CS4321 Midterm Project: Littoral Environment Classification with Transfer Learning

Madina Petashvili
Department of Computer Science
Naval Postgraduate School
Monterey, CA
`madina.petashvili@nps.edu`

Alexander Huang
Department of Computer Science
Naval Postgraduate School
Monterey, CA
`alexander.huang@nps.edu`

Gitlab repository: https://gitlab.nps.edu/cs4321/su23/team_hp_cs4321_midterm

1 Introduction

Naval operations, characterized by their strategic importance and complex nature, rely profoundly on the acquisition of precise and timely environmental information [1]. Such information, ranging from hydrographic surveys to reconnaissance data, forms the bedrock of decisions and maneuvers, especially when the operations venture into the littoral zone [2]. As the Navy continues to integrate advanced technologies into its operational fabric, the use of machine learning, particularly transfer learning, has emerged as a pivotal tool to enhance the Navy's capability to interpret environmental data swiftly [3].

In the quest to optimize image classification tasks relevant to naval environments, we embarked on a project using transfer learning techniques with three renowned deep learning models: ResNet50, VGG16, and MobileNetV2 [4] [5] [6]. We employed PyTorch as our deep learning framework, given its flexibility and efficiency in handling such models [7]. Our project took inspiration from the enlightening findings of the "Do Better ImageNet Models Transfer Better?" paper by Simon Kornblith et al. in 2018 [8]. Several key takeaways from Kornblith's work informed our approach:

Utility of ImageNet-Trained Networks: The paper underscores the worth of ImageNet-trained networks in transfer learning, emphasizing that the optimal network selection can significantly impact the outcome. While models with higher ImageNet accuracy tend to exhibit superior transfer accuracy, they might also impose heavier computational demands during training and deployment phases [8].

Significance of Regularization: Regularization techniques, which can thwart overfitting, play a dual role. While they can enhance a model's performance, they might, if not chosen judiciously, impede the model's ability to generalize features. This insight made us reconsider and refine the regularization techniques we implemented [10].

Dataset Size in Transfer Learning: Kornblith's paper explains the role of the target dataset's size [8]. Larger datasets bolster the network's learning potential, fostering its generalization capability. However, fine-tuning becomes indispensable when working with smaller datasets to attain peak performance.

In essence, the research by Kornblith and his team [8] offered a robust foundation for our project. By harnessing the salient points from the paper and adopting a tailored approach, we aim to develop a system that would not only enhance the Navy's access to environmental data but would also contribute significantly to mission-critical operations in the littoral zone and beyond [9] [1] .

2 Methods

2.1 Dataset

The U.S. Geological Survey (USGS) and Naval Postgraduate School (NPS) collected the coastal dataset, consisting of 6400 total images of size 299 by 299 pixels. The images were further split into a training dataset of 4,800 images, a validation dataset of 800 images, and a testing dataset of 800 images. Images are labeled in eight classes: sandy beach, marsh, tidal flat, coastal waterway, rocky coast, cliff, man-made structures, and dunes.

2.1.1 t-Distributed Stochastic Neighbor Embedding (t-SNE) Visualization and Principal Component Analysis

t-SNE plots visualize data manifolds by converting affinities of data points to probabilities and fitting probability distributions in the original hyperspace into low-dimensional t-distribution representations by minimizing the Kullback-Leibler (KL) divergence between the probabilities in the original hyperspace and the probabilities in the embedded space. It provides an indication of how well a model can produce separable embeddings prior to classification and thus the quality of decision boundaries in hyperspace for the classifier. Raw data can indicate whether the data is linearly separable prior to model training and aid in model selection. However, generating t-SNE plots can be computationally expensive, and plots are inconsistent between restarts with different seeds due to the stochastic nature of the method [12].

PCA plots visualize the amount of variance in data by projecting multi-dimensional data into fewer dimensions. 2- or 3- dimensional visualizations with PCA can then be used to draw conclusions about the data for model selection or model embeddings, similar to t-SNE visualizations. Generating PCA plots may be memory intensive and may require partial fitting on smaller batches of data rather than loading a complete dataset into memory [13].

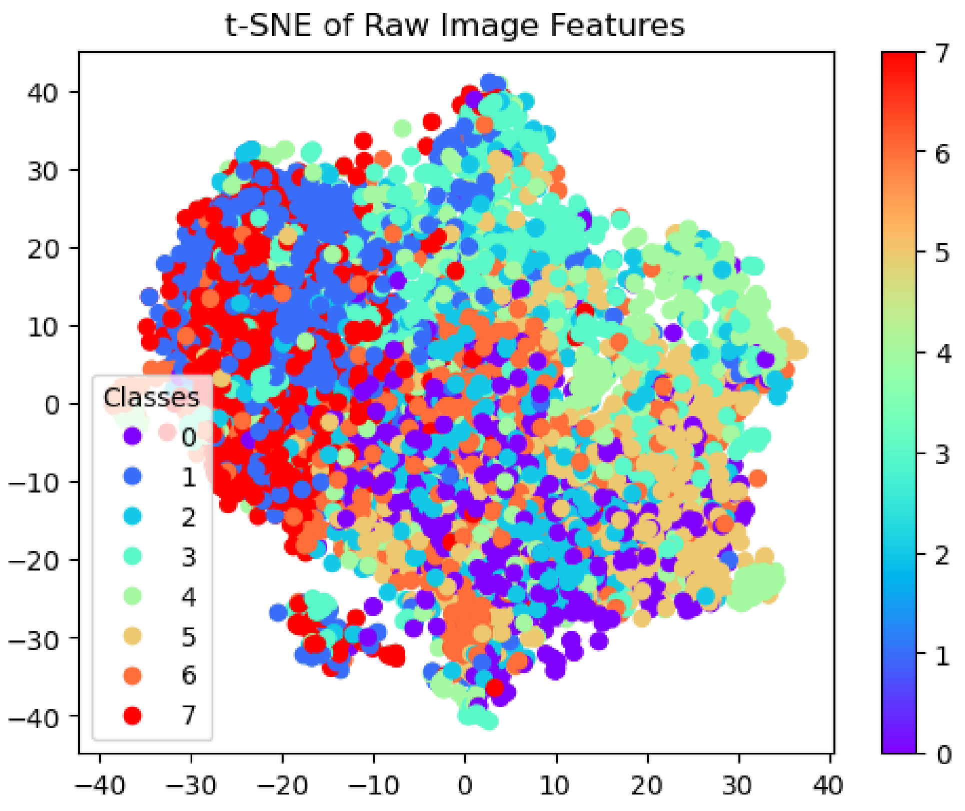


Figure 1: Raw data t-SNE plot.

Figure 1 shows the t-SNE and PCA plots for the raw littoral image data. There are eight total classes, numbered 0 to 7: coastal cliffs, coastal rocky, coastal waterway, dunes, man-made structures, salt marshes, sandy beaches, and tidal flats. From the t-SNE and PCA visualizations of the raw data, we concluded the data would not be linearly separable. Deep neural networks are commonly used to model non-linear data, and have previously been shown to perform well on image classification.

2.2 Transfer Learning

Transfer learning is a powerful technique that allows for the utilization of a pre-trained model on a new task. The core idea is to leverage the knowledge gained while training on one task and applying it to a different but related task. This approach can be particularly beneficial when we have limited labeled data for the new task.

