was entered as the search query. An example of an incorrect search result was the Pokémon called a "Durant". When Durant was used as search query, the bing-image-downloader library displayed results corresponding to Kevin Durant, a famous basketball player.

Once the data was collected, all Pokémon had no less than five images to train from. There were roughly 5,500 images that were deemed fair for machine learning. However, with the issue mentioned earlier, this was underfitting our machine learning models. To mitigate this, the dataset was recreated. Using the similar data

collection method above, a total of 20 images were collected for each Pokémon. This resulted in a total of 14,420 images that were collected for the dataset. Another change that was made to the collection of data was using the query "X pokemon png", where the X was replaced with the Pokémon that was in a spreadsheet containing the names of all the Pokémon. The query contained "pokemon png" to have a higher chance of pulling images that had no background so that the models would not be skewed during training. This spreadsheet has been created by Alopez247 on Kaggle. This dataset is used by our Pokémondium for displaying all the statistics for each Pokémon once it has been identified.

Continuing with the dataset reconstruction, approximately 4,300 images were filtered out from the dataset leaving a total of 10,073 images across the 721 Pokémon. The reason why a .png file was included in the search query was to filter out any potential backgrounds that the library may have picked up. Therefore, it was important that the query contained "pokemon png" while collecting the training images of Pokémon. The background of an image would have possibly diminished the performance of the training algorithms, specifically with K-Nearest Neighbor, which is very susceptible to changes of the color patterns in an image. K-Nearest Neighbor algorithm will be explained in a future subsection.

Learning Environment

The learning environment for the models we described involves supervised learning at every step and for all algorithms. It involves the images that constitute the dataset, as well as the .csv file that contains the stats of the Pokémon and which the website gathers information from to produce the output. The agent in our algorithm interacts with the images and learns from the patterns and colors of each Pokémon.

Experiment Implementation explained alongside Learning Parameters

1. CNNs: We used InceptionResNetV2 as our base model. As an inbuilt model, it could not be unfrozen to reveal its kernel size, strides, padding etc. But it was extended using three additional layers:
   a. The `GlobalAveragePooling2D()` layer, which reduced each feature map to a single value by averaging all values in the feature map, providing a fixed-size output;
   b. A fully connected dense layer with 1024 nodes and ReLU Activation, which performed a linear transformation on the incoming image data followed by an element-wise ReLU activation. It introduced non-linearity (through ReLU) to the network and helped learn complex patterns from the image.
   c. An output dense layer with the same number of nodes or units as the number of classes (721) with softmax activation. This calculated class probabilities for our multi-class classification.

   Our CNN model plateaued at ten epochs. It used a batch size of thirty-two, and a training to testing ratio of 80 to 20 percent.

2. Transfer Learning with Pretrained Models **:** We have used the `run_pretrained(user_input)` function to initialize the pretrained CLIP-ViT-B-32 model using `SentenceTransformer` library. It was finetuned for our dataset.

   The images of all the Pokémon were read into the model, then the directory names of the Pokémon were read in as the labels of each Pokémon. As the model was pre-trained, it was able to detect the labels of the Pokémon automatically. To prove that this worked, the images of the Pokémon were shuffled, and the labels were input into the model, where they still predicted the Pokémon correctly.

Fortunately, since this algorithm was a "pre-trained model", as the name implies, it was the simplest to implement. It simply creates encodings for the images that we tested and got the labels that were already encoded before. It then computed the cosine similarity between them. The result of the highest cosine similarity value is what is output as the predicted label of the test image.

3. K-Nearest Neighbors: This is the algorithm which we implemented from scratch, that is, without using any machine learning libraries.
   a. Our algorithm performed best at k = 4, so the default value of the K-NN algorithm was preset to k = 4. This hyperparameter could be changed.
   b. Next, Numpy was used to read in all the images as Numpy arrays. These are special arrays that are normally not found in Python. They store images in the pixel format in RGB.
   c. After this operation was performed, the array was flattened, meaning that it was converted into a one-dimensional array. This was done for all the images, then their labels were assigned to each dataset and written to a text file.
   d. With the model trained, it was ready for testing. The test file was read in through a separate function, split into 25% testing, 75% training data, then for every image in the testing data, the Euclidean Distance was calculated between each image and everything in the training data.
   e. The first four Euclidean Distances were placed inside of an array. For the fifth distance (and onward), each distance was compared with every element in the array. When one distance was shorter than any element in the array, it would replace the one in the array and continue to the next image.
   f. Finally, once the tested image was complete, the counts of all the predicted labels were totaled, and the highest count was the prediction. Any ties were randomly selected labels.

