Midterm Checkpoint

Group 3: Aaditya Reddy Anugu, Justin Kang, Nathaniel Alexander Koehler, Patrick Soo, Zhixuan Wang

Introduction

Geoguesser is a game in which players are randomly placed somewhere in Google Street View and need to guess what their exact location is.

Previous literature [3][5] discusses using techniques like CNNs and transfer learning to analyze other image datasets to identify pneumonia in x-rays and find skin lesions and skin cancer from pictures. These papers have used these techniques to identify features that can help models classify an image. However, one of the shortcomings previous literature emphasizes is a lack of sufficient data in training models.

The following dataset, GeoLocation - Geoguessr Images (50K), found through Kaggle, contains 50,000 streetview images of the world, with every image belonging to 1 of 150+ countries. The data itself is not uniform as there are more images within certain countries compared to others, but we plan to combine datasets and prune folders with insufficient data.

Problem Definition

We are interested in seeing if we can train a model to accurately perform this task of identifying key objects that belong to only specific parts of the world, and correctly identifying which country the street view image is from.

This brings us to our problem - there may be certain circumstances in which it would be helpful to determine a relative location given a set of images, such as crime investigations. Thus, our motivation towards a potential solution to this is to start by using the Geoguessr dataset found through Kaggle, and train the dataset to determine which country it is in.

Methods

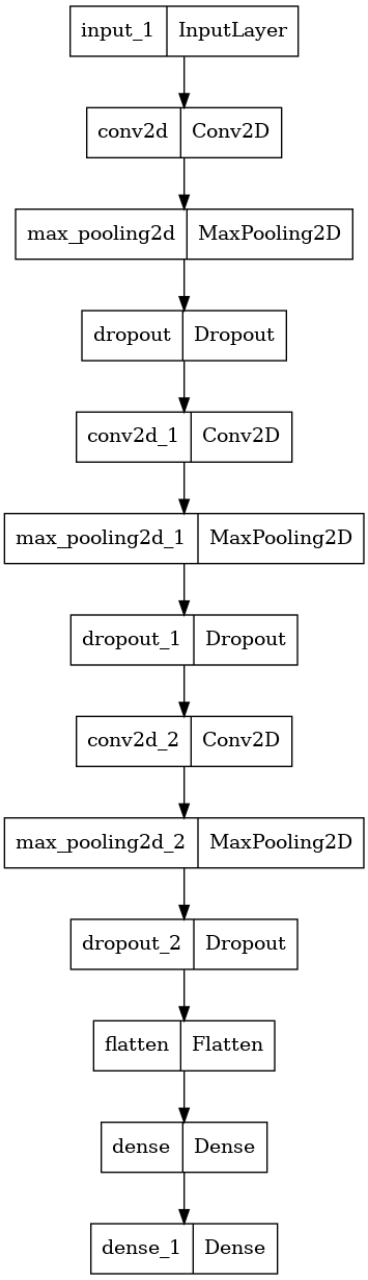
Data Cleaning and Image Preprocessing

Before our preprocessing step, we decided to first clean the dataset deleting non-uniform resolution images, and then deleted folders (classes) that had less than 100 images, as we believed it would be hard to classify images of those classes because of the small amount of data given. Lastly, due to the large dimensionality of the dataset, we decided to resize every remaining image into ⅓ of its original size.

Next, for the actual preprocessing step, we implemented standardization across the entirety of the remaining data. Due to issues with the memory when creating the datasets, we decided to utilize Tensorflow Datasets, which helped with memory as it doesn't load the entirety of the dataset in the variable at once. We divided the dataset into 70% training, 15% validation, and 15% testing. We fit a standard scaling (z-score) layer from Tensorflow Keras to the training set and transformed all three sets of data with this fitted layer. This resulted in our standardized training, validation, and testing datasets.