We used a transfer learning approach to train deep neural network classifiers on the littoral environment images.We used three pretrained models for feature extraction: VGG16, ResNet50, and MobileNetV2. All three networks were trained on the ImageNet dataset of over 1.2 million images from the ImageNet Large Scale Visualization Recognition Challenge [14]. Because the ImageNet dataset contained 1000 classes, we define a model usable for our specific domain and dataset of eight classes by removing the classification sections of each model with our own and freezing the feature extraction sections. The resulting model can then be applied to a more specific problem while retaining the generalizability advantages of training on a much larger dataset.

2.3 Data Processing

In deep learning, especially in the field of computer vision, preprocessing and augmenting data can significantly improve the model's performance and its ability to generalize to unseen scenarios [15]. With this understanding, we established a data processing pipeline tailored for the littoral environment images dataset.

Image Augmentation (Training Dataset Only):

Random Resize Cropping: We randomly crop the input image to different sizes before resizing it to a predefined size. This process aids the model in becoming invariant to the scale and position of objects in the image, crucial for handling diverse object appearances in images [16].

Horizontal Flipping: By randomly flipping images, the model's robustness to the directionality of objects is enhanced. It's a technique particularly beneficial to ensure models don't inherit any unintentional biases present in the training data regarding object orientations [17].

Consistent Transformations (Training, Validation, and Testing Datasets):

Resizing: Resizing all images to 224 × 224 pixels was required for compatibility with the ImageNet models used, which expect this resolution as input [18].

Normalization: Neural networks tend to perform better when input data falls within a standardized range. Normalization ensures more consistent activations throughout the network, fostering better training [19].

Incorporating these preprocessing steps, our models are exposed to a diversified representation of the dataset and consistently formatted data, optimizing training efficiency.

2.4 Fixed Feature Classification

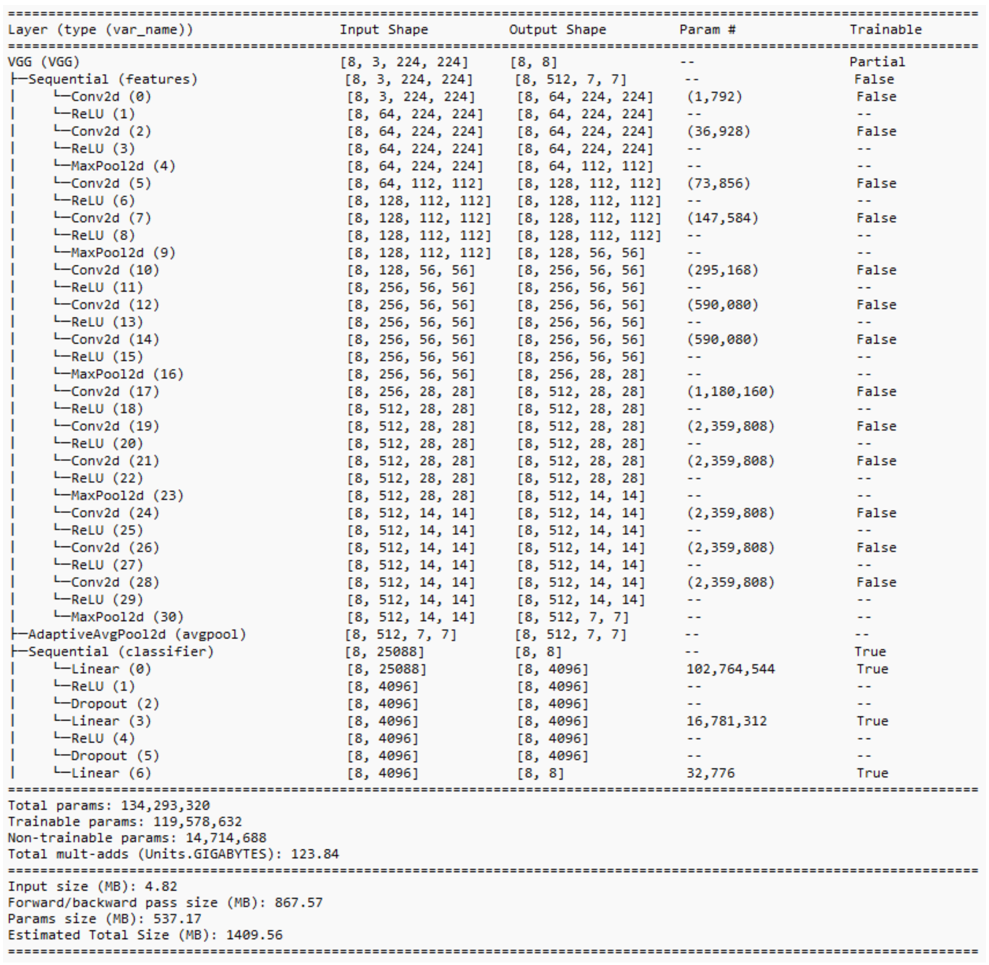
The pre-trained model is solely used as a feature extractor in this approach. The entire architecture, except for the classification layer, is frozen. A new classifier suitable for the target task is then appended and trained.

For all experiements, we used common hyperparameters as listed in Table 1. Model-specific changes during fine-tuning will be discussed for each fine-tuned model in Section 2.5.

Common Experiment Hyperparameters	
Hyperparameter	Value
Training Epochs	40
Batch Size	8
Optimizer	SGD
Learning rate	0.001
Momentum	0.9

Table 1: Common experiment hyperparameter values.