4. Support Vector Machines: SVMs are not well-suited to image data, but work well for smaller datasets and "plottable" data. We should have discarded the algorithm in the initial phase itself and taken, in the place of it, some other algorithm like random forests, which would have begotten better results than SVM. Yet we kept SVM because we wanted to showcase its implementation as a learning opportunity for those who read this paper, so that they avoid using SVM for image recognition.

   We first attempted to use transfer learning here, with CNN as our base model or old classifier. Its layers were left frozen, and SVM was appended as the new classifier. It used svm package from sklearn library, and its kernel (hyperplane) was chosen as linear. But that model often gave an accuracy of zero percent.

   So we built a new SVM model. Within this we created an SVC classifier and trained the model. The model input the predicted image (the user input) and called the predict function where the input for it was the user's image. Then predictions were related to the user. It gave slightly better accuracy values but gave immensely high computation times.

Outcomes

To ensure there was no overfitting, the models were trained on test data and their respective accuracies were recorded. The models continued to be accurate, showing there was no overfitting. However, there was severe underfitting in support vector machine model and in k-nearest neighbor implementation, and the reason we found out was that both algorithms are not very well suited to image recognition.

Support Vector Machines and K-Nearest Neighbor algorithms had the least accuracy since the algorithms were only able to classify based on color. This caused them to be easily thrown off by images of Pokémon if they had the smallest changes to their environments, such as: complex and colorful backgrounds, shading or no shading, or if it were pixel art.

To test our newly formed algorithms for error we built a small test set containing 46 images of six Pokémon. We built another test subset over all images of the first generation of Pokémon we had, which came to about 1700 images. This was to test the algorithm's performance over medium to large datasets. Finally, each algorithm was run on the entire dataset of 10,073 images. The following are our experiment-wise outcomes:

CNNs:

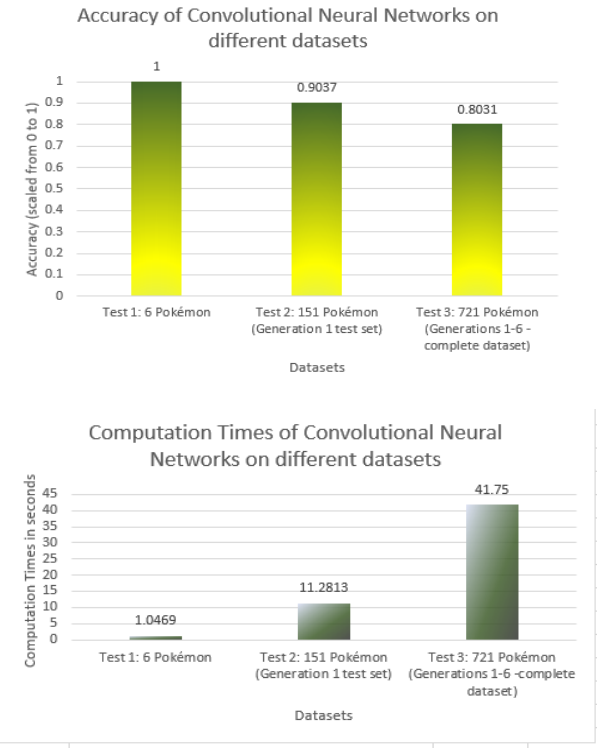


Fig 6 and 7: Accuracy and computation times, respectively, of CNNs over different datasets

Transfer learning with Pretrained Models:

