Land Use and Land Cover Classification of Satellite Imagery

Yeshwanth Somu Andrew Loeber Zhifei Dong

University of California, Berkeley

Abstract

Land use and land cover (LULC) classification is a critial process used in remote sensing and geographic information systems (GIS) to categorize and map the different types of land use and land cover within a specific geographic area. This can include categories like forests, wetlands, urban areas, agricultural fields, water bodies, and more. The classification helps to describe and analyze how land is being used and what types of surfaces or covers exist on the Earth's surface. Machine learning is commonly employed in the land cover classification by providing automated and efficient methods to process large amounts of satellite or aerial imagery data. In this study, we extract both color and GLCM texture features in combination with EfficientNetV2 mean channel activation values from the images, and employ machine learning methods, in the form of Logistic Regression and XGBoost models, for image classification of land use and land cover in 10 categories. We performed a hyperparameter search on both models. The best model, Logistic Regression on three simple features and one complex feature, gives 95.4% test accuracy. We also identified an efficient model, XGBoost on three simple features, that gives 93.2% test accuracy.

Keywords: Geospatial data, satellite imagery, classification, feature extraction, machine learning

1. Introduction

The concept of land use and land cover classification has been around for several decades, evolving with advancements in technology. Initially, it relied heavily on field surveys and the interpretation of aerial photographs. The advent of satellite remote sensing in the latter half of the 20th century revolutionized land use and land cover classification, providing a more efficient and extensive way to map and monitor the Earth's surface. The use of satellite images allowed for more frequent updates and a better understanding of large-scale patterns and changes. Organizations utilize the satellite imagery through sources such as NASA, Google Images, Maxar and EuroSAT, for various reasons, including monitoring environmental changes, guiding urban planning decisions, and assessing the impacts of human activities on natural resources, etc. This geographical data usually consists of gigabytes of data, so analyzing it all visually or manually can be a painstaking process. With the increasing computing power, machine learning has emerged as a pivotal tool in land use and land cover classification. Machine learning algorithms can efficiently process and analyze vast amounts of data, identifying patterns and making predictions about land use and land cover types. These algorithms have significantly improved the accuracy, speed, and scalability of land use and land cover classification, enabling more detailed and frequent monitoring. Popular machine learning models in land use and land cover classification include convolutional neural networks (CNN), support vector machines (SVM), and random forests. In our study, we employed image feature extraction and machine learning models on the satellite imagery data with labeled land use and land cover classes.

2. Dataset

2.1 Overview

For our work, we selected the EuroSAT dataset, a publicly-available collection of aerial satellite photographs depicting a diverse array of European landscapes, all of which were taken by the European Space Agency's Satellite-2 system. Although EuroSAT offers a full-spectrum version encompassing all 13 spectral channels captured by this satellite technology, we selected the RGB version for simplicity of use. Images were provided as 64x64 pixel RGB JPEG files with a reported scale of 10 meters per pixel, meaning that each pixel square represents a real-world area of roughly 100 m^2.

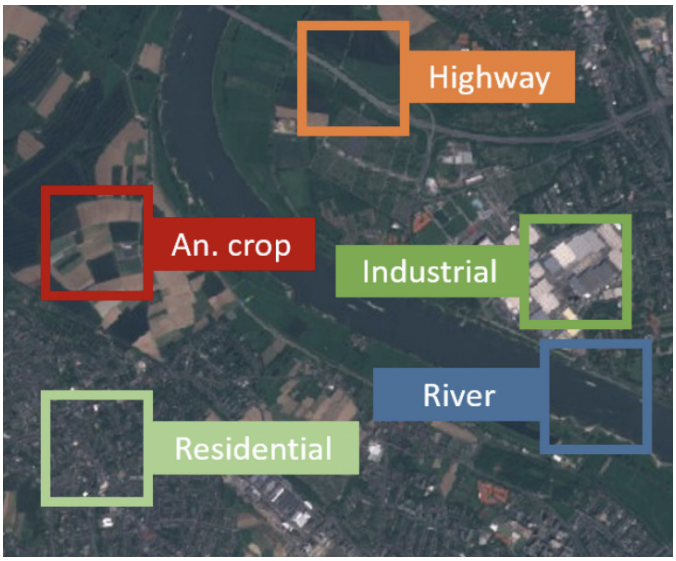


Figure 1. Selection of labeled image patches from a larger-scale satellite photo

Each image in the dataset is categorized into one of 10 distinct classes. These classes span a spectrum from highly urbanized areas, like industrial parks and residential zones, to natural landscapes, including forests, rivers, and lakes. The complete list of these classes is provided in Table 1 below.

Class Label	Class Description	Image Count
Industrial	Industrial parks, office buildings	2500
Residential	Housing developments, suburban neighborhoods	3000
Highway	Areas containing a large highway	2500
AnnualCrop	Seasonally-rotated croplands	3000
PermanentCrop	Year-round croplands	2500
Pasture	Grass-covered grazing grounds	2000
HerbaceousVegetation	Shrublands, scrublands	3000
Forest	Wooded areas densely packed with trees	3000
River	Areas containing a river	2500
SeaLake	Sections of oceans, lakes, and ponds	3000

Table 1. Class labels, descriptions, and sample sizes

We partitioned the complete EuroSAT image dataset into training, validation, and test sets using a 70/15/15 scheme, stratifying by class label to ensure that each split contained the same distribution of classes. Consequently, the training set comprises 18,900 images, and the validation and test sets contain 4,050 images each.

2.2 Class Examples

A crucial initial step in any image classification task involves a thorough visual inspection of the dataset. Figure 2 illustrates this, presenting five representative images for each class label to aid in understanding their visual features.

Certain classes are immediately distinguishable to the human eye. For example, "Industrial" images typically showcase large, white, rectangular buildings amidst expanses of gray asphalt. "Pasture" images, on the other hand, are characterized by vast areas of bright green, occasionally interspersed with small clusters or thin lines of trees.

But some classes are trickier to tell apart. Rivers and highways both manifest as stark, narrow lines cutting through surrounding vegetation, cropland, and buildings - leading to a lot of overlap between “River” and “Highway” images. “HerbaceousVegetation” images with especially dense plant cover can look a lot like forests. Even “Forest” images sometimes are only really distinguishable from those in “SeaLake” by paying close attention to the subtle bumpy texture indicating the presence of tree cover.