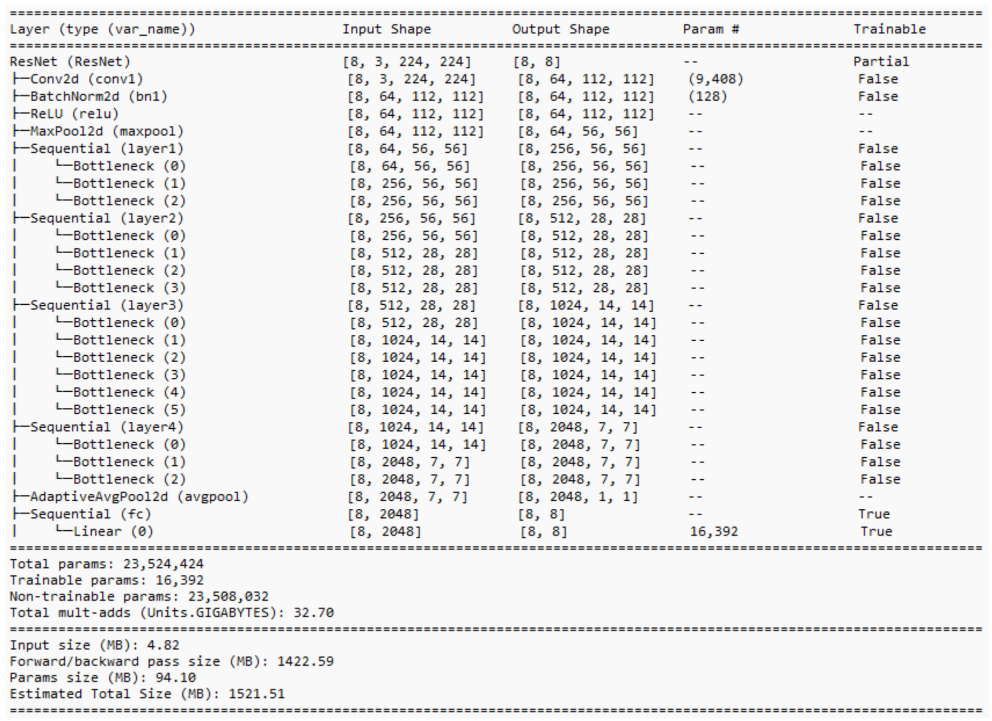
VGG16: The entire feature extraction part of VGG16 was frozen, transforming it into a feature extractor for littoral images. A new classifier, apt for our eight classes, was trained atop these features. Figure 2 shows the neural network architecture used for fixed feature classification with VGG 16 feature extraction. For fixed feature classification with this model, we froze the entire feature extraction section of the model and replaced the final linear layer of the classifier section with a 1000-class output with an eight-class output.



```
==========================================================================================================
Layer (type (var_name))                  Input Shape          Output Shape         Param #              Trainable
==========================================================================================================
VGG (VGG)                                [8, 3, 224, 224]     [8, 8]               --                   Partial
├─Sequential (features)                  [8, 3, 224, 224]     [8, 512, 7, 7]       --                   False
│    └─Conv2d (0)                        [8, 3, 224, 224]     [8, 64, 224, 224]    (1,792)              False
│    └─ReLU (1)                          [8, 64, 224, 224]    [8, 64, 224, 224]    --                   --
│    └─Conv2d (2)                        [8, 64, 224, 224]    [8, 64, 224, 224]    (36,928)             False
│    └─ReLU (3)                          [8, 64, 224, 224]    [8, 64, 224, 224]    --                   --
│    └─MaxPool2d (4)                     [8, 64, 224, 224]    [8, 64, 112, 112]    --                   --
│    └─Conv2d (5)                        [8, 64, 112, 112]    [8, 128, 112, 112]   (73,856)             False
│    └─ReLU (6)                          [8, 128, 112, 112]   [8, 128, 112, 112]   --                   --
│    └─Conv2d (7)                        [8, 128, 112, 112]   [8, 128, 112, 112]   (147,584)            False
│    └─ReLU (8)                          [8, 128, 112, 112]   [8, 128, 112, 112]   --                   --
│    └─MaxPool2d (9)                     [8, 128, 112, 112]   [8, 128, 56, 56]     --                   --
│    └─Conv2d (10)                       [8, 128, 56, 56]     [8, 256, 56, 56]     (295,168)            False
│    └─ReLU (11)                         [8, 256, 56, 56]     [8, 256, 56, 56]     --                   --
│    └─Conv2d (12)                       [8, 256, 56, 56]     [8, 256, 56, 56]     (590,080)            False
│    └─ReLU (13)                         [8, 256, 56, 56]     [8, 256, 56, 56]     --                   --
│    └─Conv2d (14)                       [8, 256, 56, 56]     [8, 256, 56, 56]     (590,080)            False
│    └─ReLU (15)                         [8, 256, 56, 56]     [8, 256, 56, 56]     --                   --
│    └─MaxPool2d (16)                    [8, 256, 56, 56]     [8, 256, 28, 28]     --                   --
│    └─Conv2d (17)                       [8, 256, 28, 28]     [8, 512, 28, 28]     (1,180,160)          False
│    └─ReLU (18)                         [8, 512, 28, 28]     [8, 512, 28, 28]     --                   --
│    └─Conv2d (19)                       [8, 512, 28, 28]     [8, 512, 28, 28]     (2,359,808)          False
│    └─ReLU (20)                         [8, 512, 28, 28]     [8, 512, 28, 28]     --                   --
│    └─Conv2d (21)                       [8, 512, 28, 28]     [8, 512, 28, 28]     (2,359,808)          False
│    └─ReLU (22)                         [8, 512, 28, 28]     [8, 512, 28, 28]     --                   --
│    └─MaxPool2d (23)                    [8, 512, 28, 28]     [8, 512, 14, 14]     --                   --
│    └─Conv2d (24)                       [8, 512, 14, 14]     [8, 512, 14, 14]     (2,359,808)          False
│    └─ReLU (25)                         [8, 512, 14, 14]     [8, 512, 14, 14]     --                   --
│    └─Conv2d (26)                       [8, 512, 14, 14]     [8, 512, 14, 14]     (2,359,808)          False
│    └─ReLU (27)                         [8, 512, 14, 14]     [8, 512, 14, 14]     --                   --
│    └─Conv2d (28)                       [8, 512, 14, 14]     [8, 512, 14, 14]     (2,359,808)          False
│    └─ReLU (29)                         [8, 512, 14, 14]     [8, 512, 14, 14]     --                   --
│    └─MaxPool2d (30)                    [8, 512, 14, 14]     [8, 512, 7, 7]       --                   --
├─AdaptiveAvgPool2d (avgpool)            [8, 512, 7, 7]       [8, 512, 7, 7]       --                   --
├─Sequential (classifier)                [8, 25088]           [8, 8]               --                   True
│    └─Linear (0)                        [8, 25088]           [8, 4096]            102,764,544          True
│    └─ReLU (1)                          [8, 4096]            [8, 4096]            --                   --
│    └─Dropout (2)                       [8, 4096]            [8, 4096]            --                   --
│    └─Linear (3)                        [8, 4096]            [8, 4096]            16,781,312           True
│    └─ReLU (4)                          [8, 4096]            [8, 4096]            --                   --
│    └─Dropout (5)                       [8, 4096]            [8, 4096]            --                   --
│    └─Linear (6)                        [8, 4096]            [8, 8]               32,776               True
==========================================================================================================
Total params: 134,293,320
Trainable params: 119,578,632
Non-trainable params: 14,714,688
Total mult-adds (Units.GIGABYTES): 123.84
==========================================================================================================
Input size (MB): 4.82
Forward/backward pass size (MB): 867.57
Params size (MB): 537.17
Estimated Total Size (MB): 1409.56
==========================================================================================================
```

Figure 2: VGG16 fixed feature classifier architecture.