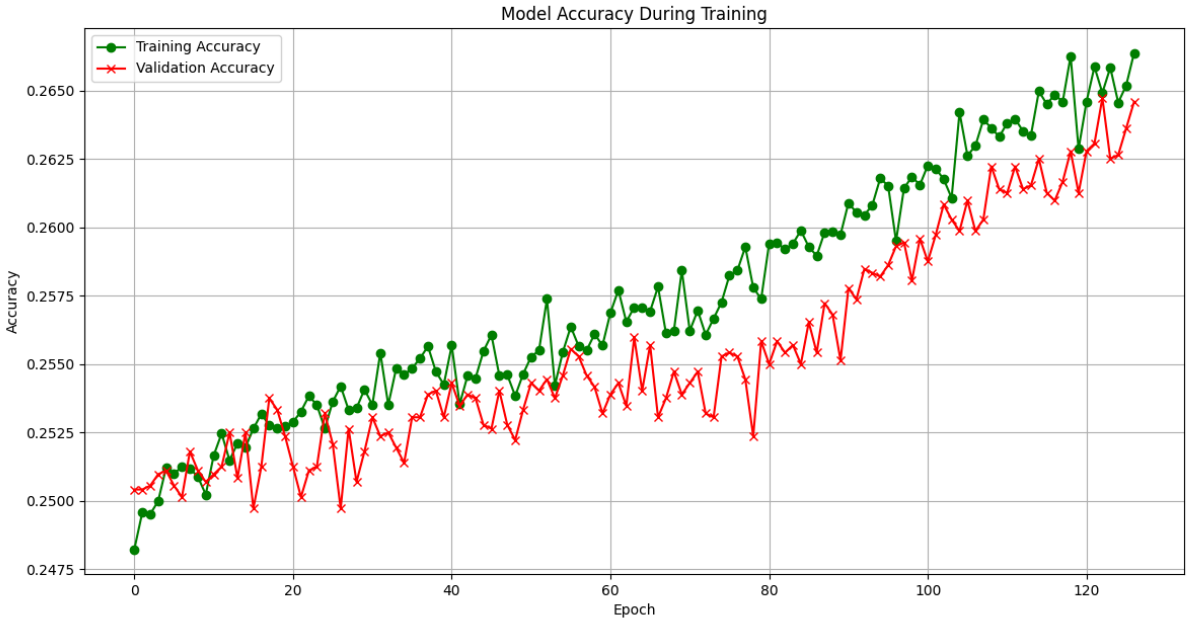
Machine Learning Algorithm/Model Implemented

For our ML Algorithm for this checkpoint, we used a Convolutional Neural Network, which was a type of supervised learning. We chose to do CNN because of its efficacy with handling image data. Widely known image classification models such as the ResNet or DenseNet also employ convolution layers. Conv2D layers take filters to extract the information and essentially summarize them into a pixel. MaxPooling layers have been known to perform well with Conv2D layers, and they also reduce the dimensions of the image. The model architecture is shown in the diagram below. In order to prevent overfitting, we used L1 regularizer and Dropout layers. The final layer has a softmax function that allows the image classification. As for the activation functions of the Conv2D layers, we used ReLU, as it is faster than Sigmoid to compute and also doesn't have Sigmoid's vanishing gradient issue.