Figure 2. Selection of five sample images per class label

3. Feature Extraction

Our next step was to determine which image features to extract for use as the inputs to our classifier models. Following our initial review of the EuroSAT dataset, we identified color and texture as particularly promising areas to explore. For example, an abundance of gray often indicates the presence of paved asphalt, so we would expect images with a high prevalence of

gray tones to be more likely to be a member of the “Industrial,” “Residential,” or “Highway” classes. Similarly, images with a very flat texture across the majority of the frame are unlikely to be anything other than a member of the “SeaLake” class.

For our color-based features, we chose average RGB pixel values to capture the overall redness, greenness, and blueness of the images. Additionally, we utilized K-Means color bin histograms to represent the frequency of pixels matching a set of representative colors. We believed that these two types of features, combined, could efficiently describe the global color profile of each image and highlight key differences in coloration between classes.

For our texture-based feature, we decided to extract statistical properties from each image's Gray-Level Co-occurrence Matrix (GLCM). Unlike the spatially invariant nature of our color features, GLCM concentrates on the relationships between each pixel and its immediate neighbors, making it well-suited for capturing textural characteristics.

Lastly, as our complex feature set, we opted to extract mean channel activation values from a pre-trained EfficientNetV2 model checkpoint. We expected that some of the abstract features it had learned to detect while training on ImageNet would also be discriminative when applied to EuroSAT imagery. As the base model embeddings were fairly high-dimensional, we utilized a shallow neural network to compress the original dense embeddings into a low-dimensional sparse representation.

3.1 Average RGB Pixel Values

The first set of feature vectors we gathered were the average RGB pixel values per image. Specifically, we took the mean RGB value for each (64, 64, 3) image, resulting in a final feature vector of length 3. To understand how colors differ between labels, we also averaged each pixel’s

value across all images in each label. This gave us an "average" image per label, which we then plotted the histograms of, shown in Figure 3. There was a lot of variability in colors from labels to labels. The figure below shows the histograms for 6 key labels. We observed a variety of average pixel values in "Industrial", which is likely due to the fact that "Industrial" images contain widely-varying proportions between roadways and buildings. However, the other images showed distinct peaks of specific pixel values per color channel.

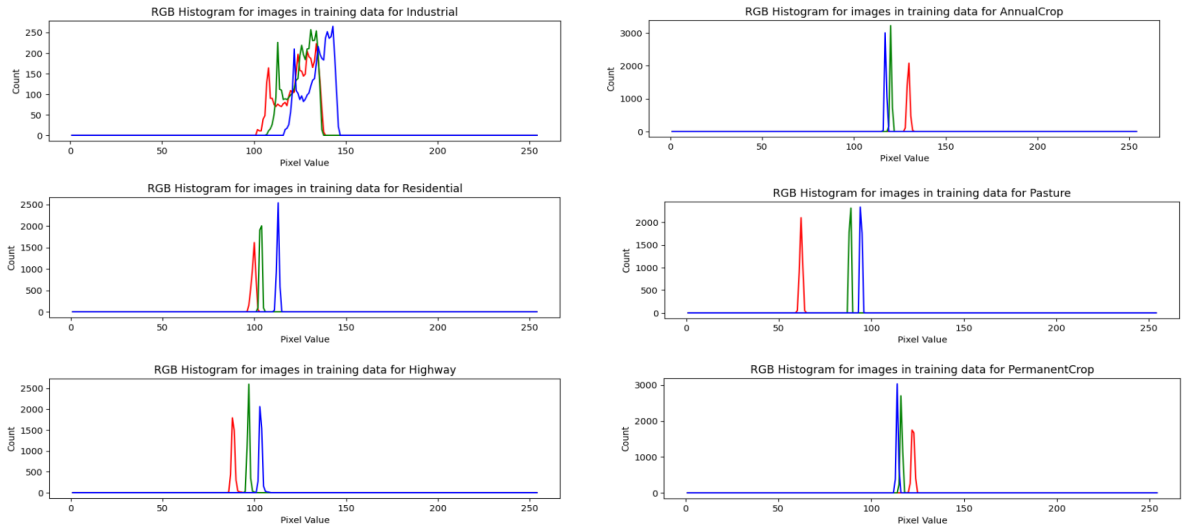


Figure 3. Histogram plots showing average image RGB value by label

Nevertheless, we observed great variation in pixel values between classes. Next, we calculated the average pixel value of each of the RGB channels and stored it as a row per image for each of the training, validation, and test sets.

3.2 K-Means Color Bin Histograms

K-Means is a popular clustering algorithm that partitions input data into distinct, non-overlapping clusters based on their features. This makes it a natural choice for compressing color information in images, as it allows us to reduce the 16.8M possible pixel values in 24-bit

RGB space down to an arbitrarily chosen number of the k most representative colors present. Our approach is motivated by the desire to capture fine-grained color information in a similar manner to 3D color histograms without being subject to its limitations - specifically, its rigidity in defining cutoff points, its inclusion of color bins which may only rarely manifest in aerial satellite imagery, and the explosive increase in dimensionality as the number of bins per channel is increased (even a modest selection of 8 bins per channel results in $8^3 = 512$ total bins).

In the typical application of K-Means color quantization, each image is dealt with in isolation, so the resulting k colors are completely different from sample to sample. However, our objective differs; we aim to determine the optimal k colors across the entire training set, maintaining consistent color bins across all samples. To achieve this, we reshaped the training images into a 2D array, with the first dimension representing each pixel, and the second dimension representing the color channel (R, G, B). These were then converted into the CIELAB color space, which separates color information into lightness, red-green, and blue-yellow components. CIELAB is attractive here due to both its increased perceptual uniformity and its decoupling of luminance from chrominance. But most importantly, the behavior of K-Means clustering is heavily influenced by the choice of distance metric, and Euclidean distances between color vectors are considered to be more meaningful in LAB space.

The training dataset encompasses over 77 million pixels, making the conventional K-Means algorithm impractically resource-intensive. To get around this issue, we utilized scikit-learn's implementation of mini-batch K-Means, which leverages small, random data batches for each clustering iteration, significantly cutting down computation time and memory requirements. We chose $k = 64$, resulting in 64 discrete color bins that represent the training images' overall color palette.