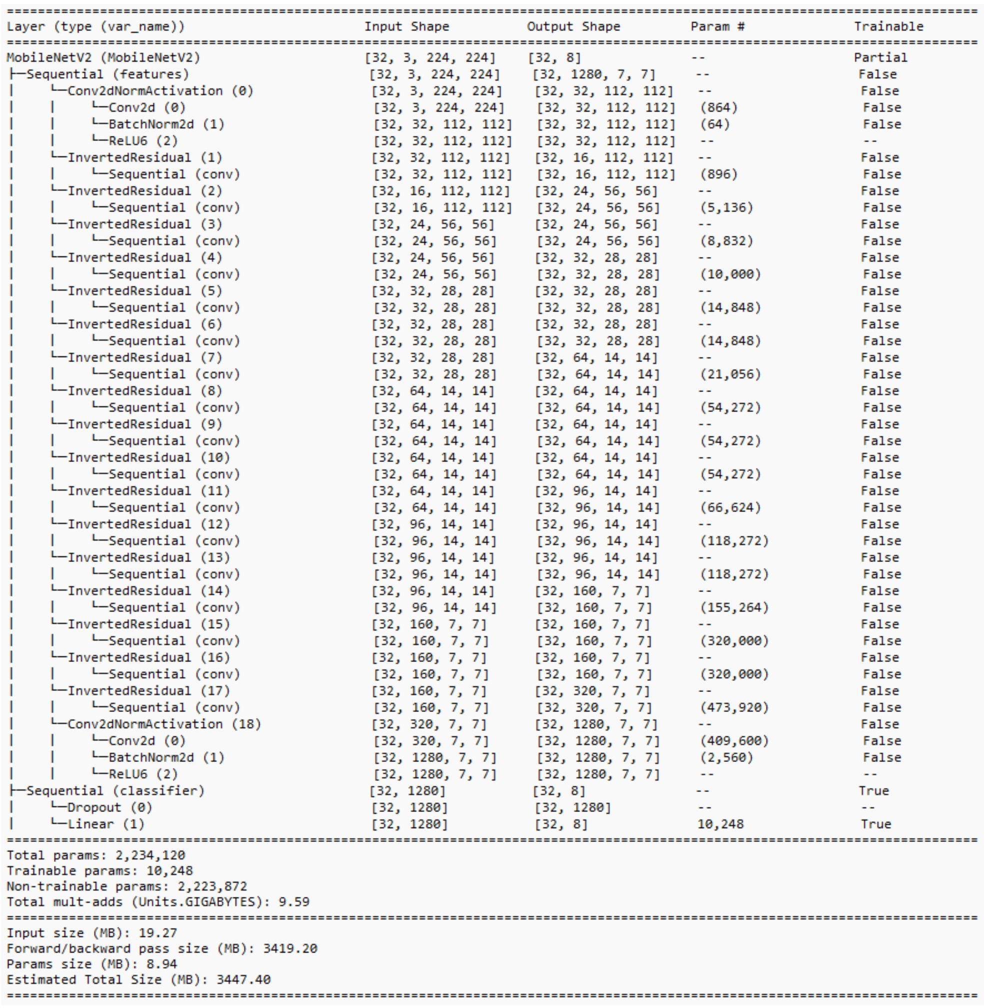
ResNet50: All four pretrained feature extraction layers of the deep ResNet50 architecture were retained and used as a fixed feature extractor. A custom classifier was appended and trained for our dataset. Figure 3 shows the resulting model architecture.



```
==========================================================================================================================
Layer (type (var_name))                  Input Shape          Output Shape         Param #              Trainable
==========================================================================================================================
ResNet (ResNet)                          [8, 3, 224, 224]     [8, 8]               --                   Partial
├─Conv2d (conv1)                         [8, 3, 224, 224]     [8, 64, 112, 112]    (9,408)              False
├─BatchNorm2d (bn1)                      [8, 64, 112, 112]    [8, 64, 112, 112]    (128)                False
├─ReLU (relu)                            [8, 64, 112, 112]    [8, 64, 112, 112]    --                   --
├─MaxPool2d (maxpool)                    [8, 64, 112, 112]    [8, 64, 56, 56]      --                   --
├─Sequential (layer1)                    [8, 64, 56, 56]      [8, 256, 56, 56]     --                   False
│    └─Bottleneck (0)                    [8, 64, 56, 56]      [8, 256, 56, 56]     --                   False
│    └─Bottleneck (1)                    [8, 256, 56, 56]     [8, 256, 56, 56]     --                   False
│    └─Bottleneck (2)                    [8, 256, 56, 56]     [8, 256, 56, 56]     --                   False
├─Sequential (layer2)                    [8, 256, 56, 56]     [8, 512, 28, 28]     --                   False
│    └─Bottleneck (0)                    [8, 256, 56, 56]     [8, 512, 28, 28]     --                   False
│    └─Bottleneck (1)                    [8, 512, 28, 28]     [8, 512, 28, 28]     --                   False
│    └─Bottleneck (2)                    [8, 512, 28, 28]     [8, 512, 28, 28]     --                   False
│    └─Bottleneck (3)                    [8, 512, 28, 28]     [8, 512, 28, 28]     --                   False
├─Sequential (layer3)                    [8, 512, 28, 28]     [8, 1024, 14, 14]    --                   False
│    └─Bottleneck (0)                    [8, 512, 28, 28]     [8, 1024, 14, 14]    --                   False
│    └─Bottleneck (1)                    [8, 1024, 14, 14]    [8, 1024, 14, 14]    --                   False
│    └─Bottleneck (2)                    [8, 1024, 14, 14]    [8, 1024, 14, 14]    --                   False
│    └─Bottleneck (3)                    [8, 1024, 14, 14]    [8, 1024, 14, 14]    --                   False
│    └─Bottleneck (4)                    [8, 1024, 14, 14]    [8, 1024, 14, 14]    --                   False
│    └─Bottleneck (5)                    [8, 1024, 14, 14]    [8, 1024, 14, 14]    --                   False
├─Sequential (layer4)                    [8, 1024, 14, 14]    [8, 2048, 7, 7]      --                   False
│    └─Bottleneck (0)                    [8, 1024, 14, 14]    [8, 2048, 7, 7]      --                   False
│    └─Bottleneck (1)                    [8, 2048, 7, 7]      [8, 2048, 7, 7]      --                   False
│    └─Bottleneck (2)                    [8, 2048, 7, 7]      [8, 2048, 7, 7]      --                   False
├─AdaptiveAvgPool2d (avgpool)            [8, 2048, 7, 7]      [8, 2048, 1, 1]      --                   --
├─Sequential (fc)                        [8, 2048]            [8, 8]               --                   True
│    └─Linear (0)                        [8, 2048]            [8, 8]               16,392               True
==========================================================================================================================
Total params: 23,524,424
Trainable params: 16,392
Non-trainable params: 23,508,032
Total mult-adds (Units.GIGABYTES): 32.70
==========================================================================================================================
Input size (MB): 4.82
Forward/backward pass size (MB): 1422.59
Params size (MB): 94.10
Estimated Total Size (MB): 1521.51
==========================================================================================================================
```

Figure 3: ResNet50 fixed feature classifier architecture, output truncated for brevity.

MobileNetV2: Using the lightweight MobileNetV2 as a feature extractor, its classifier was replaced and tailored for the eight classes in our dataset. Figure 4 shows the architecture used, with feature extraction layers frozen.



