

Model 28 Final Report

Sai Chamarty, Thomas Liu, Karen Lui, Taylor Sambajon, Jason Zhao

I. INTRODUCTION AND BACKGROUND

Autonomous robots and intelligent agents are increasingly deployed in a variety of real-world contexts—ranging from search-and-rescue missions in unstructured wilderness to delivery drones navigating urban streets. In all of these scenarios, a fundamental perceptual challenge is *environment classification*: the ability to correctly identify whether a given visual scene represents a natural landscape (e.g., forest, mountain, glacier, sea) or a human-made setting (e.g., buildings, street). Accurate environment classification enables a mobile robot to adjust locomotion parameters (e.g., walking gait on uneven terrain vs. rolling on pavement), select appropriate sensors, or choose safe waypoints to minimize obstacles. For instance, a humanoid robot operating outdoors must recognize when it transitions from a paved walkway into grass or loose gravel so that it can modify its balance and joint stiffness dynamically.

While modern deep learning techniques—particularly convolutional neural networks (CNNs)—have produced state-of-the-art results on large, labeled image datasets, many published works focus on controlled satellite or aerial imagery. In practice, a robot’s camera may capture highly variable scenes: changing lighting conditions, partial occlusions (e.g., tree branches, parked cars), and diverse camera angles (e.g., low-angle shot from a kneeling robot). These factors can significantly degrade the performance of models trained on clean, curated photos. Therefore, it is important to compare simpler, more interpretable baseline methods (such as Logistic Regression and shallow multilayer perceptrons) against deeper CNN architectures to understand how much real-world variability influences classification accuracy in each case.

In this project, we use the Intel Image Classification Dataset—consisting of approximately 24,000 balanced RGB images at 150×150 resolution, evenly split into six classes: buildings, forest, glacier, mountain, sea, and street. Our primary objectives are:

- 1) **Baseline Evaluation:** Train a Logistic Regression model (on flattened pixel vectors) and a shallow MLP (with a single hidden layer) to establish lower bounds on performance, while quantifying how linear vs. mildly nonlinear methods cope with color- and texture-based distinctions.
- 2) **Deep CNN Development:** Design and train a convolutional neural network that learns spatial filters to detect edges, patterns, and region-level features. We will compare architectures of varying depth and filter sizes to determine the best accuracy–complexity trade-off.
- 3) **Generalization Analysis:** Perform extensive exploratory data analysis (EDA) to understand

class distributions, channel statistics, and typical failure modes. We will then measure how well each model generalizes to held-out validation data and user-provided test images, highlighting scenarios in which deeper networks outperform or underperform relative to simpler baselines.

- 4) **Deployment Demonstration:** Package the top CNN model into a lightweight Flask web interface. Users (or a robot’s camera feed) can upload a new environment image and immediately receive a predicted label with confidence. This real-time demo showcases how a robot might query a live scene classification service to inform navigation decisions.

Our contributions include both an empirical comparison of three classification paradigms (linear, multi-layered perceptron, deep CNN) on a realistic, balanced dataset, and a demonstration of deployment feasibility in a web-based interface. By examining classification errors—such as confusing mountain and glacier scenes when snow covers rocky outcrops—we will identify key challenges in color and texture ambiguity. Finally, we discuss how our findings can inform future work on lightweight on-board inference for embedded robotics.

The remainder of this report is organized as follows. Section II reviews related work in environment and scene classification, highlighting existing CNN architectures and feature-extraction techniques. Section III describes the Intel dataset and presents our EDA results. Section IV details feature selection for Logistic Regression and MLP baselines, followed by CNN architecture, training procedures, and hyperparameter tuning. Section V compares model performance, analyzes confusion matrices and failure cases, and presents qualitative examples. We conclude in Section VI with a summary of key findings and directions for future research.

II. LITERATURE REVIEW

A. Machine Learning in Environment Processing

The paper “Prediction and Processing of Environment Geography Images using Deep Learning Techniques” [1] evaluates a custom CNN against an ImageNet-based model on a four-class satellite dataset (land, water, cloud, green area). On the controlled dataset (5,603 images), the CNN scored 90.6% accuracy and the ImageNet model 99.5%, but both dropped to 70% on real-time images, highlighting generalization challenges in real-world scenarios. This directly supports our inclusion of both shallow and deep CNNs—and underscores our need to validate against user-uploaded photos to ensure robust performance.

B. Classification with Small Convolutional Neural Networks

Pazoki & Farokhi [2] employ a small neural architecture mixing MLP and CNN layers to distinguish five rice-grain varieties. After segmenting grain from the background, they extracted color (RGB, HSV, YCbCr) and morphological features (area, axis lengths), achieving over 95% accuracy on their controlled dataset. As noted above, strong preprocessing and feature engineering can significantly boost performance, suggesting we should similarly explore color- and shape-based augmentations before feeding images into our baselines.

C. Larger Networks and Overfitting

Nocentini et al.'s [3] MCNN15 model—fifteen 3×3 convolutional layers grouped into three blocks—reaches 94% on Fashion-MNIST but only 40–60% on more varied datasets, illustrating the trade-off between depth and dataset complexity. They also convert some inputs to grayscale to reduce channel dimensionality. As with the geography study, real-world variability hurts deep networks, and channel reduction (e.g., RGB → grayscale) may be a practical contingency if full-color models overfit or struggle to generalize.