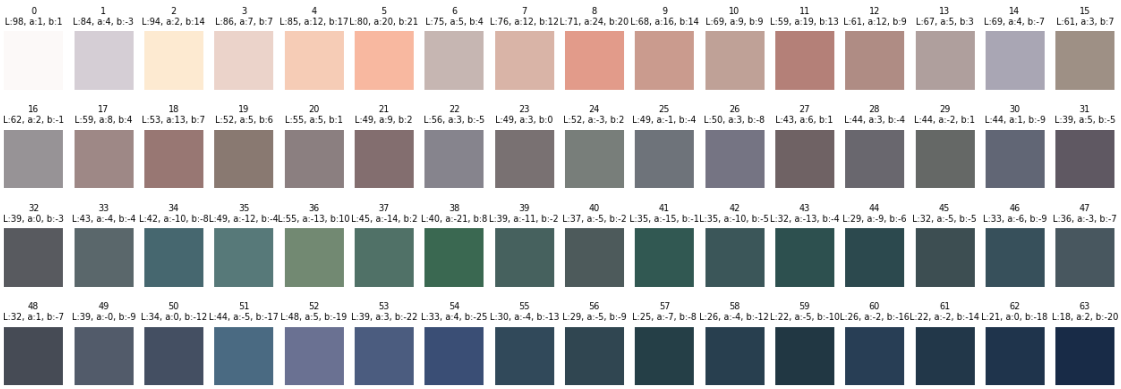


Figure 4. K-Means cluster centroids representing final 64-color palette

To convert this into a usable image feature for modeling, all pixels within an image are assigned to the closest K-Means color bin, then we calculate the proportions of how many pixels fell under each bin, culminating in an l_1-normalized 64-dimensional vector.

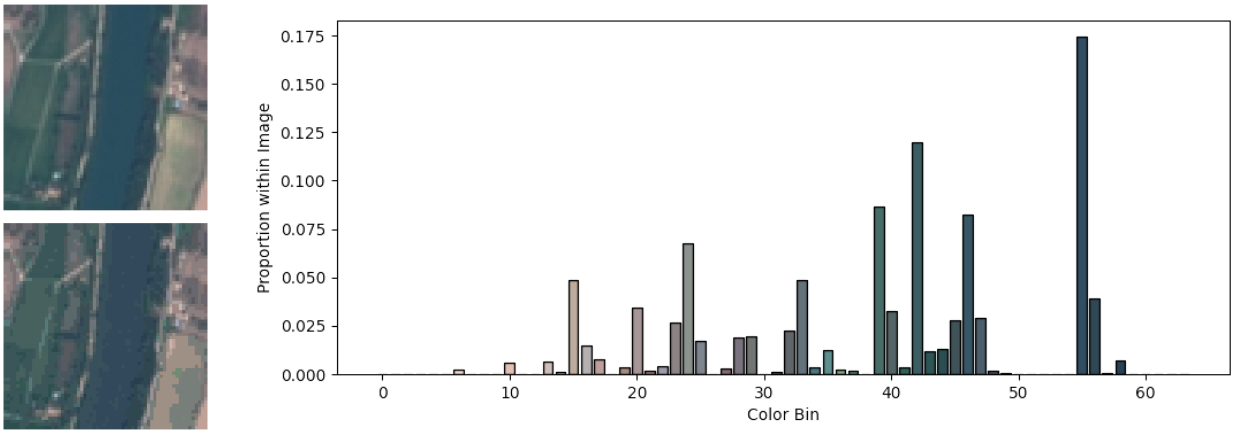


Figure 5. Sample image (upper left), K-Means color quantized version (lower left), and column chart showing proportion of each color bin within the sample image (right)

3.3 Gray-Level Co-occurrence Matrix

The Gray Level Co-occurrence Matrix (GLCM) is a powerful image processing tool for conducting texture analysis, finding use in various applications such as medical imaging, remote

sensing, and machine vision. At its core, it is a statistically-grounded methodology that considers how brightness levels are distributed amongst pairs of pixels within an image, while also accounting for the spatial relationships between these pairs.

A singular GLCM is expressed as a square matrix, where the number of rows and columns is equal to the number of gray levels, *G*, present in the image. For example, if the number of gray levels is 256, the matrix shape will be 256 x 256. The matrix element $P(i, j \mid \Delta x, \Delta y)$ is the relative frequency with which two pixels, separated by a pixel distance $(\Delta x, \Delta y)$, occur within a given neighborhood, one with intensity *i* and the other with intensity *j*.

To construct a GLCM, first a pixel distance and direction must be selected. Then, for each pair of pixels in the image separated by the chosen distance and direction, increment the value of the matrix element corresponding to the gray levels of these pixels. A standard final step is to normalize the GLCM so that its elements sum to a value of one. This process can easily be extended such that multiple GLCMs are constructed from the same base image using different distances and/or directions, creating a richer basis of textural representation.

Figure 6 below shows visually which pixels are selected for comparison with the center pixel when pixel distance is set to 1, and the directions are set to [0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°].

135°	90°	45°
180°	Center Pixel	0°
225°	270°	315°

Figure 6: Visual representation of selected comparison pixels under various specified directions

The raw GLCM itself is useful, but given its inherently high dimensionality, it's often the statistical measures derived from it that turn out to be the most useful. The GLCM properties we chose to use in our study comprised the following:

1. **Contrast**: $\sum_{i,j} P(i, j)(i - j)^2$

   Measures the local variations in the GLCM using squared difference.

2. **Dissimilarity**: $\sum_{i,j} P(i, j)|i - j|$

   Measures the local variations in the GLCM, similar to Contrast, but takes the absolute difference, making it less sensitive to larger differences.

3. **Homogeneity**: $\sum_{i,j} P(i, j)/(1 + (i - j)^2)$

   Measures the closeness of the distribution of elements in the GLCM to the GLCM diagonal.

4. **Energy (or Angular Second Moment)**: $\sum_{i,j} P(i, j)^2$

   Sum of squared elements in the GLCM.

5. **Correlation**: $\sum_{i,j} P(i, j)(i - \mu_i)(j - \mu_j)/(\sigma_i \sigma_j)$

   Measures how correlated a pixel is to its neighbor over the whole image, noting that μ and σ represent the mean and standard deviation of all gray values within the associated set.

In order to extract GLCM features from our images, we first needed to convert our RGB images into grayscale. Then, we constructed gray level co-occurrence matrices for each using a pixel distance of one and enabling the setting for using the symmetrical implementation, allowing us to specify only four directions [0°, 45°, 90°, 135°] while still capturing information

across all eight directions as shown in Figure 6. For each of the four symmetrical GLCMs constructed per image, we calculated the five GLCM statistical properties listed above, resulting in a final feature vector of length 20 - very compactly representing images' textural qualities.

