Urban Land Use Classification from Public Domain Imagery*

Mahesh Arumugam, Mackenzie Austin, Ahad Momin

281 Final Project

Summer 2023

Abstract

In this project, we investigate the urban land use dataset from UC Merced that consists of 21 land use categories from various urban areas across the United States. We explore various features of the images, starting from simple ones such as mean color values of the color channels and histogram of oriented gradients, to complex features such as bag of visual words, contours of an image, and embeddings from pre-trained deep neural network models (e.g., VGG16 and ResNet101). We perform principal component analysis to reduce the dimensionality of our dataset for better generalizability. We identify features suitable for classifying the images into their respective classes through experiments with various classification models. Finally, we compare and contrast these classification models for efficiency (with respect to the time required for training and prediction) and model accuracy.

1 Introduction

Our project focuses on classifying the UC Merced land use dataset [1] that is manually extracted from USGS national map urban imagery collection. This dataset consists of 2100 manually labeled images across 21 land use classes with 100 images in each class, each with dimensions of 256x256 and a pixel resolution of 1 foot. Figure 1 shows few land use images with their corresponding labels.

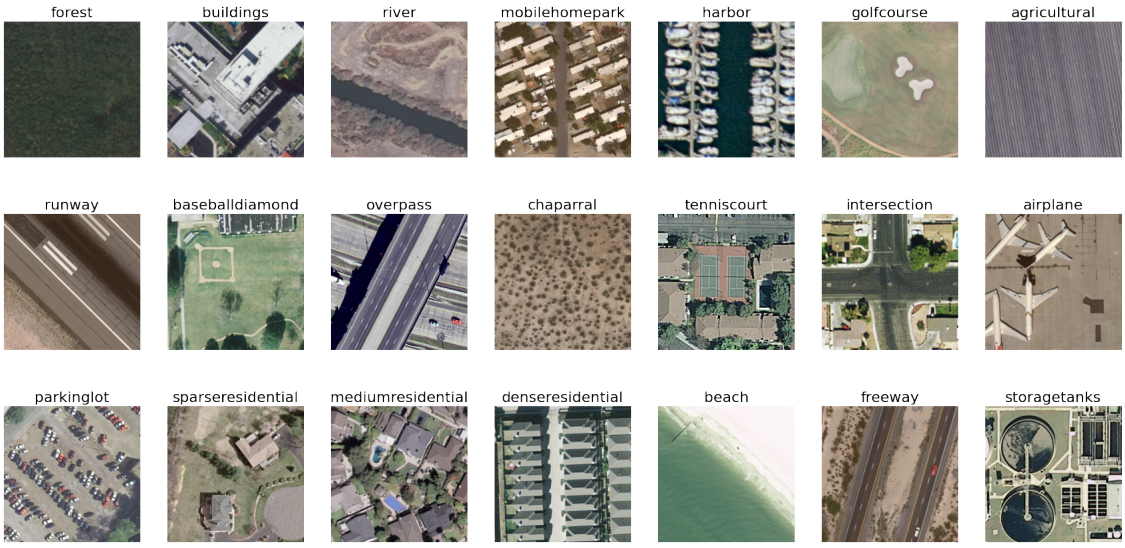


Figure 1: Example image from each category

*We refer the reader to the notebook for code and analysis at: https://bit.ly/445t0dP. For an extended version of this notebook, we refer the reader to: https://bit.ly/3OpSJYy. The GitHub repository of our project is at: https://github.com/aumahesh-mids/281-MA/tree/main.

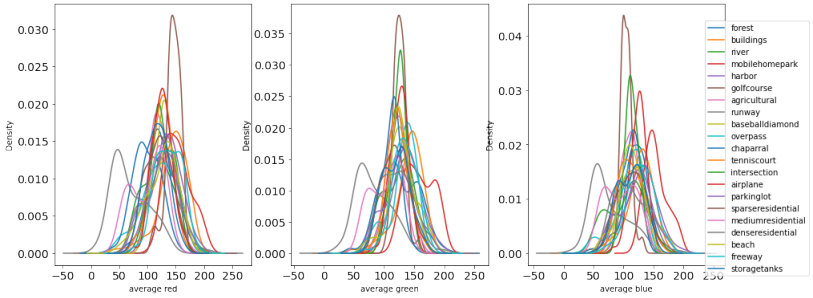
We employ feature extraction techniques such as HSV and RGB color analysis, Histogram of Oriented Gradients (HOG), and Gray-Level Co-occurrence Matrix (GLCM). In addition, we also incorporate advanced feature extraction approaches such as Bag of Visual Words (BOVW), counting contours of an image, and embeddings from pre-trained deep learning model (e.g., VGG16 and ResNet101). For classification, we utilize Logistic Regression, Random Forest, XGBoost, and SVM models. In our initial approach, we achieve an accuracy of approximately 60% without ResNet101 embedding features. However, by integrating pre-trained ResNet101 embeddings, accuracy significantly improves to more than 95%. After testing different combinations of features, the key features for optimal results include Bag of Visual Words, Mean color (RGB & HSV), and ResNet embeddings. To enhance efficiency and generalization without sacrificing accuracy, we perform PCA transformations. This reduces dimensionality while preserving essential data characteristics. In this paper, we present our methodological approach, experimental findings, and the potential for accurate urban land use classification from satellite imagery.