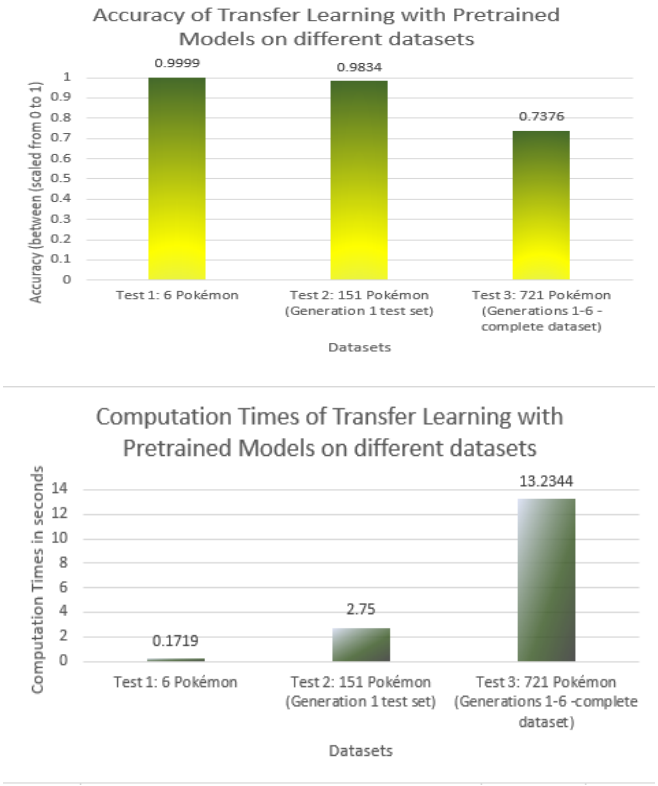
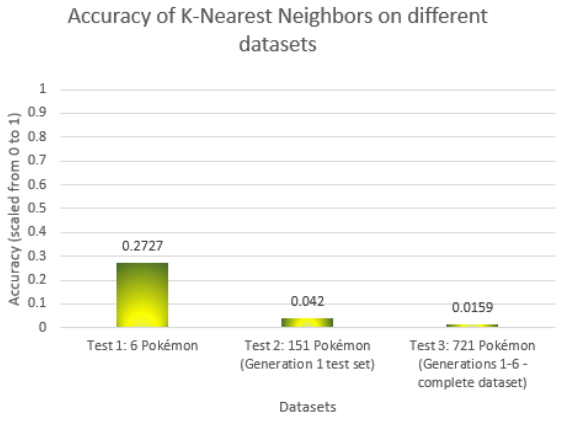


Fig 8 and 9: Accuracy and computation times of Transfer Learning over different datasets

K-Nearest Neighbors



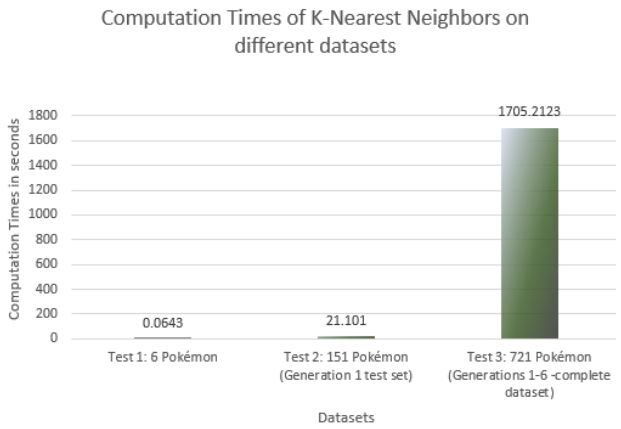
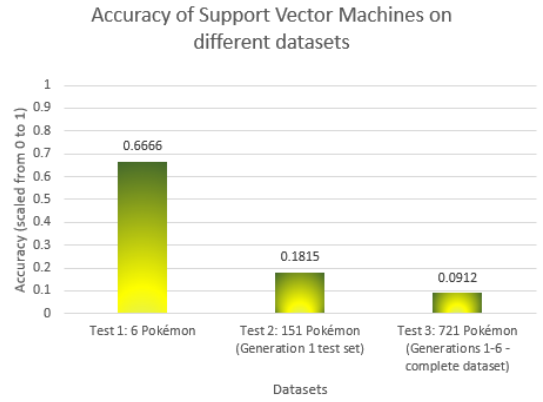


Fig 10 and 11: Accuracy and computation times of K-NN over different datasets

Support Vector Machines:



