# Classification of Natural Scene Images Using Deep Learning

DNSC 4280: Machine Learning

Tsion Temesgen - tgtemesgen03@gwmail.gwu.edu
Jiwon Yun - jiwon78@gwmail.gwu.edu
Acadia Grenier - acadiagrenier@gwmail.gwu.edu

Professor Srinivas Prasad
George Washington University
December 14, 2025

## 1. Abstract

Image classification of natural scenes is a crucial task in applications such as geographical landmark labeling, image-based retrieval systems, and eco-scientific analysis. This paper proposes to investigate the performance of deep learning models for image classification into six classes: Buildings, Forest, Glacier, Mountain, Sea, and Street. Utilizing a database of 25,000 images, the performance of two convolutional neural network models, namely ResNet50 and EfficientNet-B0, with application of transfer learning, data augmentation, and ensembling, is explored. Moreover, a Language Model (LLM) is also utilized for additional semantic deductions. Even though models accessed inherent patterns in images by means of deep learning approaches, the issue of accurate image categorization for unknown instances remains, with low classifications of about 17.5%, since the models face issues discerning between visually similar images, as well as issues of imbalance in the classes. However, the LLM showed accurate classifications of 91.18% for small data, indicating its effectiveness for semantic support purposes but not for image analysis purposes. The findings of this paper suggest prominent upcoming proceedings for improvement of image classifications by means of advanced data augmentations, application of techniques such as ensemble learning, use of attention techniques, or implementation of transformers for enhanced image classifications.

## 2. Introduction

Classification of natural image scenes is important for various applications such as geographic labeling, image search based on image contents, and other environmental observations. Incorrect image classification reduces the reliability of various applications that often require accurate image classifications. This research work aims to study the effectiveness of deep learning models

for image classification into six classes: Buildings, Forest, Glacier, Mountain, Sea, and Street. The key research aim is to test the efficiency of deep learning models on accurate image classifications with good performance on unseen data. This is clearly a crucial step for improving the dependability of geographic tagging systems, image-based retrievals, or any type of environmental observation applications.

The dataset contains about 25,000 natural images taken globally. Each image is fixed at a resolution of 150 x 150 pixels. It comprises one of the following six classes: Buildings, Forest, Glacier, Mountain, Sea, or Street. It is split into three pieces: a training database of about 14,000 images for training the models, a testing database of 3,000 images for testing the performance of models, and a prediction database of about 7,000 images for optional testing or prediction. It is diverse enough to train deep learning models for testing their abilities on unseen pictures.

## 3. Data Preparation

Images were loaded from the directory using the image_dataset_from_directory() function with automatically generated labels. A train-validation split of 80-20 was used to test the performance of the models. Image preprocessing involved resizing the images to a fixed 224x224 pixel resolution with three color channels to meet the input specifications of pre-trained models.

Additionally, pixel values were normalized between 0 and 1. Real-time data augmentations were used to enhance the robustness of the models by adding random flipping, rotation, zooming, contrast, and translation. Optimized techniques such as caching and prefetching were used to

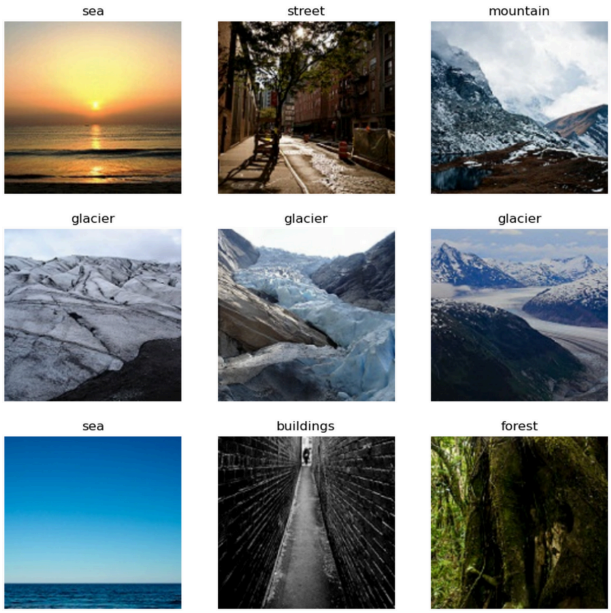
improve the efficiency of training. Visualizations of the training data samples with their augmentations illustrate the variability of the training data.