2 Feature Extraction

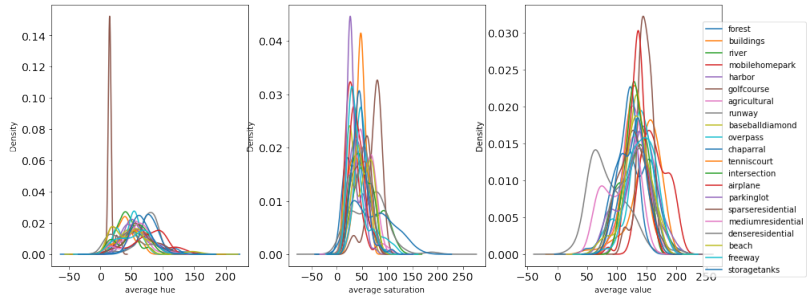
In this section, we discuss the features we considered for inclusion in our classification. We discuss the motivation and the findings for feature selection.

2.1 Average Pixel Color

Due to the nature of the images, we assumed average values in RGB channels would be a useful feature to explore first. Because some of our images are mostly green, where others are mostly blue or tan, color could help to distinguish classifications into smaller categories. The same would hold true for HSV means, differentiating primarily via the hue channel, but the saturation and value channels could also prove useful.



(a) Mean RGB



(b) Mean HSV

Figure 2: Distribution of mean color values of the images

From the distributions shown in Figures 2(a) and 2(b), we can see, surprisingly, similar distributions for most categories except some outliers like the peak in the red channel for sparse residential category, or the peak in the hue channel for golf course category.

2.2 Histogram of Oriented Gradients (HOG)

HOG extracts edges oriented in different directions for segments of the image and concatenates them together. The size of the segments can be specified by the user in pixels. HOG helps with object detection and proved useful for differentiating certain classifications like airplane, storage tanks, beach, and river. We explored HOG for both grayscale and for each RGB channel.

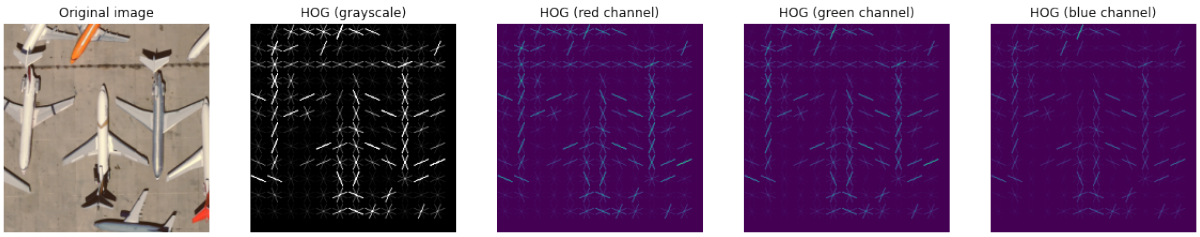


Figure 3: HOG features for one example image with (20, 20) pixels per cell and (2, 2) cells per block.

Figure 3 shows the HOG representation of a sample airplane image. The shape of the airplane is clearly preserved in the HOG representation in grayscale and in color channels. The intensity of the gradients for blue channel are not as visible as red channel or grayscale. We explore HOG due to this observed differentiation.

2.3 Gray-Level Co-occurrence Matrix (GLCM)

GLCM [2] considers pairs of pixels in varying orientations to one another specified by the user. We choose these orientations in $0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}$ radians. GLCM calculates how often pairs of pixels with specific values and in specific orientations occur in an image. This is then used to determine information about contrast, dissimilarity, homogeneity, and correlation. Because our images have a tile-able nature, similar to images of textures, GLCM was a natural choice for a feature.

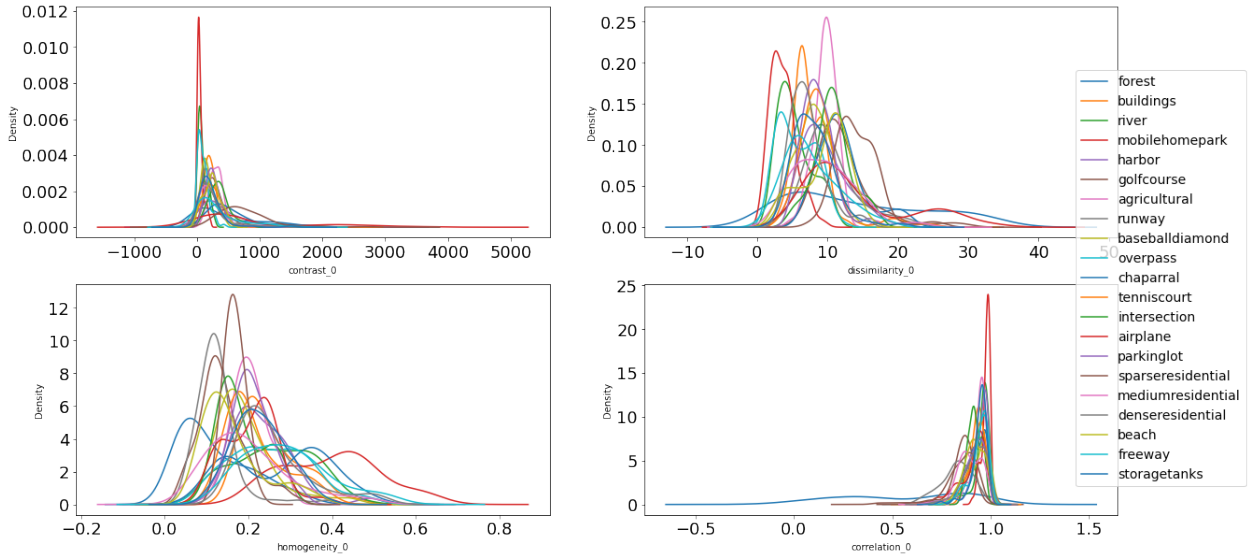


Figure 4: Distribution of contrast, dissimilarity, homogeneity, and correlation at distance 1 with orientation=0

Figure 4 shows the distributions of contrast, dissimilarity, homogeneity, and correlation for the 0 radians. We see that the distribution shows some distinguishable classifications particularly in homogeneity.

