CSC 2611 – 131

Final Project Report

Michael Wood, Alec Weinbender, Pranaav Paladugu

12/13/2024

Pokémon Image Classifier

Abstract

Our group decided to create a model that would predict from an image a Pokémon from the first generation. In the first generation of Pokémon there are 151 unique Pokémon, meaning our model should have 151 classes. We trained our model on images taken from several Kaggle datasets We combined a few different datasets to better distribute images among the 151 classes. Because of the similarity between two classes, we removed them both from the dataset, resulting in a final 149 classes. To create training and testing splits we stratified the data into new training and testing directories with an 80-20 split of the images for their respective classes. After creating our baseline model, we achieved a validation accuracy of 75%. At this point we started to create different models with Michael creating the optimized model, Alec creating a classical learning model, and Pranaav creating an MLP model. With the optimized model, we achieved an accuracy of 88.83%. We achieved this by tuning the hyperparameters and augmenting some data. We also hand removed many images from the dataset that were not accurate to the classes. The classical learning model that we used was K-Means. With K-Means, we achieved an accuracy of about 4%. With the MLP model, we achieved an accuracy of about 0.6%.

Introduction

The goal of this project is to apply the machine learning techniques and AI tools covered in this class to a real-world project. To meet the requirements, we needed to write a project proposal and then develop a baseline model. After creating our baseline model, we needed to optimize it using various techniques covered in class. After optimizing our model, we needed to compare the results between other classical and state-of-the-art machine learning algorithms.

Our group wanted to create a computer vision model so we decided to create a model that would predict the kind of Pokémon from an image. To ensure that the datasets would not get too large, we decided to only train our models on Pokémon from the first generation. This meant that there would be 151 classes. Our group searched on Kaggle for datasets containing images of Pokémon from the first generation and found many. The datasets provided images of many different sizes. Because the images also have full RGB scale, our model is trained on images with the shape

of (224, 224, 3). The folders for each class generally did not have the same number of images in each. They typically had about 300 images for each class. Many of the images come from different sources: in-game, television shows, trading cards.

To easily train and test our models, we needed to stratify the data that we took from Kaggle into two different folders for training and testing. We did this by iterating over each image in the dataset and placing 80% of them in the training set, while putting 20% of them in the validation set. During this process, we also used preprocessing on our images to fit them to the shape of 224x224x3. We had many faulty image types in the dataset like gifs or corrupted files. This proved to be a problem because our model kept crashing when being trained. To fix this we ran through each file and deleted the faulty ones that the model could not train on.

Baseline Model

Our baseline model uses transfer learning, taking from the VGG16 model with the ImageNet weights. Transfer learning allows our model to train much more quickly as it essentially transfers learned features like edges to our model. Our baseline model was trained on 150 classes with about 100 images in each class. A pooling layer was added onto the model to reduce the dimensions of the input data. A final dense layer with 150 classes for the 150 Pokémon was added onto the model. This allows the model to have 150 unique options to predict.

We didn't tune any of our hyperparameters and ran our baseline model with 5 epochs and a batch size of 32. We were able to achieve a validation accuracy of around 75%. It took about 30 minutes to run this. Most of the time was spent sorting the files into training and testing folders during runtime. At this point in the project, we had not created concrete folders that would hold the training and testing images before runtime. We were running it at around 5 minutes per epoch with 1 GPU.

Approaches to Model Improvement

After training our baseline model we added two more datasets from Kaggle into the dataset that we were using. This resulted in about 54,000 images. However, a lot of the image files were faulty or wrong file types like gifs. We wrote multiple python scripts to remove faulty files or gifs. After removing those files and removing other files that didn't represent the classes well, we had about 52,000 images. Male and female Nidoran were not included in our datasets because the authors of our datasets removed them. This was likely because the two classes were very similar, with almost the same colors and same shape. After configuring our dataset, we reprocessed and stratified our data into training and testing splits. The split was 80% training and 20% testing.

