APS360

Classifying Pokémon Generations from Sprites

Team 10, Final Report

Word Count: 1775

Due on July 12, 2020

Professor S. Colic

Shashwat Panwar	Connor Lee	Prerak Chaudhari	Siddarth Narasimhan
1004168839	1004974414	1005114760	1005129102

Introduction

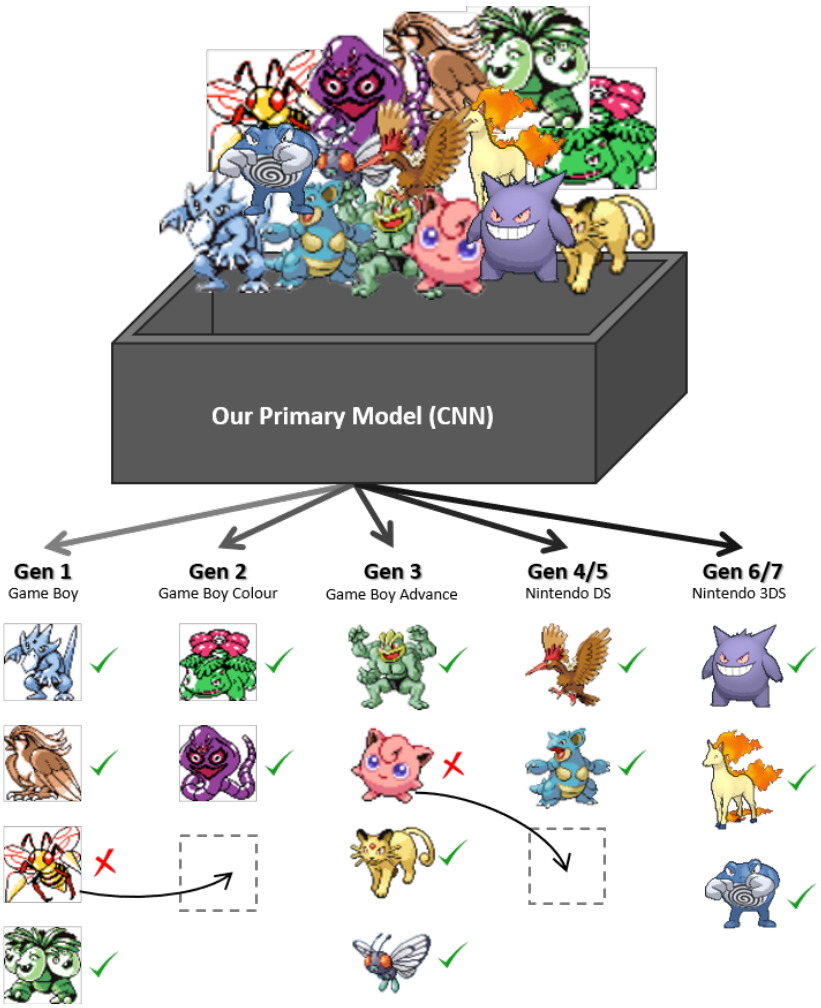

Figure 1: High-Level Visual Representation of our project and goal as of the progress report. Sprites of Pokémon across the years scraped/collected from databases are fed through our CNN which will be able to classify them on the basis of varying art-styles!

Project Goal	This project aims to be able to identify Pokémon sprites from various generations of games. The classes we have chosen are based on the hardware that each game was released for. The application for this project is to serve as an auto-classifier for Pokémon in live streams, so that categorizing the gaming hardware can be streamlined.
Motivation	There has been little (almost none, in fact) work done in classifying Pokémon in this way. As many of us have grown up with Pokémon in our childhoods, we find this to be an interesting topic to apply machine learning. We may learn something new about the nature of each generation's artwork and design. This problem could also be extended to other forms of game hardware identification in the future.
Machine Learning Appropriateness	Currently, there are no classification projects such as this related to machine learning. This project is a perfect application for machine learning due to the nature of image processing and complex classification problems. A machine learning algorithm will likely be able to learn features and trends that a human or other types of algorithms would not be able to distinguish.