III. DATASET DESCRIPTION AND EXPLORATORY DATA ANALYSIS

A. Dataset Overview

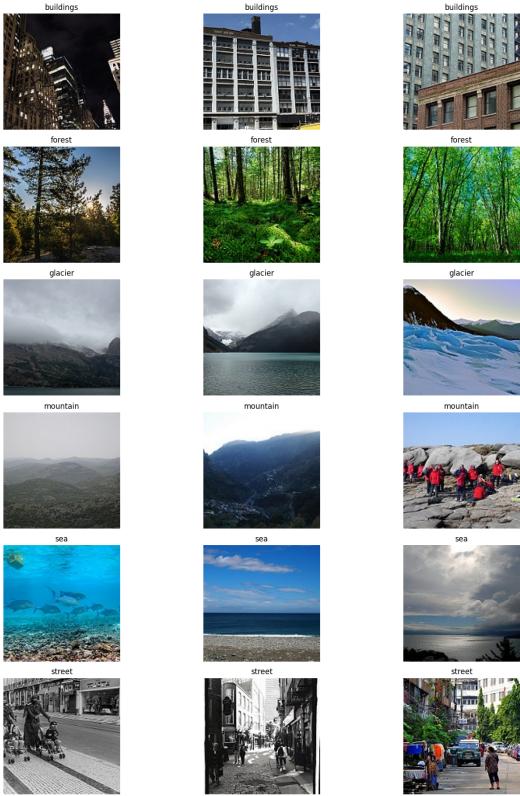


Fig. 1. Overview of classification

Our project involves the classification of images into classes based on the location they depict. The dataset contains thou-

sands of images belonging to one of six distinct classes: buildings, forest, glacier, mountain, sea, and street. These class labels are given for images in the training dataset, and by listing a few examples from each class (see Figure 1), we can obtain an overall understanding of the problem structure and the project's main objectives.

B. Dataset Structure and Distribution

1) Data Formatting: In the dataset overview, we observed that most of the images are of the same size (150x150x3). In other words, the majority of the dataset is formatted the same: 150x150 images with 3 channels for red, green, and blue (RGB). Thus, we decided to use these images as the "default" format for pre-processing. We convert each image into a usable data point by flattening it into a 1D vector of its RGB values.

- X is the feature matrix. It's a NumPy array of shape (N, 67500), where:
 - N is the total number of images you loaded.
 - 67500 comes from flattening each 150x150x3 RGB image into a one-dimensional vector $150 \times 150 \times 3 = 67500$.
 - Each row of X corresponds to one image and contains its pixel values in the order: $[R_1, G_1, B_1, R_2, G_2, B_2, \dots, R_i, G_i, B_i]$ where each triplet is the red, green, and blue channels of a single pixel.
- y is the label vector. It's a NumPy array of shape (N,), where each entry is an integer in $[0 \dots 5]$ indicating which class that image belongs to:
 - For example, if


```
classes = ['buildings', 'forest',
                     'glacier', 'mountain', 'sea',
                     'street']
```

 then a y value of 2 means 'glacier'.

So in effect, there is flattened feature vector per image in X, and the matching class index in y. Together, $(X[i], y[i])$ is the feature-label pair that will be fed into Logistic Regression or MLP models.

2) Outliers and Special Cases: Although most images in the training set follow the 150x150x3 format, not all of them do. It is very important to identify any such images, as they must be resized during pre-processing. If not, this would lead to data points with the incorrect dimensions. For example, a rectangular 150x100x3 image flattens out into a 45000-feature input vector, while a grayscale 150x150x1 image flattens out into 22500-feature input vector. Both of these cases would cause the feature matrix – and the entire classification process – to crash due to a shape mismatch. To prevent this, we list out every unique image size in the training set and identify any images that will require special handling.

Image counts by size:

- 150x150x3: 13986
- Other (150x113x3, 150x135x3, 150x111x3, etc.): 48

3) *Class Distribution:* It is also advisable to perform an analysis on the training set's demographics. It is important to check that no image class has significantly more instances than any of the others. If an image classifier is trained too heavily on one class of image, unnecessary bias can be introduced. The class distribution, seen below, shows that there are no significant differences in the number of images in each class. We may therefore use the full dataset when training our models.

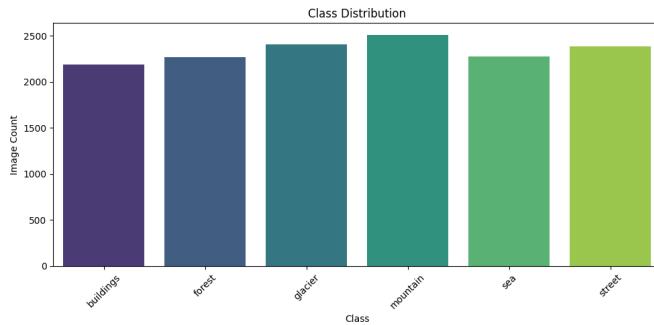


Fig. 2. Class distribution of images

TABLE I
IMAGE COUNTS BY CLASS

Class	Count
Buildings	2191
Forest	2271
Glacier	2404
Mountain	2512
Sea	2274
Street	2382

C. RGB and Vibrance Analysis

In most digital images, the color of each pixel is determined by 3 RGB values, corresponding to the amount of red, green, and blue light to display. The mean of the 3 RGB values determines the overall intensity of the pixel to the viewing eye, which is known as vibrance. An image is comprised of 3 color channels corresponding to red, green, and blue. By isolating each of these channels, we can see only the red, green, or blue light in an image. Both the vibrance and the relative intensity of the color channels in an image can lend some interesting insight into what class the image might belong to.

