A. Kumar, A. Zutshi, M. Shull, S. Andrews
April 18th, 2025
DATASCI 281

Classifying Real-World Flags Using Synthetic Data

Context

This project explores the task of classifying national and ideological flags, including rarer extremist group flags, using computer vision techniques. Our objective was twofold: first, to build a classifier trained entirely on synthetically generated flag images, and second, to assess the model's ability to generalize to real-world flag images found in uncontrolled environments.

The motivation behind this approach stems from the limited availability of labeled real-world data for many flag types, especially those representing fringe or extremist groups. These flags are often difficult to source at scale, and even when available, they appear under varying conditions – partially occluded, distorted by wind, or captured in poor lighting. Synthetic data, on the other hand, offers a scalable and customizable way to build large, labeled datasets, enabling researchers to simulate the visual diversity and complexity of real-world scenes.

This task reflects broader challenges in low-resource classification, particularly in domains like digital forensics and open-source intelligence. In these fields, analysts often face the daunting task of reviewing massive volumes of images to identify key visual indicators despite having limited real-world training data available to build automated tools. Our project explores whether synthetically generated data can be used to train models that assist with this process, helping to prioritize and surface relevant images when labeled examples are too scarce to support traditional supervised learning approaches.

To tackle the problem, we used Blender, a 3D modeling and rendering tool, to generate a large dataset of synthetic flag images with realistic cloth physics, motion, lighting variation, and camera perspective shifts. We extracted both classical and deep learning features – specifically Histogram of Oriented Gradients (HOG), Hue, Saturation, Value (HSV) color histograms, Local Binary Patterns (LBP), and Convolutional Neural Network (CNN) feature vectors from a pre-trained ResNet model – and trained a variety of classifiers using SVM and Random Forest architectures. Finally, we evaluated model performance on both synthetic test images and a curated set of real-world images collected from public sources to better understand how well our synthetic training data transfers to real-world scenarios.

Datasets

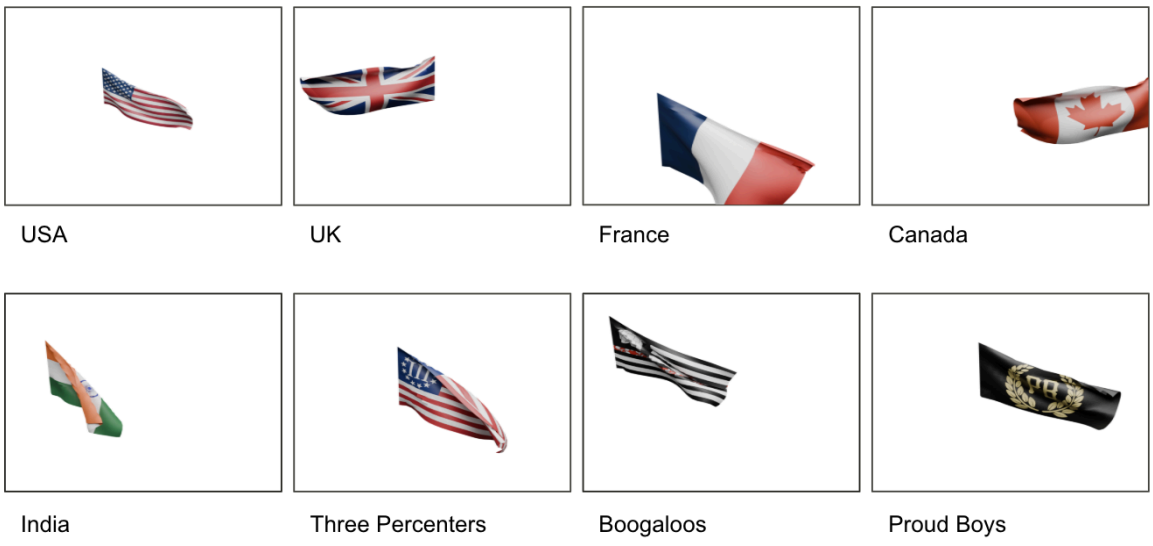
Our dataset consists of two parts: a large set of synthetically generated flag images used for

training, finetuning, and testing our models; and a separate set of unseen real-world test images. Link here: 281_flags_dataset

Synthetic Data for Training, Validating, and Testing Models

We used Blender, a free and open-source 3D modeling and rendering tool, to generate a large synthetic dataset of national and ideological flags. Each flag was modeled as a cloth mesh and subjected to physics-based simulation with wind forces applied, replicating realistic wave-like motion and deformation. To simulate the kinds of variation found in real-world imagery, we randomized lighting, camera angles, focal lengths, and cloth dynamics for each render. This introduced diversity in visual perspective, occlusion, and lighting conditions across the dataset.

Flags were rendered on transparent backgrounds at a resolution of 512 by 512 pixels and later resized to 224 by 224 for training. We incorporated multiple aspect ratios to reflect the true dimensions of different flags. In total, we generated 400 training images per flag category across eight categories, including rarer extremist group flags. This synthetic pipeline allowed us to scale data generation, making it possible to train our model on varied, labeled data even in the absence of a large real-world dataset.



Real-World Data for Testing Final Classifier Models

For our real-world dataset, we collected images using both the Unsplash API and manual Google Image Search. Our goal was to capture flags in natural, in-the-wild settings like parades, protests, and urban environments — places where flags often appear in news and public imagery.

A. Kumar, A. Zutshi, M. Shull, S. Andrews
April 18th, 2025
DATASCI 281

We focused on variation: flags that weren't always centered, fully visible, or perfectly lit. This gave us images with realistic challenges like motion blur, occlusion, harsh lighting, and unusual angles. We collected and used approximately 50 images per class for our national flag categories. For the extremist flags, we manually gathered examples through Google Image Search, as Unsplash and similar platforms prohibit content associated with extremist groups.


