Analysis of Modeling Techniques for Country Image Classification

Albert Crooks crooksa1 Computer Science Senior
Santiago Rodriguez-Papa rodsan Computer Science Senior
Simon Situ situsimo Computer Science Junior
Johnny Okoniewski okoniew4 Computational Data Science Senior

Abstract

In this paper we attempt to create a machine learning model that can accurately determine the location of an image. This was inspired by the game GeoGuessr, in which players have to determine where they are on the planet solely through images of the surrounding environment. We developed two different multi-model pipelines: one using an autoencoder and SVM; and the other using a convolution neural network based on VGG-16. To simplify the learning model, we attempted to predict the country of origin rather than the precise coordinates where the images were taken. Previous works have used models trained extensively on a very large number of samples, but due to memory and space constraints, we opted for a pre-existing data set of 50,000 images. Since this data set was quite skewed (24.03% of images belonged to one class and 48 classes had less than 50 samples) we subsampled the data set to 11 countries that had over a thousand images. To further improve upon our VGG-16 model we used data augmentation to simulate a larger data set. We only used 11 classes of locations, so we cannot directly compare with the related works of DeepGeo and PlaNet.

1 Introduction

The game GeoGuessr places the player on a random street somewhere in the world. The goal of the player is to guess either the country or precise location of the image.

There are different rule sets that can be used to limit the amount of interaction players have with the Google Street View panorama. One such rule states that no interaction whatsoever can be done—the player cannot rotate their viewpoint, zoom in the panorama, or move positions—and therefore must base their guess on a single still image.

We created a model that—given a picture— predicts the country of origin using publicly available data. If our model were improved, it could aid in improving modern geo-location methods—currently based on GPS and Wi-Fi SSIDs—to pinpoint positions with higher accuracy. A successful model might also give novel insight into similar architectures and geographies across cities and countries.

Although most (if not all) machine learning projects start with good intentions, there are many examples where even though the original goals are benign, they end up being used for harmful or unethical purposes. We have, after much consideration, determined that this task is not dangerous in it's previous implementations, or in our implementation. We believe this is so because the model can only determine the country (or in DeepGeo's case, by state) that the photo belongs to. A negative application of this implementation is such that a bad actor trying to find a location of someone based solely on using the model, would only be able retrieve the country. We believe the worst-case scenario would be a government hostile to free speech using the program to determine if some political activist was in a their country, and then taking additional measures outside of the capabilities or intentions of our model to silence the activists. However, this seems extremely unlikely since there are much more reliable ways to find someone's country, such as someone's cell phone number or IP Geolocation lookup. But a more likely scenario, and far less harmful would be someone using this program to cheat in GeoGuessr. The implications of this are not drastic as there are other, more reliable, methods that could give the answer to an interested user. It could however negatively impact the quality of the game as well as reduce the amount of true scores on the leader board.

We divided the work equally among the four of us to accomplish this goal using the hours we have worked on the project such that each of us have put in equal time. We designed the system this way because it allows each member to be represented equally by effort rather than assigning tasks that can vary in time commitment.

2 Related Works

2.1 PlaNet

PlaNet (?) is a model that, when given a query photo, produces a discrete probability distribution over a portion of the Earth's surface. The part of the Earth that is used for predictions is tiled in a way that smaller tiles are given to densely populated areas, and larger tiles are given to sparsely populated areas. Each tile represents a class for the model to predict the probability that the given photo was taken in respective tile. A variety of LSTM-based (Long short-term memory) models were also experimented with to improve on the single photo query. Instead of a single photo, a sequence of photos of the same or similar location are given to the model to reduce the potential ambiguity from a standalone image. The paper found that LSTM-based models outperformed the single image model, and using longer sequences of photos yielded better performance.

2.2 DeepGeo

DeepGeo (?) uses a deep neural network model based on ResNet architecture and is trained using 0.5 million different images taken from various locations in each U.S. state. This model was used in order to investigate how geographical, ecological, and man-made features generalize for random location prediction. On the other hand, only taking the top guess gives an accuracy of This model is able to achieve an accuracy 20 times bet r than chance, and has a 71.87% of correctly predicting the state when the top 5 best guesses are taken. On the other hand, when only taking the top guess, the model is only 38.32% accurate. The baseline method used was CNN classification, and then used The Residual Network (ResNet) which uses residual blocks to achieve a better gradient propagation, which enables training much deeper networks. The network was implemented in Tensorflow and used the Adam algorithm for training. The hyper-parameters are: learning rate of 0.001, β_1 set to 0.9, β_2 set to 0.999, and $\varepsilon = 10^{-8}$. In addition, there is a weight decay of .0001, and the model was trained for 50 epochs using a 90/10 train/testing split, which uses the epoch with the highest validation accuracy for testing.