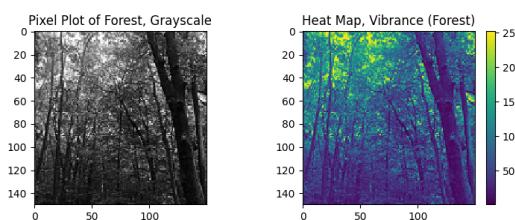


Fig. 3. Vibrance analysis

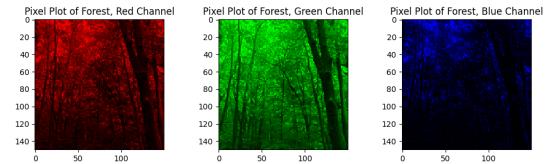


Fig. 4. RGB channels

We can observe some rough trends from these pixel plots of RGB and vibrance, especially when taking preexisting domain knowledge into consideration. For instance, we could infer that forest images, like the one depicted above, have more green (due to the abundance of plants and trees) and less vibrance and blue (due to the trees blocking out the sky). We might also assume that sea images have the most blue (for obvious reasons), or that mountain images show a gradient in vibrance from bottom to top (mountain rock on the bottom, sky on the top). Of course, not all images in each class will follow these trends, but this analysis serves as a basic proof of concept of how images could be classified according to their RGB values.

A more mathematically robust insight into RGB-related trends can be derived from statistical analysis. One of the easy places to start would be the mean and standard deviation of red, green, and blue across all images in one class. By plotting these values for each class, we can observe some more general patterns about RGB values compared to the semantic analysis from earlier.

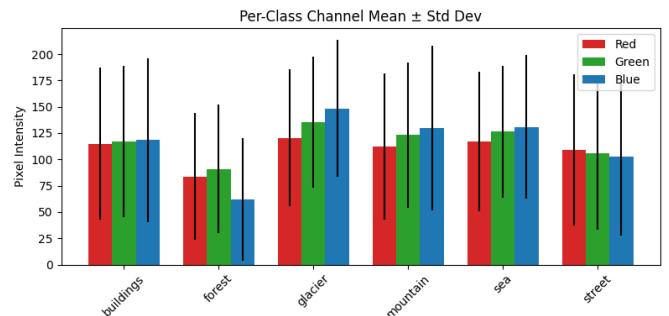


Fig. 5. Statistical analysis of RGB

These bar graphs help corroborate some of our earlier observations, especially regarding forest images: less vibrance overall, with green being the most prominent color and blue being the least prominent. Other trends can also be seen:

- Glacier images tend to have the most vibrance. This may be because they are mostly white in color (red, green, and blue all at maximum).
- Interestingly, sea images do not have the most blue or the highest proportion of blue among all classes. This goes against what we had predicted during our earlier analysis.
- Buildings and street images have the highest proportion of red. This makes sense, as red is generally not a common color in natural settings.

- Buildings and street images also have the highest variance across all color channels, which is also reasonable, given that they tend to encompass a much wider range of colors.

And so on. These RGB-related trends show the numerical patterns in the image data that a classification model might pick up on during training, and by conducting statistical analysis on RGB values, we are able to achieve a more complete understanding of this process.

IV. PROPOSED METHODOLOGY

The three approaches we used for classifying our Intel Image dataset were Multinomial Logistic Regression, Multi-Layer Perceptron, and Convolutional Neural Network. We will be evaluating and comparing the performance of these three algorithms in regards to accuracy to identify which model is most accurate in classifying image classes of buildings, forest, glacier, mountain, sea, and street.

A. Logistic Regression

In this section, we describe our approach for creating our Multinomial Logistic Regression (MLR) model.

MLR similar to standard Logistic Regression requires our input data to be a one-dimensional vector. Since our Intel Image dataset contains images associated with one of the six environmental classes, we had to first preprocess the image data. Due to memory and hardware limitations, unlike the other two models, we were compromised to resize our images to 64x64 dimensions instead of 150x150. We flattened the image data into 64x64x3 pixels in a 1-D vector and normalized the pixel values between [0,1] by dividing our 1-D vector by 255.

Since we work with categorical image data, we utilized label encoding to represent our categorical outcome variable with numbers between 0-5. Once we converted our labels into numbers, we split our dataset into 64% for training data, 16% for validation data, and 20% for testing data. After splitting our dataset, we one hot encoded our target variables.

Our MLR model was implemented using a simple neural network consisting of an input layer taking in our 1-D flattened image vectors and a single dense output layer with the softmax activation function generating the probability distribution of our six labels. The model is then configured using Stochastic Gradient Descent (SGD) as it's optimizer for finding the optimal weights associated with our features and to minimize our categorical cross-entropy loss function using the hyperparameters: learning rate, regularization type, regularization value, batch size, and momentum.