USA


UK


France


Canada


India

Three Percenters


Boogaloos

Proud Boys

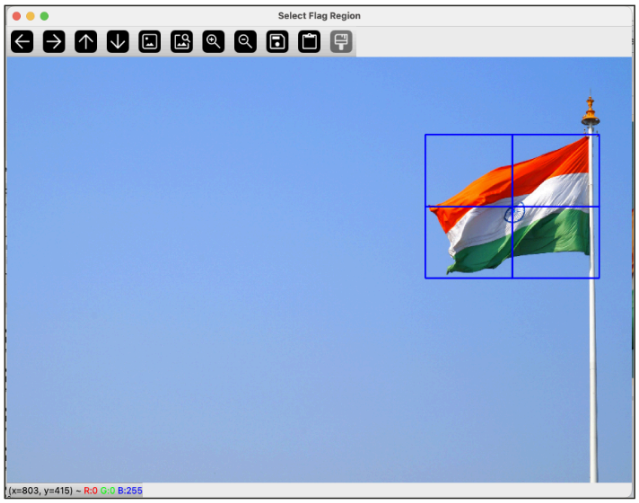
Preprocessing

Cropping, Scaling, and Padding

Synthetic images were resized from 512 by 512 pixels to 224 by 224 pixels to match the input size expected by standard CNN architectures like ResNet, which we used for feature extraction.

For our real-world images, we manually reviewed each one to filter out irrelevant results. Since the retrieved images varied widely in size and composition, we used a Python GUI we wrote to crop and isolate flags by hand. The cropped flags were then scaled and padded first to fit 512 by 512 pixels then ultimately converted to 224 by 224 pixels to match the format of our synthetic images. These real-world examples were reserved for testing only, allowing us to evaluate how well the model generalizes beyond the synthetic data it was trained on.

A. Kumar, A. Zutshi, M. Shull, S. Andrews
April 18th, 2025
DATASCI 281



Cropping Scaling Padding

Train-Test-Validation Split

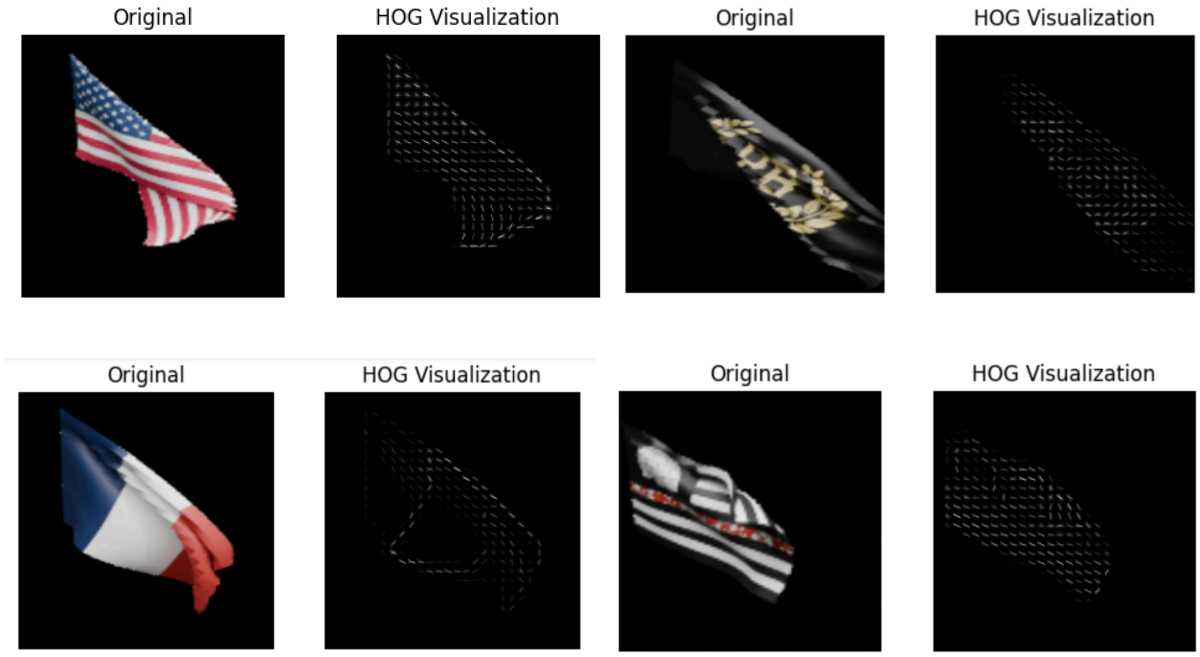
To prepare the synthetic dataset for training and evaluation, we implemented a two-stage stratified split. First, we set aside 20% of the synthetic data for testing. From the remaining 80%, we allocated 20% for validation, which was used for hyperparameter tuning and model selection. This resulted in a final split of 64% training, 16% validation, and 20% testing. The real-world dataset was reserved entirely for testing.