3 Methodology

3.1 Data set

Our initial data set consisted of 49,997 unique 1536×662 3-channel images from 124 different countries taken from Google Street View, totaling 6.77 GB (?). The data set was not expanded further due to memory and storage constraints. The data set is quite unbalanced, with approximately 24.03% of the images belonging to the `United States` class, and 48 classes having less than 50 images, of which 13 have exactly one. An ideally uniform data set would hold $\frac{1}{124} = 0.806\%$ of samples per class.

Initially, we performed preprocessing to speed up training and reduce memory usage. The images were first converted to gray-scale using the xxxx method, cropped to remove all graphical elements from the Google Street View UI, and then cropped further to achieve a square aspect ratio. We then resized the images down to 128 by 128 pixels, flattened them into a 16,384 length 1-D array, and normalized the pixel values from $(0,255)$ integer values to $(0,1)$ 32-bit floating point values.

3.2 PCA + SVM

We started our experimentation by applying Principal Component Analysis (PCA) from the `scikit-learn` library to the data set (?). This algorithm is used to reduce the dimensionality of a data set while preserving its most salient features. Our initial experiments aimed to achieve a final dimensionality of $d = 128$, which is equivalent to a $128\times$ reduction in feature size. However, the PCA algorithm was unable to converge to a solution as our large data set could not fit fully in memory. As such, we then opted to instead use Incremental Principal Component Analysis (IPCA). IPCA is a batched variation of PCA used to perform feature reduction on the images while avoiding the need to keep the whole data set in working memory.

The next step in our original pipeline was to simply classify the data set using its principal components. However, we first wanted to fine-tune the optimal number of principal components to aim for [other word], as this would result in a more robust model. For this initial exploration, we settled on a Support Vector Machine (SVM) classifier. This classifier aims to find decision boundaries that maximize the gap between categories (?). The implementation used was Scikit-learn's `sklearn.svm.SVC` with a constant regularization parameter $C = 1.0$ and a Radial Basis Function (RBF) kernel [[also cite this]]. We then shuffled the data set and split it into 80% testing and 20% training. Finally, we varied the target number of principal components $d \in \{64, 128, 256\}$ and measured the training and testing accuracy while keeping the classifier constant.

The above resulted in a training accuracy of 27.62% and a testing accuracy of 25.31% for 64 components, a training accuracy of 24.26% training accuracy and testing accuracy of 23.21% for 128 components, and a training accuracy of 24.13%

training accuracy and a testing accuracy of 23.81% testing accuracy for 256 components. Despite $d = 64$ having higher training and testing accuracy, we opted to work with 128 dimensions instead. This is because the speed improvements were not considerable compared to 64 dimensions, and we feared the lower feature size would impede later classification.

Figure 1: The original image of a road in Aland

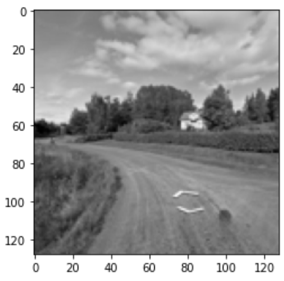

Figure 2: Sized down and grey-scaled image of a road in Aland, used as input into our models.

Once we had settled on the feature reduction part of our model, we moved onto classification. We began by harnessing the Perceptron Learning Algorithm (PLA, or most commonly, Perceptron). We used the Perceptron algorithm from sklearn which handles multi-class classification. We used the default hyperparameters for this model. This yielded a training accuracy of 3.46% and a testing accuracy of 3.38%.

Given our quite poor results, we determined the resulting data set after IPCA was not linearly separable. We thus moved onto using a Support Vector Machine together with the kernel trick in order to be able to classify nonlinear data. We performed a grid search with the Support Vector Machine on the hyperparameters of C and kernel function. The values of C used were: 0.01, 0.05, 0.1, 0.5, 1, 2, 5, 10; the kernels used were: RBF, sigmoid, quadratic polynomial, cubic polynomial, and quartic polynomial. The grid search found that the best testing accuracy was achieved with a C set to 2 and the kernel set to the RBF kernel.

128x128 128-PCA + SVM Training Metrics	
Accuracy	0.40206
Macro-Precision	0.75366
Macro-Recall	0.12557
Macro-F1	0.17639

Figure 3: Training metrics for the SVM on the 128-component PCA transformed images.