To find the optimal hyperparameter values for our model we performed a grid search on these values:

The chosen set of values were constrained to reduce training time due to hardware limitations while also maintaining a broad enough of hyperparameter values to configure our model with. The grid search ran 3 fold cross-validation on every combination of parameters.

Once the optimal hyperparameters were chosen, we trained our model on these values for 100 epochs tracking the accuracy

TABLE II
HYPERPARAMETER GRID FOR MODEL OPTIMIZATION

Hyperparameter	Values Explored
Learning rate	{0.001, 0.0001}
Regularization type	{L1, L2}
Regularization strength (λ)	{0.01, 0.1}
Momentum	{0.0, 0.5, 0.9}
Batch size	{16, 32, 64}

and loss for our training and validation data obtaining the final accuracy from our model.

B. Multilayer Perceptron (MLP)

This section will discuss our methodology and approach to our Multilayer Perceptron model (MLP).

Beginning with our input, this model also takes the preprocessed image data as-is (150x150x3 flattened into a vector). This data format is ideal for our MLP, since our MLP cannot take into account any spatial hierarchies in the images. Instead, it uses the value of each individual pixel as a feature to train on. The data is then split into training and testing sets in an 80:20 ratio, and a 2-layer MLP is constructed.

The MLP model itself consists of four layers- one input, one output, and two hidden. Each layer excluding the output layer uses ReLU activation and includes a dropout layer, while the output layer uses the softmax activation function. The input layer begins with 256 neurons, with each subsequent training layer halving that amount until our final output layer with 6 neurons. The exact dimensions of the model can be seen in the provided table below:

TABLE III
SHOWN: BASE MODEL FOR MLP CONSTRUCTION.

Layer (type)	Output Shape	Param #
Input (Dense)	(None, 256)	17,280,256
dropout (Dropout)	(None, 256)	0
Hidden 1 (Dense)	(None, 128)	32,896
Hidden 1 (Dropout)	(None, 128)	0
Hidden 2 (Dense)	(None, 64)	8,256
Hidden 2 (Dropout)	(None, 64)	0
Output (Dense)	(None, 6)	390

Once the base model was created, we moved onto hyper-tuning parameters. The initial plan was to use a tuning grid to find the best combination of parameter values, however due to computational and time limitations this had to be done by hand. The data was split in half in order to speed up the tuning section, and four models, including the base model, were created. The parameters tuned for this model were the **dropout rate** and the **learning rate**, with the final model having adjusted epochs as well as batch size to account for the size of the entire dataset. Upon the completion of all models, we then proceeded to graph the loss and accuracy of each one in order to visualize the best model.

C. Convolutional Neural Network (CNN)

The methodology follows a structured pipeline beginning with data preparation and ending with model evaluation and visualization of results.

First, we loaded our preprocessed image data, which had been resized and normalized to ensure uniformity. Initially, the images were stored as flattened vectors; however, since CNNs require images in their original spatial format, we reshaped these vectors into three-dimensional tensors corresponding to the height, width, and color channels of each image. This step is important for allowing the convolutional layers to effectively extract spatial features.

Next, the dataset was divided into training, validation, and testing subsets. The training set is used to fit the model, while the validation set helps monitor the model's performance during training, allowing for adjustments to prevent overfitting. The test set, which remains unseen during training and validation, is reserved for the final unbiased evaluation of the model's accuracy.

TABLE IV
CNN ARCHITECTURE SUMMARY

Layer	Param #
Input (150x150x3)	0
Conv2D (32 filters)	896
MaxPooling2D	0
Conv2D (64 filters)	18,496
MaxPooling2D	0
Conv2D (128 filters)	73,856
Flatten	0
Dense (128 units)	4,735,104
Output (6 units)	774
Total:	4,829,126

The CNN architecture was then constructed using several key layers. Table IV shows the layout of the CNN model. It begins with convolutional layers that apply multiple filters to detect important features such as edges and textures. These are followed by max-pooling layers that reduce the spatial dimensions, helping to lower computational load and reduce overfitting by summarizing feature presence. After multiple convolutional and pooling blocks, the feature maps are flattened into a one-dimensional vector and passed through fully connected dense layers. The final output layer uses a softmax activation function to generate probabilities corresponding to each of the six environment classes, enabling multi-class classification.

To train the model, we compiled it with the Adam optimizer, a widely used algorithm that adjusts weights efficiently to minimize classification errors. The loss function selected was sparse categorical cross-entropy, appropriate for integer-labeled multi-class problems. Training continued for a fixed number of epochs, during which the model's accuracy and loss were tracked on both training and validation sets to ensure steady improvement and detect potential overfitting.

Once training was complete, we plotted training curves showing how accuracy evolved over epochs, which helps assess convergence and training stability.

V. EXPERIMENTAL RESULTS AND EVALUATION

A. Logistic Regression

From what was previously mentioned in the methodology section, our multinomial logistic regression model (MLR) utilized grid search to find the optimal hyperparameter values for our model and trained our model using stochastic gradient descent tracking the accuracy and loss on the training and validation data. Due to hardware limitations, we fixed our training to 100 epochs.