Feature Extraction

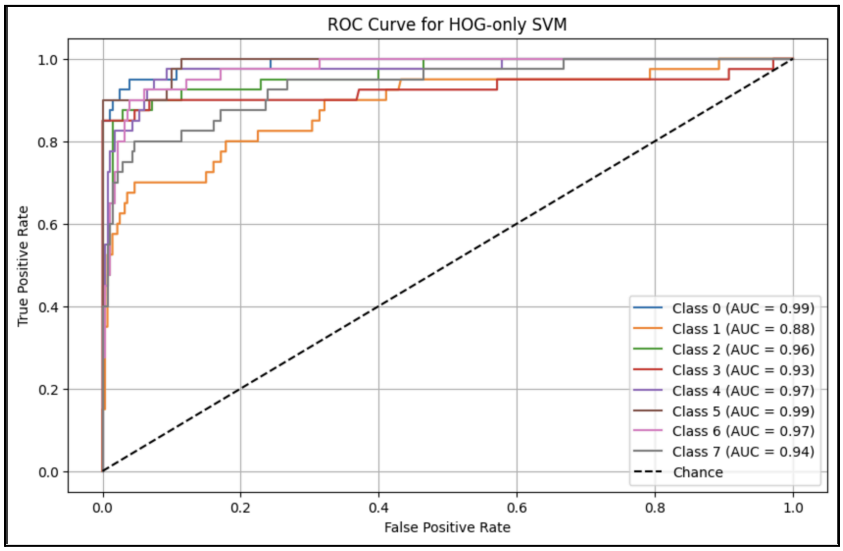
To capture different visual characteristics of each flag, we implemented a combination of classical and deep learning-based feature extraction techniques. Our classical methods included Histogram of Oriented Gradients (HOG), HSV color histograms, and Local Binary Patterns (LBP), each offering a unique perspective on shape, color, and texture. To complement these, we also used a convolutional neural network — specifically ResNet50 pretrained on ImageNet — as a feature extractor. By removing the final classification layer and collecting the output from the last average pooling layer, we were able to generate compact feature vectors that captured higher-level visual patterns. We evaluated each feature type individually and in combination to understand their strengths and how they contribute to flag classification performance.

A. Kumar, A. Zutshi, M. Shull, S. Andrews
April 18th, 2025
DATASCI 281

Histogram of Oriented Gradients (HOG)

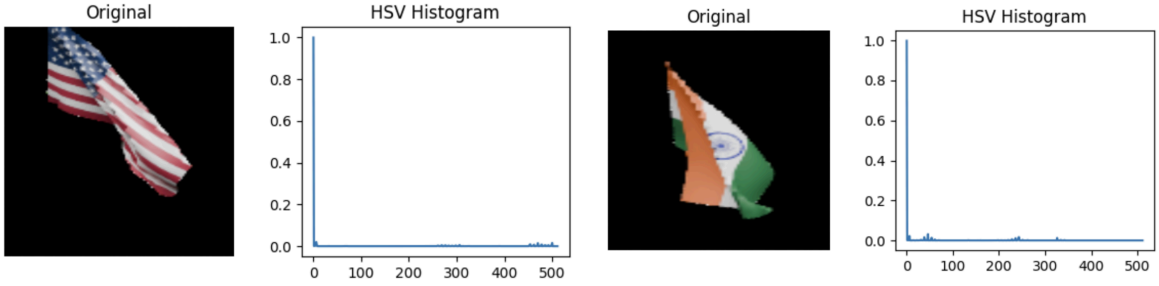


HOG captures the overall shape and outlines in an image by focusing on edges and directions of gradients. It worked well in our project because many flags have strong, recognizable shapes like the maple leaf on the Canadian flag or the wheel on the Indian flag. HOG features led to strong results for the synthetic set, with AUC scores between 0.88 and 0.99 across all flag categories in our training set, showing it could reliably pick up on those large shapes.

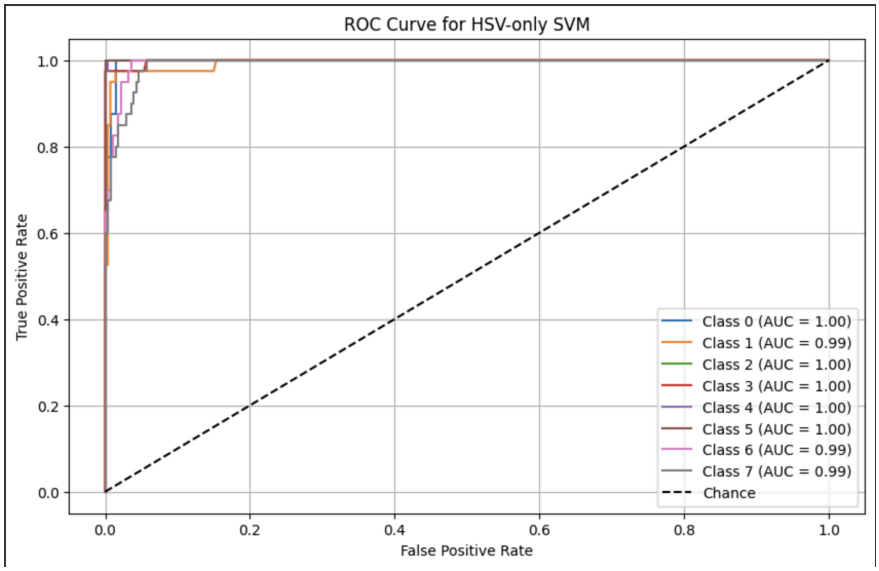


As we can see in this image, the ROC scores for the HOG feature are fairly consistent across the 8 classes with high performance across the board. This indicates that the model was successful in differentiating between the large shapes present in the flags. The 2 highest performing classes: Class 0 and Class 4 represent India and Boogaloo, respectively, which makes sense as the Indian flag has a distinct wheel in the center of their flag and Boogaloo has an igloo and tree in their flag, obviously very unique.