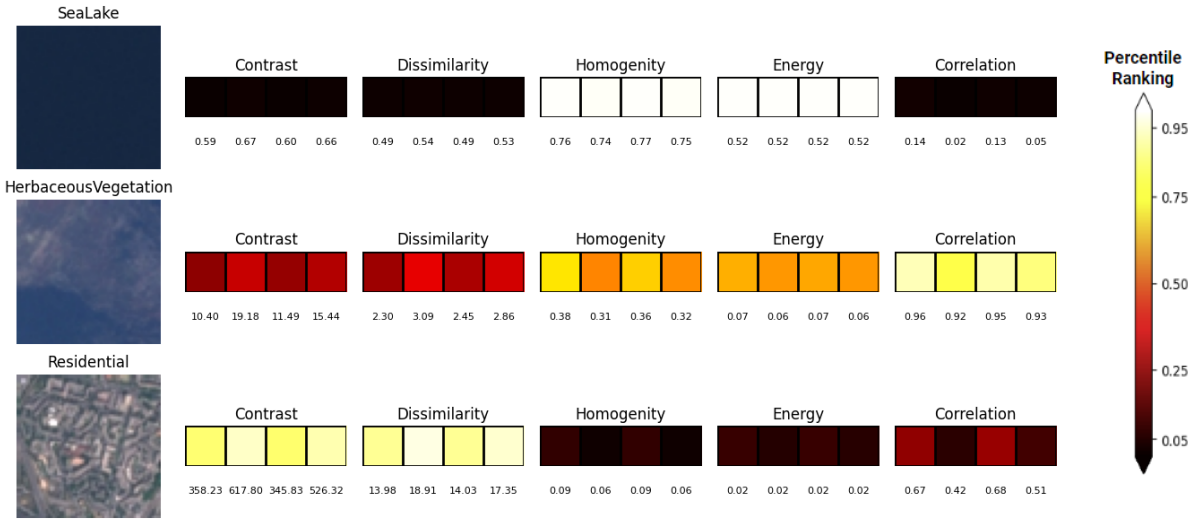


Figure 7. Heatmap visualization of GLCM properties for sample images

3.4 EfficientNetV2 Embeddings

EfficientNetV2 is a family of convolutional neural networks developed by two Google researchers to be competitive with state-of-the-art model architectures in terms of accuracy on image classification tasks while requiring significantly fewer parameters, which in turn leads to markedly faster training and inference speeds. They accomplished this by combining a novel progressive training strategy with a neural architecture search (NAS) process that "jointly optimize[s] training speed and parameter efficiency" according to their 2021 paper.

Beyond its proven strengths on accuracy and speed, EfficientNetV2 was an enticing option to use as a high-level feature detector for myriad additional reasons. Key among these was its excellent public deployment via the Keras API, which enabled us to quickly download model

checkpoints which had been pre-trained on ImageNet as well as pre-specify the type of feature pooling to use for producing embeddings. EfficientNetV2 is also largely resolution-agnostic, and several models are available which can run inference on 64x64 pixel images right out of the box. We ultimately selected the EfficientNetV2B3 model as our embedding engine, as it was the largest one available which could natively handle 64x64 resolution, and its download and inference speeds were negligibly slower as compared to its lighter-weight counterparts.

We started by downloading a pre-trained EfficientNetV2B3 checkpoint set to use average pooling. Under this setup, images passed in are propagated through each layer of the model as normal until reaching the final convolutional layer, at which point each input image has been transformed into a tensor with 4x4 spatial resolution and 1536 channels. Then, a channelwise mean pooling operation is performed, resulting in a 1536-dimensional vector. We applied this to our training, validation, and test sets before saving the outputs as numpy arrays for future use.

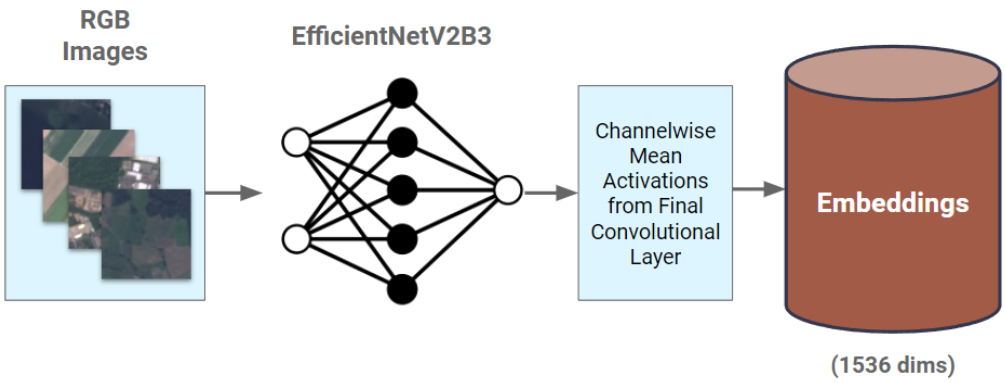


Figure 8. Schematic of the initial embedding extraction process

While the raw embeddings were not so high-dimensional that they could not feasibly be used as inputs to our classifier models, we felt there would still be advantages to compressing them into a more compact representation. To list a few, faster convergence times during training, decreased RAM usage, and the suppression of feature variability lacking utility for our specific image classification task. To accomplish this, we applied supervised dimensionality reduction via artificial neural network (ANN), training a shallow ANN with one 64-unit hidden layer to predict class labels given raw embedding vectors as inputs, before extracting ReLU activations from the hidden layer to act as our final 64-dimensional compressed embeddings. A great deal of regularization was necessary to prevent the model from prematurely converging on a suboptimal solution, which we implemented using gaussian noise injection and dropout. Following completion of the training process on our encoder model (achieving an accuracy of 93.9% on validation data), we ran each of our dataset splits through it and saved the results for later use.

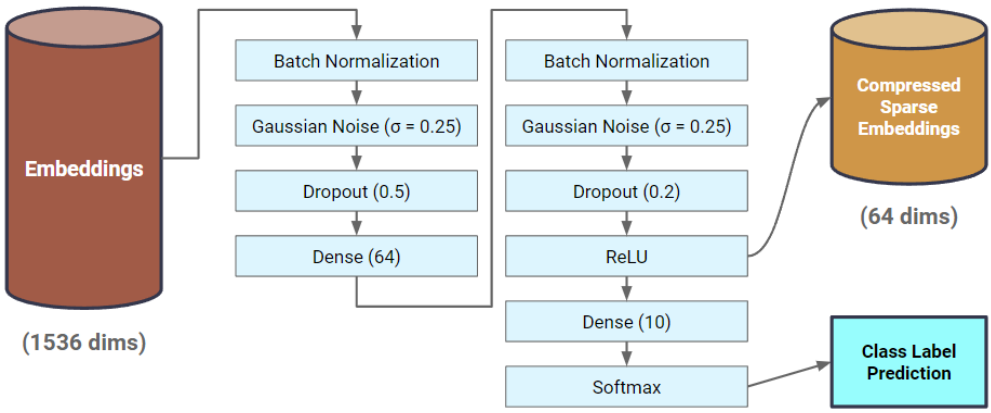


Figure 9. Schematic of our ANN-aided supervised dimensionality reduction process

3.5 Low-Dimensional Feature Projections

A great way to gain intuition for how a set of data points is distributed in feature space when its dimensionality is too high to cram into a single plot is to first apply a dimensionality reduction technique to project it down into two dimensions. Many different techniques are available for creating such a projection, but for the following visualizations of each of our feature sets, we used Principal Component Analysis (PCA) and Uniform Manifold Approximation and Projection (UMAP).