## 4. Approach

An ensemble-oriented modeling strategy was explored to improve robustness and classification performance by leveraging multiple deep learning architectures. Specifically, EfficientNet-B0, known for its lightweight and optimized design, and ResNet50, a deeper and more expressive model, were evaluated in parallel to assess their complementary strengths. In addition, a Language Model (LLM) was incorporated as an auxiliary source of predictive information rather than as a direct vision-based classifier. Transfer learning with ImageNet-pretrained weights was employed to enhance training efficiency and generalization. To mitigate overfitting, data augmentation, dropout, and early stopping were applied. Comparative analysis of EfficientNet-B0 and ResNet50 highlights their differing capacity–efficiency trade-offs, informing the potential value of ensemble-based extensions.

## 5. Deep Learning Models for Image Scene Classification

***Sample Visualization:***




***Data Augmentation:***

***ResNet50(24M parameters (heavy, powerful)):***

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_layer_2 (InputLayer) | (None, 224, 224, 3) | 0 |
| rescaling_1 (Rescaling) | (None, 224, 224, 3) | 0 |
| data_augmentation (Sequential) | (None, 224, 224, 3) | 0 |
| resnet50 (Functional) | (None, 7, 7, 2048) | 23,587,712 |
| global_average_pooling2d (GlobalAveragePooling2D) | (None, 2048) | 0 |
| dropout (Dropout) | (None, 2048) | 0 |
| dense (Dense) | (None, 256) | 524,544 |
| dropout_1 (Dropout) | (None, 256) | 0 |
| dense_1 (Dense) | (None, 6) | 1,542 |

**Total params:** 24,113,798 (91.99 MB)

**Trainable params:** 526,086 (2.01 MB)

**Non-trainable params:** 23,587,712 (89.98 MB)

***EfficientNet-B0(4.3M parameters (lightweight, efficient)):***

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_layer_4 (InputLayer) | (None, 224, 224, 3) | 0 |
| rescaling_1 (Rescaling) | (None, 224, 224, 3) | 0 |
| data_augmentation (Sequential) | (None, 224, 224, 3) | 0 |
| efficientnetb0 (Functional) | (None, 7, 7, 1280) | 4,049,571 |
| global_average_pooling2d_1 (GlobalAveragePooling2D) | (None, 1280) | 0 |
| dropout_2 (Dropout) | (None, 1280) | 0 |
| dense_2 (Dense) | (None, 256) | 327,936 |
| dropout_3 (Dropout) | (None, 256) | 0 |
| dense_3 (Dense) | (None, 6) | 1,542 |

**Total params:** 4,379,049 (16.70 MB)

**Trainable params:** 329,478 (1.26 MB)

**Non-trainable params:** 4,049,571 (15.45 MB)

### 5.1. Sample Scene Classification Results

This image presents some examples of data in the image dataset and their respective predictions of what the scene is, such as sea, street, mountain, glacier, buildings, or forest. These examples

illustrate the types of visual features the models attempted to learn, such as texture, color, and structural cues, particularly for visually similar classes like mountains and glaciers. This is achieved through the key learning of features by the models, such as texture, color, and structure.

### 5.2. Data Augmentation Impact

This image portrays how data augmentation assists in enhancing robustness. Variations such as grayscale transformation, image distortion, and perspective transformations are shown to occur on the same street scene image. It is observed that despite the image transformations, the network is still centered on structural inconsistencies such as building boundaries and depth information. This acknowledges that data augmentation aids in making the network less sensitive to lighting and color variations.

### 5.3. Model Architecture of ResNet50

This is an image of the transfer learning architecture for image classification with the support of ResNet50. This is a deep convolutional neural network with about 25 million parameters. This is a complex neural network that consumes more computational power. The input image is of size 224 x 224 x 3. It is normalized and then augmented before it is processed using the pre-trained ResNet50. The convolutional layers of this neural network are fixed, with additional custom layers for the classification of the six targets. This neural network consumes more memory due to the number of parameters it possesses.