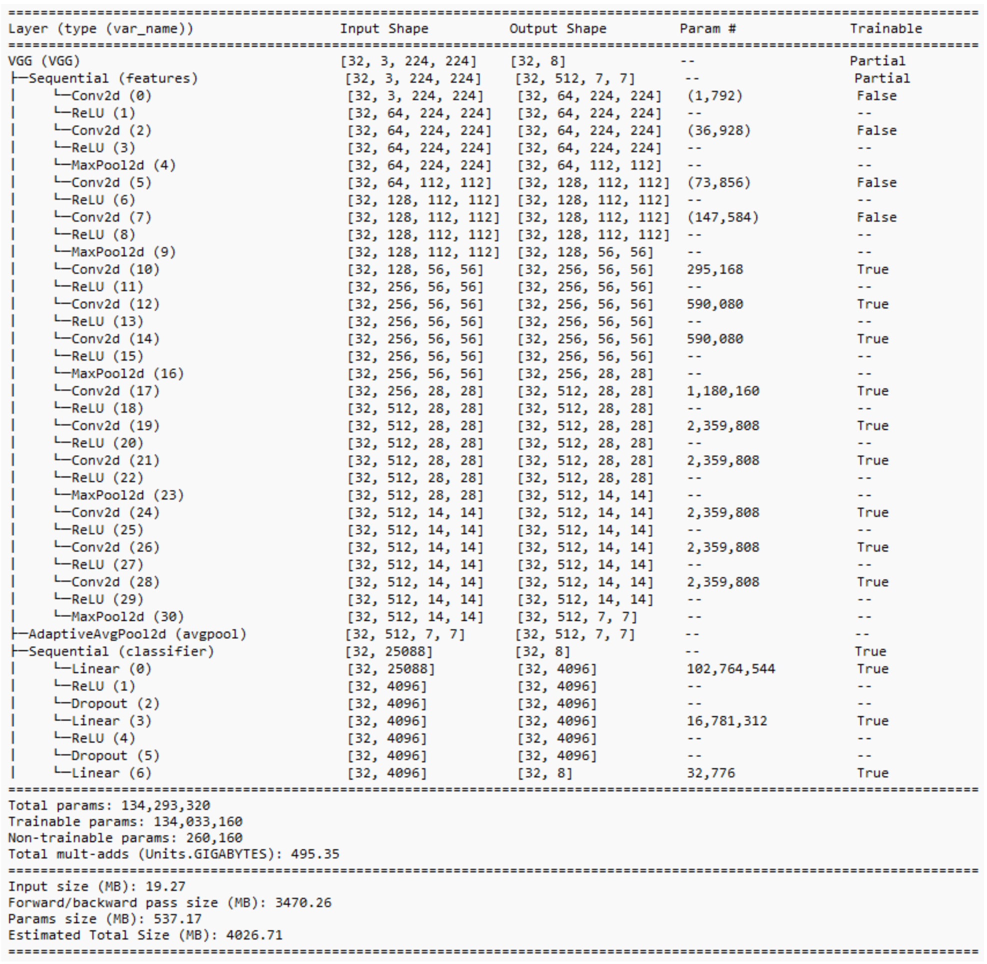
```
==========================================================================================================
Layer (type (var_name))                  Input Shape          Output Shape         Param #              Trainable
==========================================================================================================
MobileNetV2 (MobileNetV2)                [32, 3, 224, 224]    [32, 8]              --                   Partial
├─Sequential (features)                  [32, 3, 224, 224]    [32, 1280, 7, 7]     --                   False
│    └─Conv2dNormActivation (0)          [32, 3, 224, 224]    [32, 32, 112, 112]   --                   False
│    │    └─Conv2d (0)                   [32, 3, 224, 224]    [32, 32, 112, 112]   (864)                False
│    │    └─BatchNorm2d (1)              [32, 32, 112, 112]   [32, 32, 112, 112]   (64)                 False
│    │    └─ReLU6 (2)                    [32, 32, 112, 112]   [32, 32, 112, 112]   --                   --
│    └─InvertedResidual (1)              [32, 32, 112, 112]   [32, 16, 112, 112]   --                   False
│    │    └─Sequential (conv)            [32, 32, 112, 112]   [32, 16, 112, 112]   (896)                False
│    └─InvertedResidual (2)              [32, 16, 112, 112]   [32, 24, 56, 56]     --                   False
│    │    └─Sequential (conv)            [32, 16, 112, 112]   [32, 24, 56, 56]     (5,136)              False
│    └─InvertedResidual (3)              [32, 24, 56, 56]     [32, 24, 56, 56]     --                   False
│    │    └─Sequential (conv)            [32, 24, 56, 56]     [32, 24, 56, 56]     (8,832)              False
│    └─InvertedResidual (4)              [32, 24, 56, 56]     [32, 32, 28, 28]     --                   False
│    │    └─Sequential (conv)            [32, 24, 56, 56]     [32, 32, 28, 28]     (10,000)             False
│    └─InvertedResidual (5)              [32, 32, 28, 28]     [32, 32, 28, 28]     --                   False
│    │    └─Sequential (conv)            [32, 32, 28, 28]     [32, 32, 28, 28]     (14,848)             False
│    └─InvertedResidual (6)              [32, 32, 28, 28]     [32, 32, 28, 28]     --                   False
│    │    └─Sequential (conv)            [32, 32, 28, 28]     [32, 32, 28, 28]     (14,848)             False
│    └─InvertedResidual (7)              [32, 32, 28, 28]     [32, 64, 14, 14]     --                   False
│    │    └─Sequential (conv)            [32, 32, 28, 28]     [32, 64, 14, 14]     (21,056)             False
│    └─InvertedResidual (8)              [32, 64, 14, 14]     [32, 64, 14, 14]     --                   False
│    │    └─Sequential (conv)            [32, 64, 14, 14]     [32, 64, 14, 14]     (54,272)             False
│    └─InvertedResidual (9)              [32, 64, 14, 14]     [32, 64, 14, 14]     --                   False
│    │    └─Sequential (conv)            [32, 64, 14, 14]     [32, 64, 14, 14]     (54,272)             False
│    └─InvertedResidual (10)             [32, 64, 14, 14]     [32, 64, 14, 14]     --                   False
│    │    └─Sequential (conv)            [32, 64, 14, 14]     [32, 64, 14, 14]     (54,272)             False
│    └─InvertedResidual (11)             [32, 64, 14, 14]     [32, 96, 14, 14]     --                   False
│    │    └─Sequential (conv)            [32, 64, 14, 14]     [32, 96, 14, 14]     (66,624)             False
│    └─InvertedResidual (12)             [32, 96, 14, 14]     [32, 96, 14, 14]     --                   False
│    │    └─Sequential (conv)            [32, 96, 14, 14]     [32, 96, 14, 14]     (118,272)            False
│    └─InvertedResidual (13)             [32, 96, 14, 14]     [32, 96, 14, 14]     --                   False
│    │    └─Sequential (conv)            [32, 96, 14, 14]     [32, 96, 14, 14]     (118,272)            False
│    └─InvertedResidual (14)             [32, 96, 14, 14]     [32, 160, 7, 7]      --                   False
│    │    └─Sequential (conv)            [32, 96, 14, 14]     [32, 160, 7, 7]      (155,264)            False
│    └─InvertedResidual (15)             [32, 160, 7, 7]      [32, 160, 7, 7]      --                   False
│    │    └─Sequential (conv)            [32, 160, 7, 7]      [32, 160, 7, 7]      (320,000)            False
│    └─InvertedResidual (16)             [32, 160, 7, 7]      [32, 160, 7, 7]      --                   False
│    │    └─Sequential (conv)            [32, 160, 7, 7]      [32, 160, 7, 7]      (320,000)            False
│    └─InvertedResidual (17)             [32, 160, 7, 7]      [32, 320, 7, 7]      --                   False
│    │    └─Sequential (conv)            [32, 160, 7, 7]      [32, 320, 7, 7]      (473,920)            False
│    └─Conv2dNormActivation (18)         [32, 320, 7, 7]      [32, 1280, 7, 7]     --                   False
│    │    └─Conv2d (0)                   [32, 320, 7, 7]      [32, 1280, 7, 7]     (409,600)            False
│    │    └─BatchNorm2d (1)              [32, 1280, 7, 7]     [32, 1280, 7, 7]     (2,560)              False
│    │    └─ReLU6 (2)                    [32, 1280, 7, 7]     [32, 1280, 7, 7]     --                   --
├─Sequential (classifier)                [32, 1280]           [32, 8]              --                   True
│    └─Dropout (0)                       [32, 1280]           [32, 1280]           --                   --
│    └─Linear (1)                        [32, 1280]           [32, 8]              10,248               True
==========================================================================================================
Total params: 2,234,120
Trainable params: 10,248
Non-trainable params: 2,223,872
Total mult-adds (Units.GIGABYTES): 9.59
==========================================================================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 3419.20
Params size (MB): 8.94
Estimated Total Size (MB): 3447.40
==========================================================================================================
```

Figure 4: MobileNetV2 fixed feature classifier architecture.