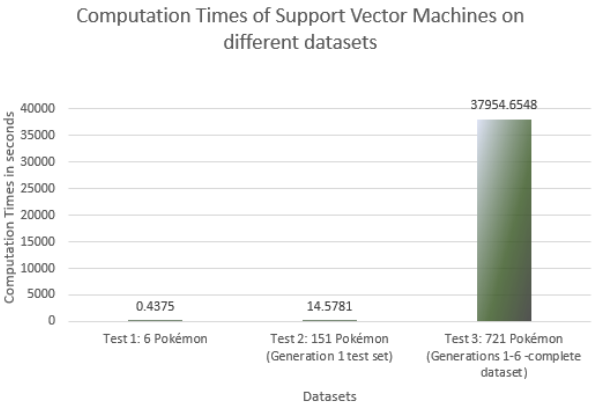
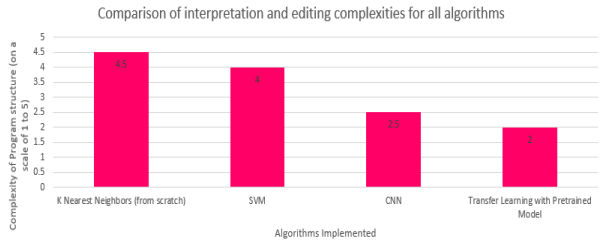


Fig 12 and 13: Accuracy and computation times of SVM over different datasets

Relative complexity of each of our algorithms:



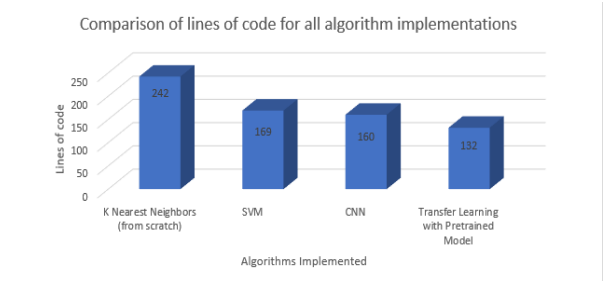
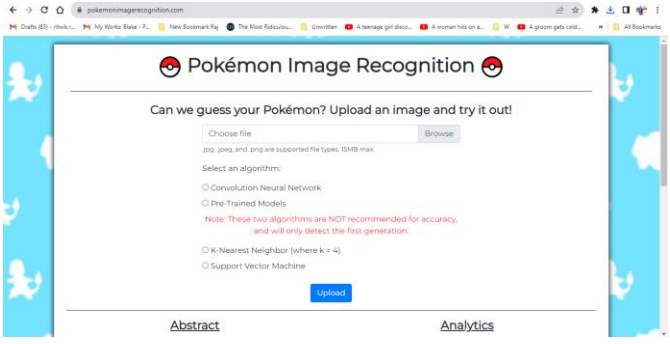


Fig 14 and 15: Relative complexity of our four algorithms

The high complexity of some of our algorithms and the injunction of implementing at least one of the models without using machine learning libraries presented roadblocks for us to implement metrics like precision, recall, F1 score, Brier score, and area under the Receiver Operating Characteristics curve for the models.

Our website:



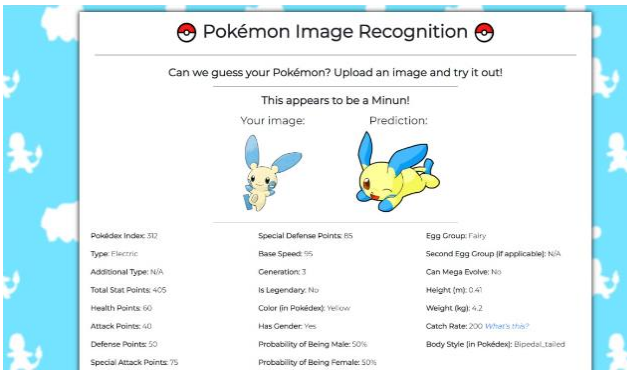


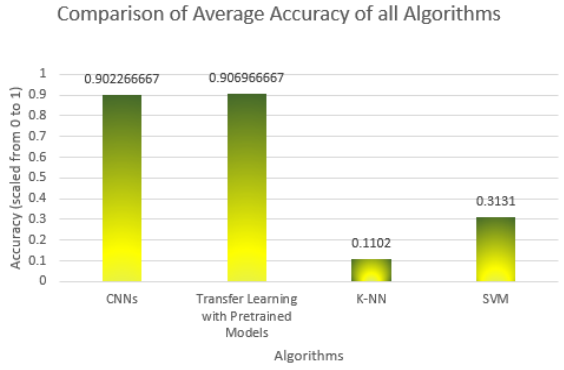
Fig 16 and 17: Screenshots from the website that hosts the Pokémondium

Our website, uses PHP, HTML, CSS, JavaScript, and the Bootstrap framework in its design and implementation. It showcases the abstract of our paper and the accuracy and computation times for our algorithms, so that the user is informed beforehand as to which algorithm should be chosen to get a more accurate and timely result for the user's input image.