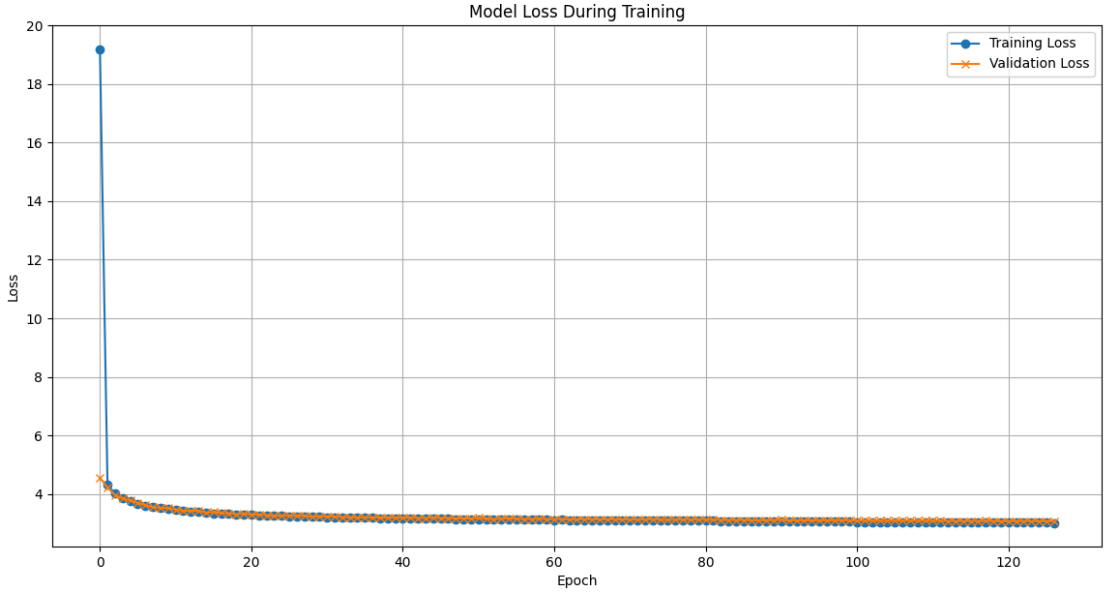


Results and Discussion

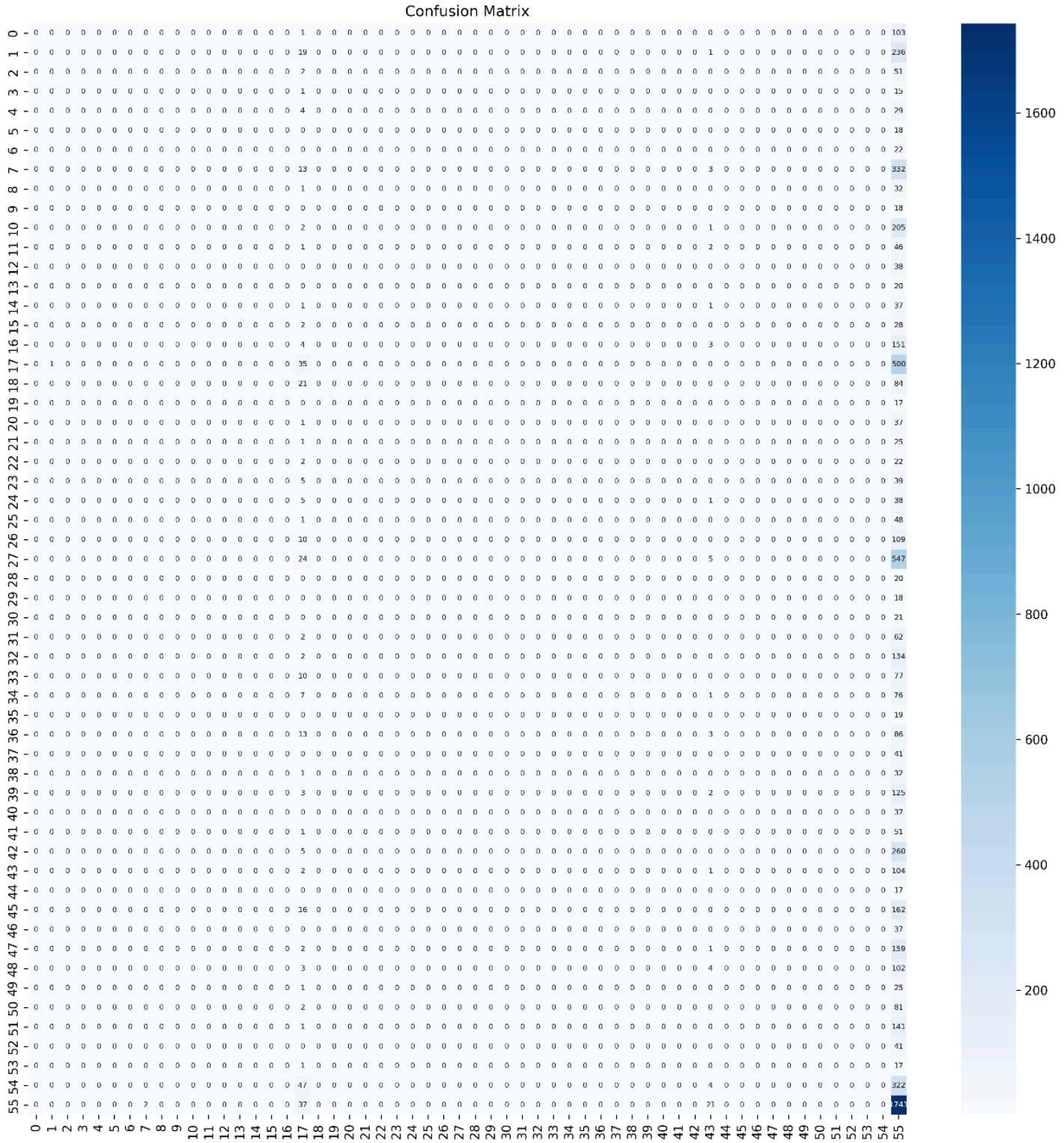
[Figure 1.2] Model Accuracy vs. Training Epoch



[Figure 1.2] Model Loss vs. Training Epoch



[Figure 1.3] Confusion Matrix


Confusion Matrix
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55
1600
1400
1200
1000
800
600
400
200