2.4 Object Count

Object count [3] is a function which looks at the contours in a given image and counts them. This provides an estimate for the number of objects in an image. After some classification tests with only HOG (cf. Section 2.2) and GLCM (cf. Section 2.3) features, we noticed frequent misclassifications of buildings as dense residential. We employ object count as a way to remedy this misclassification, since we expect there would be more objects in dense residential than in buildings.

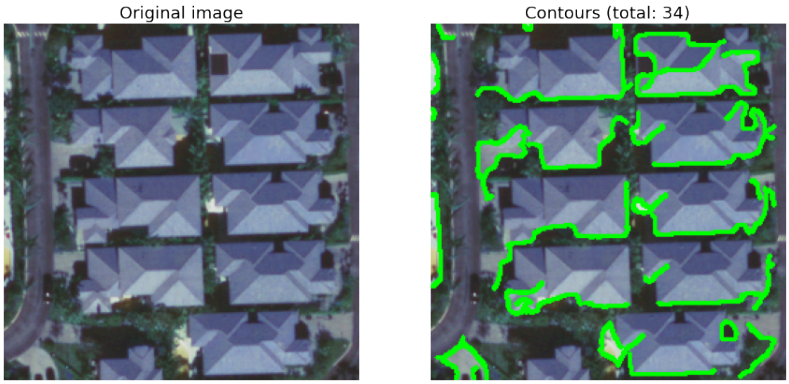


Figure 5: Counting contours of an image

Figure 5 shows an example of object count feature. And, Figure 6 shows the distribution of object count feature across the categories in our dataset. We note that there are some distinguishable classifications. While most objects have very few contours, some have very high contours (in the order of $100 - 400$) that might help in distinguishing those categories better.

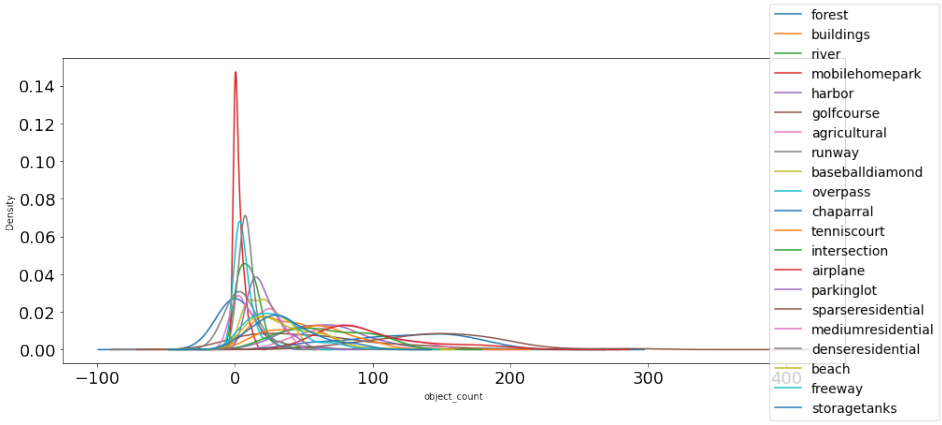


Figure 6: Distribution of object counts in the images.

2.5 Bag of Visual Words (BOVW)

BOVW [4] is a complex feature with many parts. This is similar to bag of words process in natural language processing domain. To compute BOVW, we proceed as follows.

1. We extract the keypoints and descriptors using ORB [5], a very fast binary descriptor and keypoints detector that is noise tolerant. Keypoints are points in an image which do not change when the image is rotated, expanded or scaled. Descriptors are vector representations of an image patch found at a keypoint.

2. Keypoints and descriptors are paired to create a visual word. Similar keypoints are grouped together using K-means; these clusters are then added to a codebook. The codebook functions as a dictionary or vocabulary for an image. This helps to represent an image as a finite set of visual words.

3. We count the frequency of these visual words to create a histogram.

4. Finally, TF-IDF [6] is performed to reduce the weight of common visual features, and increase or maintain the weight of uncommon or unique visual features.

Figure 7 shows an example image along with the keypoints discovered by ORB. And, Figure 8 shows the distribution of the first 4 visual words across the categories in our dataset. For most categories, the densities of these visual words are near zero, while there are some categories with a small density (greater than 0). These visual words help in distinguishing the category.