PCA works by finding a linear transformation of the input matrix such that the transformed matrix retains as much of the variance present in the input as possible, with its columns sorted in descending order from left to right by how much of this variance they account for. In contrast to the inherent linearity of PCA, UMAP is a non-linear manifold learning technique which first constructs a topological representation of how each data point is connected with its nearest neighbors in high-dimensional space, instantiates a new set of embedding points in low-dimensional space, then iteratively pulls embedding points together or pushes them apart such that the topology of the embedding points matches the topology of the original points as much as possible.

Shown below in Figure 10 are the results from applying PCA and UMAP to each of our feature vectors (with the exception of average RGB pixel value, since it is already only 3-dimensional).

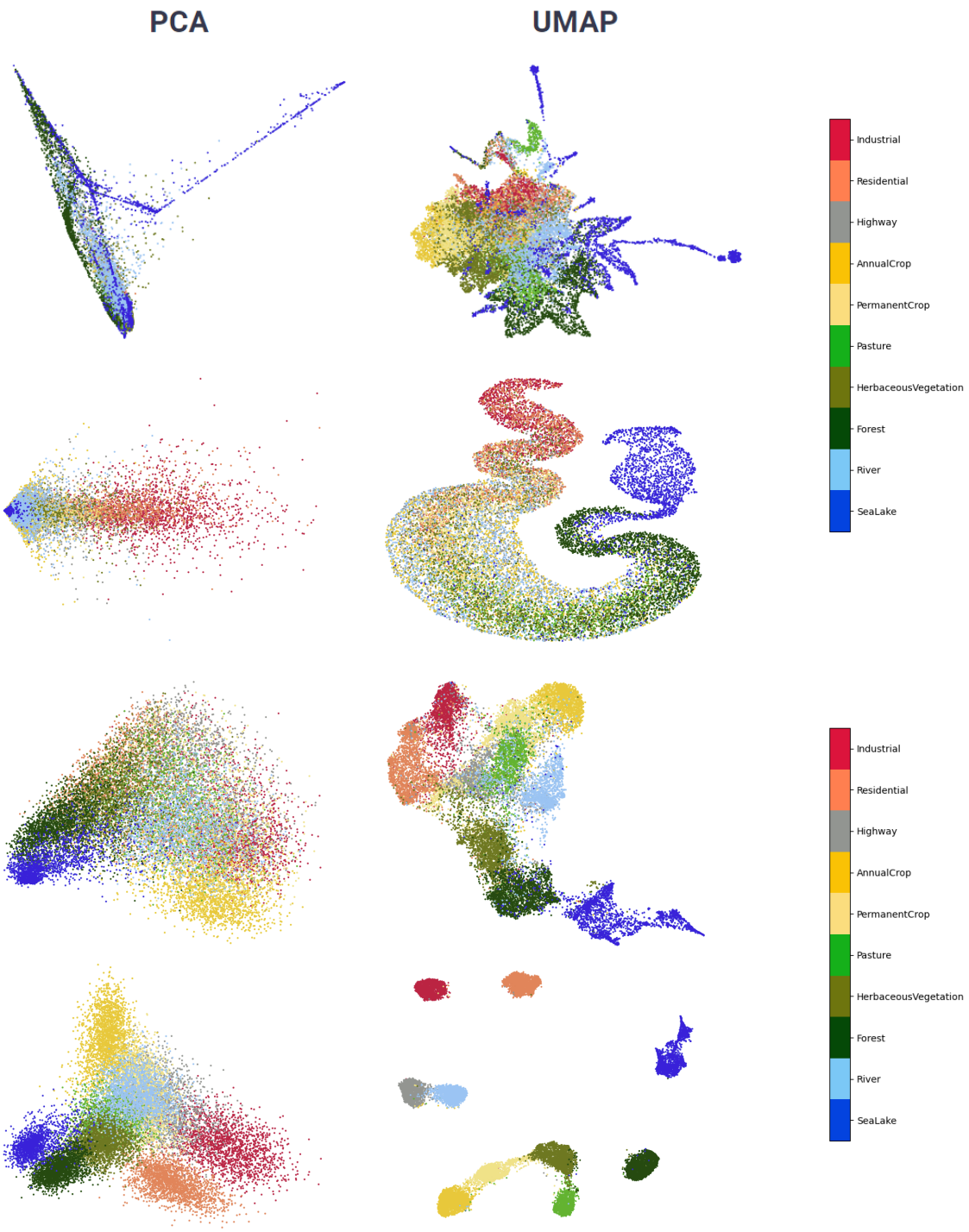


Figure 10. PCA and UMAP projections of training images' features
Listed in order from top to bottom:
(1) K-Means color bin histograms, (2) GLCM properties,
(3) original EfficientNetV2 embeddings, (4) compressed EfficientNetV2 embeddings

The visual representations of these feature sets show striking differences from each other that reveal much about their natures. K-Means color bin histograms appear to be a poor fit for PCA, with the “SeaLake” class taking up a large section of the projection, and all other classes squished together into a small area, indicating that it may not be easy to get quality classifications with a linear model. Its UMAP projection showcases much more class separation, but classes are not neatly gathered together - in many cases they are scattered haphazardly over the embedding space in smaller clumps. GLCM properties are also unusual in that both projections favor placing “SeaLake” and “Industrial” images on opposite ends of the spectrum, with the others gradually filling the space in between. This is likely due to the fact that GLCM contrast has a much larger scale than the other GLCM properties, and therefore is commanding more of PCA & UMAP’s attention. The original EfficientNetV2 embeddings show muddled but consistently-located class separation in the PCA plot, which is much starker with UMAP. The compressed embeddings (as would be expected with features constructed using supervised dimensionality reduction) show strong separation with PCA, and near-perfect separation with UMAP.