[Figure 1.4] Precision Score Chart

	precision	recall	f1-score	support
0	0.0000	0.0000	0.0000	104
1	0.0000	0.0000	0.0000	256
2	0.0000	0.0000	0.0000	53
3	0.0000	0.0000	0.0000	16
4	0.0000	0.0000	0.0000	33
5	0.0000	0.0000	0.0000	18
6	0.0000	0.0000	0.0000	22
7	0.0000	0.0000	0.0000	348
8	0.0000	0.0000	0.0000	33
9	0.0000	0.0000	0.0000	18
10	0.0000	0.0000	0.0000	208
11	0.0000	0.0000	0.0000	49
12	0.0000	0.0000	0.0000	38
13	0.0000	0.0000	0.0000	20
14	0.0000	0.0000	0.0000	39
15	0.0000	0.0000	0.0000	30
16	0.0000	0.0000	0.0000	158
17	0.1122	0.0653	0.0825	536
18	0.0000	0.0000	0.0000	105
19	0.0000	0.0000	0.0000	17
20	0.0000	0.0000	0.0000	38
21	0.0000	0.0000	0.0000	26
22	0.0000	0.0000	0.0000	24
23	0.0000	0.0000	0.0000	44
24	0.0000	0.0000	0.0000	44
25	0.0000	0.0000	0.0000	49
26	0.0000	0.0000	0.0000	119
27	0.0000	0.0000	0.0000	576
28	0.0000	0.0000	0.0000	20
29	0.0000	0.0000	0.0000	18
30	0.0000	0.0000	0.0000	21
31	0.0000	0.0000	0.0000	64
32	0.0000	0.0000	0.0000	136
33	0.0000	0.0000	0.0000	87
34	0.0000	0.0000	0.0000	84
35	0.0000	0.0000	0.0000	19
36	0.0000	0.0000	0.0000	102
37	0.0000	0.0000	0.0000	41
38	0.0000	0.0000	0.0000	33
39	0.0000	0.0000	0.0000	130
40	0.0000	0.0000	0.0000	37
41	0.0000	0.0000	0.0000	52
42	0.0000	0.0000	0.0000	265
43	0.0185	0.0093	0.0124	107
44	0.0000	0.0000	0.0000	17
45	0.0000	0.0000	0.0000	178
46	0.0000	0.0000	0.0000	37
47	0.0000	0.0000	0.0000	162
48	0.0000	0.0000	0.0000	109
49	0.0000	0.0000	0.0000	26
50	0.0000	0.0000	0.0000	83
51	0.0000	0.0000	0.0000	142
52	0.0000	0.0000	0.0000	41
53	0.0000	0.0000	0.0000	18
54	0.0000	0.0000	0.0000	373
55	0.2542	0.9667	0.4025	1803
accuracy			0.2462	7226

macro avg	0.0069	0.0186	0.0089	7226
weighted avg	0.0720	0.2462	0.1067	7226

[Figure 1.5] # Images In a Class

(Note: This graph goes from 1 to 56 instead of 0 to 55)

Our confusion matrix shows very little relationship in terms of having a visible diagonal. This possibly implies that the models were not accurate in predicting each class, and were heavily predicting class 55/56 (United States), which was most likely due to the disproportionate amount of data it had compared to the other classes.

Analysis of Convolutional Neural Network

Overall, the visualizations show and imply that the model's accuracy was very low, and it can be inferred that it is most likely linked to the way we cleaned our data and or preprocessed it. As previously mentioned, our dataset was also not uniform; it shows within the confusion matrix with the lack of diagonal and heavy weightage on class 55. Furthermore, the precision score chart (Figure 1.5) showed that class 55 had a staggering high difference in precision in comparison to the others because it had over 12,000 images while most others had around an average of 200~600 images, which made the model more biased in predicting class 55.

As for the model itself, the first two visualizations indicated that the model was doing well on the data in relation to the disproportionate dataset. Figure 1.1 showed that the validation and training accuracy were both going up, and both were increasing at a relatively same rate and value (disregarding outliers such as epoch 22 and 79). This meant that the model was learning, as the accuracy kept on improving overall as the number of epochs increased.

For Figure 1.2, the model loss graph is very similar to an average model loss graph. It has a sharp dip at the beginning, in which the validation and training loss converges towards a small

value in the end. Because the loss of the training and validation showed convergence, this gave signs that the model was not overfitting the data.

As a result, the accuracy and loss graphs showed that the model was consistent in handling the data, but the model was biased because it was trained on a high number of images within class 55. This shows just how important data cleaning and preparation is, and if the data itself is not good, it will most likely imply that the model training will not do well either.

Next Steps

As for next steps, we plan on focusing more on the how we should handle and prepare the data before training the model, and also utilize new preprocessing methods as well. As previously mentioned, our data was not uniform; this proved to have a negative effect on the model, as it would be able to classify labels that had much more images correctly than ones without. We thus plan on making the data more uniform by either providing more images to the dataset and or trimming more images from folders that had too many. We could also potentially utilize the class weights feature in Tensorflow.

GanttChart (1)

Contribution Table:

Group Member	**Contributions**
Aaditya Anugu	Intro, Background, Gantt Chart
Justin Kang	Methods, Results, & Discussion
Nathaniel Koehler	Methods, Github Pages Website
Patrick Soo	Methods, Results, & Discussion
Zhixuan Wang	Methods, Results, & Discussion