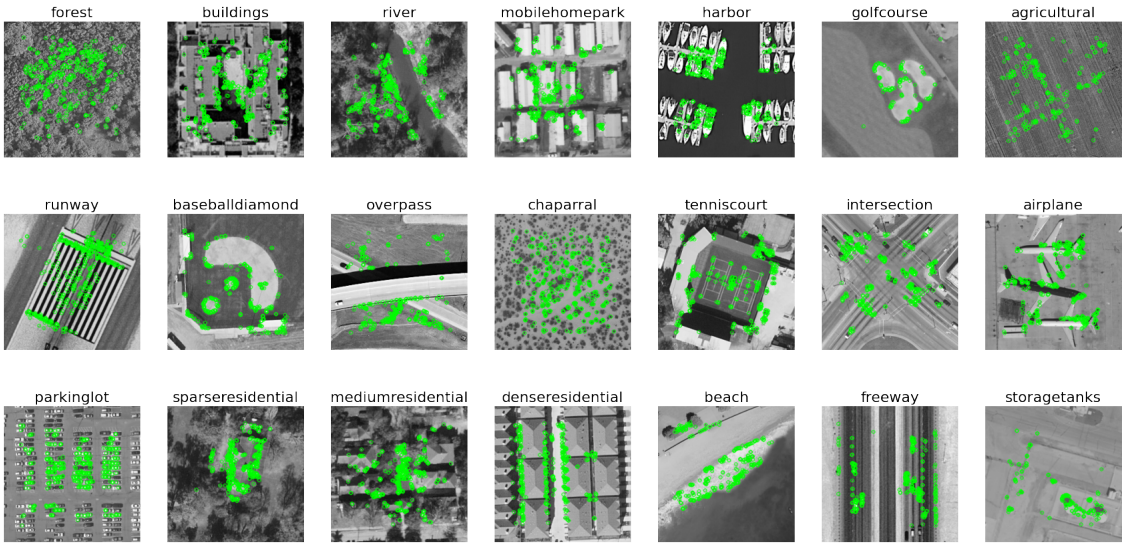


Figure 7: Keypoints identified by OpenCV ORB method

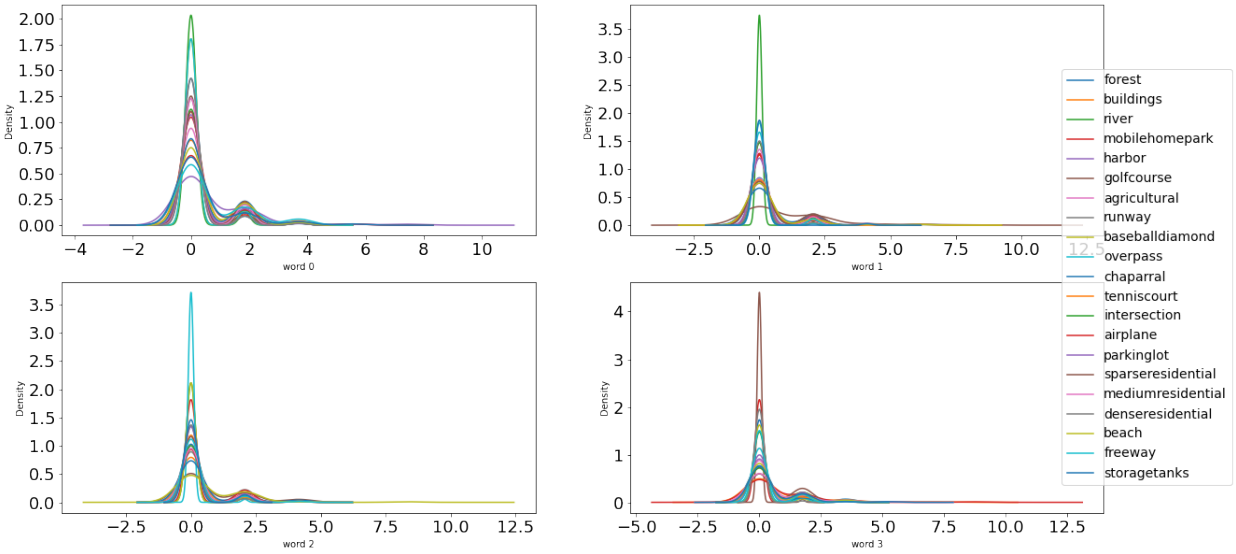


Figure 8: Distribution of the first four words

2.6 Feature Embeddings from Pre-trained Neural Network Models

We considered embeddings from two pre-trained deep neural network architectures for our feature extraction.

VGG16. Visual Geometry Group 16 (VGG16) [7] is a convolutional neural network (CNN) architecture. It consists of 16 layers, including 13 convolutional layers and 3 fully connected layers. VGG16 is known for its simplicity and uniformity, which led to its widespread use and understanding. This process helps in extracting hierarchical feature representations at different levels of abstraction. The features obtained from the convolutional layers are flattened and passed through fully connected layers to make predictions. VGG16 has shown impressive accuracy in different classification challenges, making it a convenient choice for us to generate embedding features for our dataset.

ResNet101. Residual Network 101 (ResNet101) [8] is another CNN architecture that addresses the issues of vanishing gradients encountered in deeper networks. It introduces skip connections or residual blocks, allowing the network to learn residual mappings that are then added to the output of previous layers. This enables the network to learn and build more accurate and deeper representations. ResNet101 employs a similar process as VGG16. ResNet101 has demonstrated superior performance on various image classification benchmarks, making it a convenient choice for us to generate embedding features for our dataset.

When using these models for our classification tasks, we use the embeddings from these networks. These embeddings capture high-level features of the input image, which we then use as input to our classifiers. Using embeddings from pre-trained models showed increased accuracy in our classification tasks.

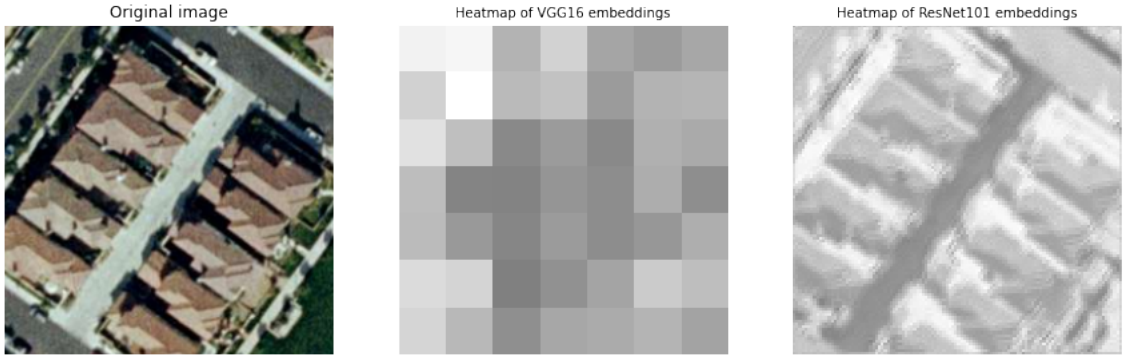


Figure 9: Heatmap of VGG16 and ResNet101 embeddings for one image

Figure 9 shows the heatmap of a visual representation of the embedded features on the image when passed through ResNet101 and VGG16 layers, which is a lower-dimensional representation of the image that captures its important attributes.