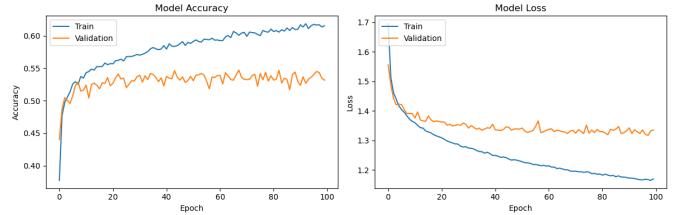


Fig. 6. Training and validation accuracy (left) and loss (right) over 100 epochs using categorical cross-entropy as the loss function

From Figure 6, we can see that in both model accuracy and model loss graphs, there's a lot of fluctuations between every epoch which is expected as we are using a stochastic gradient descent approach for optimizing our weights associated with our features. Examining our model accuracy graph, the training accuracy was steadily improving throughout the epochs, while the validation accuracy started to plateau 60 epochs in. At epoch 100, we see the training (62%) and validation accuracy (53%) converging as both of their curves are flat and space between these two lines indicating no overfitting from our training. Similarly in Figure 6 for our model loss graph, our training and validation loss curves decrease steadily with both curves flattening out at epoch 100 with a loss of 1.4 from our validation curve and 1.2 for our training curve.

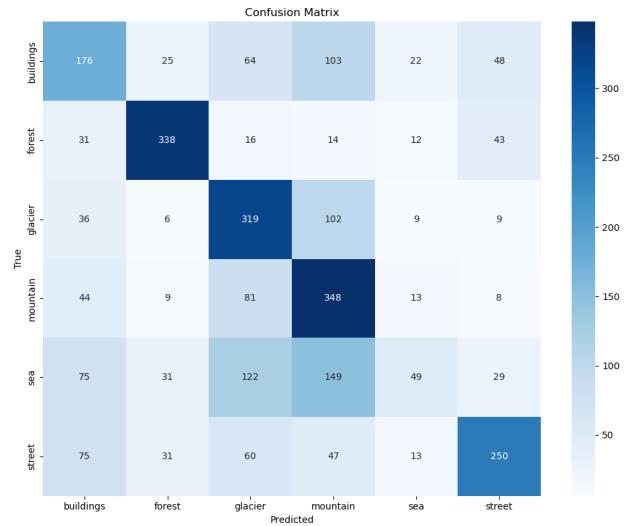


Fig. 7. Confusion Matrix of our six environment classes

TABLE V
PRECISION, RECALL, AND F1-SCORE FOR EACH CLASS

Class	Precision	Recall	F1-Score
Buildings	0.40	0.40	0.40
Forest	0.77	0.74	0.76
Glacier	0.48	0.66	0.56
Mountain	0.46	0.69	0.55
Sea	0.42	0.11	0.17
Street	0.65	0.53	0.58

From Table V and Figure 7, our model achieved the highest precision, recall, and F1-score for the forest class with respective scores of 0.77, 0.74, and 0.76, indicating confident predictions. In contrast however, our model had trouble identifying the sea class recording scores of 0.42, 0.11, and 0.17 for the precision, recall, and F1-score respectively. From Figure 7, an actual sea image was commonly predicted as a glacier (122 times) and a mountain (149 times) which seems logical since these classes share many of the same color distribution. This is further supported by the data in Figure 7 where there are 81 misclassifications of mountains as a glacier image and 102 of misclassifications of glaciers as a mountain image.

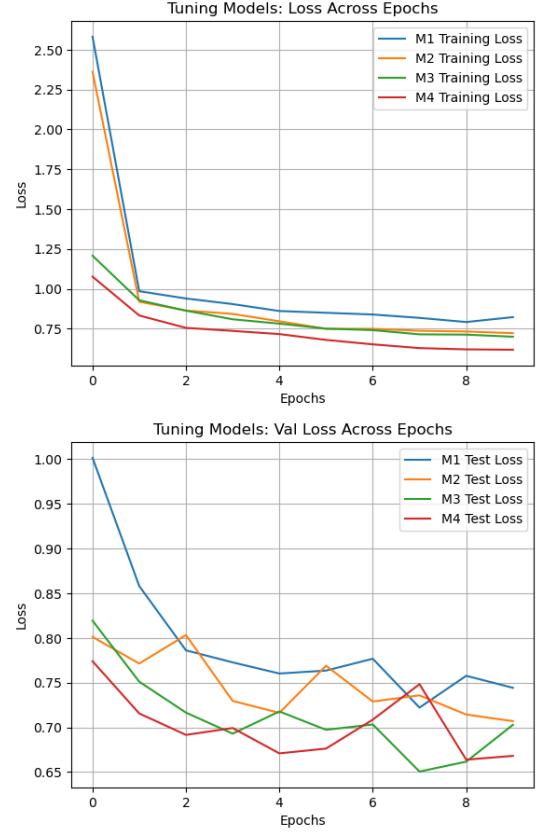


Fig. 8. Tuning loss over 10 epochs.

B. Multilayer Perceptron (MLP)

Each model in our MLP was trained using a different combination of dropout rates and learning rates on a tuning dataset that was half the size of the original dataset. The table below shows the rates used for each model. As mentioned

TABLE VI
MODELS AND TUNING PARAMETERS

Model	Dropout Rate	Learning Rate
Model One (Default)	0.2	0.001
Model Two	0.1	0.001
Model Three	0.2	0.0001
Model Four	0.1	0.0001

before, our intention was to tune the MLP model using a tuning grid, but space and time constraints limited us to hand tuning. The above rates were the values we ultimately decided to tune on. Once tuning was done for each model, we graphed the loss and accuracy in separate graphs to visualize which model had the best performance and to determine which one would become our final model.