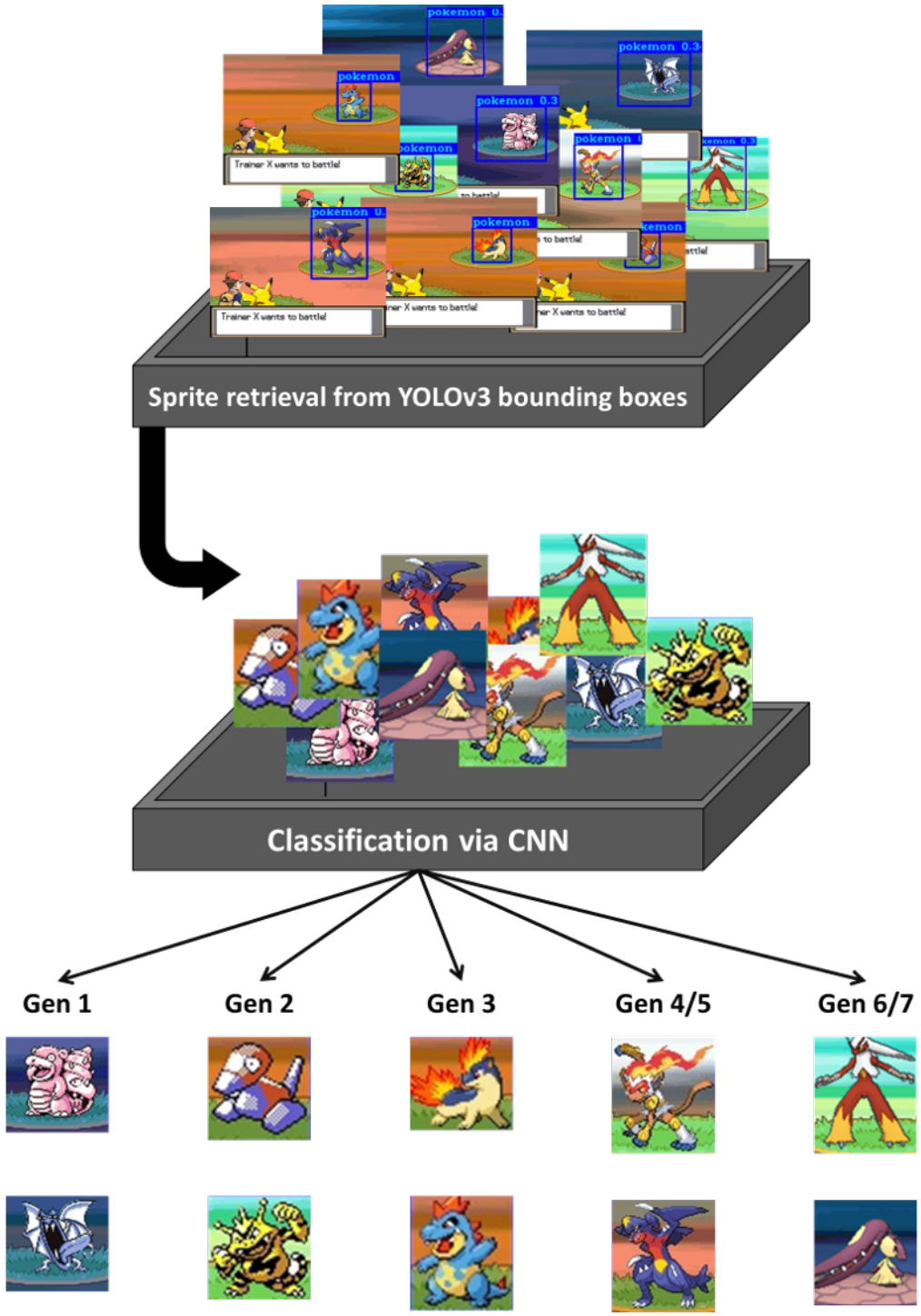

Figure 2: Addding onto figure 1; the Inclusion of sprite retrieval from YOLOv3 bounding boxes on in-game battle scenes for a more appropriate set of training and testing input images.

Setting up and usage of YOLOv3 [12] to improve the applicability of the project is expanded upon in "Evaluating the Model on New Data".

Background and Related Work

There is evidence of little work (in fact, almost none) in the classification of Pokemon generations. The furthest people have gone is classifying the Pokemon themselves, their types, identifying legendary Pokemon, determining battle viability, or creating new Pokemon with GANs [1][2][3]. Seeing existing Pokemon datasets allowed us to streamline our data collection process.

Data Processing

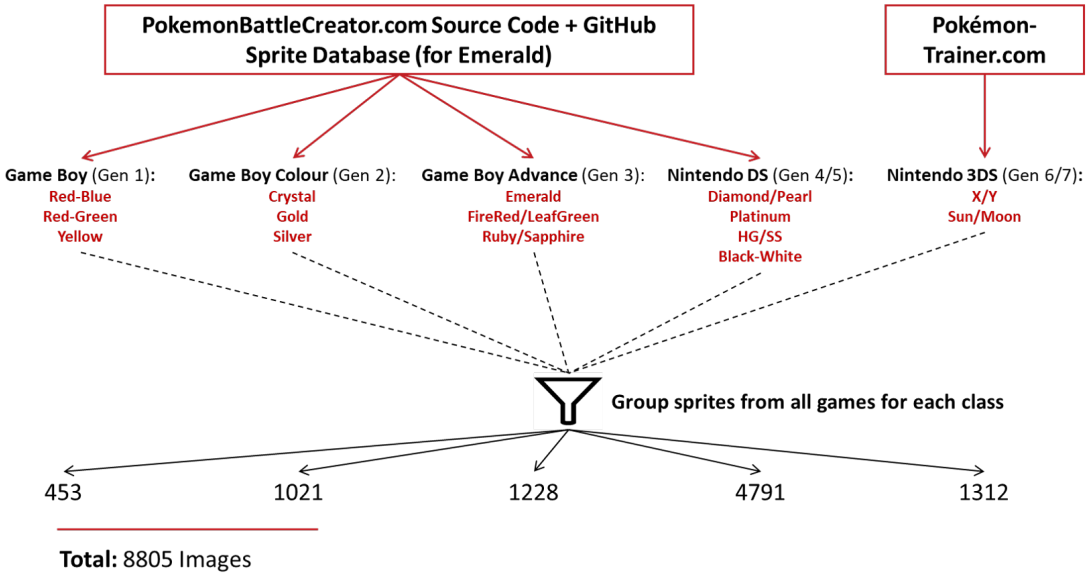

Figure 3: Pokemon game sprites are scrapped and sorted based on the game console they were released on. Duplicates are deleted using image hashing.

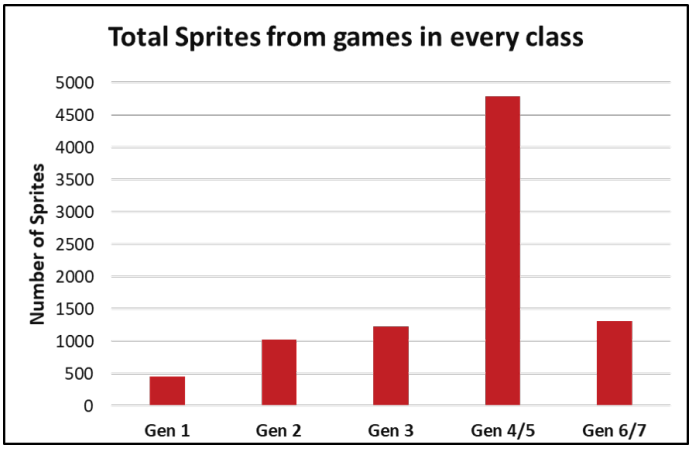

Figure 4: Distribution of raw sprites across our 5 classes.