128x128 128-PCA + SVM Training Metrics	
Accuracy	0.26060
Macro-Precision	0.03894
Macro-Recall	0.02047
Macro-F1	0.01982

Figure 4: Testing metrics for the SVM on the 128-component PCA transformed images.

3.3 Autoencoder + SVM

Considering the relatively-low performance of the `PCA + SVM` model above, we looked into replacing the Principal Component Analysis for an autoencoder. Our intention was to improve the performance of the overall model by improving the quality of the features being fed into our classification algorithm. In this case, we use "quality" to refer to how meaningful or representative they are compared to the initial data set. Autoencoders have a wide variety of uses, ranging from noise reduction to anomaly detection. They are a great fit when it comes it comes to dimensionality reduction, as they usually outperform other comparable methods (**?**).

We opted to use a Stacked Convolutional Autoencoder. We chose a stacked autoencoder in place of a more traditional three-layer autoencoder as it usually yields better accuracy and more robust results despite sacrificing memory usage and training time (**?**). This design choice allowed us to use multiple convolutional layers, which usually yield higher accuracy when it comes to images compared to other methods (**?**). This is because the convolutional layers are able to take advantage of the spatial locality inherent to 2-D images—information which is completely lost during the flattening step prior to a PCA. Note that "accuracy" is measured differently in autoencoders compared to other models—in this case, we use accuracy to refer to how well the model can reproduce the original input. While an autoencoder consists of a separate encoder and decoder, we will only use the encoder part to feed into the SVM.

The encoder model consisted of 11 total layers, taking an 128×128 grayscale image and turning it into a 128-long vector. It has five maximum pooling layers—which downsample their input—and five convolutional layers. It was modeled using TensorFlow/Keras (?) and trained with the Adam optimization algorithm using binary crossentropy for its loss function. We achieved a training accuracy of 76.3% and a testing accuracy of 23.2% over 20 epochs, indicating that the model was likely overfitting. We decided to harness He Normal kernel initialization—which initializes the neuron weights with a "truncated normal distribution centered on 0 with $\sigma = \sqrt{\frac{2}{n_{\text{inputs}}}}$" (?)—and L2 kernel regularization—which applies a penalty based on layer activity—on all `Conv2D` layers, as well as `Dropout()` layer—which randomly sets a certain fraction of neuron edges to zero in order to increase robustness—after every `MaxPooling2D` layer. However, none of these mechanisms were able to combat overfitting.

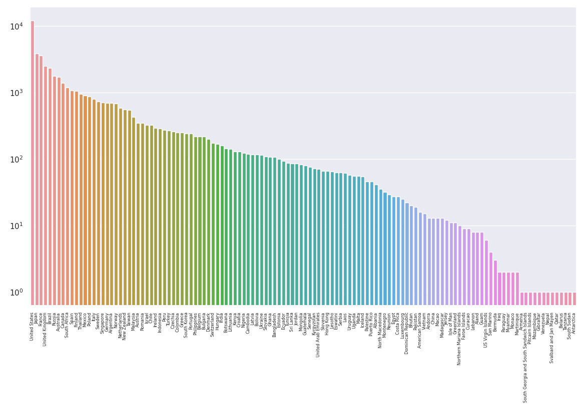

Figure 5: Sample count per class in the original data set, sorted by sample count; y-axis is logarithmic.

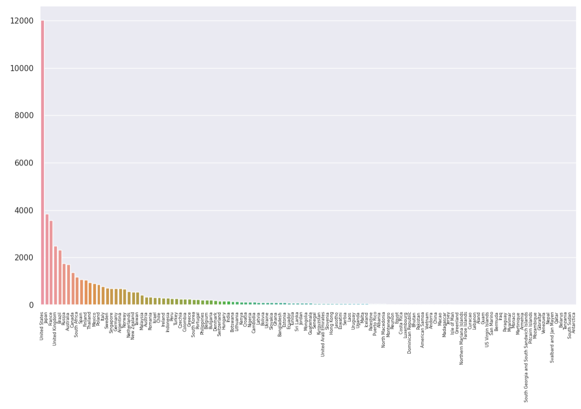

Figure 6: Sample count per class in the original data set, sorted by sample count; y-axis is not logarithmic. The really low number of samples in the later classes can be observed.