2.5 Fine Tuning with Transfer Learning

In fine-tuning, we initially use the pre-trained model and slightly adjust its architecture to make it suitable for our task. Once the architecture is adapted, the model undergoes further training on the new dataset.

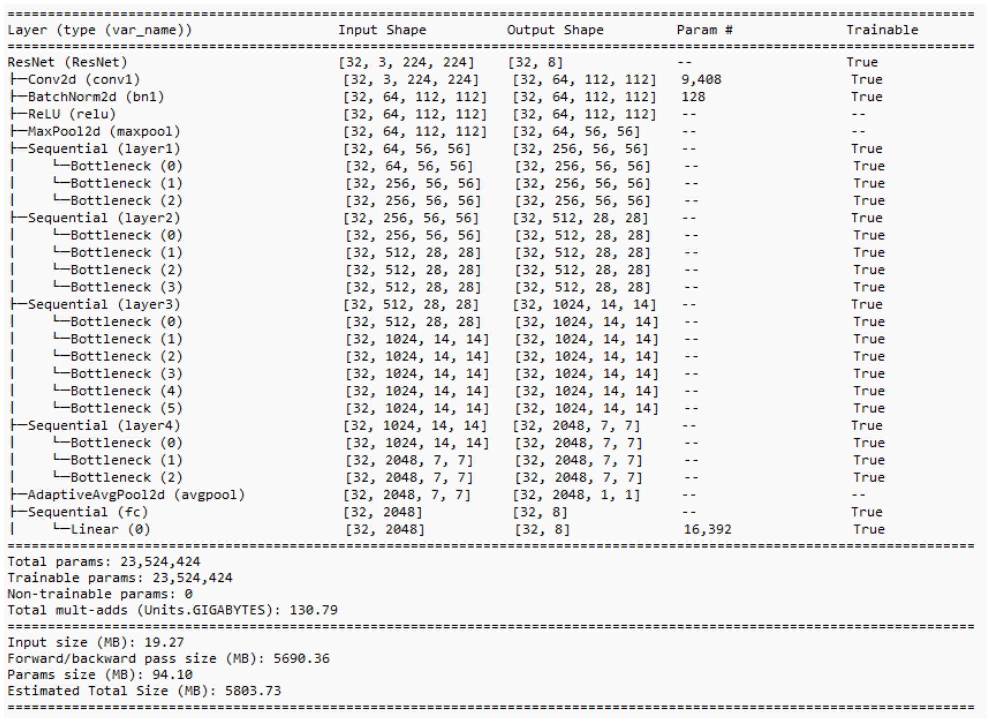
VGG16: VGG16, known for its deep yet straightforward architecture, was initially fine-tuned for our eight-class problem. While the early convolutional blocks capturing basic image features were frozen, the subsequent layers, including the classifier, were tailored and trained for our dataset [5]. In our experiments, we unfroze layers from the bottom-up, in groups of two convolutional layers ending with maximum pooling layers. We found the best performance by unfreezing three such groups of layers from the bottom. 5 shows the resulting architecture.



```
===================================================================================================================
Layer (type (var_name))                  Input Shape          Output Shape         Param #              Trainable
===================================================================================================================
VGG (VGG)                                [32, 3, 224, 224]    [32, 8]              --                   Partial
├─Sequential (features)                  [32, 3, 224, 224]    [32, 512, 7, 7]      --                   Partial
│    └─Conv2d (0)                        [32, 3, 224, 224]    [32, 64, 224, 224]   (1,792)              False
│    └─ReLU (1)                          [32, 64, 224, 224]   [32, 64, 224, 224]   --                   --
│    └─Conv2d (2)                        [32, 64, 224, 224]   [32, 64, 224, 224]   (36,928)             False
│    └─ReLU (3)                          [32, 64, 224, 224]   [32, 64, 224, 224]   --                   --
│    └─MaxPool2d (4)                     [32, 64, 224, 224]   [32, 64, 112, 112]   --                   --
│    └─Conv2d (5)                        [32, 64, 112, 112]   [32, 128, 112, 112]  (73,856)             False
│    └─ReLU (6)                          [32, 128, 112, 112]  [32, 128, 112, 112]  --                   --
│    └─Conv2d (7)                        [32, 128, 112, 112]  [32, 128, 112, 112]  (147,584)            False
│    └─ReLU (8)                          [32, 128, 112, 112]  [32, 128, 112, 112]  --                   --
│    └─MaxPool2d (9)                     [32, 128, 112, 112]  [32, 128, 56, 56]    --                   --
│    └─Conv2d (10)                       [32, 128, 56, 56]    [32, 256, 56, 56]    295,168              True
│    └─ReLU (11)                         [32, 256, 56, 56]    [32, 256, 56, 56]    --                   --
│    └─Conv2d (12)                       [32, 256, 56, 56]    [32, 256, 56, 56]    590,080              True
│    └─ReLU (13)                         [32, 256, 56, 56]    [32, 256, 56, 56]    --                   --
│    └─Conv2d (14)                       [32, 256, 56, 56]    [32, 256, 56, 56]    590,080              True
│    └─ReLU (15)                         [32, 256, 56, 56]    [32, 256, 56, 56]    --                   --
│    └─MaxPool2d (16)                    [32, 256, 56, 56]    [32, 256, 28, 28]    --                   --
│    └─Conv2d (17)                       [32, 256, 28, 28]    [32, 512, 28, 28]    1,180,160            True
│    └─ReLU (18)                         [32, 512, 28, 28]    [32, 512, 28, 28]    --                   --
│    └─Conv2d (19)                       [32, 512, 28, 28]    [32, 512, 28, 28]    2,359,808            True
│    └─ReLU (20)                         [32, 512, 28, 28]    [32, 512, 28, 28]    --                   --
│    └─Conv2d (21)                       [32, 512, 28, 28]    [32, 512, 28, 28]    2,359,808            True
│    └─ReLU (22)                         [32, 512, 28, 28]    [32, 512, 28, 28]    --                   --
│    └─MaxPool2d (23)                    [32, 512, 28, 28]    [32, 512, 14, 14]    --                   --
│    └─Conv2d (24)                       [32, 512, 14, 14]    [32, 512, 14, 14]    2,359,808            True
│    └─ReLU (25)                         [32, 512, 14, 14]    [32, 512, 14, 14]    --                   --
│    └─Conv2d (26)                       [32, 512, 14, 14]    [32, 512, 14, 14]    2,359,808            True
│    └─ReLU (27)                         [32, 512, 14, 14]    [32, 512, 14, 14]    --                   --
│    └─Conv2d (28)                       [32, 512, 14, 14]    [32, 512, 14, 14]    2,359,808            True
│    └─ReLU (29)                         [32, 512, 14, 14]    [32, 512, 14, 14]    --                   --
│    └─MaxPool2d (30)                    [32, 512, 14, 14]    [32, 512, 7, 7]      --                   --
├─AdaptiveAvgPool2d (avgpool)            [32, 512, 7, 7]      [32, 512, 7, 7]      --                   --
├─Sequential (classifier)                [32, 25088]          [32, 8]              --                   True
│    └─Linear (0)                        [32, 25088]          [32, 4096]           102,764,544          True
│    └─ReLU (1)                          [32, 4096]           [32, 4096]           --                   --
│    └─Dropout (2)                       [32, 4096]           [32, 4096]           --                   --
│    └─Linear (3)                        [32, 4096]           [32, 4096]           16,781,312           True
│    └─ReLU (4)                          [32, 4096]           [32, 4096]           --                   --
│    └─Dropout (5)                       [32, 4096]           [32, 4096]           --                   --
│    └─Linear (6)                        [32, 4096]           [32, 8]              32,776               True
===================================================================================================================
Total params: 134,293,320
Trainable params: 134,033,160
Non-trainable params: 260,160
Total mult-adds (Units.GIGABYTES): 495.35
===================================================================================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 3470.26
Params size (MB): 537.17
Estimated Total Size (MB): 4026.71
===================================================================================================================
```

Figure 5: VGG16 fine-tuned classifier architecture.