3.6 Combined Feature Vectors

The final step needed before entering the model fitting phase was to combine together our features. In order to see what kind of performance we could achieve without the use of a large CNN, we constructed two sets of combined feature vectors: a “basic” set and a “full” set. The basic set only contains the three basic image features: average RGB pixel values, K-Means color bin histograms, and GLCM properties, which forms a vector with a length of 87 (3+64+20). The full set adds in the compressed EfficientNetV2B3 embeddings, forming a vector with a length of

151 (3+64+20+64). Standard normalization was applied to the full feature vector to reduce the impact of imbalance feature scaling, then we applied PCA to examine the cumulative explained variance ratios shown in Figure 11 below. To reach 90% cumulative explained variance, approximately 60 PCA components are required.

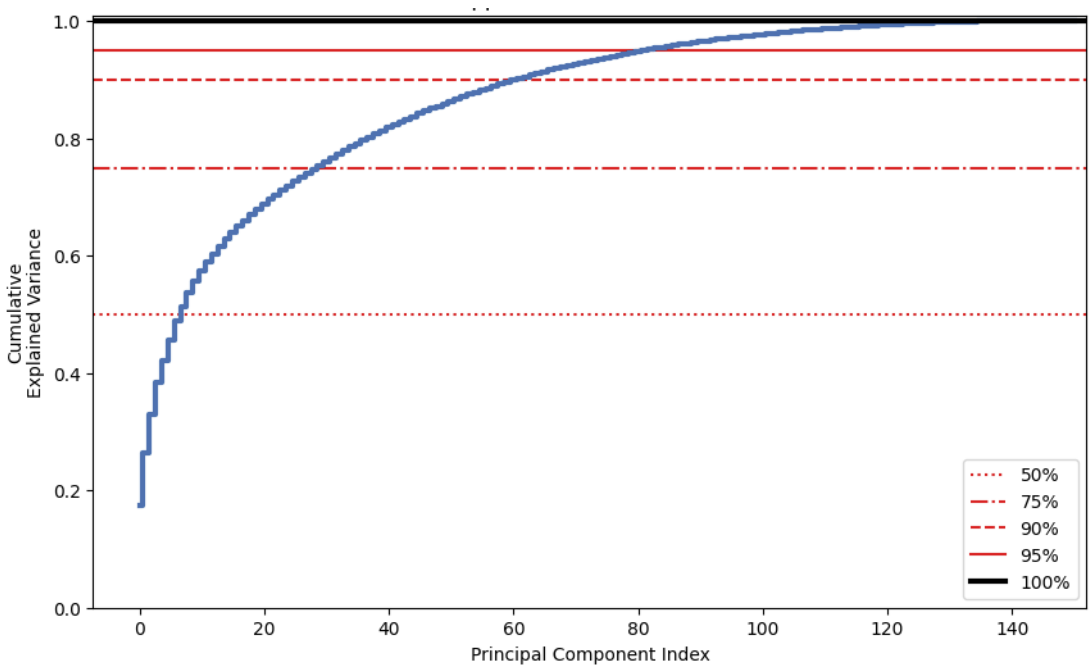


Figure 11. Cumulative explained variance plot of PCA applied on the full feature set

4. Classification

4.1 Initial Modeling

Once we compiled all the individual feature vectors, we input all feature sets (Color, GLCM, and compressed EfficientNetV2 embeddings) separately through a logistic regression model as a baseline. Our logistic regression model used a softmax activation function for enabling multiclass classification. The model's output was an array of probabilities of each row

being one of the 10 classes. From this, we chose the highest probability to be the predicted class for validation and testing. The results of the three feature sets are shown below in Table 2.

Feature Set	Hyperparameters	Validation Accuracy
Color Features	*C* = 100, max_iter = 2000	70.7%
GLCM	*C* = 1; *max_iter* = 2000	72.1%
EfficientNetV2	*C* = 100; *max_iter* = 2000	92.7%

Table 2. Validation accuracy fitting feature sets in isolation

The compressed EfficientNetV2 feature set performed much better than the color-based and GLCM feature sets, achieving a 92.7% validation accuracy. This was expected, as this was a complex feature set sourced from a modern CNN with well-established strength on image classification tasks. The GLCM features only resulted in a validation accuracy of 72.1%, which was lower than expected. Nevertheless, we were confident that our combined features would yield great results, expecting that the reinjection of lower-level image features would prove an effective complement to the highly-processed nature of the compressed EfficientNetV2 embeddings.

4.2 Model Tuning

After normalizing our basic and full combined feature sets using min-max scaling, we proceeded with inputting them into our two classifiers: logistic regression and XGBoost. To achieve the most robust models possible, we performed hyperparameter tuning on both model types via grid search.

For the logistic regression model, we primarily tuned the regularization parameter *C* and number of maximum iterations *max_iter*. The regularization parameter is used to control how much the model's weights are penalized by l2 regularization. A larger *C* value means less regularization, which can cause the model to have very large parameters, reduced generalization ability, and an increased chance of overfitting. We tuned the following values of *C* and *max_iter*:

$$C\text{: } [0.01,\ 0.1,\ 1,\ 10,\ 100,\ 1000,\ 10000]$$

$$max\_iter\text{: } [100,\ 250,\ 500,\ 1000,\ 3000]$$

Next, we used an XGBoost model because we wanted to use a more complex and nonlinear machine learning model for the purposes of multi-classification. XGBoost brings an ensemble element by adding new models to correct the errors of existing models until no further improvements can be made. We observed non-linearity in the PCA & UMAP visualizations of the feature vectors as shown in Figure 10. This was a significant facet to XGBoost that logistic regression lacks. It can handle this nonlinearity present in our features. One consequence of XGBoost is an increased computation time due to the algorithm's structure. There were many hyperparameters available to tune, however we focused on tuning the number of trees that are fit (*n_estimators)*, the depth of each tree (*max_depth*), and the learning rate (*learning_rate*). We present the grids used in this hyperparameter tuning as follows.

$$n\_estimators\text{: } [50,\ 100,\ 200]$$

$$max\_depth\text{: } [3,\ 5,\ 7]$$

$$learning\_rate\text{: } [0.01,\ 0.1,\ 0.3]$$

With the help of the grid search, we iterated through each combination of hyperparameters to get the best possible accuracy performance on the validation set. The results from our best-performing configurations are shown below in Table 3.