It is significant to note that the website is only functional from a local machine, as it requires intense processing power which can only be provided with dedicated hosting. Dedicated hosts are very expensive and there is no funding to host this website from a web host. The website may be accessed here:

www.pokemonimagerecognition.com

Conclusions



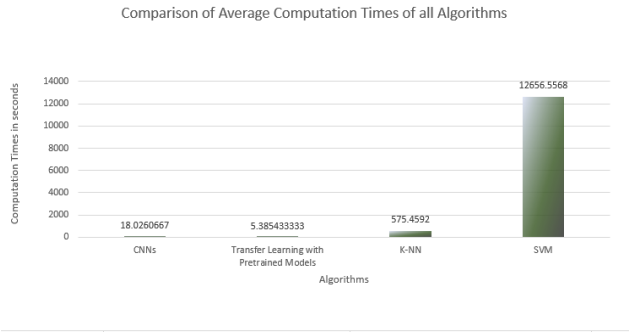


Fig 18 and 19: Accuracies and computation times for all models compared with each other

Average accuracy and computation time, respectively, for each algorithm

1. Convolutional Neural Networks (90.2% and ~18.02 seconds)
2. Transfer Learning with Pretrained Models (90.7% and ~5.4 seconds)
3. Support Vector Machines (31.31% and ~12656.56 seconds)
4. K-Nearest Neighbor (11.02% and ~575.49 seconds)

Transfer Learning with Pretrained Models algorithm performed the best on both metrics, computation time and accuracy. CNN's results were very similar to Transfer Learning on the metric of accuracy, but it took longer computation times. CNN's accuracy was 0.47% lesser than the accuracy of Transfer Learning with Pretrained Models. SVM and K-NN took immensely long to execute as well as provided much poorer performance compared to the Transfer Learning and CNN models because they are not suited for image recognition. They are not good at processing image data, especially large tensors. Since image data has large tensors, curse of dimensionality was a problem with K-NN. The performance of SVM shows that it is much less suited to large datasets with many classes compared to the other three algorithms.

Our hypothesis that CNNs would be the best model was disproven but with a negligible margin of error, and we attribute this purely to probability. Perhaps if the algorithms are trained

on the same dataset in a fresh environment, CNNs would perform better.

Future Work

1. Metrics like precision, recall, F1 score, Brier score and area under the receiver operating characteristics curve may be used to better compare the performance of the machine learning algorithms we have employed.
2. The algorithms can be trained on all known Pokémon, which currently number more than a thousand.
3. Once the best algorithm(s) is/are finalized, we may use them for more widely useful tasks like medical image recognition, for example, to diagnose cardiopathies from ECG images.