Given the above results, we determined we would be unable to train a proper Perceptron model with our highly-skewed data set—the overfitting problem was likely caused both by a high percentage of samples belonging to the `United States` class, as well as the extremely low percentage of samples belonging to other classes. We decided our next step should be to produce a more-robust data set by subsampling the current, larger data set. Based on the distribution of the original data set (Figure 5), we decided to generate two new data sets: one which includes all classes with over 1000 samples—nicknamed Data set A—and one which includes all classes with over 1000 samples, but only 1000 samples per class—nicknamed Data set B.

3.4 Stable Diffusion Variational Autoencoder

We then decided to revisit our earlier autoencoder idea. Given that training an autoencoder from scratch proved difficult with the limited and skewed set of data—and that both Data set A and B are even further reduced—we opted to make use of an already-existing implementation. After comparing various models, we came across the work of Rombach et. al. in Stable Diffusion, a latent diffusion model for image synthesis (?). The model for the text-to-image process is made up of three main components: a `ClipText` model for text encoding, a `UNet` model for information diffusion through the latent space, and an variational autoencoder's (VAE) Decoder, which generates the final image from the latent space (?). Given that their model is well-known for its performance and quality (?), we extrapolated that the encoder of their VAE model would be good as well.

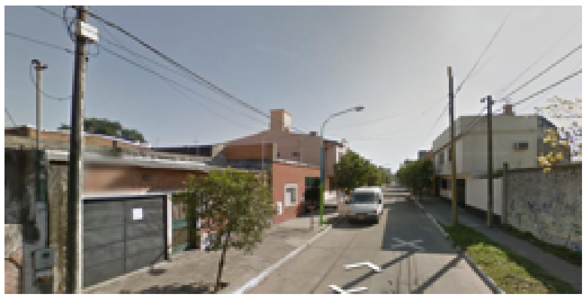

Figure 7: Original image fed into the Stable Diffusion VAE: a road in Argentina.

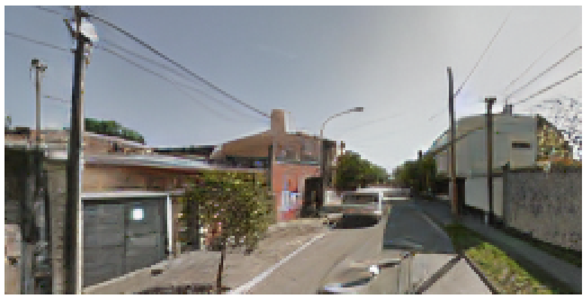

Figure 8: Reconstructed image from the VAE latent space.

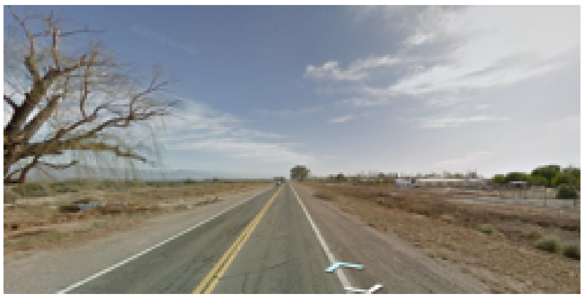
Figure 9: Original image fed into the Stable Diffusion VAE: a road in Chile.

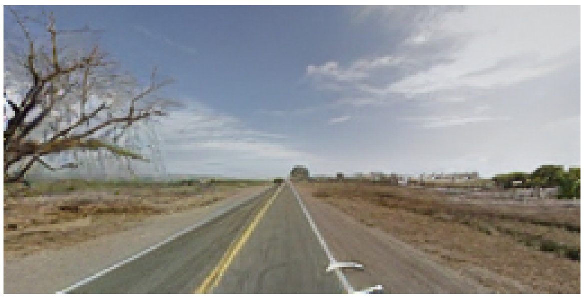
Figure 10: Reconstructed image from the VAE latent space.

Using the `diffusers` library, we were able to load the model and weights of the `CompVis/stable-diffusion-v1-4` VAE. We then preprocessed Data Set A according to its requirements and then selected a random image for the model to encode–decode. Because this is a VAE, the latent representation used must be sampled from the decoder's distribution. We assume that the sampled latent representation The initial image and result can be seen in (Figure 8). A second attempt can be seen in (Figure 10) We then repeated this experiment a handful of times. Since these reconstructions were incredibly successful, we concluded that our hypothesis—that the Stable Diffusion VAE model was well-trained—was correct, and thus assumed the latent space would encode sufficient data to classify the images. After encoding the whole of Data set A at various resolutions (64×64, 128×128, 128×256, and 256×256) using 3-channel color, we proceeded to attempt at classifying the compressed images.