With a clear imbalance among the classes, our first instinct was to augment the first 453 images from each class as that was the minimum population of a class (gen 1).

Figure 5: Sample Charizard (#006) and Pikachu (#025) sprites prior to any image augmentation.

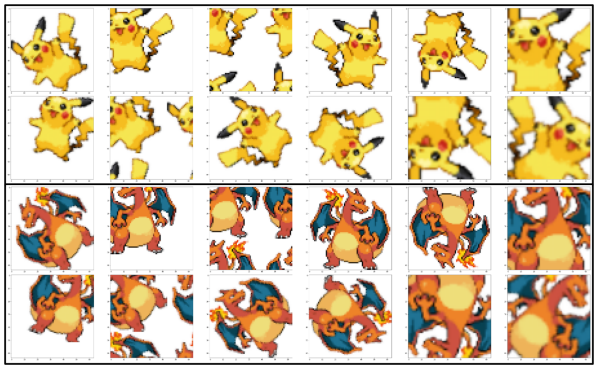

Figure 6: Examples of our 12 sets of augmentations on the sprites in Figure 4

After preliminary testing using our newly augmented dataset, we found that it did not help nearly as much as we had hoped so we scrapped this and veered toward using proportionally weighted class weights which inevitably gave us the exceptional results we presented.

For consistency among classes, we resized all sprites to 120 px by 120 px as that was the largest size in our scraped data. This made sure details in later gens were not lost due to image compression. Additionally, we added in-game battle background to the transparent layer of the resized sprites to ensure our model had as realistic of a training input as possible.

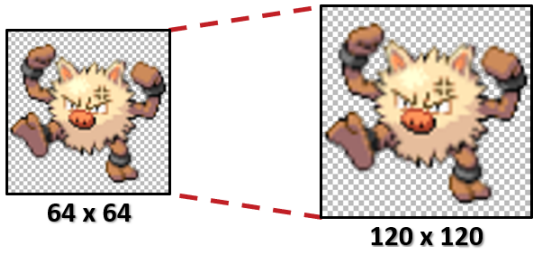

Figure 7: The sprites in our dataset have varying sizes with the newer generations nearly doubling the older ones in dimensions. For consistency, all images are re-scaled to 120x120. See the Primeape (#057) example above.

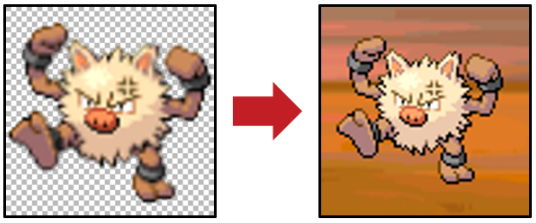

Figure 8: Sample Primeape (#057) sprite is pasted over a random in-game battle background.

Baseline Model

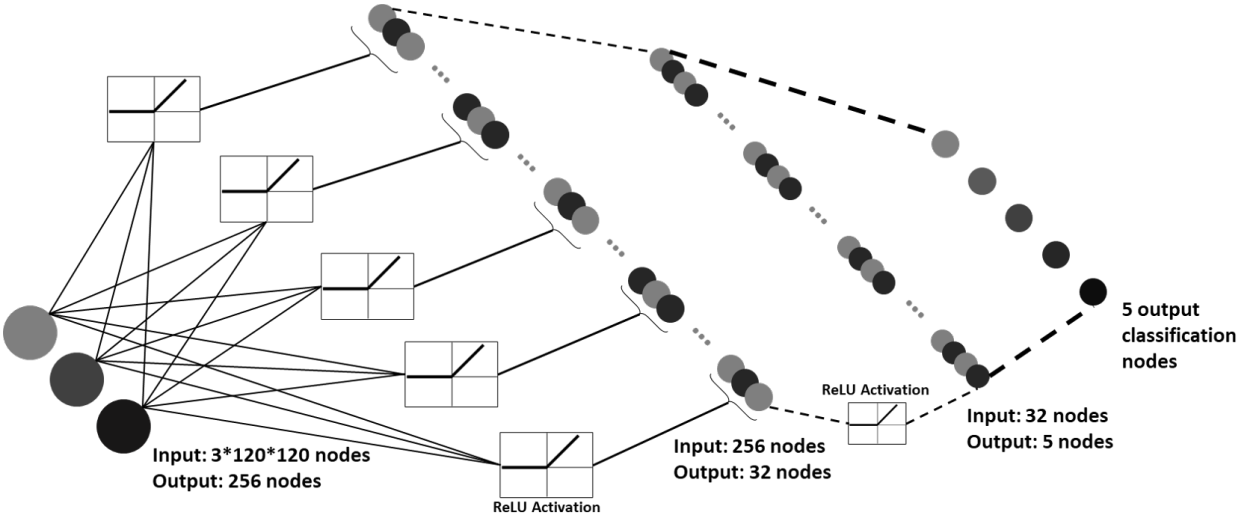

Figure 9: Visual Representation of Baseline ANN Architecture;
Total Parameters $= (3 \times 120 \times 120 \times 256 + 256) + (256 \times 32 + 32) + (32 \times 5 + 5) = 11067845$.
Layers: 3 fully-connected layers

Architecture

Figure 9 shows our final CNN structure.