After having our data correctly set up, we decided to tune the hyperparameters that we used when training our baseline model. We knew that we needed to increase the number of epochs used, because we hadn't seen overfitting in our baseline model. We also halved the batch size that we used in the baseline model to 16. This means that the model's parameters are updated twice as quickly as they were in the baseline model. We trained the model a few times using the updated dataset and changed hyperparameters, and we found that after about 10 epochs the model would

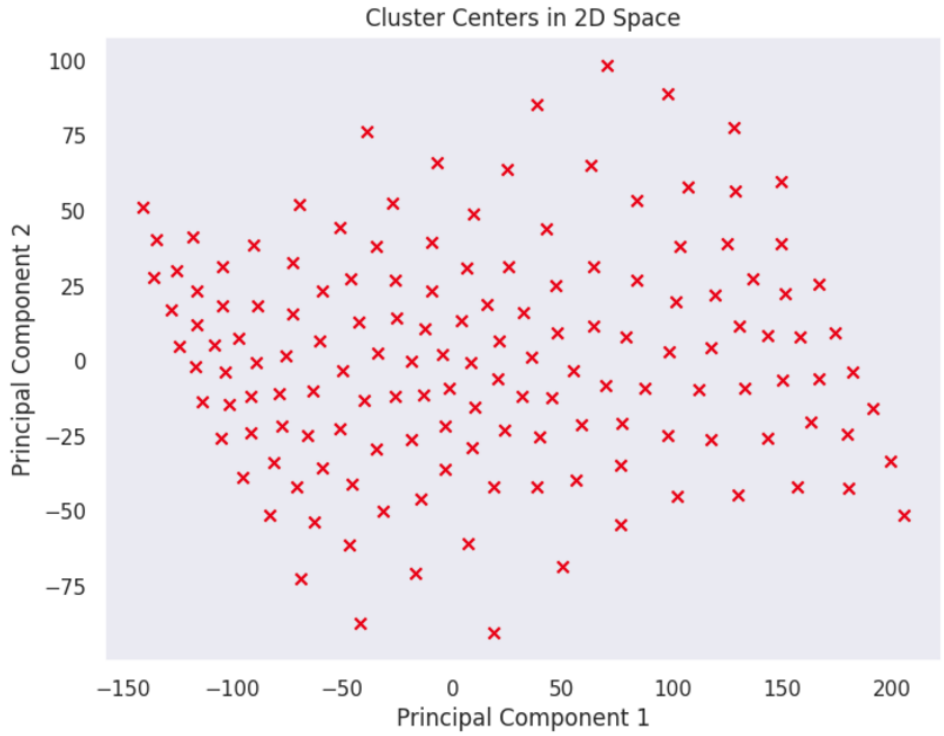
become overfit. Our final hyperparameters for our optimized model was 10 epochs and a batch size of 16. We also performed data augmentation where images were rotated and shifted to prevent overfitting. Finally, we allowed for fine tuning in our model where we unfroze the entire base model to continue training. We trained our final optimized model with 8 V100 GPUs resulting in an overall runtime of about 44 minutes. The average time per epoch was about 120 seconds.

To further analyze our results, we set up a Jupyter Notebook space that would allow us to see how the model would perform on each specific Pokémon. We found that the model best predicted purple Pokémon (accuracies above 90%) while it would do the worst on yellow and orange Pokémon (accuracies below 10%).

Comparison Models

K-Means Model

We set up a Jupyter Notebook to easily create the K-Means model. At this point we had already stratified the data, but the images needed to be preprocessed for our new model. After reshaping and normalizing our images, we used principal component analysis on the images. Using PCA on our images allowed us to pick out the most important pieces of information given from our images. It also heavily saves time used to train the model, which is especially important when the model is only being trained on a CPU. We set the PCA to 2. This greatly reduces the dimensionality of the data taken from the images from 150,528 to 2. However, setting the PCA to 2 allows us to easily visualize the clusters used when predicting the Pokémon. After all this processing of the images and the data we started to train the model. We fit the K-Means model to 149 clusters for the 149 classes we were trying to predict. We then trained the model on the training and testing sets.



To predict how well our K-Means model worked, we assigned ground truth labels to our clusters. This works by having each class determine which cluster the majority of their datapoints were closest to. This way each cluster had a specific Pokémon that it was assigned to. We calculated the accuracy of our model by running through our testing dataset and seeing if the new datapoints matched with the correct cluster.

This model achieved an accuracy of about 4%. Although this is very low, it is much better than a random guess from the model. It takes about 30 minutes to train this model if the data has not been preprocessed. We believe that this model could have been improved if we had not lowered the number of components to 2 through PCA. It is common to lower the number of components to the range of about 100-500. We believe that our CNN model performed so much better than our K-Means model because CNNs are much more adapted to image classification models than classical machine learning models are.