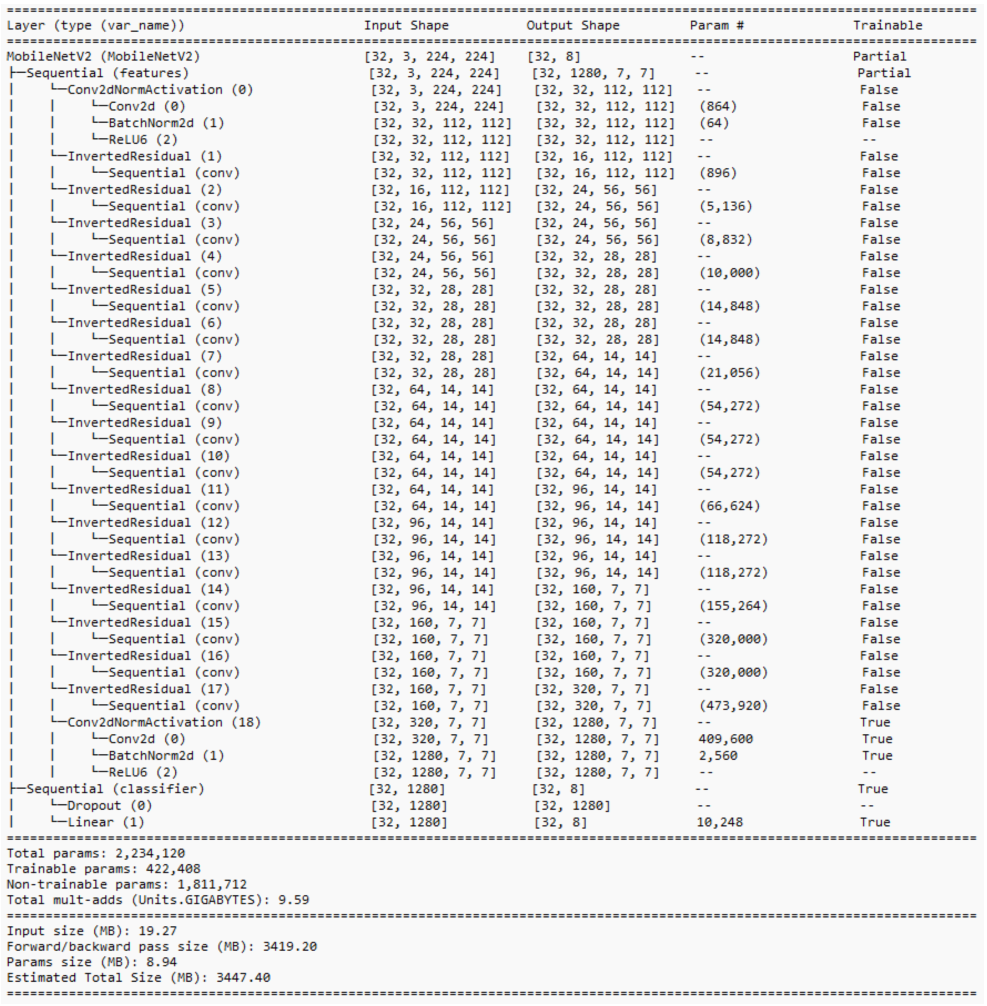
ResNet50: ResNet50, characterized by its deep architecture and skip connections, was adapted similarly. The earlier blocks were frozen, while the latter blocks and the classifier were fine-tuned to cater to our domain-specific needs [4]. We conducted iterative trials unfreezing each sequential layer from the bottom, and obtained the best results by unfreezing all layers. Figure 6 shows the resulting architecture.



```
=====================================================================================================================
Layer (type (var_name))                   Input Shape          Output Shape         Param #              Trainable
=====================================================================================================================
ResNet (ResNet)                           [32, 3, 224, 224]    [32, 8]              --                   True
├─Conv2d (conv1)                          [32, 3, 224, 224]    [32, 64, 112, 112]   9,408                True
├─BatchNorm2d (bn1)                       [32, 64, 112, 112]   [32, 64, 112, 112]   128                  True
├─ReLU (relu)                             [32, 64, 112, 112]   [32, 64, 112, 112]   --                   --
├─MaxPool2d (maxpool)                     [32, 64, 112, 112]   [32, 64, 56, 56]     --                   --
├─Sequential (layer1)                     [32, 64, 56, 56]     [32, 256, 56, 56]    --                   True
│    └─Bottleneck (0)                     [32, 64, 56, 56]     [32, 256, 56, 56]    --                   True
│    └─Bottleneck (1)                     [32, 256, 56, 56]    [32, 256, 56, 56]    --                   True
│    └─Bottleneck (2)                     [32, 256, 56, 56]    [32, 256, 56, 56]    --                   True
├─Sequential (layer2)                     [32, 256, 56, 56]    [32, 512, 28, 28]    --                   True
│    └─Bottleneck (0)                     [32, 256, 56, 56]    [32, 512, 28, 28]    --                   True
│    └─Bottleneck (1)                     [32, 512, 28, 28]    [32, 512, 28, 28]    --                   True
│    └─Bottleneck (2)                     [32, 512, 28, 28]    [32, 512, 28, 28]    --                   True
│    └─Bottleneck (3)                     [32, 512, 28, 28]    [32, 512, 28, 28]    --                   True
├─Sequential (layer3)                     [32, 512, 28, 28]    [32, 1024, 14, 14]   --                   True
│    └─Bottleneck (0)                     [32, 512, 28, 28]    [32, 1024, 14, 14]   --                   True
│    └─Bottleneck (1)                     [32, 1024, 14, 14]   [32, 1024, 14, 14]   --                   True
│    └─Bottleneck (2)                     [32, 1024, 14, 14]   [32, 1024, 14, 14]   --                   True
│    └─Bottleneck (3)                     [32, 1024, 14, 14]   [32, 1024, 14, 14]   --                   True
│    └─Bottleneck (4)                     [32, 1024, 14, 14]   [32, 1024, 14, 14]   --                   True
│    └─Bottleneck (5)                     [32, 1024, 14, 14]   [32, 1024, 14, 14]   --                   True
├─Sequential (layer4)                     [32, 1024, 14, 14]   [32, 2048, 7, 7]     --                   True
│    └─Bottleneck (0)                     [32, 1024, 14, 14]   [32, 2048, 7, 7]     --                   True
│    └─Bottleneck (1)                     [32, 2048, 7, 7]     [32, 2048, 7, 7]     --                   True
│    └─Bottleneck (2)                     [32, 2048, 7, 7]     [32, 2048, 7, 7]     --                   True
├─AdaptiveAvgPool2d (avgpool)             [32, 2048, 7, 7]     [32, 2048, 1, 1]     --                   --
├─Sequential (fc)                         [32, 2048]           [32, 8]              --                   True
│    └─Linear (0)                         [32, 2048]           [32, 8]              16,392               True
=====================================================================================================================
Total params: 23,524,424
Trainable params: 23,524,424
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 130.79
=====================================================================================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 5690.36
Params size (MB): 94.10
Estimated Total Size (MB): 5803.73
=====================================================================================================================
```

Figure 6: ResNet50 fine-tuned classifier architecture, output truncated for brevity.