### 5.4. EfficientNetB0 Model Architecture

This image shows the EfficientNetB0-based model that strikes a good trade-off between effectiveness and efficiency. Like in the ResNet50 model, the image is rescaled and then augmented before feature extraction. EfficientNetB0 yields a 7 × 7 × 1280 feature map that is then pooled and classified using a lightweight dense layer classifier. This model is highly efficient since it requires only 4.38 million total parameters, which is much smaller compared to the number of parameters in the ResNet50 model, while still being effective in image classification.

## 6. LLM-Assisted Predictions

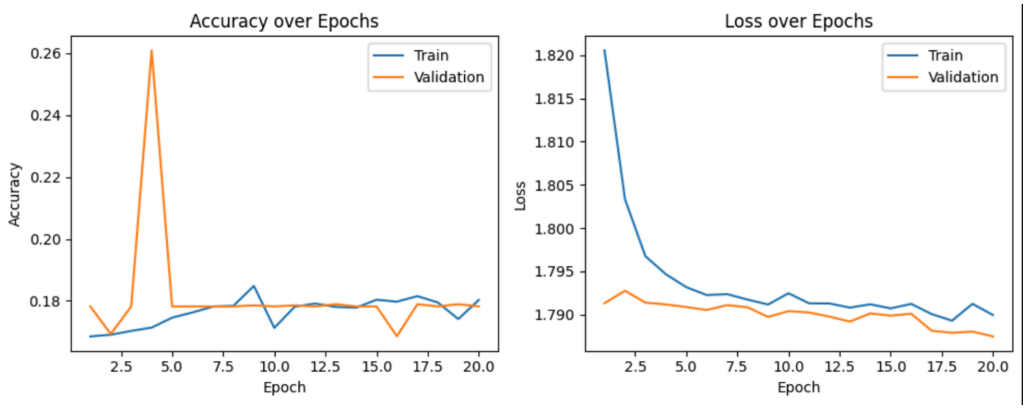
| Filename | LLM Label |
|---|---|
| 10003.jpg | glacier |
| 10006.jpg | building |
| 10007.jpg | forest |

A Language Model was employed to generate class predictions for a subset of images based on available training metadata, and batch processing was used to efficiently handle the data. The LLM's predictions were compared against the corresponding true labels, yielding an accuracy of 91.18% across 204 evaluated images. These results suggest that LLM-based predictions may provide useful supplementary semantic signals; however, the LLM should be interpreted as a supportive reference rather than a standalone vision-based image classifier. The table shows the LLM semantic labeling of image files, each with a dominant visual category taken from the filename. The nature of the labels, glacier, building, and forest, suggests that the model captures high-level scene context rather than fine-grained object details. The scenes recognized include salient levels within two quite different environmental domains: natural and urban, with

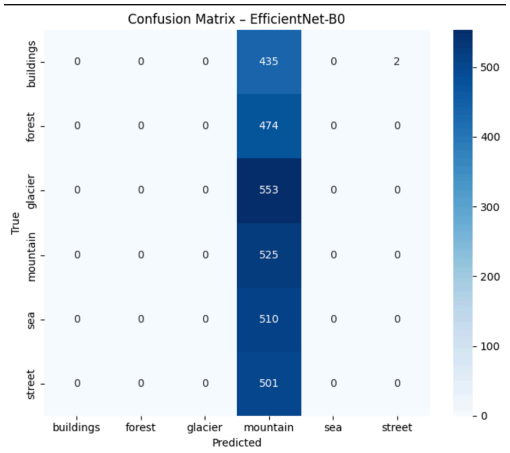
separation between categories. Although the sample is small, the diversity of labels suggests that the model can generalize across different landscapes. For a more detailed evaluation, this labeling would have to be checked against ground-truth annotations to assess the accuracy and consistency at scale.

## 7. Results

During training, the performance of the model did not show much improvement in terms of accuracy over the number of epochs, representing poor learning of features. While there was a drop in loss, there was no progress in accuracy, implying that the model had learned the general pattern of images but failed to distinguish boundaries between classes. The test accuracy for the final model was at 17.5%, which is almost random. Examination of the confusion matrix and the classification report showed that the training data were mostly classified into one dominant class, representing poor class distribution. Graphs of accuracy and loss with respect to the number of epochs showed that while there was a loss in training, the training accuracy was poor.