Feature Set	Feature Set	Hyperparameters	Training Time (s)	Train Accuracy	Validation Accuracy
Logistic Regression	Basic	*C* = 10000 *max_iter* = 3000	70.3	87.3%	85.7%
	Full	*C* = 1 *max_iter* = 100	4.4	99.2%	**95.2%**
XGBoost	Basic	*learning_rate* = 0.1 *max_depth* = 7 *n_estimators* = 200	51.5	100%	93.2%
	Full	*learning_rate* = 0.1 *max_depth* = 3 *n_estimators* = 100	18.9	99.7%	94.7%

Table 3. Summary of logistic regression and XGBoost training and fitting

We observed the logistic regression model with the full feature set to have the best performance on the validation set, achieving 95.2% accuracy. It was also the fastest model to train, taking only 2.4 seconds, beating out the XGBoost model, which took over 4X longer to train on the full feature set. This was surprising to us, as we did not expect the logistic regression model to handle non-linear relationships in the features very well. The most likely cause of this speedy convergence time is the way the EfficientNetV2B3 embeddings were compressed, as the simple ANN was itself trained to predict class labels after applying a linear projection and softmax operation, a very close match to how this logistic regression generates its predictions. We expected the XGBoost model to perform the best among all the models, due to its increased complexity and ability to capture non-linearity in the features.

4.3 Discussion

4.3.1 Results

Our best model (logistic regression trained on the full feature set) performed well when evaluated on the test set, achieving an accuracy of 95.4%. A confusion matrix summarizing its test set predictions given in Figure 12. Overall, every class had a very low confusion rate with accuracies > 90%. The "SeaLake" images gain the highest accuracy (99.3%), as they contain the most consistent color and texture across the image. The maximum mislabeling was 4.3%, in the case of "River" images being confused for "Highway", as they both have clear belt-shaped patterns through the image.

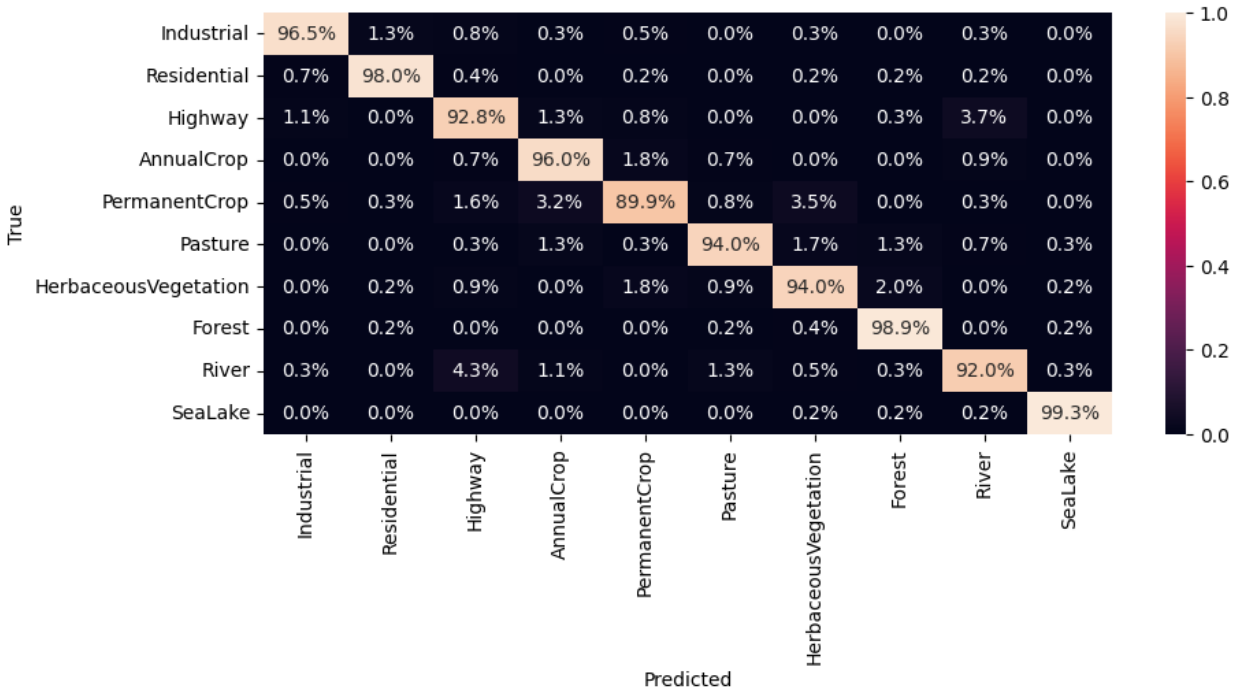
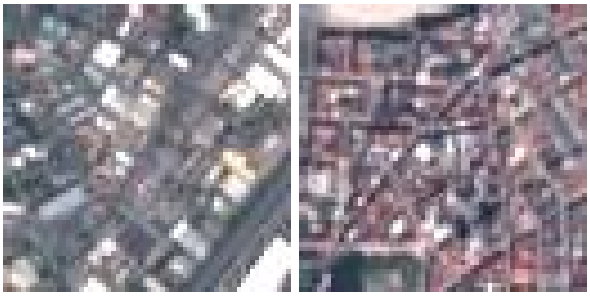


Figure 12. Confusion matrix for the full-feature logistic regression model's test set predictions

1. “Industrial” is mostly mislabeled as “Residential” (1.3%). It makes a lot of sense that images from these two classes often look very similar as shown in the figure below (“Industrial” on the left, “Residential” on the right). There is also 0.8% of “Industrial” predicted as “Highway”, because some “Industrial” images do contain roads. Similarly, “Residential” is mostly mislabeled as “Industrial” and secondly mislabeled as “Highway.”



2. As expected, “Highway” images are mostly mislabeled as “River” (3.7%). Meanwhile, it is noteworthy that there are a considerable amount of “Highway” images predicted as “AnnualCrop” and “PermanentCrop”, simply because many “Highway” images contain crop fields as we can see below.

3. The “AnnualCrop” images are mostly mislabeled as “PermanentCrop,” as these two classes both share the large rectangle patterns. In comparison, “PermanentCrop” images

are mostly mislabeled as “HerbaceousVegetation”, as many “PermanentCrop” images contain irregular shapes of crop fields that are very similar to vegetation as shown below (“PermanentCrop” on the left, “HerbaceousVegetation” on the right).


4. “Pasture” images are often mislabeled as “HerbaceousVegetation,” “Forest,” and “AnnualCrop,” as they are usually green like “HerbaceousVegetation” and “Forest,” and sometimes contain clear rectangle patterns like “AnnualCrop.”

4.3.2 Limitations

Listed below are a few possible factors limiting model performance:

1. Class Labels

   “AnnualCrop,” “PermanentCrop” and “Pasture” images share very similar rectangular patterns, making discerning between them difficult in many cases.

2. Image Resolution

   “Industrial” and “Residential” images usually contain roads, which can cause them to get mislabeled as “Highway”. The low resolution of the EuroSAT images limits the possibilities for capturing the difference between highways and simple roadways.