2.7 Principal Component Analysis (PCA)

PCA is an approach to reduce the dimensionality of a dataset. The primary purpose of this process is twofold: (1) to mitigate overfitting and (2) enhance computational efficiency. By linearly transforming the data through the correlation matrix, PCA projects it onto a new coordinate system, while ensuring that the maximum amount of variation is encapsulated within few components. In our analysis, we employed PCA to transform all the aforementioned features. This approach allowed us to create a visual representation of the fraction of total variance explained versus the number of principal components. Figure 10 illustrates the relationship between the total variance and the total number of principal components.

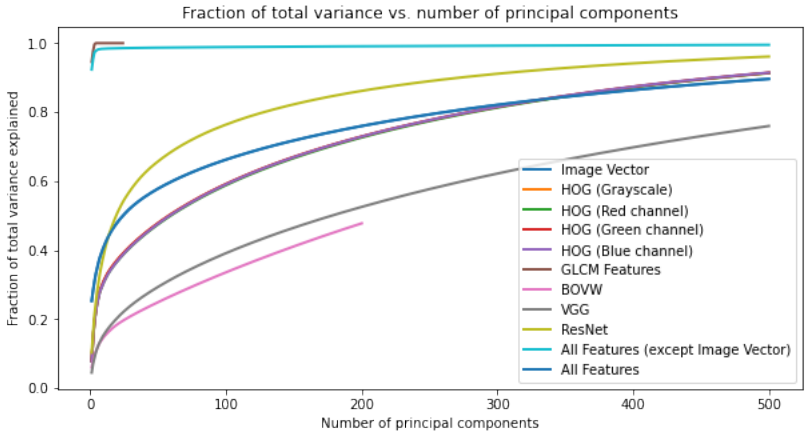


Figure 10: Explained variance plot of the principal components from various feature vectors

Amongst all the individual features with 100 or more dimensions, PCA on the ResNet101 embedding feature conserved the highest variance with the first few components. Additionally, when all features were combined, the total number of components experienced a significant reduction. To address the issue of overfitting stemming from the inclusion of all features, we made the decision to incorporate only BOVW, Mean Color (RGB, HSV), and ResNet101 features for training. As shown in Figure 11, this selection resulted in an impressive 95% variance explained by just 20 components, which we decided to keep for training models.

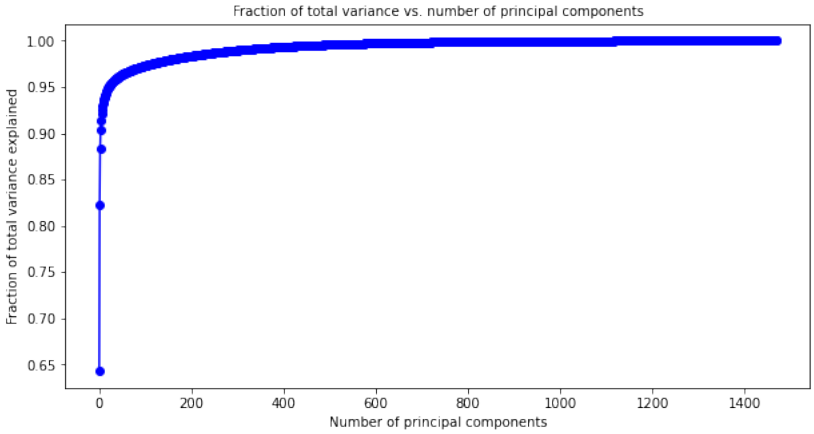
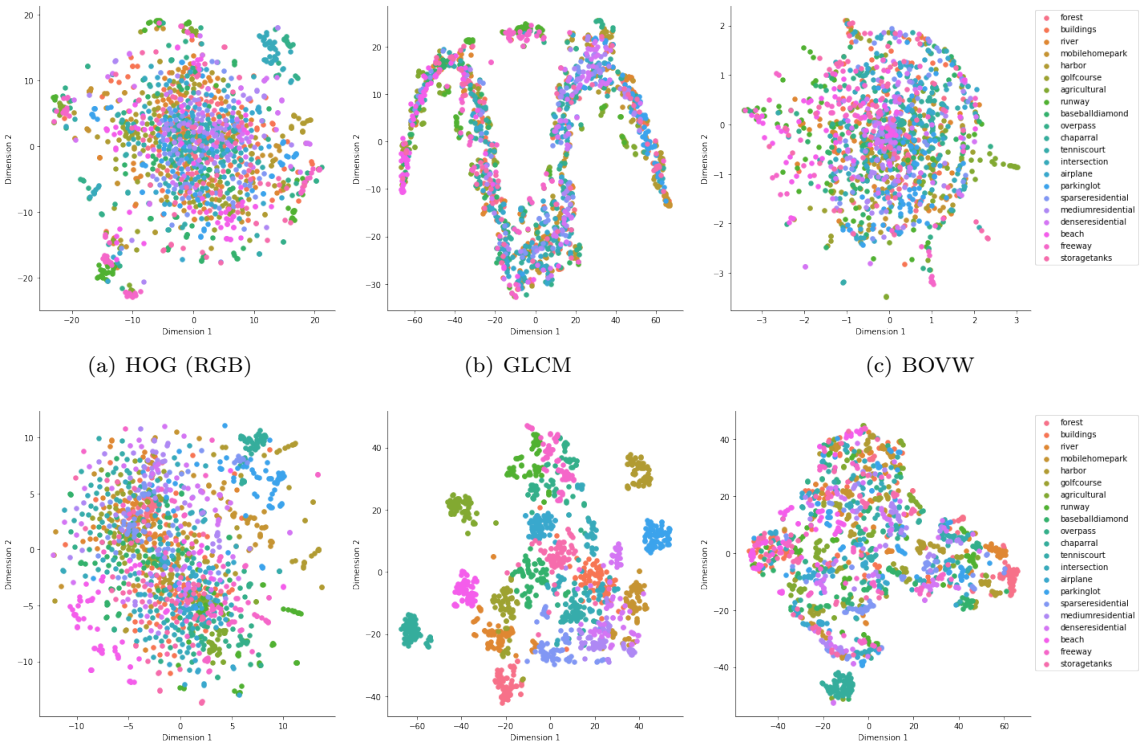