Hue, Saturation, Value (HSV) Histograms



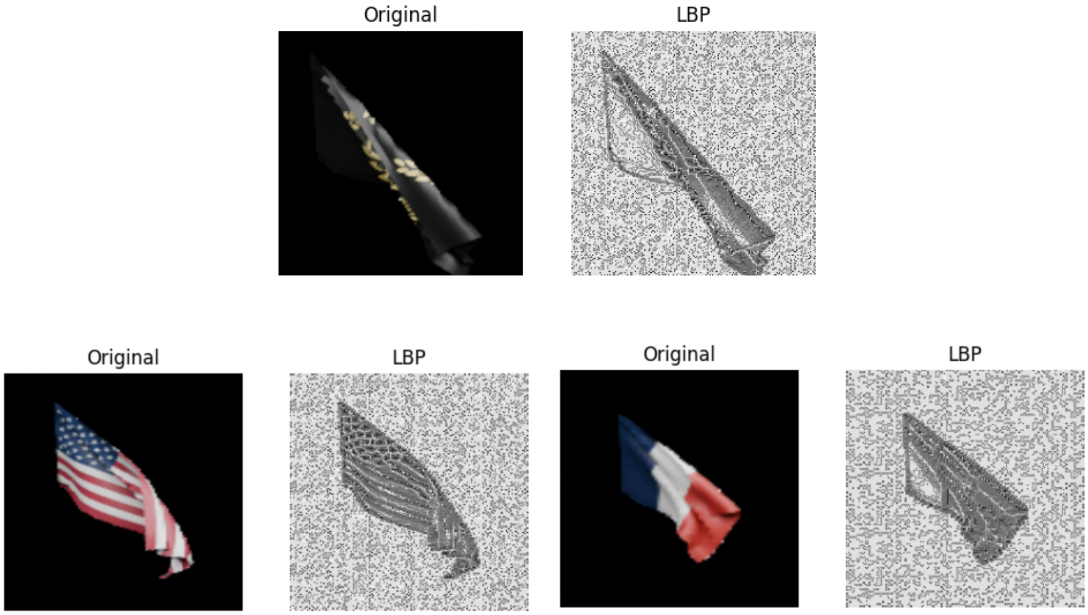
HSV histograms focus on the colors in an image. Since our synthetic training images were rendered with no background and had clean lighting, HSV performed extremely well for the synthetic set. All eight flag categories had AUC scores above 0.99 on the synthetic set. However, this also made the model rely too much on color. To help prevent overfitting and improve performance on real-world test data (which includes more significant lighting and background noise), we added regularization during training.



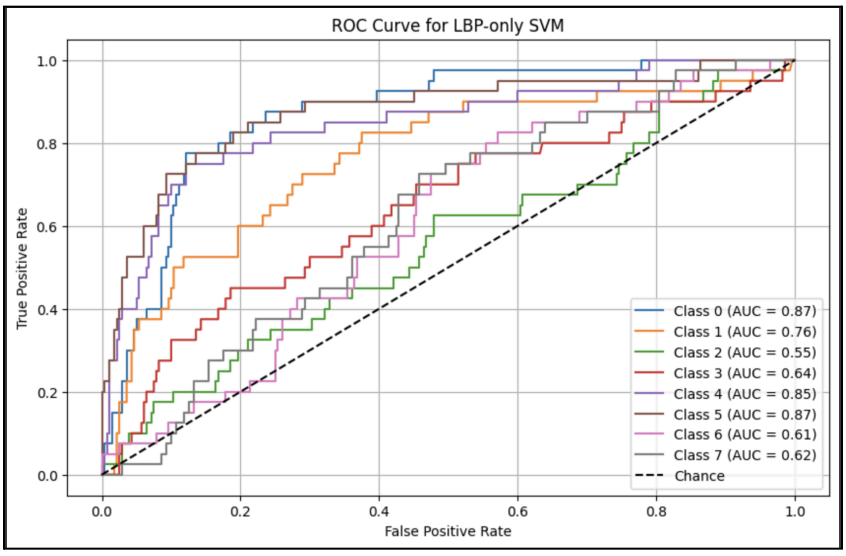
We can see that the performance of the HSV feature is consistent and extremely high across all features (> 0.99 for all features). This is due to the fact that the synthetic train set contains flags

with minimal lighting variance and no background noise, allowing for HSV to distinguish the color of the flags extremely well.

Local Binary Patterns (LBP)



LBP captures texture by comparing small local patterns in an image, which we thought might help distinguish flags. However, it didn't perform as well as HOG or HSV, likely because our synthetic flags were rendered with relatively smooth surfaces and had much less lighting variation than real-world images; these conditions limit the texture-based information LBP depends on.