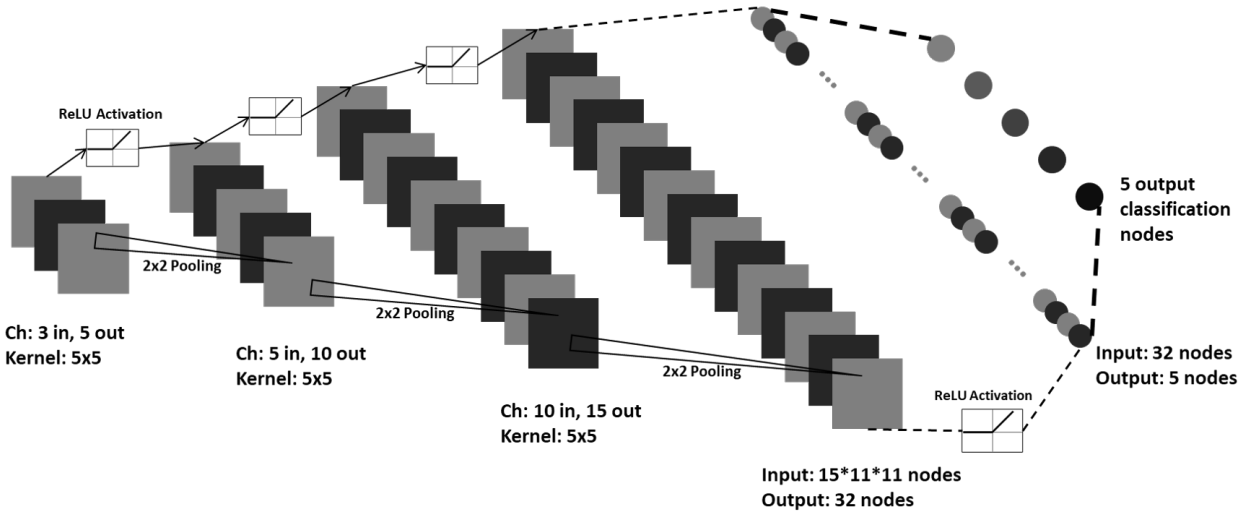

Figure 10: Visual Representation of CNN Architecture;
Total Parameters $= (3 \times 5 \times 5 \times 5 + 5) + (5 \times 10 \times 5 \times 5 + 10) + (10 \times 15 \times 5 \times 5 + 15) + (15 \times 11 \times 11 \times 32 + 32) + (32 \times 5 + 5) = 63682$.
Layers: 4 convolutional layers, 3 (2×2) max-pooling layers, 1 fully-connected layer

Figure 10 shows an alternate model we ran for reference using transfer learning from AlexNet, a formidable image classifier.

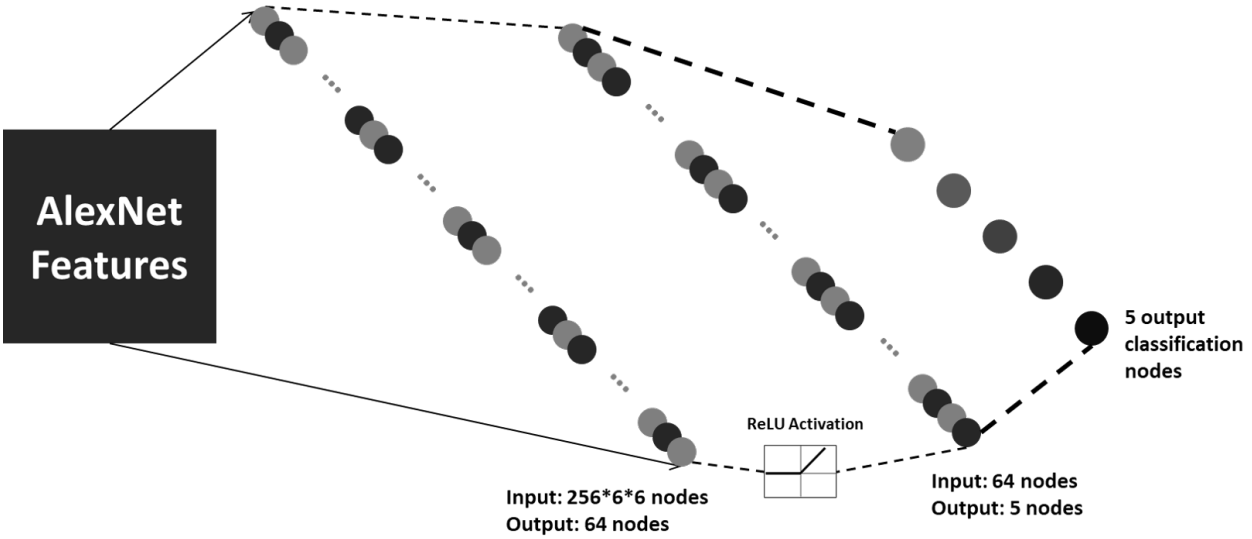

Figure 11: Visual Representation of Transfer Learning Architecture from AlexNet (alternate primary model);
Total Parameters $= (256 \times 6 \times 6 \times 64 + 64) + (64 \times 5 + 5) = 590213$.
Layers: AlexNet Feature Maps, 2 fully-connected layers

Evaluating the Model on New Data

In order to ensure that our testing data has not been seen by our classifier before, we constructed a new testing dataset cropping bounding boxes detected by YOLOv3. YOLOv3 was used for Pokémon localization since it does very well in identifying distinct objects around various backgrounds. Since bounding box detection is not hard-coded, slight variances in how YOLOv3 detects boxes (such as partial detection and excessive detection seen in Figure 12) ensures that our testing set is not seen by our classifier (which has sprites always cropped in the center).

Figure 12: YOLOv3 is trained on numerous images of in-game Pokemon battles. Detected bounding boxes are cropped out to generate a realistic testing set of in-game Pokemon sprites

Figure 13: YOLOv3 achieved a detection accuracy of 80.1% on new, unseen images.