References

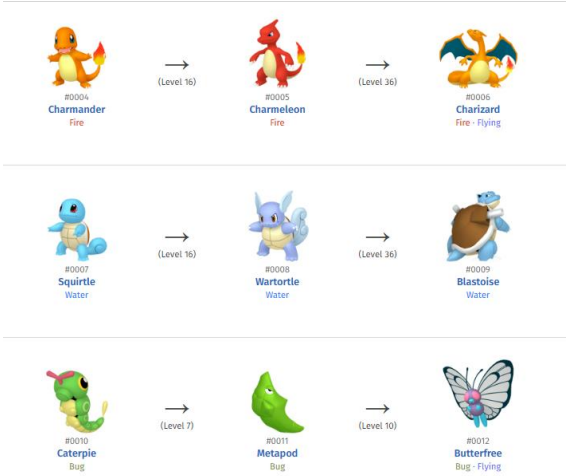
1. Pathak, A. R., Pandey, M., Rautaray, S. S. (2018). Application of deep learning for object detection. Procedia Computer Science, 132, 1706–1717. https://doi.org/10.1016/j.procs.2018.05.144
2. Srivastava, S., Divekar, A. V., Anilkumar, C., Naik, I., Kulkarni, V., & Pattabiraman, V. (2021). Comparative analysis of deep learning image detection algorithms. Journal of Big Data, 8(1). https://doi.org/10.1186/s40537-021-00434-w
3. Blogs, C. A. (2022). Classifying Pokémon Images with Machine Learning | CodeAI. Medium. https://medium.com/m2mtechconnect/classifying-pok%C3%A9mon-images-with-machine-learning-79b9bc07c080
4. Subbaih V., Pokemon Image Dataset. (2018). Kaggle. https://www.kaggle.com/datasets/vishalsubbiah/pokemon-images-and-types
5. Chirra, V. R. R., Syeda, A., Kolla, N., Ghanta, N., & Muvva, S. (2023). Pokepedia: Pokemon Image Classification using transfer Learning. Review of Computer Engineering Studies, 10(1), 14–19. https://doi.org/10.18280/rces.100103
6. J. J. Chu, Z. Shan and X. Sun, Attribute prediction of Pokémon images based on convolutional neural network. 2023 IEEE 2nd International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA), Changchun, China, 2023, pp. 129-134, doi: 10.1109/EEBDA56825.2023.10090499. https://ieeexplore.ieee.org/document/10090499
7. Kvpratama., Pokemon Images Dataset. (2020). Kaggle. https://www.kaggle.com/datasets/kvpratama/pokemon-images-dataset
8. Asokan R., The complete Pokemon Images data set. (2021). Kaggle. https://www.kaggle.com/datasets/arenagrenade/the-complete-pokemon-images-data-set
9. Szegedy, C, Ioffe, S., Vanhoucke, V., Alemi, A. (2016). Inception-V4, Inception-ResNet and the impact of residual connections on learning. arXiv.org. https://arxiv.org/abs/1602.07261v2
10. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015. https://doi.org/10.48550/arXiv.1512.03385
11. Chouhan V (2023). ML Introduction to Transfer Learning. https://www.geeksforgeeks.org/ml-introduction-to-transfer-learning/
12. Donges, N. (2019). What is transfer learning? Exploring the popular deep learning approach. Built In. https://builtin.com/data-science/transfer-learning
13. Yang, I., & Prayogo, H. (2022). Efficient Reliability analysis of structures using symbiotic organisms Search-Based Active Learning Support Vector Machine. Buildings, 12(4), 455. https://doi.org/10.3390/buildings12040455

14. Musolf, A. M., Holzinger, E. R., Malley, J. D., & Bailey-Wilson, J. E. (2021). What makes a good prediction? Feature importance and beginning to open the black box of machine learning in genetics. Human Genetics, 141(9), 1515–1528. https://doi.org/10.1007/s00439-021-02402-z

Appendix

An Introduction to Pokémon

A Pokémon is a fictional creature which is the basis for certain Japanese and Japan-inspired games and cartoons. It has stats, a type, a generation, and an evolution paradigm, as shown in the figure underneath:



A list of the categories under which a Pokémon can fall is as under:

Existing Pokédexes compared to our 'Pokédex', the Pokémondium, on the metric of number of Pokémon:

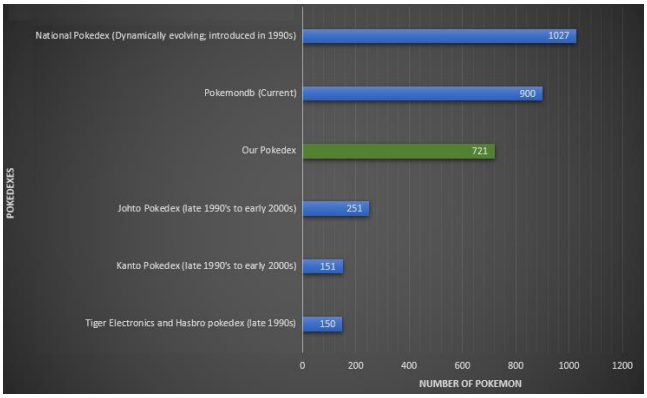
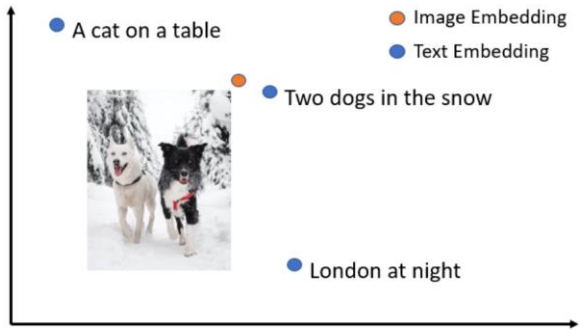


Figure showing how text and image embeddings are combined using `SentenceTransformers` (The following figure is a sample from the `SentenceTransformers` documentation):



The above figure shows how someone can use `SentenceTransformers` library to read-in an image from the user, then detect a label that is closely plotted to the image that was fed into it.