3.5 Stable Diffusion VAE Encoder + SVM

We trained new models with the compressed representations of each image in the data set encoded by the VAE. For 64x64 images, the encoded representations were flattened from 1x4x16x4 tensors into 256 feature sets. We compared the performance between Perceptron, SVM, and a fully connected feed forward neural network. To counteract the imbalance in the data set, the classes are weighted inversely proportionally to their representation in the data set such that the class weighting is balanced. We used the same hyper-parameters for Perceptron that we used for the Perceptron that we trained on the PCA transformed images. Before testing an SVM using sklearn, we performed a grid search for the C hyper-parameter that gives the best macro-F1 score and found that a value of 1.5 performed the best and used it for all of our tests involving an SVM. For the grid search, we used a bagging classifier using three SVM estimators, each trained on a random third of the data set to speed up the search. We acknowledge that the C value found using the bagging classifier may not be the same that would be found if we used a single SVM, but we only needed a decent value to use. The RBF kernel was used and the gamma parameter is set to "scale". The best macro-F1 score came from the PCA + SVM model trained on 128x128 encoded images (Figure 11).

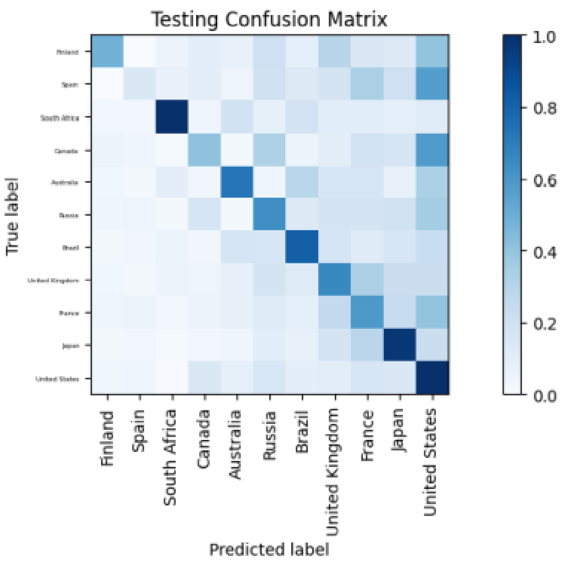

Figure 11: The confusion matrix on the testing set from the SVM trained on the 128x128 encoded images. The predicted labels are ordered from least represented to most represented in the data set.

64x64 SVM Training Metrics	
Accuracy	0.79948
Macro-Precision	0.77927
Macro-Recall	0.91751
Macro-F1	0.83230

Figure 12: Training metrics for the SVM on the 64x64 encoded images.

64x64 SVM Testing Metrics	
Accuracy	0.37826
Macro-Precision	0.31292
Macro-Recall	0.32608
Macro-F1	0.31403

Figure 13: Testing metrics for the SVM on the 64x64 encoded images

64x64 PCA + SVM Training Metrics	
Accuracy	0.78868
Macro-Precision	0.76755
Macro-Recall	0.91002
Macro-F1	0.82145

Figure 14: Training metrics for the PCA + SVM pipeline on the 64x64 encoded images.

64x64 PCA + SVM Testing Metrics	
Accuracy	0.37147
Macro-Precision	0.30861
Macro-Recall	0.32489
Macro-F1	0.31061

Figure 15: Testing metrics for the PCA + SVM pipeline on the 64x64 encoded images.

128x128 SVM Training Metrics	
Accuracy	0.85958
Macro-Precision	0.84461
Macro-Recall	0.94816
Macro-F1	0.88819

Figure 16: Training metrics for the SVM on the 128x128 encoded images.

128x128 SVM Testing Metrics	
Accuracy	0.41532
Macro-Precision	0.33231
Macro-Recall	0.33949
Macro-F1	0.33214

Figure 17: Testing metrics for the SVM on the 128x128 encoded images.

3.6 Fine-tuning and transfer learning VGG-16

In addition to the above, we found it interesting to answer whether our `Encoder + SVM` accuracy could be improved upon with a different model, or whether there existed inherent limitations in our data set that would stop any model from making further progress. We determined that the spatially-local nature of the individual images mentioned in the `Autoencoder + SVM` subsection was worth exploring. Since we had learned in the aforementioned section that our data set was not large enough to train a convolutional model from scratch without overfitting, we began to look for alternatives. After some time researching, we stumbled upon the concepts of fine-tuning and transfer learning.

128x128 PCA + SVM Training Metrics	
Accuracy	0.85039
Macro-Precision	0.83420
Macro-Recall	0.94359
Macro-F1	0.87949

Figure 18: Training metrics for the PCA + SVM pipeline on the 128x128 encoded images.