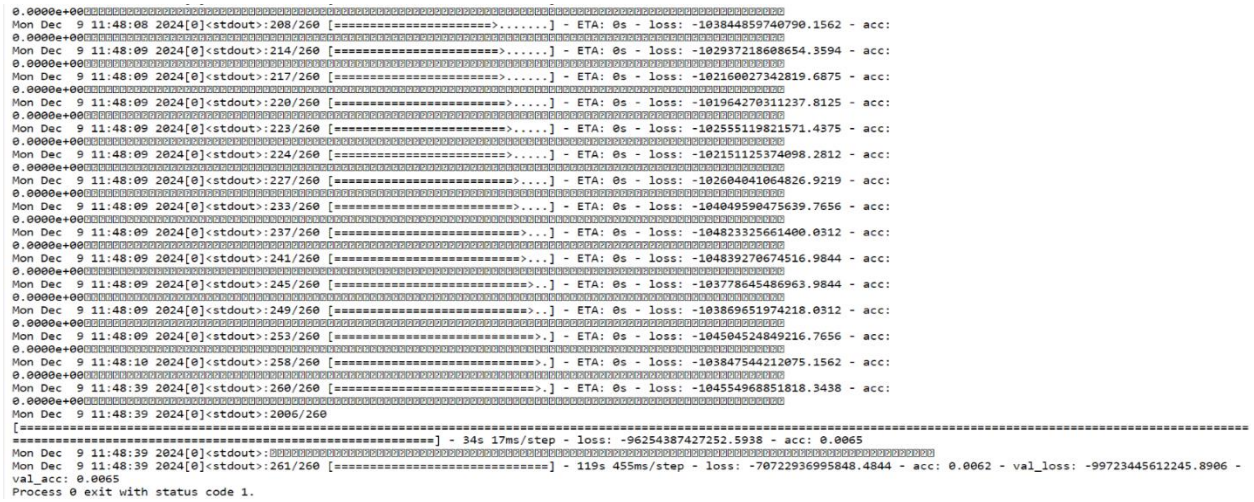
MLP Model

We developed an algorithm in Jupyter notebook to implement and evaluate a Multilayer Perceptron (MLP) model. A MLP is a part of the deep neural network (DNN) family, and particularly suited for clustering low-variance data. We need to split the dataset into training and testing sets to check for overfitting. The architecture of the MLP model was designed with a depth of 1 to 2 layers, training with over 5 to 10 epochs, using a batch size of 32.

Our dataset consisted of 149 classes, which meant model requires a deeper network to return the real training accuracies, time, and complexity. However, we only use a simplified 2-layer architecture to accelerate the training. While deeper layers are not strictly necessary for all outputs, lack of depth can affect the model's ability to generalize. The performance of the MLP model was poor, achieving accuracy of only 0.65%. This low accuracy can be reasoned as high variance and high-dimensional nature of the dataset (images), which is impractical part for the model.

The MLP model is not an ideal choice when it comes to an image classification task as compare other models such as baseline or optimized. It is all due to limited depth and inability to handle high-dimensional data effectively which can result in poor performance. However, there is a possibility to improve the model's accuracy by experimenting with additional layers, increasing the

number of epochs, or using other techniques which can handle high-variance data effectively.



Conclusion

The goal of this project was to use our skills learned in class to create a model to accurately predict a Pokémon from a given image. We were able to create a CNN model which accurately predicted the Pokémon approximately 89% of the time. We also created 2 comparison models, a KNN model and MLP model, which accurately predicted Pokémon 4% and 0.6% of the time respectively. Some patterns we noticed that more purple Pokémon were the most accurate whereas more yellow Pokémon were the least accurate Pokémon. We also noticed that Pokémon with very similar evolution lines were a lot less accurate than Pokémon who change drastically when they evolve.

Some limitations that we ran into was that not all the data was in the right folders. We had to try to move those images to the right folders, but we were not able to go through all 52,000 images and remove all bad images. Some other limitations we ran into was that most of the datasets we found didn't have 2 of the Pokémon, so we weren't able to do the entire 1st generation. Another problem we had was that some of the classes had a lot more images than others such as Pikachu and Evee. In the future we could expand our model to be able to recognize any Pokémon from all 9 generations rather than just the first generation. Our model would likely be a lot less accurate as there are currently 1,025 Pokémon which is a lot more output classes to predict than our current 149.