Quantitative and Qualitative Results

Baseline Model

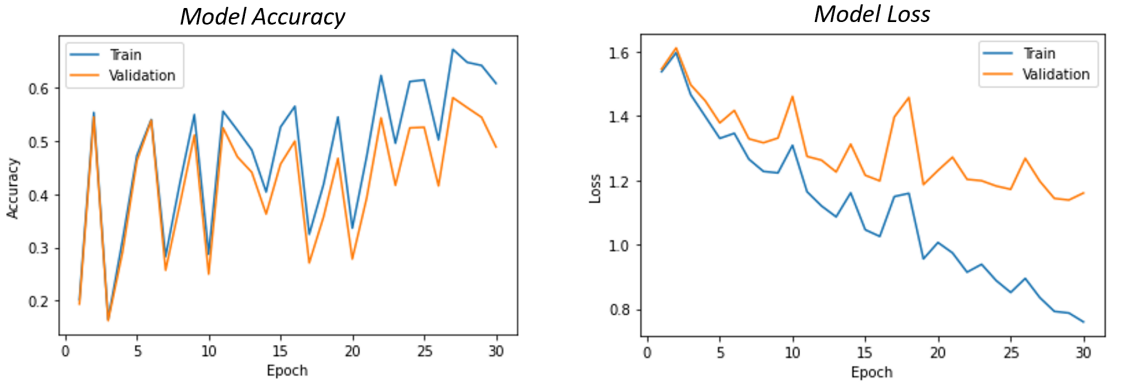

Figure 14: **Final Training Accuracy:** 0.608, **Final Validation Accuracy:** 0.4887, **Final Testing Accuracy: 0.27**, **Final Training Loss:** 0.76, **Final Validation Loss:** 1.16, **Final Testing Loss:** 2.47.

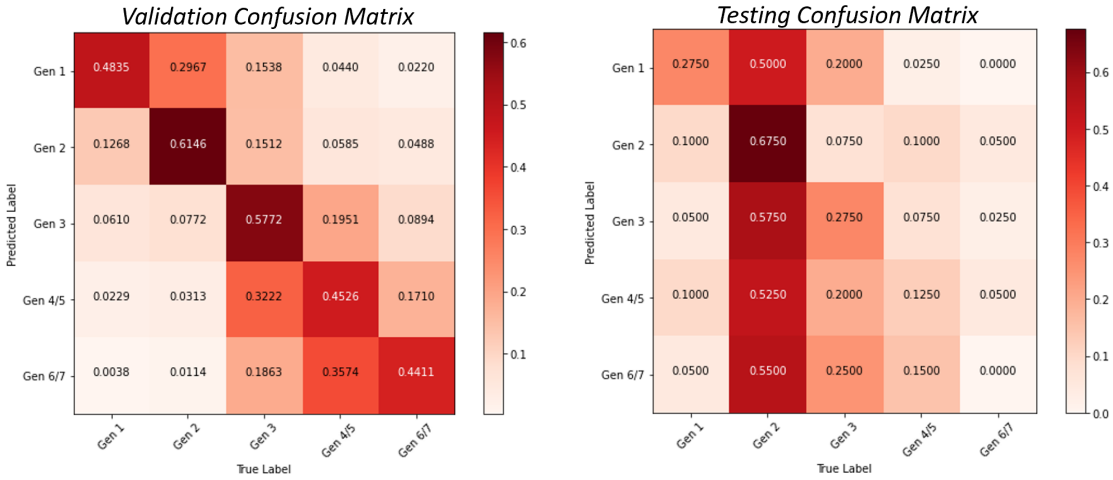

Figure 15: ANN Confusion Matrices. Note that the confusion matrices are normalized across classes for readability.

The naive ANN baseline model does rather poorly on the validation set and even worse on the testing set. This is expected as its hyper-parameters are not optimized and it does not utilise any spatial reasoning to make deductions about the sprites. The baseline model also has several fluctuations between epochs, which shows the difficulty in its classification ability. For this reason, we decided to not train it past 30 epochs to save effort, and since we did not believe it would ultimately not produce the best results.

Primary Model

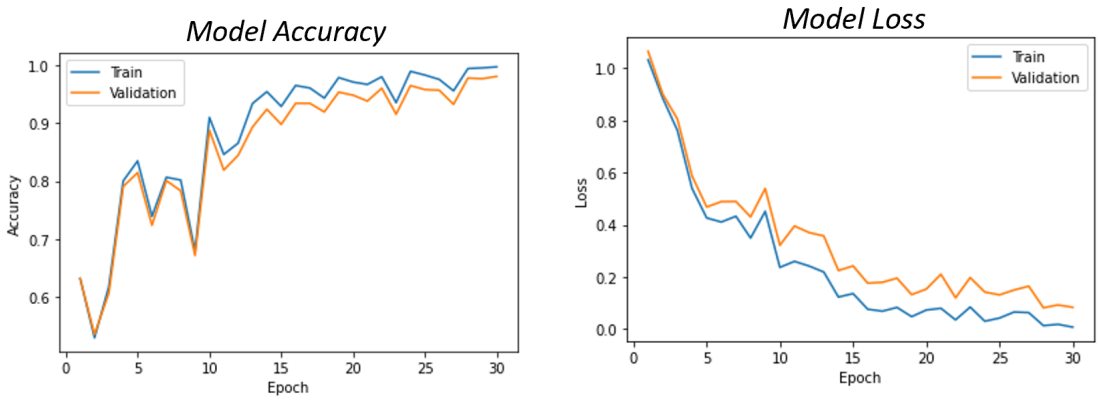

Figure 16: **Final Training Accuracy:** 0.9972, **Final Validation Accuracy:** 0.9807, **Final Testing Accuracy:** 0.74, **Final Training Loss:** 0.008, **Final Validation Loss:** 0.083, **Final Testing Loss:** 1.39.