The figure above shows how each model's loss compares to one another. Notably, the default model (model One) in blue has marginally the highest loss compared to the more tuned models. There is a sharp decline in loss from epoch one to epoch two in models One and Two, whereas models Three and Four have a smoother loss convergence. The validation loss for each model does not model the smoother patterns of the loss itself, but it is still worth noting that model One has the highest loss and validation loss. Models Two and Three have very similar loss patterns between each other, and model Four has the lowest loss overall.

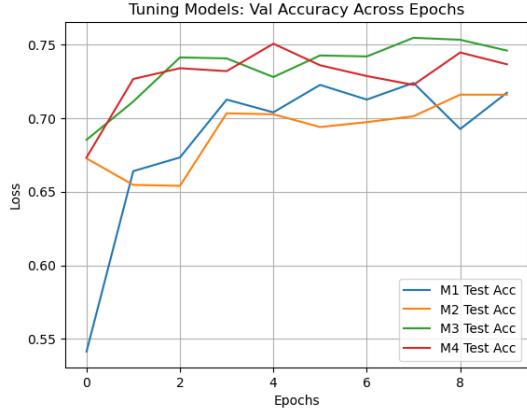
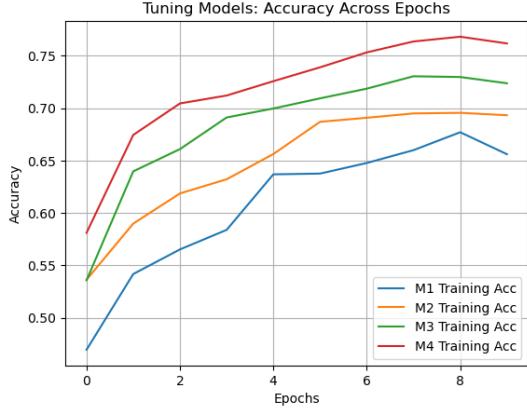


Fig. 9. Tuning accuracy over 10 epochs.

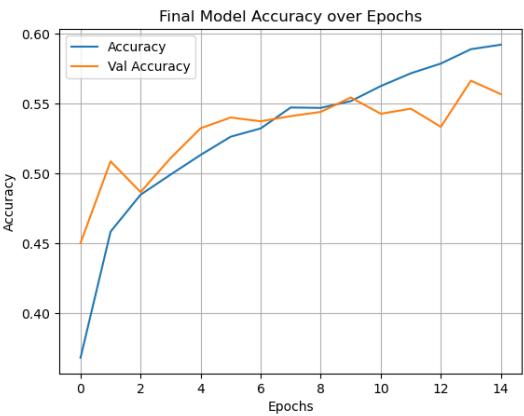
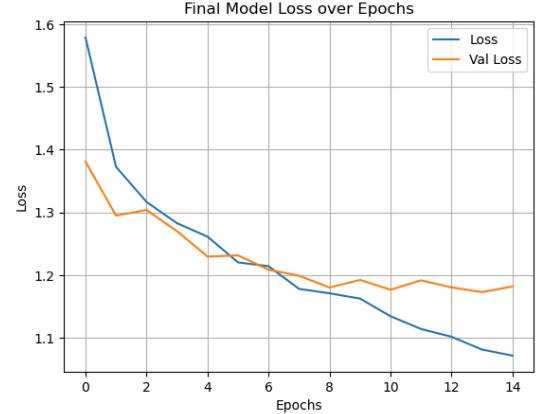


Fig. 10. Final model loss and accuracy over 15 epochs.

The figures above shows how each model's accuracy compares to one another. As expected, the untuned default model One has the lowest accuracy and validation accuracy among all of the models trained. The accuracy itself appears to gradually increase with each new model and ends with model Four being the most accurate. Interestingly, model Two's validation accuracy ended up around the same as model One, and models Three and Four appear to almost alternate with model Three having the highest validation accuracy. Ultimately though, model Four wound up with the highest accuracy and the lowest loss as a combination of both model Two's dropout rate and model Three's learning rate.

With this data, we were able to create our final MLP model with a dropout rate of **0.1** and a learning rate of **0.0001**. Since prior testing with grid tuning (not included) showed that training on larger datasets resulted in slower convergence, the final model was trained using 15 epochs instead of 10, and a batch size of 25 instead of 30. The change in batch size was largely due to memory constraints despite a smaller batch size also meaning slower training.

With our final model trained, we were then able to visualize the loss and accuracy as depicted in Figure 10.

At 15 epochs, the final model showed decent convergence for both accuracy and loss. The final accuracy hovered around 55-60%, which was expected of our MLP since it cannot take into account spatial hierarchies. Note that the final accuracy and loss for our model began to diverge from the validation at around epoch 9. This implies that despite being slower to converge and slower to train on a larger dataset, 15 epochs may have resulted in overfitting. For the final step in our visualization, we generated a confusion matrix as seen below. In this matrix, the categories $\{0, 1, 2, 3, 4, 5\}$ correspond to the classes $\{\text{building}, \text{forest}, \text{glacier}, \text{mountain}, \text{sea}, \text{street}\}$.