Figure 11: Explained variance of the principal components of the dataset used for classification

2.8 T-distributed Stochastic Neighbor Embedding (t-SNE) Visualization

T-distributed Stochastic Neighbor Embedding (t-SNE) is similar to PCA in that it reduces dimensions. However, t-SNE is a method meant specifically for visualizing high-dimensional data in a lower dimensional map. t-SNE helps to visualize how feature extractions are clustering the data, and thus helps users determine the best feature maps to use.

Figure 12 shows the t-SNE visualization for our dense features like HOG, GLCM, BOVW, VGG16 and ResNet101 embeddings, and our final features (i.e., BOVW, Mean RGB and HSV, and ResNet101). From the t-SNE visualization, it is clear that ResNet is phenomenal in distinguishing between the categories in our dataset.



(a) HOG (RGB) (b) GLCM (c) BOVW

(d) VGG16 (e) ResNet101 (f) Final Features

Figure 12: t-SNE visualization of first 2 components of various features on the training dataset

3 Classification

In this section, we present the various classification models we trained and analyzed. As discussed in Section 2, we select the following features: (1) bag of visual words (BOVW), (2) mean RGB color channels, (3) mean HSV channels, and (4) embeddings from ResNet101 model. Our final feature vectors include 2254 dimensions. After PCA, we only use the first 20 principal components.

3.1 Results

We train 4 models: (1) Logistic Regression, (2) Random Forest, (3) XGBoost, and (4) Support Vector Machine (SVM). We split our dataset into 3 sets: (1) training set, consisting of 70% (or 1470 images) of the dataset, (2) validation set, consisting of 20% (or 420 images) of the dataset, and (3) test set, consisting of 10% (or 210 images) of the dataset. Table 1 shows the classification accuracy of our models when trained with the full dataset (consisting of 2254 dimensions). Note that in these models we did not tune any hyperparameters. We refer the reader to the notebook for more detailed report and analysis of these models at: https://bit.ly/3OpSJYy.

Model	Accuracy		
	Training Set	Validation Set	Test Set
Logistic Regression	1.000000	0.971429	0.971429
Random Forest	0.981633	0.857143	0.880952
XGBoost	1.000000	0.888095	0.923810
SVM (Linear)	1.000000	0.966667	0.976190
SVM (RBF)	0.863946	0.783333	0.790476

Table 1: Model performance on full dataset with 2254 features and no hyperparameter tuning

Next, we tune relevant hyperparameters in each of these models using grid search and use accuracy as the scoring metric to evaluate the performance. Table 2 shows the parameter(s) we tune and the value(s) that provide the best accuracy on the cross validation set. We then train our models using these hyperparameters and evaluate the performance on validation and test set. Table 3 shows the classification accuracy of our models. We refer the reader to the notebook for more detailed report and analysis of these models at: https://bit.ly/445t0dP.

Model	Parameter	Search Values	Best Value	Best Accuracy	Search Time
Logistic Regression	`max_iter`	`100, 500, 1000, 2000`	`2000`	0.85170	57.068344s
Random Forest	`max_depth`	`2, 4, 6, 8, 10`	`10`	0.83605	40.242700s
	`n_estimators`	`10, 50, 100, 200`	`200`		
XGBoost	`max_depth`	`2, 4, 6, 8, 10`	`10`	0.82517	243.260105s
	`n_estimators`	`10, 50, 100, 200`	`200`		
SVM	`C`	`1, 2, 5, 10`	`1`	0.85238	8.016668s
	`kernel`	`'linear', 'rbf'`	`'linear'`		

Table 2: Hyperparameter tuning (Grid Search) for our models

Model	Best Hyperparameters	Accuracy		
		Training Set	Validation Set	Test Set
Logistic Regression	`max_iter: 2000`	0.940816	0.859524	0.885714
Random Forest	`max_depth: 10, n_estimators: 200`	0.995918	0.859524	0.828571
XGBoost	`max_depth: 10, n_estimators: 200`	1.000000	0.823810	0.842857
SVM	`C: 1, kernel: linear`	0.989116	0.864286	0.857143

Table 3: Model performance on 20 principal components after hyperparameter tuning

Logistic Regression. Logistic Regression is an efficient and interpretable classifier widely used for classification tasks. It models a linear relationship between independent variables and the dependent variable using the logistic function to estimate probabilities. Due to its simplicity and interpretability, we selected Logistic Regression for our land use classification dataset. In our experiments, this model demonstrated impressive accuracy, achieving 94% on training, 85% on validation, and 88% on the test set (cf. Table 3).

Typically, it is hard to obtain complex relationships using logistic regression. Our model, however, does very well. Logistic regression exhibits difficulty in classifying intersections (cf. Figure 14(a)), but most of the other models also struggle with the same classification, it is therefore difficult to determine the precise reason for the misclassification.