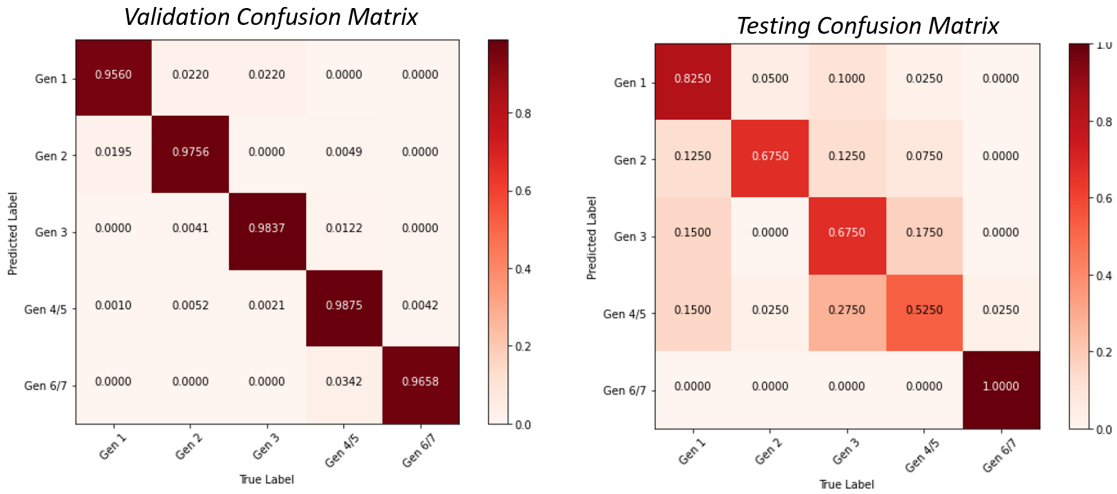

Figure 17: CNN Confusion Matrices. Note that the confusion matrices are normalized across classes for readability.

Upgrading to a CNN model allows our network to analyze spatial and geometric similarities in the pictures which greatly improved results. The CNN also produced some interesting feature maps, to help differentiate between classes (see Fig. 19)! The testing accuracy is the best out of all 3 models; 74% testing accuracy is therefore deemed acceptable. Notice how the CNN model does much better within 30 epochs than the ANN model.

Transfer Learning Model

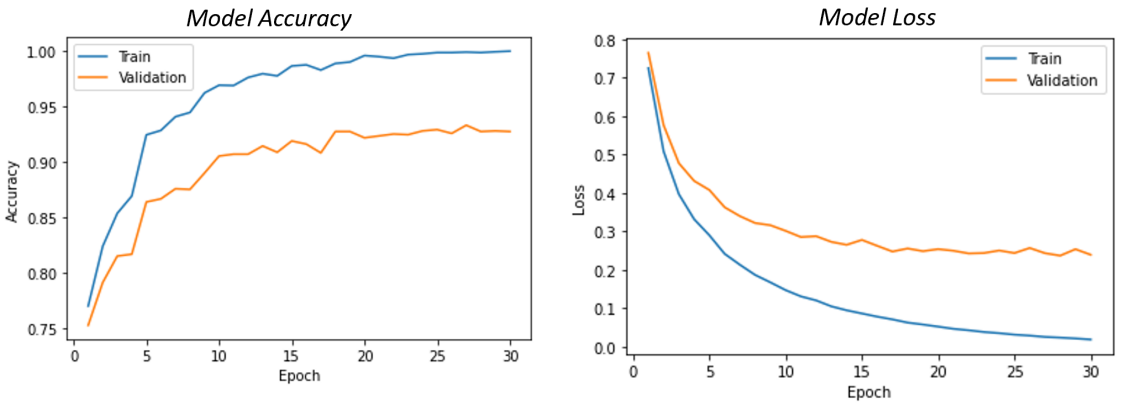

Figure 18: **Final Training Accuracy:** 0.9999, **Final Validation Accuracy:** 0.9274, **Final Testing Accuracy:** 0.685, **Final Training Loss:** 0.018, **Final Validation Loss:** 0.24, **Final Testing Loss:** 1.03.

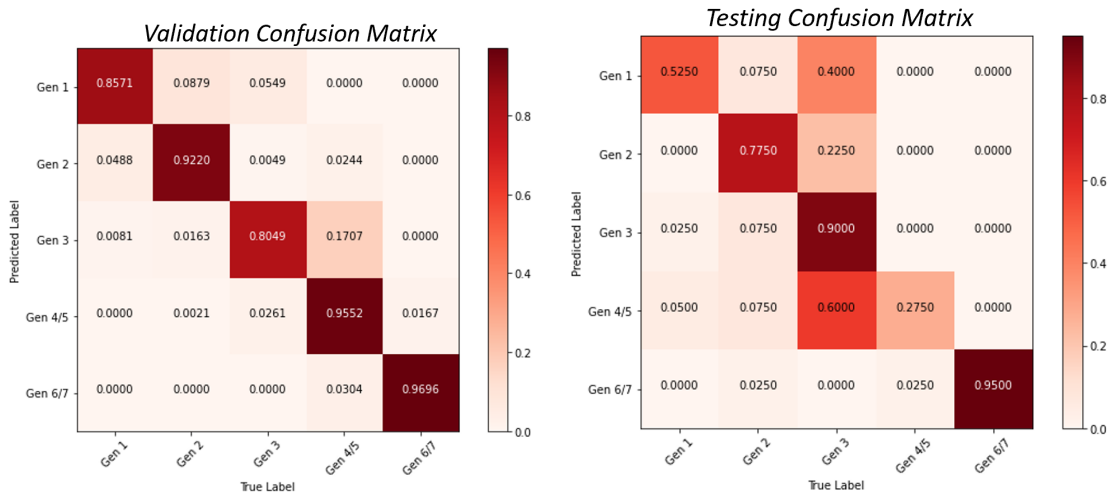

Figure 19: AlexNet Confusion Matrices. Note that the confusion matrices are normalized across classes for readability.

Using pre-trained AlexNet features to classify our images was a major improvement compared to the baseline model. There are also significantly less fluctuations in the curve when compared to the CNN. However, it did not perform as well on the testing set. In addition to dimensionality, another reason for this drastic discrepancy is over fitting onto the training data as indicated by the plateaued validation loss and accuracy curves.