The accuracy plot shows that the training as well as the validation accuracy is low, with hovering between 17-18% for both training and validation. This is an indication that the model is not learning any patterns from the data. There is a small peak for the validation accuracy in the initial stages of training, which seems to be random. From the loss plot, there is a steady drop in both training and validation loss with the plots being very close. This means that, although the training is occurring (the model is trying to minimize loss), it is not resulting in better classifications. It is also noticed that there is no overfitting but rather underfitting, given the proximity of training and validation loss plots. These trends collectively indicate that the current model is too simple or not well specified or that it does not have informative features. Some of the ways the current models could be improved include: improving the capacity of the current models, adjusting the learning rates or training for a longer period, or even improving feature engineering.




| | precision | recall | f1-score | support |
|---|---|---|---|---|
| buildings | 0.00 | 0.00 | 0.00 | 437 |
| forest | 0.00 | 0.00 | 0.00 | 474 |
| glacier | 0.00 | 0.00 | 0.00 | 553 |
| mountain | 0.18 | 1.00 | 0.30 | 525 |
| sea | 0.00 | 0.00 | 0.00 | 510 |
| street | 0.00 | 0.00 | 0.00 | 501 |
| accuracy | | | 0.17 | 3000 |
| macro avg | 0.03 | 0.17 | 0.05 | 3000 |
| weighted avg | 0.03 | 0.17 | 0.05 | 3000 |

This confusion matrix shows that there is a prediction collapse in the prediction results, since the network almost always predicts any image with “mountain”, irrespective of its actual class. All other classes: “buildings”, “forest”, “glacier”, “sea”, “street” are never correctly classified into their respective classes. This means that the rows are mapped to a single column. This is also

reflected in the classification report. “mountain” class recall is 1.00, implying that all “mountain” images are classified correctly, but precision is low (0.18), since many other images are incorrectly classified as “mountain.” Precision, recall, and F1-score for other classes are 0.00, implying that no image of any class is classified correctly. Overall accuracy is 17%, consistent with the number of mountain image examples in the data set, clearly not meaningful learning. Macro F1-scores of approximately 0.05 also speak of poor performance.

## 8. Conclusion

This study highlights the challenges of natural scene image classification using deep learning models, even when applied to moderately large datasets. Although architectures such as ResNet50 and EfficientNet-B0 were able to capture general visual patterns, they failed to effectively separate individual scene categories, resulting in low classification accuracy and pronounced class bias.

An exploratory experiment involving a Language Model (LLM) demonstrated high agreement with ground-truth labels on a limited subset of images. However, this result should be interpreted cautiously, as the LLM was not used as a direct vision-based classifier and did not replace convolutional feature extraction. Instead, the LLM experiment suggests potential value as a supplementary semantic reference rather than as a standalone solution.

Overall, the findings indicate that deep learning alone does not guarantee high classification performance without careful attention to preprocessing, class balance, and model fine-tuning.

While LLM-assisted approaches may offer complementary insights, robust image classification remains dependent on well-trained vision models and appropriate experimental design.

## 9. Future Work

Further improvements are needed, targeting several strategies for overall performance. First, data augmentation should be enhanced by incorporating more transformations, including color jittering, random cropping, perspective warping, and other adjustments that would allow models to learn more invariant features. Class imbalance needs to be handled appropriately; otherwise, techniques such as oversampling, weighted loss functions, or generation of balanced batches may reduce the over-dominance of a single class. Fine-tuning of pre-trained models could be further explored by unfreezing more layers, hence allowing networks to learn more dataset-specific features that should lead to better class separation. Exploring more sophisticated ensemble methods that combine multiple deep learning architectures with LLM predictions may leverage the complementary strengths of each model, resulting in improved classification accuracy.

Furthermore, using attention mechanisms or transformer-based hybrid architectures could further enhance feature extraction from complex natural scenes. Finally, LLMs can also be used to help interpret features, semantic reasoning, or pre-classification labeling to provide further benefit to the convolutional models, making the general classification pipeline more robust, generic, and accurate for unseen images.