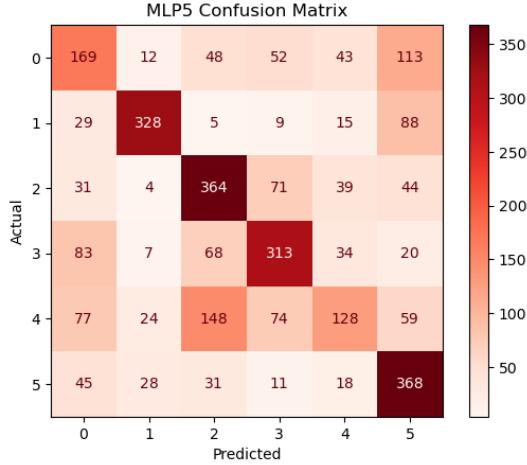


Fig. 11. Final MLP confusion matrix.

The confusion matrix in Figure 11 demonstrates that our model can identify glacier and street images with relative confidence, but it struggles with properly identifying images that should be classified as **sea** images or **building** images. It labeled a majority of the **sea** images as **glacier** images during testing, but as noted in our discussion of our dataset, both "sea" and "glacier" images have higher B-channel values than other images. It is also important to consider that street images have the most unique RGB values compared to the other classes in our dataset, and a lack of ability to account for spatial hierarchy leads to an over reliance on RGB pixel values to classify images. This is the most probable point of confusion for our MLP model.

C. Convolutional Neural Network (CNN)

To assess the performance of the convolutional neural network (CNN) on natural environment classification, we trained the model on the preprocessed Intel Image Dataset and monitored both accuracy and loss across training epochs. The CNN architecture, consisting of three convolutional-max pooling blocks followed by dense layers and dropout regularization, was trained using the Adam optimizer and early stopping to mitigate overfitting. Model checkpointing was used to retain the version with the highest validation accuracy during training.

The results are visualized in Figure 12 and Figure 13, which shows the accuracy and loss curves over 9 epochs before early stopping was triggered.

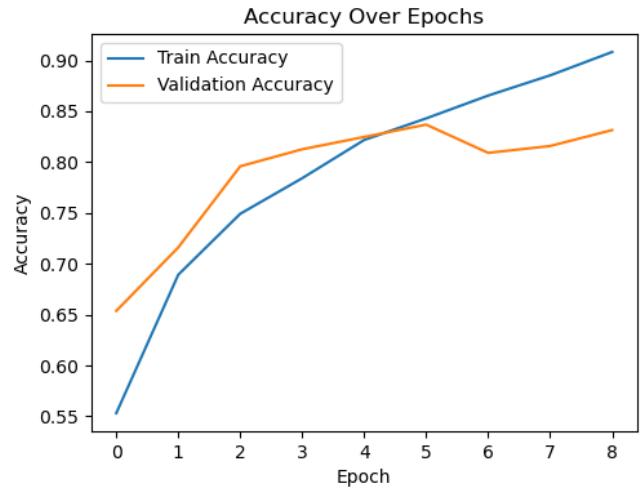


Fig. 12. Training and validation accuracy over 9 epochs.

As observed in Figure 12, training accuracy steadily improved and reached around 91%, while validation accuracy reached around 84%. This indicates improved generalization performance, although slight fluctuations in the validation accuracy may indicate mild overfitting in later epochs.

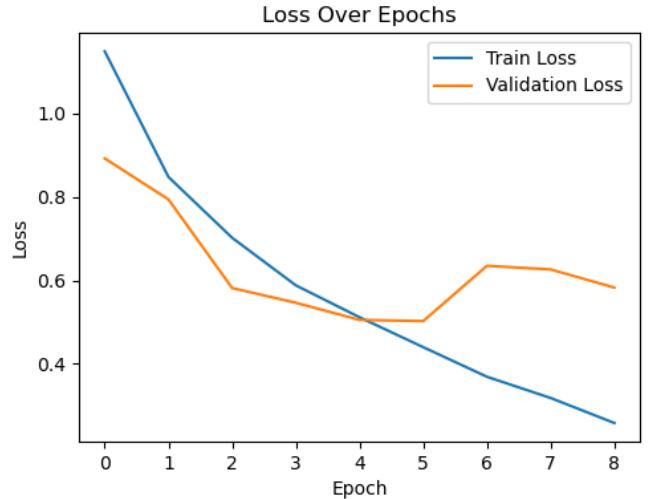


Fig. 13. Training and validation loss over 9 epochs.

Figure 13 shows a consistent decline in training loss, while validation loss fluctuated, suggesting the model began to overfit slightly in later epochs. Despite this, the early stopping mechanism ensured that the best-performing model was preserved.

Table VII presents the precision, recall, and F1-score for each class in the classification task. The forest class achieved the highest performance across all metrics, with a precision of 0.97, recall of 0.96, and F1-score of 0.96, showing strong and consistent classification accuracy. The buildings and street classes also performed well, with F1-scores of 0.83 and 0.86, respectively, suggesting reliable detection for these categories.