In addition to the issue of memorizing the training dataset, another reason why AlexNet may not have performed as well as our CNN model is because AlexNet was trained with ImageNet, with the goal of trying to classify objects from one another. Our project aims to identify stylistic differences across classes, as opposed to standard object recognition, so AlexNet may not be suitable in handling this form of classification.

Ethical Considerations

The art style for each generation is limited by the technology that powered their respective game [6]:

- Generation 1: Nintendo Gameboy (2-bit colour palette, 160 × 144 pixels screen) [7]
- Generation 2: Nintendo Gameboy Colour (15-bit colour palette, 160 × 144 pixels screen) [8]
- Generation 3: Nintendo Gameboy Advance (15-bit colour palette, 240 × 160 pixels screen) [9]
- Generations 4-5: Nintendo DS (18-bit colour palette, 256 × 192 pixels screens) [10]
- Generations 6-7: Nintendo 3DS (top screen is 800 × 240, bottom screen is 320 × 240 pixels) [11]

We acknowledge that our CNN likely picked up on these technological discrepancies and used it to distinguish between sprite classes. Our model was only trained on official artwork released by Nintendo to maintain authenticity and consistency in the art design.

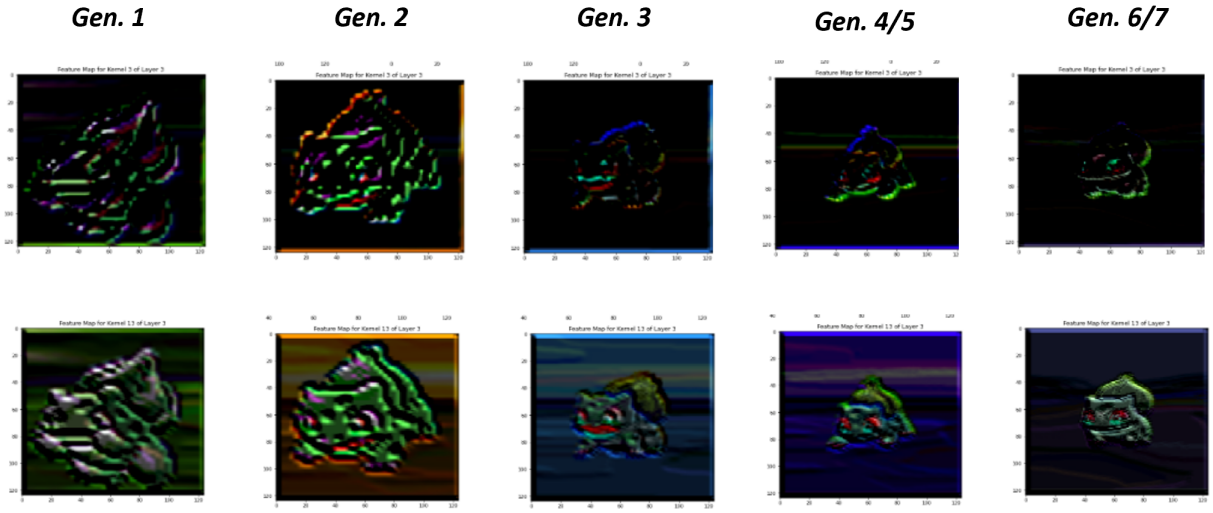

Figure 20: Feature maps of our CNN classifier across generations. Notice how the classifier picks up on more edges in the earlier generations as opposed to the later ones. Also notice how the Gen 3 and Gen 4/5 feature maps are similar, which may be the reason for some mis-classifications.

While the model was limited to official sprites, we must consider the implications of classified fan art. If it is able to correctly classify fan art drawn to mimic a certain generation, the robustness of our model's ability to deduce the stylistic differences between console generations, irrespective of artists, is validated. On the other hand, incorrect classifications indicates there are artistic differences between the fan art and official artwork, or worse, that the model was over fitted onto the training set.

One ethical consideration is the use of this classification technology in replacing human judges in fan art competitions. We have disclosed which classes the model excels at or struggles to classify. Say a participant draws a newer Pokemon from generation 7 in generation 3's art style. Our model may incorrectly classify it as either generation 4 or 5, resulting in participants being disqualified on false negatives.

Another example is how streaming sites such as Twitch may use the model to automatically classify the games Pokemon streamers are playing. If Nintendo releases a new game or is partnering with streamers to play an older game, the model may incorrectly classify some streams, reducing viewership and making it harder for users to find the content they are looking for.

Discussion

Looking at the confusion matrices, all of our models consistently misclassify Gen 1 with Gen 2, and Gen 3 with Gen 4/5 (and vice-versa), which is an expected result due to the stylistic similarities in the sprites. Gen 6/7 (the Nintendo 3DS) is classified almost perfectly in the CNN and the AlexNet models, likely due to higher-quality and vastly-different sprites in the Nintendo 3DS.

One useful takeaway from the project is that data augmentation and increasing the data set size is not always the right decision when using an imbalanced data set. Class weighting is a simpler alternative to implement that aims to change the weighting each class has on the loss function. In our particular case, there are more sprites in later Pokemon generations. These classes would have more variance in their art styles. Augmenting the earlier classes would only introduce bias into the model towards those specific images, not help it correctly infer its art styles. Furthermore, transformations such as rotations and translations do not make sense for our project as our validation and testing set has centred, upright sprites. On the other hand, class weights provide proportional emphasis to ensure that classes with less sprites have a greater effect on model tuning, eliminating biases towards any one class.

Project Difficulty / Quality

For the Progress Report, we built a model that would classify sprites with clear backgrounds for the first 151 Pokemon in each class; training and validation numbers/graphs were presented at the time. To increase model complexity and real-life applicability, we added in-game battle backgrounds to the aforementioned sprites to simulate how these Pokemon would look in game. Each sprite randomly had one of 16 different backgrounds placed behind it.