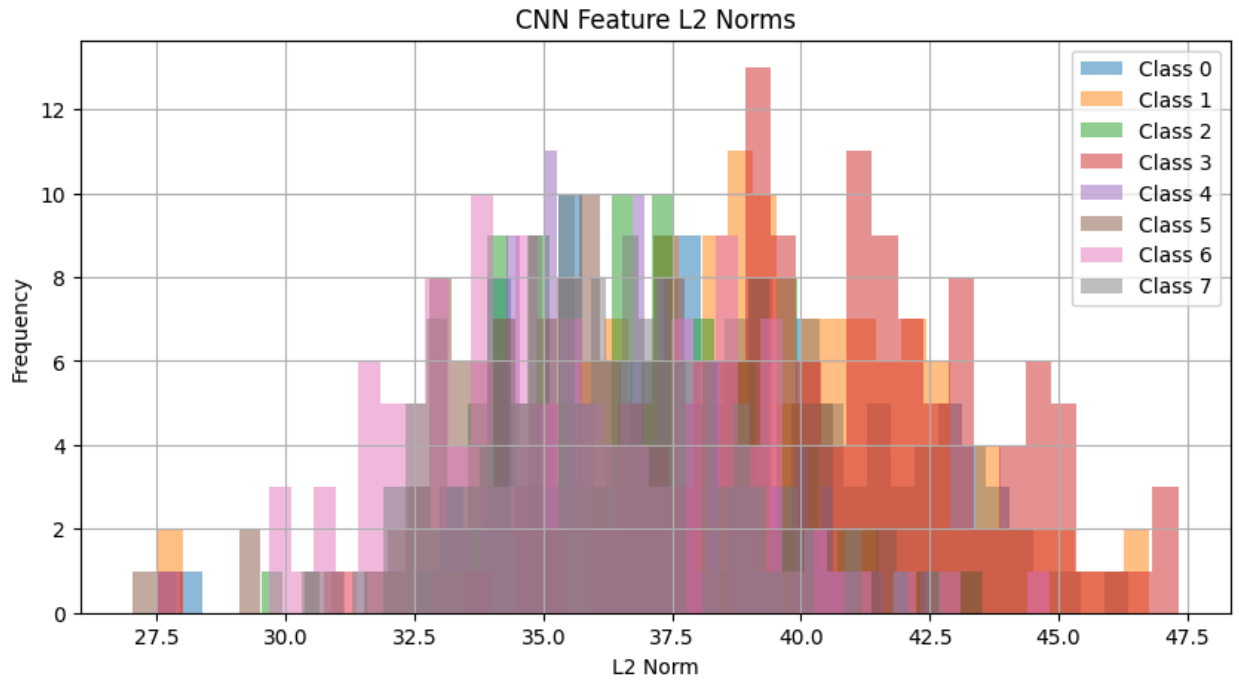


We can see in the image above that LBP is our worst performing basic feature compared to HSV and HOG. While the performance of all of the classes is still > 0.5 indicating the feature is valuable in predicting classes, this relatively poor performance is likely due to the lack of texture and distinct fine grained objects present in the synthetic train set.

Deep Features from CNN (ResNet50)

To complement our traditional image features like HOG and HSV histograms, we also extracted deep features using a convolutional neural network. We specifically used ResNet50 pretrained on ImageNet, with the top classification layer removed. This lets us use the model as a feature extractor rather than a full classifier.

We passed each image through the network, then collected the output of the final average pooling layer. This gave us a compact feature vector that captures high-level visual patterns that a CNN trained on natural images can recognize, such as shapes, textures, and spatial relationships. This feature vector was then combined with our other extracted features. Using CNN features gave us a more expressive representation of each flag, which we believe would improve performance, especially when the flags were distorted or partially occluded.



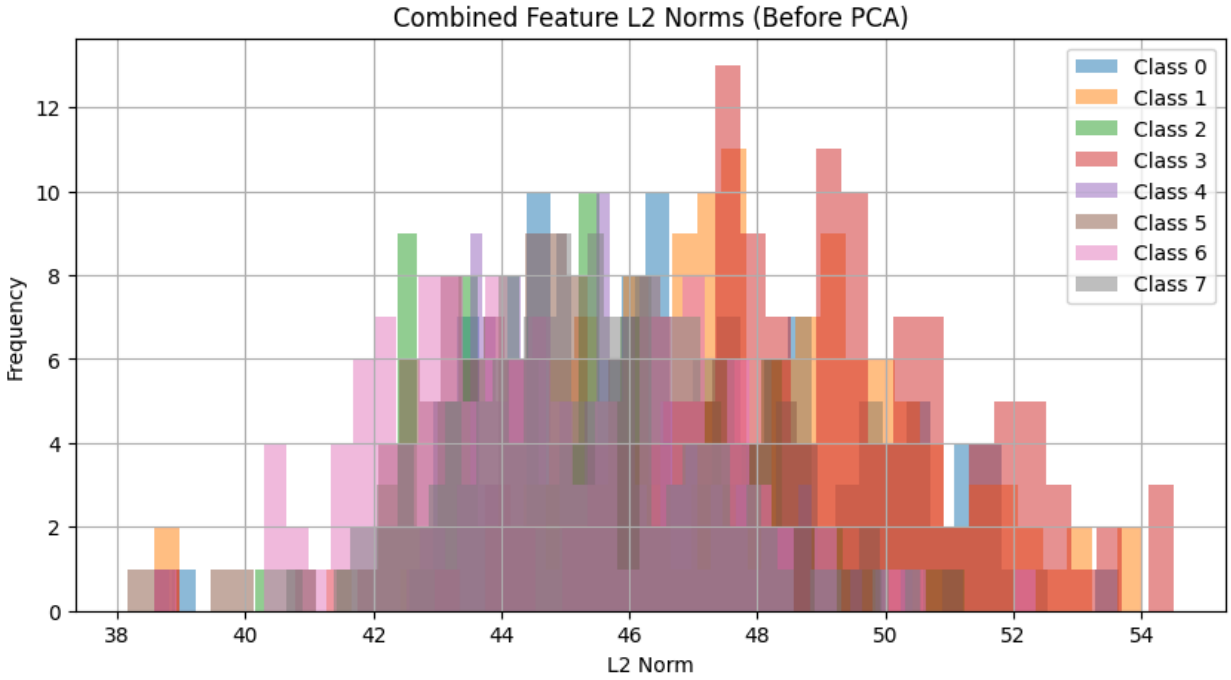
The L2 norms distribution for ResNet50 feature vectors shows some variation across classes, with some classes like class 3 tending to have higher norm values. However, the significant overlap between classes suggests that CNN norm differences are not enough to clearly separate the classes.

A. Kumar, A. Zutshi, M. Shull, S. Andrews
April 18th, 2025
DATASCI 281

Modeling and Classification Approach

Combining Extracted Features

We combined several types of features into a single vector for each image to capture both low-level visual details and high-level patterns. This included HOG for shape, HSV histograms for color, LBP for texture, and deep features from a pretrained ResNet50 model to represent more complex visual structures.



The L2 norm distribution of the combined handcrafted features shows some variation between classes, with classes like 3 and 5 having slightly higher values. But there's still a lot of overlap, which means the size of the feature vectors alone doesn't help much in telling the classes apart.