TABLE VII
PRECISION, RECALL, AND F1-SCORE FOR EACH CLASS

Class	Precision	Recall	F1-Score
Buildings	0.89	0.78	0.83
Forest	0.97	0.96	0.96
Glacier	0.74	0.80	0.77
Mountain	0.77	0.78	0.77
Sea	0.83	0.81	0.82
Street	0.83	0.89	0.86

In contrast, the glacier and mountain classes had slightly lower performance, with F1-scores of 0.77 for both. There may be room for improvement in distinguishing the glacier and mountain classes from others. The sea class demonstrated moderate performance, with an F1-score of 0.82, reflecting decent but not exceptional accuracy.

Overall, the model excels in classifying natural scenes like forests but may require further refinement for classes with similar visual features, such as glaciers and mountains.

VI. CONCLUSION AND DISCUSSION

After completing the training and evaluation of all three models, we observed that the convolutional neural network produced the best results in image classification. The CNN comprehensively outperformed the other two models across all image classes and evaluation metrics.

There are several possible reasons for this. For one, both the logistic regression and multilayer perceptron models operate on flattened, 1D input vectors, which fail to preserve the spatial and geometric features that generally define each class of image. As such, they are reduced to fitting to the general RGB values instead, making them especially prone to wrongly classifying images with similar RGB distributions. In particular, both models had trouble correctly classifying sea images, and we can see from Figure 5 that sea images do not have a very distinct RGB distribution. In addition, sea images tend to be fairly uniform in terms of shape, making them more similar on average to all other image classes.

This is where a convolutional network emerges as the superior model. Unlike the other two models, a CNN takes a full image as input rather than a flattened 1D vector, allowing it to operate on the image's actual shape, not just its RGB values. Because of this unique design and architecture, a CNN is able to detect spatial features in images like edges, corners, and textures, which are generally more indicative of an image's class than its RGB distribution. This advantage allowed our CNN to achieve much improved performance overall in the classification process, particularly with the sea images that the other models had struggled on. That said, it saw less significant gains in separating glacier and mountain images, in part due to their similar shape and profile (see section V-C). In their case, it may be that RGB values are still of some importance, particularly as glaciers and mountain images do have slightly different RGB distributions.

In any case, regardless of how well it performed relative to the other models, our CNN retains some room for

improvement. The training methodology could, for instance, benefit from improved preprocessing, feature engineering, and even model selection. The model fitting process is, as always, capable of being refined through hyperparameter tuning, cross-validation, and training the model on more data. We leave any further research on this matter to the reader's discretion.

VII. GITHUB LINK, PROJECT ROADMAP, AND TEAM ROLES

GitHub Repository and Demo Video

Our complete project codebase and documentation are available on GitHub. A live demonstration of the deployed application can be viewed on YouTube.

GitHub Repository: <https://github.com/Model-28/project>

Demo Video: https://youtu.be/trj_Kn91iGM

Roadmap:

- **Week 1 (4/28–5/02):** Project setup, environment sync, dataset download, folder structure & script stubs; complete data preprocessing and initial EDA.
- **Week 2 (5/05–5/09):** Train and tune baselines—Logistic Regression (Jason) and MLP (Taylor)—on flattened pixels; generate metrics, confusion matrices, and summary plots.
- **Week 3 (5/12–5/16):** Design, implement, and train initial CNN architectures (Karen); compare filter depths and capture training/validation curves.
- **Week 4 (5/19–5/23):** Finalize all three models—Logistic Regression, MLP, and CNN—through hyperparameter tuning and data augmentation; consolidate best-performing versions.
- **Week 5 (5/26–5/30):** Draft and polish report sections (Methods, Results, EDA, Feature Selection); integrate figures and tables; prepare demo outline.
- **Week 6 (6/02–6/06):** Record and refine the 5-minute demo video (all members); finalize LaTeX report edits, GitHub repository, and submit deliverables.

Team Roles:

- **Sai Chamarty (Team Lead):** Led meetings, managed roadmap, handled code integration, built the Flask demo, and oversaw the report and edited demo video.
- **Jason Zhao (LogReg Engineer):** Built and tuned the Logistic Regression baseline with performance metrics and plots.
- **Taylor Sambajon (MLP Engineer):** Developed and evaluated the MLP model, including training curves and confusion matrices.
- **Karen Lui (CNN Lead):** Designed and trained CNNs; optimized depth and filter parameters.
- **Thomas Liu (EDA & Data Prep):** Built preprocessing pipeline and performed EDA on class balance and channel stats.

REFERENCES

- [1] R. L, V. P, S. D, N. T, N. M, and P. Yaswanth, "Prediction and processing of environment geography images using deep learning techniques," in *2023 8th International Conference on Signal and Image Processing (ICSIP)*, 2023, pp. 1–5. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/10370035>
- [2] S. Pazoki and S. Farokhi, "Classification with small convolutional neural networks," in *Proc. 27th Iranian Conf. Electr. Eng. (ICEE)*, Yazd, Iran, 2019, pp. 1–6. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/8776982>
- [3] F. Nocentini, F. Biondi, M. Magherini, and D. Bertini, "Larger networks and overfitting: A case study with mcnn15 on fashion-mnist and more varied datasets," *Sensors*, vol. 22, no. 23, p. 9544, 2022. [Online]. Available: <https://www.mdpi.com/1424-8220/22/23/9544>
- [4] S. Chamarty, T. Liu, K. Lui, T. Sambajon, and J. Zhao, "Project: Image classification for environment geography images," <https://github.com/Model-28/project.git>, accessed: 2025-06-06.