128x128 PCA + SVM Testing Metrics	
Accuracy	0.41300
Macro-Precision	0.33467
Macro-Recall	0.34337
Macro-F1	0.33447

Figure 19: Testing metrics for the PCA + SVM pipeline on the 128x128 encoded images.

Fine-tuning and transfer learning both involve building upon the knowledge already learned by a pre-existing model. The latter involves freezing the totality of the pre-existing model and training a number of "top" layers to adapt to the new domain; the former instead involves further training the parameters from the pre-existing model (**?**). In this case, we took a hybrid approach.

We decided to use the VGG-16 Convolutional Neural Network, as it has been trained to classify 1000 classes from the ILSVRC-2012 data set (**?**).

Our sequential model consisted of the pre-trained VGG-16 model (imported through TensorFlow's `tf.keras.applications.VGG16`) without its top layers; a `GlobalAveragePooling2D` layer; a `Dense` layer with 1024 neurons, ReLU activation, with a He Normal kernel initializer, and an L2 kernel regularizer with a regularization factor of 0.001; a `Dropout` layer with a rate of 0.2; and a `Dense` prediction layer with 11 outputs (equal to the number of classes on Data Set B) and a `softmax` activation. Just like in the `Stable Diffusion VAE + SVM` model, we opted to use the full color, RGB images as they would encode the most information.

In order to further combat overfitting, we augmented Data Set B using Keras' `ImageDataGenerator`. We used a rotation range of 40, a width and height shift range of 0.2, shear and zoom range of 0.2, and enabled horizontal flips. We did a train-test split of 85:15.

We began training by freezing the pre-trained VGG-16 section,compiling the whole model with the `rmsprop` optimizer and `sparse_categorical_crossentropy` loss. We trained it for 20 epochs with a batch size of 32, achieving a training accuracy of 35.78% and a testing accuracy of 36.12% (Figure 20).

We then unfroze the last two convolutional blocks of the VGG-16 model (layers 11–18) and recompiled the model with a Stochastic Gradient Descent optimizer, with a learning rate of 0.0001 and a momentum of 0.9. We continued training on the same data set as before, and stopped at 500 epochs or when testing accuracy did not increase significantly for 20 epochs, whichever happened sooner. We achieved a training accuracy of 76.21% and a testing accuracy of 61.21% (Figure 21). Further statistical analyses are discussed in the `Experimental analysis` section.

The trained model and Jupyter Notebook used for developing and training are attached with this paper. They can be run under our Docker environment with the command `docker run -rm -it -gpus all -p 8888:8888 -e JUPYTER_TOKEN=foo -v /$(pwd)/nb:/tf rodsan0/geoguessr-cse404`.

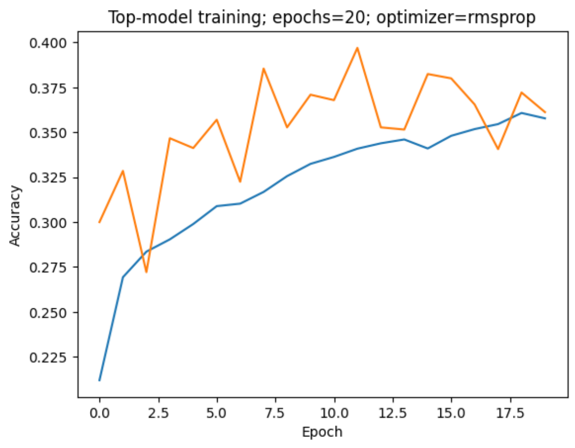

Figure 20: Graph of training (blue) and testing (orange) accuracy with respect to epochs during training of the top part of the custom VGG-16 model.

4 Experimental analysis

The SVM that was trained on data that was transformed by PCA performed much worse (Figure 4) than the SVM that was trained on the data that was encoded by the VAE encoder (Figure 13). Due to computational limits, we had to pick an unreasonable number of components for the PCA to transform the data down to, eliminating a majority of the

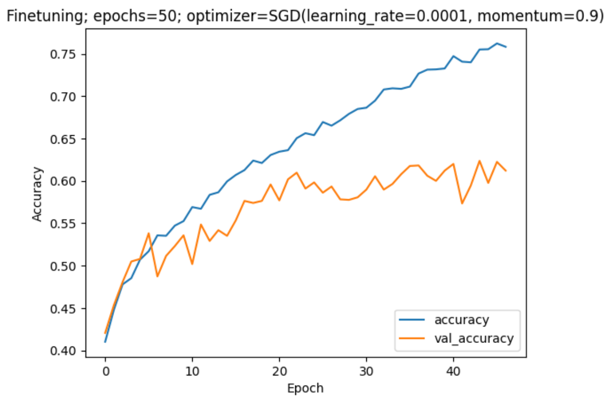

Figure 21: Graph of training and testing accuracy with respect to epochs during fine-tuning of the custom VGG-16 model.