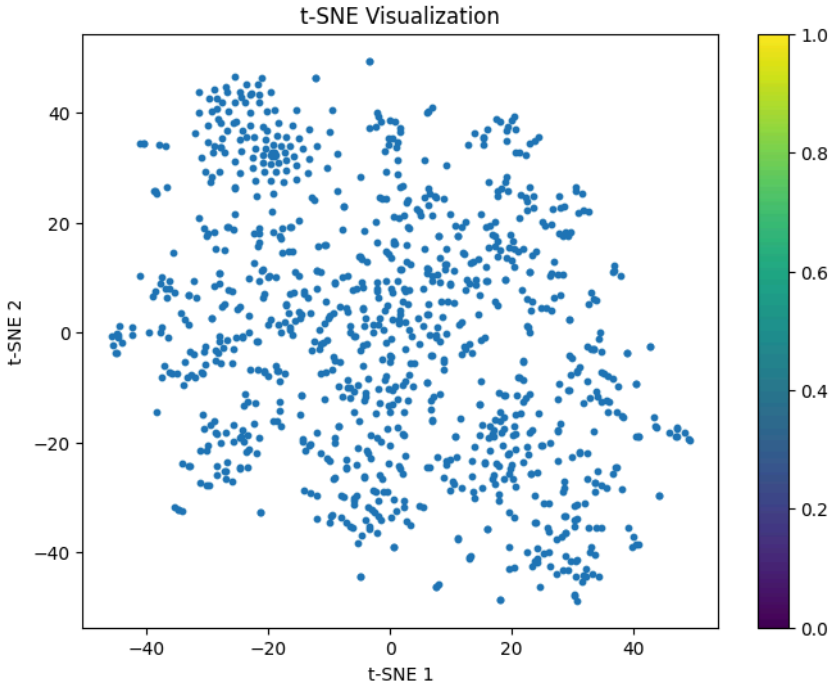
PCA Dimensionality Reduction

We used Principal Component Analysis (PCA) to reduce the dimensionality of our combined feature vectors by removing redundant or less informative components. PCA was fit on the synthetic training set, and we selected the number of components that preserved 95% of the variance. This helped improve training efficiency and reduced the risk of overfitting. The dataset after combining extracted feature data went from 28814 columns to 758 columns. Multiple variance thresholds (80%, 85%, 90%, 95%) were tested to determine whether there was a significant effect on the final test accuracy of the real-flags dataset but the change to the team's metrics was minimal.

A. Kumar, A. Zutshi, M. Shull, S. Andrews
April 18th, 2025
DATASCI 281

t-Distributed Stochastic Neighbor Embedding

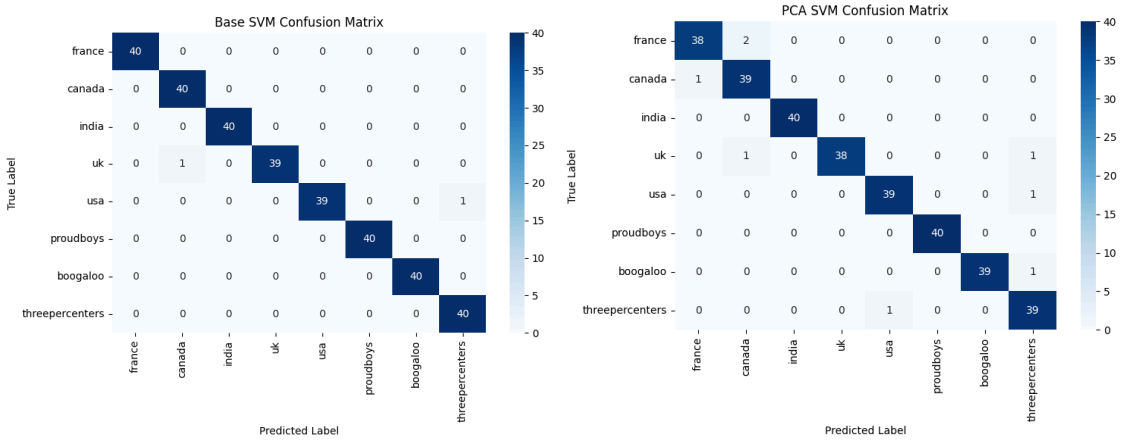
The team used T-Distributed Stochastic Neighbor Embedding (t-SNE) to visualize the distribution of our data points after dimensional reduction (28814 columns to 2 columns for each of the 1040 rows in the training set, which resulted in 1040 points). Two parameters were changed to determine whether the scatterplot visualization would offer more insights in the visualization, learning rate and perplexity. The learning rate was changed lower after each iteration of running the t-SNE dimensionality reduction to find a value that would provide low computing time when changes were not apparent to the visualization. The perplexity was both increased and decreased until groups were made apparent. Unfortunately, there is only one group that we can glean from the visualization, which can be found in the top left corner. One possible explanation for this lack of structure (besides the one group) is that many of the training flags have a lot of noise that may not be needed. This finding encouraged the team to use the PCA dimension reduced dataset for the rest of the project.



SVM-based Models

We trained a base Support Vector Machine (SVM) model using the full set of extracted features (HSV, HOG, and LBP) without any dimensionality reduction. This model achieved excellent performance with 99% accuracy and a 0.99 F1 score on the train set, effectively handling the synthetic flag variations. However, it was computationally expensive, taking over 20 seconds for training the model, likely due to the high dimensionality of the combined feature set. To improve efficiency, we applied Principal Component Analysis (PCA) prior to SVM training, reducing the feature space while retaining 95% of the variance. The PCA-reduced SVM model showed a