Random Forest. Random Forest is an ensemble learning method that employs multiple decision trees for making predictions. By combining the results of these trees through voting or averaging, Random Forest can handle complex datasets with nonlinear relationships that logistic regression may fail to capture. It is well-suited for high-dimensional feature spaces and is resistant to overfitting. Due to these characteristics, we opted to use Random Forest for our land use classification. Our experiments yielded impressive accuracy on the validation and test sets; however, overfitting was observed on the train set, as indicated in Table 3.

Random forest is typically computationally expensive (when the number of estimators are huge), and fails to determine the significance of each variable. With our feature selection, however, random forest was relatively quick compared to other models and had a high accuracy. The misclassifications in random forest were again with intersections (cf. Figure 14(b)) and this is likely a problem with the feature vectors rather than the specific classifier as it is a common misclassification across models.

Extreme Gradient Boosting (XGBoost). XGBoost is an enhanced implementation of the gradient boosting algorithm. It constructs an ensemble of weak prediction models, iteratively learning from residuals

and combining predictions to improve accuracy. With its capability to capture complex non-linear relationships and achieve accurate predictions, XGBoost exhibited remarkable accuracy on the test and validation sets; however, overfitting was observed on the train set, as shown in Table 3.

The main limitation of XGBoost is that it is prone to overfitting, particularly with smaller datasets. Even with PCA we still encountered overfitting.

Support Vector Machine (SVM). SVM is a versatile classifier that determines the best hyperplane to separate classes in the feature space. By maximizing the margin between data points of different classes, SVM effectively handles complex relationships and can map high-dimensional data to a higher-dimensional space. In our experiments, we employed both linear and RBF kernel SVM, both of which yielded excellent results (as indicated in Tables 1 and 3).

3.1.1 Model Performance

Overall, each classifier showed strengths in the land use classification task. With hyperparameter tuning, we saw some uplift in accuracy for random forest classifiers. The confusion matrix in Figure 13 provides a comprehensive overview of the performance, while Figure 14 highlights some misclassification categories.

4 Generalizability

Generalizability was a large concern throughout the experimentation process. Following common practices, we split our data into train, validation, and test sets. We chose a $70\%, 20\%, \&10\%$ split for train, validation, and test respectively and experimented on the train and validation set, using the validation accuracy and confusion matrices to inform our decisions for features selection. We only use the test set on the final feature vector and classifier choice.

Dealing with overfitting. During the initial experimentation phase we encountered some overfitting issues. The first instinct to combat overfitting would be to perform PCA and only use a small subset of components which encapsulates about 85% of the variation. In our case even with only 20 components, we see overfitting throughout our experiments; some classifiers had train accuracy at or close to 1 with validation accuracy at around 70%. Through experimentation we saw that HOG was the primary perpetrator of the overfitting, however when removing it our accuracy decreased drastically. After we added the complex feature vectors with VGG and ResNet, however, we were able to achieve extremely high accuracy with only 20 components and eliminating overfitting in almost all classifiers except for XGBoost.

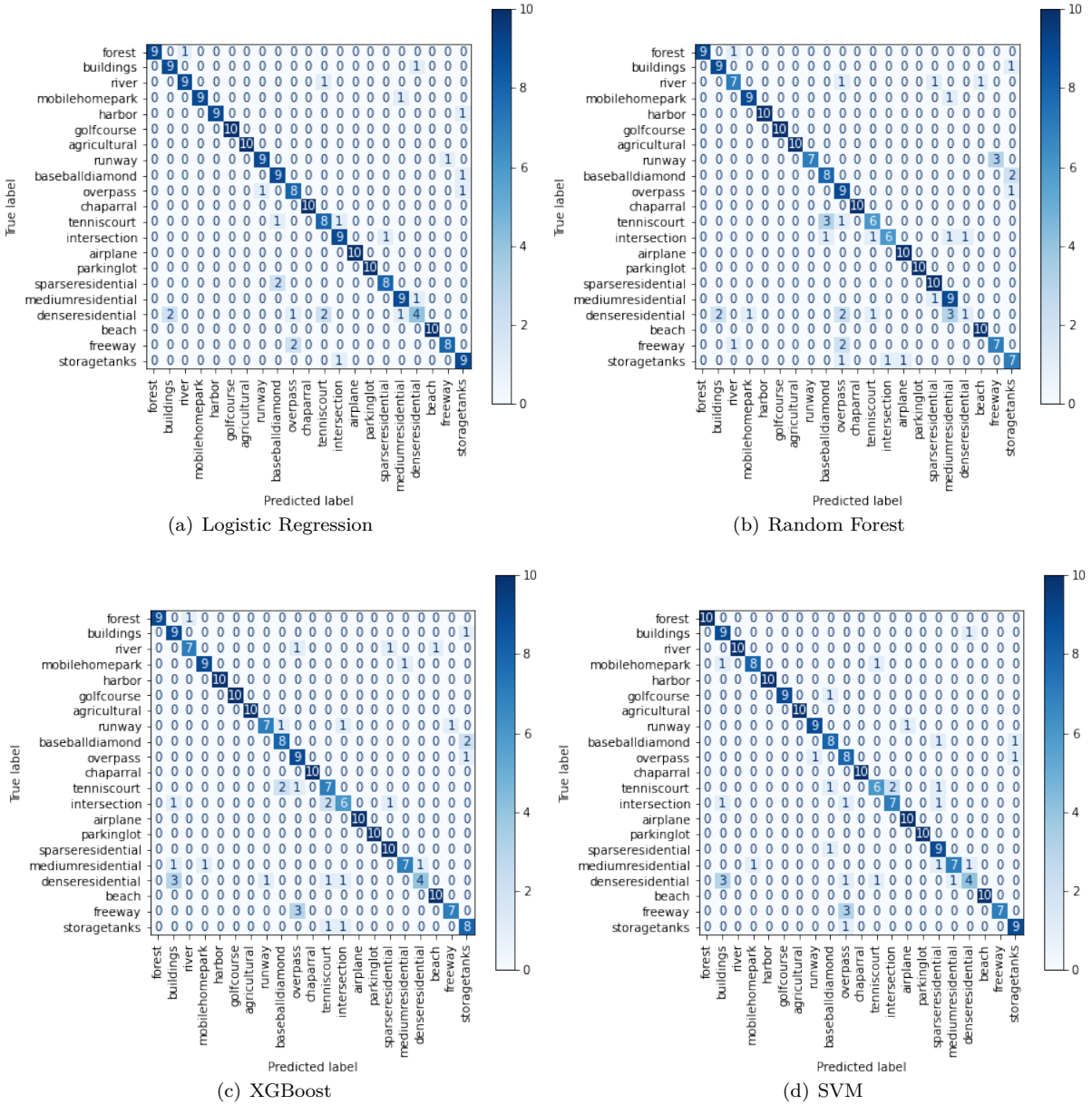
Hyperparameter tuning. We used grid search to tune our hyperparameters in search of the most accurate, efficient, and generalizable classifiers. Grid search tests each classifier on our reduced component dataset with every combination of parameters and scores them based on accuracy.