To generate a testing set, a YOLOv3 model was trained to generate bounding boxes around opponent Pokemon. This allowed us to crop out sprites from in-game battles. It also made our project more complex since we had to perform object localization and then classification, instead of just classification as we had before. Training YOLOv3 from scratch also meant that we had to build a new training dataset, which further made the project challenging.

We also tried implementing data augmentation onto our training set to balance the sprite discrepancy between classes. Some example transformations include rotation, translation and warping. This did not produce good results and we opted to use class weights instead. In fact, data augmentation and resulted in a testing accuracy of 50%.

By doing this, it made our project much more difficult than it seems. It took several attempts to actually create a successful model since we explored data augmentation techniques as well as balancing the classes for the majority of the time, which surprisingly did not end up working. It took further research to come up with the idea of class weighting, which ended up being our best choice, and producing the highest testing accuracy (74%).

Code

- Prerak
  - Pokemon sprite classifier: **Click me**
  - Data augmentation: **Click me**
  - Image hashing (see [4]): **Click me**
- Shashwat
  - Pokemon-trainer.com web-scraper: **Click me**
  - Skimage augmentation script and coloured confusion matrix generator: **Click me**
- Siddarth
  - Filename: **Click me**

Individual Contribution

Shashwat	**Prerak**	**Connor**	**Siddarth**
• Created script to scrape sprites from pokemontrainer.com and sort them by generation • Created script, using skimage, for 12 variations of physical augmentations (unused for final model) • Created visual representations of the neurel net models, coloured and normaliize confusion matrices, and other accompying diagrams/figures • Laid out final presentation and organized its structure, with constant revisions based on feedback of the team • Converted screen recorded voice-overs into a final video where tiny details needed to be sped up and/or pitch-corrected to meet the 8 minute limit • Assisted in formatting all LaTeX documents **Overall Contribution Score: 25%**	• Manually sorted sprites from PokemonBattleCreator.com source code and GitHub database into classes • Used image hashing to delete duplicate images (see [4]) • Wrote script to mask alpha channels and paste background images onto sprites • Increased size of training set by implementing Shashwat's skimage augmentation functions • Wrote code for complementary tasks such as graphs, confusion matrices and feature maps • Coded classification neural networks for ANN, AlexNet and CNN and optimized hyper-parameters • Researched and implemented class weights (see [5]) and other transfer learning models such as ResNet50 before settling on AlexNet **Overall Contribution Score: 25%**	• Initial project topic research and planning • Sourcing datasets and image databases to process and use in training • Experimented with early baseline ANN • Created an outline for Final Presentation, planned tasks for presentation • Performed project results analysis (qualitative and quantitative) for each model for presentation • Contributed to project proposal, progress report, and final report • Enhanced team morale and confidence :) **Overall Contribution Score: 25%**	• Divided up workload among group - created the Project Plan • Data scrape from all websites for the Project Proposal - made all cleaned data accessible to the group • Trained the YOLOv3 model and did background research on how to use it • Built the dataset to train YOLOv3, and labeled all the images • Format tables, images in LaTeX documents (90%) • Create outlines for Project Proposal and Progress Report • Wrote augmentation script using PyTorch • Experimented with hyper-parameter settings for our CNN (did not complete 10%) • Tested CNN performance after balancing dataset (did not complete - 15%) **Overall Contribution Score: 25%**

Table 1: Individual Contribution

References

[1] J. Feldman, "What Makes a Pokémon Legendary?," DataCamp, 2020. [Online].
Available: https://www.datacamp.com/projects/712.
[Accessed: 05-Jun-2020].

[2] Y. alouini, "Pokemons Machine Learning 101," Kaggle, 10-Sep-2018. [Online].
Available: https://www.kaggle.com/yassinealouini/pokemons-machine-learning-101.
[Accessed: 05-Jun-2020].

[3] A. Dennanni, "Creating Pokémon with Deep Learning," Medium, 29-Oct-2018. [Online].
Available: https://medium.com/neuronio/creating-pokemon-with-artificial-intelligence-d080fa89835b.
[Accessed: 05-Jun-2020].

[4] A. Rosebrock, "Detect and remove duplicate images from a dataset for deep learning," PyImageSearch, 23-Apr-2020. [Online].
Available: https://www.pyimagesearch.com/2020/04/20/detect-and-remove-duplicate-images-from-a-dataset-for-deep-learning/.
[Accessed: 09-Aug-2020].

[5] Scikit-learn.org, "sklearn.utils.class_weight.compute_class_weight." [Online].
Available: https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html.
[Accessed: 09-Aug-2020].

[6] "Generation," Bulbapedia, the community-driven Pokémon encyclopedia, 29-May-2020. [Online].
Available: https://bulbapedia.bulbagarden.net/wiki/Generation. [Accessed: 10-Jun-2020].

[7] "Game Boy," Wikipedia, 13-Jun-2020. [Online].
Available: https://en.wikipedia.org/wiki/Game_Boy#Technical_specifications. [Accessed: 13-Jun-2020].

[8] "Game Boy Colour," Wikipedia, 13-Jun-2020. [Online].
Available: https://en.wikipedia.org/wiki/Game_Boy_Color#Specifications. [Accessed: 13-Jun-2020].

[9] "Game Boy Advance," Wikipedia, 06-Jun-2020. [Online].
Available: https://en.wikipedia.org/wiki/Game_Boy_Advance. [Accessed: 13-Jun-2020].

[10] "Nintendo DS," Wikipedia, 13-Jun-2020. [Online].
Available: https://en.wikipedia.org/wiki/Nintendo_DS. [Accessed: 13-Jun-2020].

[11] "Nintendo 3DS," Wikipedia, 10-Jun-2020. [Online].
Available: https://en.wikipedia.org/wiki/Nintendo_3DS. [Accessed: 13-Jun-2020].

[12] "You Only Look Once: Unified, Real-Time Object Detection" Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, 2018. [Online].
Available: https://arxiv.org/pdf/1506.02640.pdf.