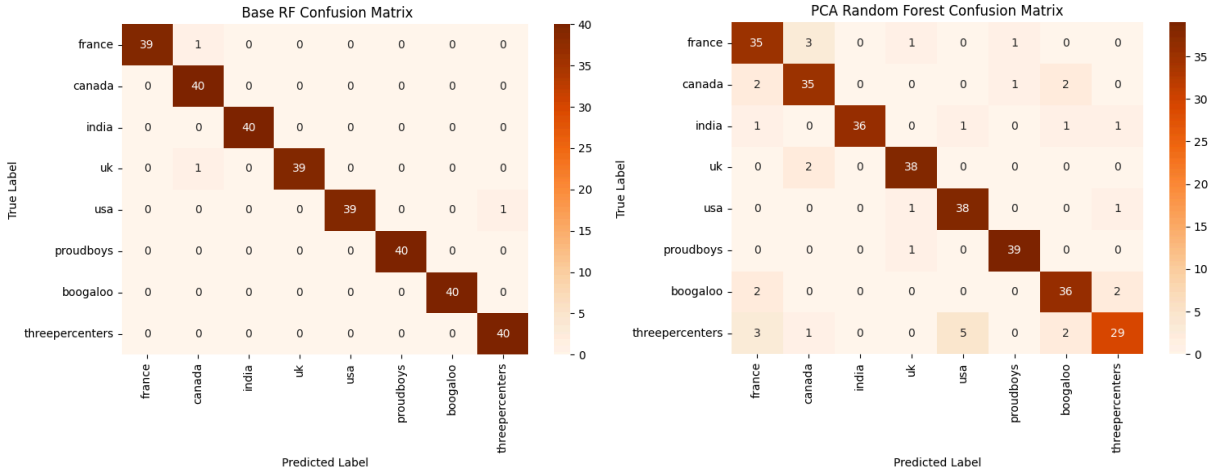
slight drop in accuracy (97%) and F1 score (0.98), but significantly improved inference speed (0.53 seconds). This trade-off highlights the benefit of dimensionality reduction in terms of computational efficiency with only minimal impact on performance, making the PCA-SVM model more practical for real-time or resource-constrained scenarios. The figures below illustrate the minor drop in accuracy between the two models for the tradeoff of having higher computational efficiency.



Random Forest Models

We trained a Random Forest classifier using the same set of features (HSV, HOG, and LBP). Using 100 trees and default settings, the model reached 99% accuracy, with precision, recall, and F1 score all at 0.99. It worked well across all classes, with no clear signs of overfitting and strong generalization to new data. Like the SVM, the confusion matrix showed that predictions were balanced across categories. Because of its strong performance and easy-to-understand structure, Random Forest served as a solid and reliable starting point for this task. However, unlike the SVM model, our model suffered in accuracy when the PCA dataset was used instead. This finding can be seen in the two confusion matrix figures below.

A. Kumar, A. Zutshi, M. Shull, S. Andrews
April 18th, 2025
DATASCI 281



Hyperparameter Tuning

We chose to run hyperparameter tuning on both the SVM and Random Forest models by implementing grid search and tuning the hyperparameters on our validation sets. We specifically ran hyperparameter tuning for both models after PCA was conducted to reduce the feature set and for the SVM model we ran grid search on the following parameters: C values of 0.05, 0.1, and 1; kernel options of linear, rbf, and polynomial; and gamma options between auto and scale.

Our best tuned SVM model had the following hyperparameters: C of 0.05, kernel was linear, and gamma was scale. This led to our tuned PCA SVM model to score 0.97 for accuracy and f1-score on the validation set and 0.97, 0.98 for test accuracy and f1-score, respectively. This represents a very small dropoff of ~ 0.01 compared to the non-PCA model, indicating that the PCA model is more efficient while maintaining a high level performance.

Regarding the Random Forest model, we implemented grid search with the following parameters: n_estimators of 200 and 300, max_depth of 20 and 30, min_samples_split of 2 and 5, min_samples_leaf of 1 and 2, and bootstrap of True or False. Our best tuned Random Forest model had the following hyperparameters: n_estimators of 300, max_depth of 20, min_samples_split of 5, min_samples_leaf of 1, max_depth of 20, and bootstrap of False. This led to our tuned PCA Random Forest model to have scores of 0.91 for accuracy and f1-score on the validation set and 0.92 for accuracy and f1-score on the test set. This is a larger dropoff from the non-PCA Random Forest model when compared to the dropoff for the SVM model, indicating that there is a tradeoff between efficiency and accuracy for the Random Forest model when implementing PCA.

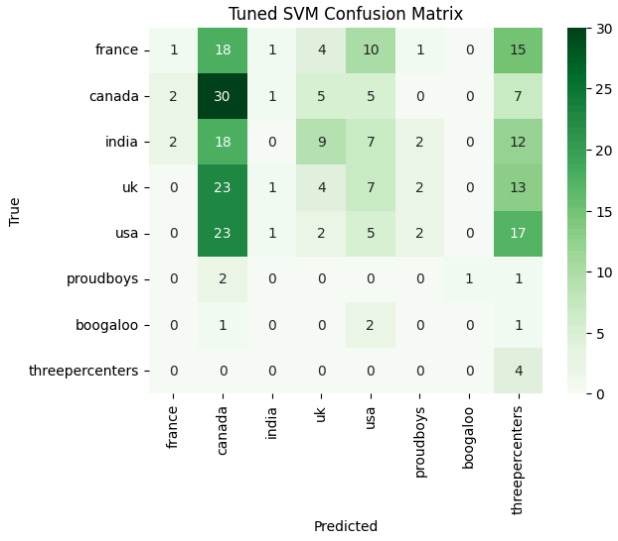
A. Kumar, A. Zutshi, M. Shull, S. Andrews
April 18th, 2025
DATASCI 281

Results and Evaluation on Synthetic and Real Data

Best Model Performance

The hyperparameter-tuned SVM model trained on PCA data achieved high accuracy on the synthetic set but performed poorly on the real-world set, with only 17% accuracy and an F1 score of 0.12. The confusion matrix shows significant misclassification across classes, likely due to challenging conditions in the test images, such as shadows, orientations, and colorful backgrounds that manual cropping could not remove. These results highlight the model's difficulty in generalizing to more realistic, less controlled scenarios. Nevertheless, with 8 classes, the model performs better than random chance, which gives some proof-of-concept that models trained on synthetic data have potential to predict on real-world data.

```
Tuned SVM Test Data Accuracy: 0.17
Tuned SVM Test Data F1 Score: 0.12
```



Test Predictions vs True Labels (SVM vs RF)