explained variance. The VAE encoder performed feature extraction on the data instead of the feature reduction by the PCA, so the features were higher in quality. The SVM models trained on the encoded 128x128 images performed (Figure 17) only slightly better than on the encoded 64x64 images (Figure 13). Despite the use of balanced class weighting, we can still observe bias towards the countries with a larger proportion of the data set in the confusion matrix from the testing set (Figure 11), although it could be the result of inter-class similarity.

The VGG-16 pipeline was determined to have a training precision of 0.9864018 and a training recall of 0.98141176. Its test precision was determined to be 0.96080995 and its training recall was determined to be 1.0. Both these values were calculated using `tf.keras.metrics.Precision()` and `tf.keras.metrics.Recall()` respectively. Using `sklearn.metrics.f1_score`, we found the training macro F1-score to be 0.803665 and the testing macro F1-score to be 0.610430. We also calculated normalized confusion matrices for the training (Figure 22) and testing (Figure 23) data sets. This model had the highest training and testing accuracy of any other model we attempted, with a training accuracy of 76.21% and a testing accuracy of 61.21% (Figure 21). Furthermore, we can observe the model making mistakes a human player would—confusing Canada for the United States, confusing European countries with each other, and not confusing Brazil or Australia with any other country.

5 Conclusion

Based on our work, we can conclude that the two models developed—`VAE Encoder + SVM` and `Fine-tuned VGG-16` achieve the original goal

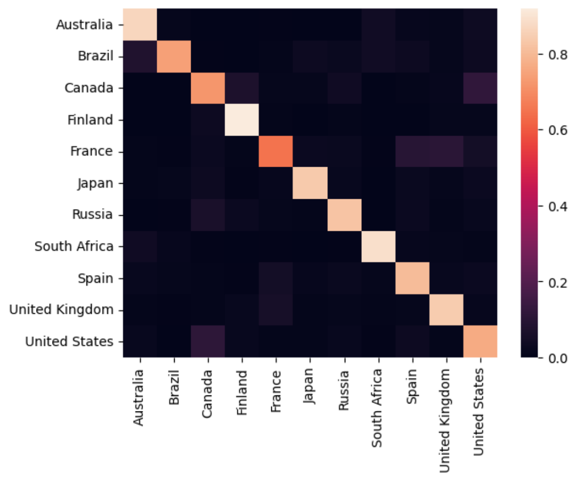

Figure 22: Confusion matrix on the training data set for the custom VGG-16 model. Counts have been normalized to [0, 1].

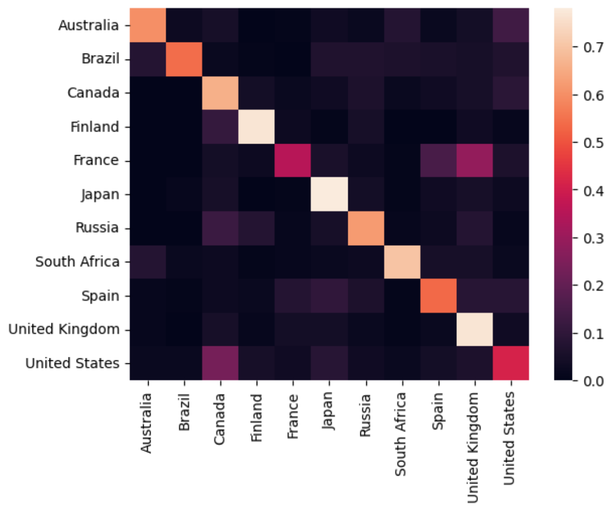

Figure 23: Confusion matrix on the testing data set for the custom VGG-16 model. Counts have been normalized to [0, 1].

we set out to do—classify Google Street View images into countries. The need to reduce the number of classes and samples stemmed from the fact our original data set was not of the highest quality. However, given the ease with which the `Fine-tuned VGG-16` model trained, we believe it will comfortably scale to more classes given a more balanced data set. However, based on the fact that this model committed the same mistakes a human player would, we can infer that it will be difficult to improve beyond human-level play.