Final results. Table 3 highlights the performance on our models on test set with 20 principal components (after hyperparameter tuning). Our results indicate that we achieved generalizability for all models.

5 Efficiency

To assess the efficiency of our classifiers, we documented the training and prediction times across all models, both with the full image dataset and with PCA (20 components) transformed dataset to our training, validation, and test splits. The training process was conducted on 2.3Hz Quad Core Intel i7 MacBook Pro with 32GB of memory. Overall, our models exhibited impressive speed and efficiency.

Table 4 shows the training time and the prediction time for the entire dataset; Logistic Regression and XGBoost took the longest by far, but by industry standards they are pretty fast. We also assess the efficiency of our classifiers with only 20 principal components of the dataset. Computing 20 principal components of the dataset took 0.22 seconds. Table 5 shows the results on the 20 principal components of the dataset for the following: (1) the time it took to tune the hyperparameters, (2) training time with the tuned hyperparameters, (3) prediction time on training, validation and test sets. XGBoost has many parameters and as expected, the



(a) Logistic Regression

(b) Random Forest

(c) XGBoost

(d) SVM

Figure 13: Confusion matrices of our classifiers on the test set.

(a) Logistic Regression

(b) Random Forest

(c) XGBoost

(d) SVM

Figure 14: Some misclassification examples on validation set from our classifiers

grid search to tune the hyperparameters of XGBoost took the longest. Its worth noting that training on reduced component dataset was a lot quicker than the full dataset.

Model	Model Training Time	Prediction Time		
		Training Set	Validation Set	Test Set
Logistic Regression	46.334399s	0.013238s	0.002927s	0.002522s
Random Forest	2.192632s	0.062170s	0.024529s	0.034632s
XGBoost	47.600663s	0.023990s	0.009291s	0.005990s
SVM (Linear)	0.724948s	1.803072s	0.522328s	0.249482s
SVM (RBF)	0.882340s	2.941856s	0.899447s	0.416643s

Table 4: Efficiency of the models (on 2.3GHz Quad-Core Intel i7 MacBook Pro with 32GB Memory) for the full dataset with 2254 features and no hyperparameter tuning

Model	Hyperparameter Tuning Time	Model Training Time	Prediction Time		
			Training Set	Validation Set	Test Set
Logistic Regression	57.068344s	6.117899s	0.000724s	0.000384s	0.000156s
Random Forest	40.242700s	1.312505s	0.174917s	0.050048s	0.044467s
XGBoost	243.260105s	5.409562s	0.012476s	0.003941s	0.003333s
SVM	8.016668s	0.389447s	0.039479s	0.015372s	0.006088s

Table 5: Efficiency of the models (on 2.3GHz Quad-Core Intel i7 MacBook Pro with 32GB Memory) for 20 principal components of the dataset

Logistic Regression and XGBoost exhibited longer training times for the entire dataset, while SVM proved to be the fastest in training, regardless of whether it was applied to the complete dataset or the PCA-transformed data. It is worth mentioning that 2,254 features with 1,470 images was relatively faster to train and did not necessitate extensive GPU usage.

6 Conclusion

In summary, the classification models evaluated in our study demonstrated impressive performance on accuracy and efficiency metrics. Overall, all models performed well, exhibiting relative speed and efficiency. However, we encountered a significant issue of overfitting with the XGBoost model, even when utilizing PCA. Notably, by leveraging embedding vectors from pre-trained deep neural network models, we achieved a substantial improvement in accuracy. In fact, our results surpassed those reported in the original paper from 2010 [1], which relied on feature extraction techniques such as GLCM, bag of visual words, etc. It is worth noting that at the time of the original study, deep neural network models were not yet available, further highlighting the advancements we have made in leveraging these powerful models for improved classification accuracy.

References

[1] Yi Yang and Shawn Newsam. "Bag-of-visual-words and spatial extensions for land-use classification". In: *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS)*. 2010.

[2] Ray. *Color, Shape and Texture: Feature Extraction using OpenCV*. https://medium.com/mlearning-ai/color-shape-and-texture-feature-extraction-using-opencv-cb1feb2dbd73. Feb. 2022.

[3] *Count number of Object using Python-OpenCV*. https://www.geeksforgeeks.org/count-number-of-object-using-python-opencv/.

[4] *Bag of Visual Words*. https://www.pinecone.io/learn/series/image-search/bag-of-visual-words/.

[5] Ethan Rublee et al. "ORB: An efficient alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[6] *Understanding TF-IDF (Term Frequency-Inverse Document Frequency)*. https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/.

[7] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[8] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].