True: boogaloo (5)	True: usa (2)	True: uk (1)	True: uk (1)	True: canada (3)	True: threepercenters (6)	True: india (0)	True: india (0)
SVM Pred: uk (1)	SVM Pred: uk (1)	SVM Pred: uk (1)	SVM Pred: uk (1)	SVM Pred: uk (1)	SVM Pred: uk (1)	SVM Pred: proudboys (7)	SVM Pred: france (4)
RF Pred: canada (3)	RF Pred: uk (1)	RF Pred: usa (2)	RF Pred: uk (1)	RF Pred: uk (1)	RF Pred: uk (1)	RF Pred: proudboys (7)	RF Pred: uk (1)

A. Kumar, A. Zutshi, M. Shull, S. Andrews
April 18th, 2025
DATASCI 281

Efficiency vs Accuracy

We first implemented both Random Forest and SVM models that both took in a combination of 3 feature vectors: HOG, HSV, and LBP, as well as deep features extracted from a convolutional neural network. These models ended up taking 20.02 and 7.23 seconds for SVM and Random Forest respectively for the base model, and we chose to not run hyperparameter tuning for these models as Google Colab ended up timing out when trying to run grid search on these extensive feature sets in both models. Our base model's performance for both Random Forest and SVM were very high on the synthetic training set as detailed above, however, there was a substantial drop-off in performance on the real-world test set.

When discussing the tradeoff between efficiency and accuracy in the context of this problem, the main efficiency boost came from our PCA implementation for both our Random Forest and SVM model that reduced the # of features in our model while still maintaining 95% of the variance present in the data. We were able to reduce the number of features from 28814 to 758 through this process which enabled us to be able to run grid search on the validation set to tune the hyperparameters for our model. This also allowed us to introduce regularization components to our models to address the overfitting of our model to the synthetic train set and make it more robust and improve performance on the test set. When running grid search for the PCA SVM and Random Forest models, it took us 43.70 seconds, and 818 seconds, respectively to tune both models. We decided that due to the minimal drop-off in performance on the synthetic training set when comparing the base SVM and Random Forest models to the PCA SVM and Random Forest models, coupled with the substantial speed improvements ~40x for SVM and ~2.5x for Random Forest, led us to utilize hyperparameter tuned PCA models as our final models for evaluation on the test real-world dataset.

Generalizability

As mentioned previously, our training set contained purely synthetic flags generated in Blender while our test set consisted of real-world images that were manually cropped to capture the entirety of the flag while minimizing the amount of background noise. When training our model on the synthetic training dataset, there were a couple of factors that led to the near perfect performance and likely overfitting of the model.

Firstly, the synthetic images contained only the flag with no background which allowed for certain features like HSV to be extremely accurate in pinpointing the color of the flag and be able to easily distinguish the different classes. This is not realistic in the real world images as while those images contain the colors of the flags, they often contain different colors present in the background that could mislead our model when evaluating on the test set. One example is a lot of Canadian flags in our test had blue backgrounds which could help explain why the model often confused the Canadian flags for USA or UK flags which contain the red and white of the Canadian flag as well as the blue of the background color.

Secondly, while we attempted to introduce distortions and lighting variability into our synthetic training set, we were unable to capture the sheer variety of lighting, orientations, zoom, texture, etc. that is present in the test set. This could potentially cause the LBP feature to be less effective as it is unable to train on the variety of the real-world images when we train the model on the synthetic train set. We attempted to mitigate this through a number of methods.

We cleaned the test set by running a python script that allowed us to crop the images to isolate each flag and minimize the background. We experimented with several different methods of isolating these flags in the test set, one method where you locate the four corners of the flag, similar to the process used to isolate license plates. The main issue with this method is that flags have many distortions due to hanging off a pole, rippling in the wind, etc. that are not present in license plates that make it difficult to determine the four corners of the flag and can distort the flag further if transformed into a square image. Ultimately, this warrants further research and was out of scope of this project, but would be very interesting to dive deeper to determine the best method to isolate these flags.

We also used PCA to reduce the number of features, therefore reducing the model complexity and addressing some of the overfitting to the training data that was occurring. We then introduced regularization terms in the hyperparameter tuning of these models after running PCA that mitigated the effect of certain features when applying the model to the test set in an effort to build a more robust model that performed better on the test set.

Discussion

Insights on Generalization from Synthetic Data

Our results suggest that synthetic data can support real-world classification tasks, but the model's ability to generalize depends heavily on the diversity and quality of the synthetic dataset. We achieved high performance when evaluating our model on the synthetic test set, indicating that the model learned useful visual patterns from the generated data. However, performance dropped significantly when classifying real-world test images. This highlighted the gap between controlled, synthetic training images and more complex, noisy real-world conditions. Still, the successful use of classical computer vision features demonstrates a promising proof of concept for using synthetic data in low-resource classification tasks.

Challenges and Limitations

As mentioned in the results section above, one of the core challenges we faced was bridging the gap between clean, synthetic training images and the complexity of noisy real-world images. While our synthetic flags were rendered against transparent backgrounds with varied though controlled lighting and perspective, the real-world images often had more visual clutter,

occlusion, blur, and general variability. We cropped flags in real-world images to isolate them, but in many cases, background content remained and may have confused our models. Moreover, the synthetic training set, though diverse, did not fully replicate the variation in lighting, texture, and deformation found in real photos. These factors likely contributed to the models' difficulty generalizing from synthetic data to real test images.

Future Work

There are many opportunities to strengthen and extend this project. Improving segmentation of the real-world test images could help reduce background noise and improve classifier accuracy. We looked into using the Segment Anything Model and fine-tuning a YOLO model with flags from OpenImages but ultimately went with a simpler cropping method for this iteration of the project. Introducing greater variation into the synthetic data used for training (more lighting conditions and orientations, for example) could also make the models more robust to real-world conditions. We could also try more hyperparameter options during the grid search tuning portion to further optimize our models for performance given more compute. Future iterations could also explore more modern computer vision techniques, such as using transformer-based models, fine-tuning convolutional networks, or incorporating contextual information to better capture subtle patterns and improve generalization to challenging test cases.