We can also conclude that our models cannot be used to unethically locate a person beyond their general country or region.

Finally, we can conclude that models that preserve the spatial locality context—such as our VAE encoder model or our VGG-16 model—perform better than models that disregard this information.

6 Future Work

We opted to limit ourselves to the original 49,997 unique images in the Kaggle dataset (**?**) since using the Google Street View API to procure any more would have been quite costly. Having access to further data could have improved our models' performance and increase the number of accurately classified countries. Despite this, there is an upper limit to the number of images that could theoretically be useful, as there is a point of diminishing returns with regards to accuracy, efficiency, and performance.

A less compressed version of the data set would have also improved training at every step. Storing the images as lossless `.pngs` instead of lossy `.jpegs` would have improved performance at the cost of disk space.

We were limited by computational power to experiment with higher numbers of PCA components for the raw images, and with higher-end machines, would likely yield better results.

We only performed image augmentation for the data used for the VGG-16 model, and not for the data that was encoded by the VAE encoder. The SVM models that were trained on the encoded data could have performed much better. We cannot confirm how much of the performance from VGG-16 is due to the transfer learning or the image augmentation of the data used to fine-tune the model for our application.

An idea during the latter stages of development was to hook up the trained model to some script or extension that would allow it to play GeoGuessr in real time. The model is quite fast at classifying images, so this would have simply required a way to continuously capture the screen or web browser, crop and resize the images down to a reasonable resolution, and sent them through the model. We expect this would enable such a program to make highly-accurate country guesses in real time. Sadly, due to a lack of time, we were unable to proceed with this experiment.

It would also be interesting to have a deeper and more careful look into our earlier, less successful models. Instead of a PCA, we could have attempted Singular Value Decomposition. Future work could involve comparing and contrasting these two methods on our earlier pipelines.

7 Acquired Skills

This project was designed with the hopes of creating a model that could accurately predict the location of a still Google Street View image. We ended up partially achieving this goal—with two widely distinct models that can somewhat accurately predict the country of origin of a random still image from Google Street View—but also gained a deepened understanding of specifically the Perceptron Learning Algorithm and Support Vector Machines. This independent research project showed us how we could take high level research done by some of the best and brightest people in the field and apply it to a novel problem we wanted to solve. The value of this cannot be overstated as new ideas and techniques come from trying new things. A curious mind or group of curious minds could use current research like we did and modify it in such a way that allows for new breakthroughs in the field.

Although we did not achieve any breakthroughs that advanced the field, there are other quantifiable measures in which our group improved. We gained valuable insight into group dynamics when solving problems using coding that is appreciative not only in academia but industry as well. The application of implementing our own model to solve a problem we created allowed us to approach the problem in a more rigorous way. We gained a more intuitive understanding of the underlying mathematical principals that make principal component analysis as well as support vector machines work than if we had only been shown the techniques mathematical proofs and or pseudo-code. It should be emphasized that the difference between a homework assignment and nearly a semester long project is quite drastic.

We thought it was interesting that the backing of most of these approaches are coined as mathematical, but in practice they seem much more statistical because we are dealing with probabilities rather than certainties in outcome. Although the difference between mathematical and statistical might seem semantic the specificity of language allows for the accurate and efficient transfer of knowledge. The outcomes of these models are also not certain but based on any given model where we decided to use machine learning algorithms which goes to show that in the real world it is more of an art form choosing a model than an exact science in our opinion.

References

Thomas Helmuth, Lee Spector, and James Matheson. 2014. Solving uncompromising problems with lexicase selection. *IEEE Transactions on Evolutionary Computation*, 19(5):630–643.

Jose Guadalupe Hernandez, Alexander Lalejini, and Charles Ofria. 2022. An exploration of exploration: Measuring the ability of lexicase selection to find obscure pathways to optimality. In *Genetic Programming Theory and Practice XVIII*, pages 83–107. Springer.

Manoj Kumar, Dr Husain, Naveen Upreti, Deepti Gupta, et al. 2010. Genetic algorithm: Review and application. *Available at SSRN 3529843*.

Matthew Andres Moreno, Santiago Rodriguez Papa, Alexander Lalejini, and Charles Ofria. 2021a. Signalgp-lite: Event driven genetic programming library for large-scale artificial life applications. *CoRR*, abs/2108.00382.

Matthew Andres Moreno, Santiago Rodriguez Papa, and Charles Ofria. 2021b. Conduit: A c++ library for best-effort high performance computing. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '21, page 1795–1800, New York, NY, USA. Association for Computing Machinery.