## References:

- Intel Image Classification Dataset
- Python libraries including Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn, TensorFlow, Keras

## Appendix - Python code:

***Sample Visualization:***

```
plt.figure(figsize=(10, 10))

for images, labels in train_ds.take(1):

    for i in range(9):

        ax = plt.subplot(3, 3, i + 1)

        plt.imshow(images[i].numpy().astype("uint8"))

        plt.title(class_names[labels[i]])

        plt.axis("off")

normalization_layer = layers.Rescaling(1./255)



normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))

image_batch, labels_batch = next(iter(normalized_ds))

first_image = image_batch[0]



print(image_batch)

print(first_image)
```

***Data Augmentation:***

```
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
```

```
    layers.RandomZoom(0.1),
    layers.RandomContrast(0.2),
    layers.RandomTranslation(0.1, 0.1)
], name='data_augmentation')

normalization_layer = layers.Rescaling(1./255)
image_batch, label_batch = next(iter(train_ds))
image = image_batch[0]
image = normalization_layer(image)

plt.figure(figsize=(10, 10))
for i in range(9):
    augmented = data_augmentation(tf.expand_dims(image, 0), training=True)[0]
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented)
    plt.axis("off")
```

***Autotuning:***

```
autotune = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=autotune)
val_ds = val_ds.cache().prefetch(buffer_size=autotune)
test_ds = test_ds.cache().prefetch(buffer_size=autotune)

from tensorflow.keras.applications import ResNet50
from tensorflow.keras import layers, Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint
```

***Resnet50 model:***

```
def create_resnet50_model(num_classes=6):
    base_model = ResNet50(
        include_top=False,
```

```
        input_shape=(224, 224, 3),
        weights='imagenet'
    )
    base_model.trainable = False
    inputs = keras.Input(shape=(224, 224, 3))
    x = normalization_layer(inputs)
    x = data_augmentation(x)
    x = base_model(x, training=False)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dropout(0.5)(x)
    x = layers.Dense(256, activation='relu')(x)
    x = layers.Dropout(0.3)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)
    model = Model(inputs, outputs)
    model.compile(
        optimizer=Adam(learning_rate=0.001),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    return model

resnet_model = create_resnet50_model(num_classes=num_classes)
resnet_model.summary()
```

***EfficientNet-B0:***

```
from tensorflow.keras.applications import EfficientNetB0

def create_efficientnet(num_classes=6, pretrained=True):
    weights = 'imagenet' if pretrained else None
    base_model = EfficientNetB0(
        include_top=False,
        input_shape=(224, 224, 3),
```

```
        weights=weights
    )
    base_model.trainable = False
    inputs = keras.Input(shape=(224, 224, 3))
    x = normalization_layer(inputs)
    x = data_augmentation(x)
    x = base_model(x, training=False)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dropout(0.5)(x)
    x = layers.Dense(256, activation='relu')(x)
    x = layers.Dropout(0.3)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)
    model = Model(inputs, outputs)
    model.compile(
        optimizer=Adam(learning_rate=0.001),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    return model

efficientnet_model = create_efficientnet(num_classes=num_classes)

efficientnet_model.summary()
```

***LLM Predictions:***

```
import os
import pandas as pd

train_path = r"C:\Users\Tsion T\Downloads\archive_39\seg_train\seg_train"

records = []
```

```
for label in os.listdir(train_path):
    class_dir = os.path.join(train_path, label)

    if not os.path.isdir(class_dir):
        continue



    for filename in os.listdir(class_dir):
        if filename.lower().endswith((".jpg", ".jpeg", ".png")):
            records.append({
                "filename": filename,
                "true_label": label
            })

df_true = pd.DataFrame(records)

print(df_true.head())
print("Total images found:", len(df_true))

output_path = r"C:\Users\Tsion T\Downloads\true_labels.csv"
df_true.to_csv(output_path, index=False)

print(f"True labels saved to: {output_path}")

import pandas as pd

df_true = pd.read_csv(r"C:\Users\Tsion T\Downloads\true_labels.csv")
df_llm = pd.read_csv(r"C:\Users\Tsion T\Downloads\llm_predictions.csv")

# merge on filename
df = df_llm.merge(df_true, on="filename", how="inner")

# accuracy
df["correct"] = df["llm_label"] == df["true_label"]
```

```
accuracy = df["correct"].mean()

print("LLM Accuracy:", accuracy)
print("Total matched images:", len(df))
```