3. Classifiers

The two classifier types we used in our study are relatively simple choices for conducting image classification work. Convolutional neural network models used directly as a classifier instead of as a feature detector may yield improved performance.

4.4 Efficiency vs Accuracy

As shown in Table 3, although our logistic regression model fit on the full feature set performed best, having the highest validation accuracy and shortest training time, it may not necessarily be considered the most efficient when accounting for the practical usage costs. It requires running an instantiation of the EfficientNetV2 CNN, which took 3 minutes to run on the nearly 19K images in the training dataset using a GPU-accelerated Google Colab environment. In reality, the XGBoost model trained on the basic feature set, which took 48.6 secs to run with a validation accuracy of 93.2%, could be considered more efficient in practice. It doesn't require running either the EfficientNetV2 model or the compression model, which takes an average of 0.5 seconds to run on single unbatched images using a CPU Colab environment, and it was still able to capture a large amount of non-linearity present in the basic image features alone.

The confusion matrix of the basic-feature XGBoost model's predictions on the test set is given in Figure 13. In comparison with the best overall model, the accuracy of this more efficient model choice is significantly lower in "Highway" (82.9% vs. 92.8%), which indicates that the complex features play a vital role in capturing the difference between the "Highway" and "Industrial"/"River" classes.

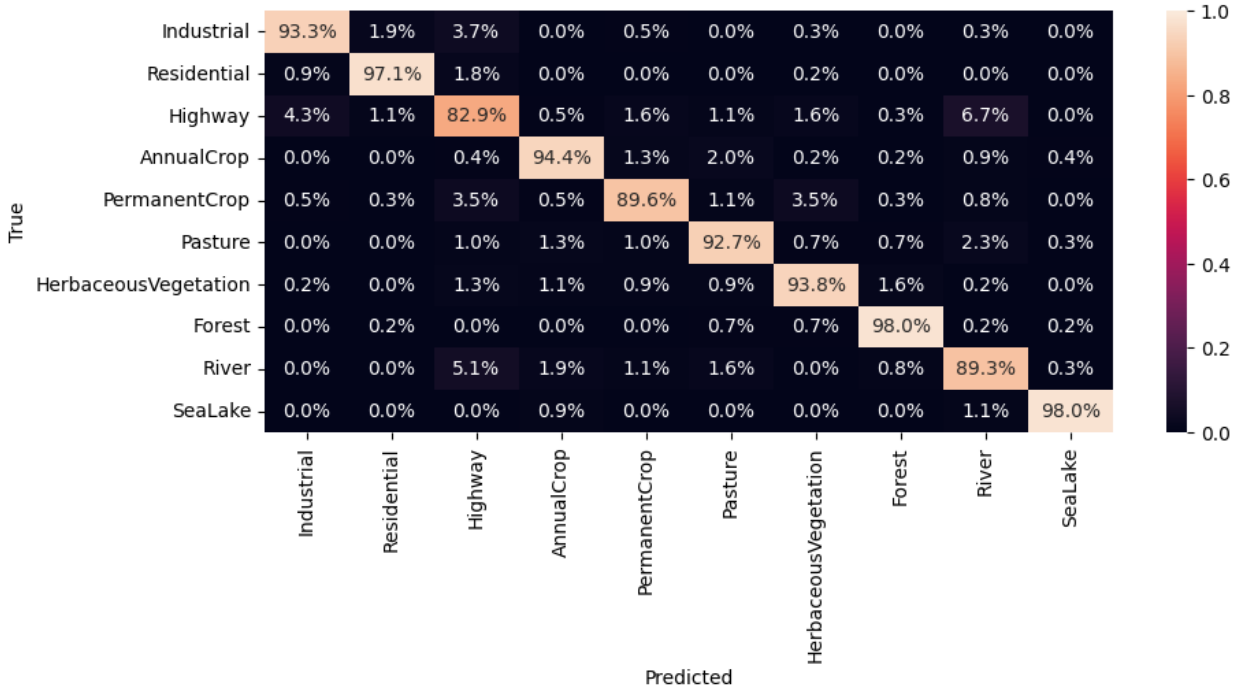


Figure 13. Confusion matrix for the basic-feature XGBoost Model on test set

4.5 Generalizability and Improvements

4.5.1 Discussion on Generalizability

In summary, we split the dataset into training, validation and test groups using a 70%/15%/15% scheme. We used the training dataset to fit the two classifiers and performed the hyperparameter search with the validation set to avoid overfitting. Lastly, we used the best-performing model configuration in terms of validation accuracy to predict the labels of the test dataset before reporting the classification results. The validation accuracy (95.2%) and test accuracy (95.4%) of this model were nearly equal, indicating that it should be expected to work consistently well on previously-unseen images. Thus, there is good evidence to conclude that we achieved solid generalizability in this study.

4.5.2 Potential Improvements

Based on the discussion on the model results and limitations above, here are a few areas of further investigation which may help improve classifier performance on EuroSAT:

1. Since “Highway” and “River” classes are often confused with each other, it could be worth exploring image processing techniques that can detect and quantify the widths of patterns that appear, since rivers are often much wider than highways. Edge detection algorithms or Fourier transformations may be helpful here.
2. In our study, we realized that pre-trained model embeddings played a vital role in improving our model’s accuracy, so there’s every reason to think that pursuing other transfer learning avenues may yield performance gains. Some options here include fine-tuning the pre-trained EfficientNetV2 model instead of extracting embeddings from it, or testing a few other pre-trained models from different families.

5. References

1. P. Helber, B. Bischke, A. Dengel and D. Borth, "EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification," in IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 12, no. 7, pp. 2217-2226, July 2019, doi: 10.1109/JSTARS.2019.2918242.
2. ESA-Sentinel2 (https://www.esa.int/Applications/Observing_the_Earth/Copernicus/Sentinel-2)
3. Tan, M., & Le, Q. (2021). EfficientNetV2: Smaller Models and Faster Training. In M. Meila & T. Zhang (Eds.), Proceedings of the 38th International Conference on Machine Learning (Vol. 139, pp. 10096-10106). PMLR. Retrieved from http://proceedings.mlr.press/v139/tan21a/tan21a.pdf
4. Land Use/Land Cover Classification (https://www.earthdata.nasa.gov/topics/land-surface/land-use-land-cover/land-use-land-cover-classification)
5. Significance of Land Use/Land Cover (LULC) Maps (https://satpalda.com/blogs/significance-of-land-use-land-cover-lulc-maps/)
6. UMAP Documentation (https://umap-learn.readthedocs.io/en/latest/)