MobileNetV2: For MobileNetV2, an architecture designed for mobile devices, the earlier inverted residual blocks were kept frozen, whereas the subsequent blocks and the classifier were fine-tuned for our task [6]. We found the highest accuracy score is attained with just the penultimate layer unfrozen. Figure 7 shows the resulting architecture.



```
==============================================================================================================
Layer (type (var_name))                      Input Shape          Output Shape         Param #              Trainable
==============================================================================================================
MobileNetV2 (MobileNetV2)                    [32, 3, 224, 224]    [32, 8]              --                   Partial
├─Sequential (features)                      [32, 3, 224, 224]    [32, 1280, 7, 7]     --                   Partial
│    └─Conv2dNormActivation (0)              [32, 3, 224, 224]    [32, 32, 112, 112]   --                   False
│    │    └─Conv2d (0)                       [32, 3, 224, 224]    [32, 32, 112, 112]   (864)                False
│    │    └─BatchNorm2d (1)                  [32, 32, 112, 112]   [32, 32, 112, 112]   (64)                 False
│    │    └─ReLU6 (2)                        [32, 32, 112, 112]   [32, 32, 112, 112]   --                   --
│    └─InvertedResidual (1)                  [32, 32, 112, 112]   [32, 16, 112, 112]   --                   False
│    │    └─Sequential (conv)                [32, 32, 112, 112]   [32, 16, 112, 112]   (896)                False
│    └─InvertedResidual (2)                  [32, 16, 112, 112]   [32, 24, 56, 56]     --                   False
│    │    └─Sequential (conv)                [32, 16, 112, 112]   [32, 24, 56, 56]     (5,136)              False
│    └─InvertedResidual (3)                  [32, 24, 56, 56]     [32, 24, 56, 56]     --                   False
│    │    └─Sequential (conv)                [32, 24, 56, 56]     [32, 24, 56, 56]     (8,832)              False
│    └─InvertedResidual (4)                  [32, 24, 56, 56]     [32, 32, 28, 28]     --                   False
│    │    └─Sequential (conv)                [32, 24, 56, 56]     [32, 32, 28, 28]     (10,000)             False
│    └─InvertedResidual (5)                  [32, 32, 28, 28]     [32, 32, 28, 28]     --                   False
│    │    └─Sequential (conv)                [32, 32, 28, 28]     [32, 32, 28, 28]     (14,848)             False
│    └─InvertedResidual (6)                  [32, 32, 28, 28]     [32, 32, 28, 28]     --                   False
│    │    └─Sequential (conv)                [32, 32, 28, 28]     [32, 32, 28, 28]     (14,848)             False
│    └─InvertedResidual (7)                  [32, 32, 28, 28]     [32, 64, 14, 14]     --                   False
│    │    └─Sequential (conv)                [32, 32, 28, 28]     [32, 64, 14, 14]     (21,056)             False
│    └─InvertedResidual (8)                  [32, 64, 14, 14]     [32, 64, 14, 14]     --                   False
│    │    └─Sequential (conv)                [32, 64, 14, 14]     [32, 64, 14, 14]     (54,272)             False
│    └─InvertedResidual (9)                  [32, 64, 14, 14]     [32, 64, 14, 14]     --                   False
│    │    └─Sequential (conv)                [32, 64, 14, 14]     [32, 64, 14, 14]     (54,272)             False
│    └─InvertedResidual (10)                 [32, 64, 14, 14]     [32, 64, 14, 14]     --                   False
│    │    └─Sequential (conv)                [32, 64, 14, 14]     [32, 64, 14, 14]     (54,272)             False
│    └─InvertedResidual (11)                 [32, 64, 14, 14]     [32, 96, 14, 14]     --                   False
│    │    └─Sequential (conv)                [32, 64, 14, 14]     [32, 96, 14, 14]     (66,624)             False
│    └─InvertedResidual (12)                 [32, 96, 14, 14]     [32, 96, 14, 14]     --                   False
│    │    └─Sequential (conv)                [32, 96, 14, 14]     [32, 96, 14, 14]     (118,272)            False
│    └─InvertedResidual (13)                 [32, 96, 14, 14]     [32, 96, 14, 14]     --                   False
│    │    └─Sequential (conv)                [32, 96, 14, 14]     [32, 96, 14, 14]     (118,272)            False
│    └─InvertedResidual (14)                 [32, 96, 14, 14]     [32, 160, 7, 7]      --                   False
│    │    └─Sequential (conv)                [32, 96, 14, 14]     [32, 160, 7, 7]      (155,264)            False
│    └─InvertedResidual (15)                 [32, 160, 7, 7]      [32, 160, 7, 7]      --                   False
│    │    └─Sequential (conv)                [32, 160, 7, 7]      [32, 160, 7, 7]      (320,000)            False
│    └─InvertedResidual (16)                 [32, 160, 7, 7]      [32, 160, 7, 7]      --                   False
│    │    └─Sequential (conv)                [32, 160, 7, 7]      [32, 160, 7, 7]      (320,000)            False
│    └─InvertedResidual (17)                 [32, 160, 7, 7]      [32, 320, 7, 7]      --                   False
│    │    └─Sequential (conv)                [32, 160, 7, 7]      [32, 320, 7, 7]      (473,920)            False
│    └─Conv2dNormActivation (18)             [32, 320, 7, 7]      [32, 1280, 7, 7]     --                   True
│    │    └─Conv2d (0)                       [32, 320, 7, 7]      [32, 1280, 7, 7]     409,600              True
│    │    └─BatchNorm2d (1)                  [32, 1280, 7, 7]     [32, 1280, 7, 7]     2,560                True
│    │    └─ReLU6 (2)                        [32, 1280, 7, 7]     [32, 1280, 7, 7]     --                   --
├─Sequential (classifier)                    [32, 1280]           [32, 8]              --                   True
│    └─Dropout (0)                           [32, 1280]           [32, 1280]           --                   --
│    └─Linear (1)                            [32, 1280]           [32, 8]              10,248               True
==============================================================================================================
Total params: 2,234,120
Trainable params: 422,408
Non-trainable params: 1,811,712
Total mult-adds (Units.GIGABYTES): 9.59
==============================================================================================================
Input size (MB): 19.27
Forward/backward pass size (MB): 3419.20
Params size (MB): 8.94
Estimated Total Size (MB): 3447.40
==============================================================================================================
```

Figure 7: MobileNetV2 fine-tuned classifier architecture.

3 Results

3.1 Fixed Feature Classification

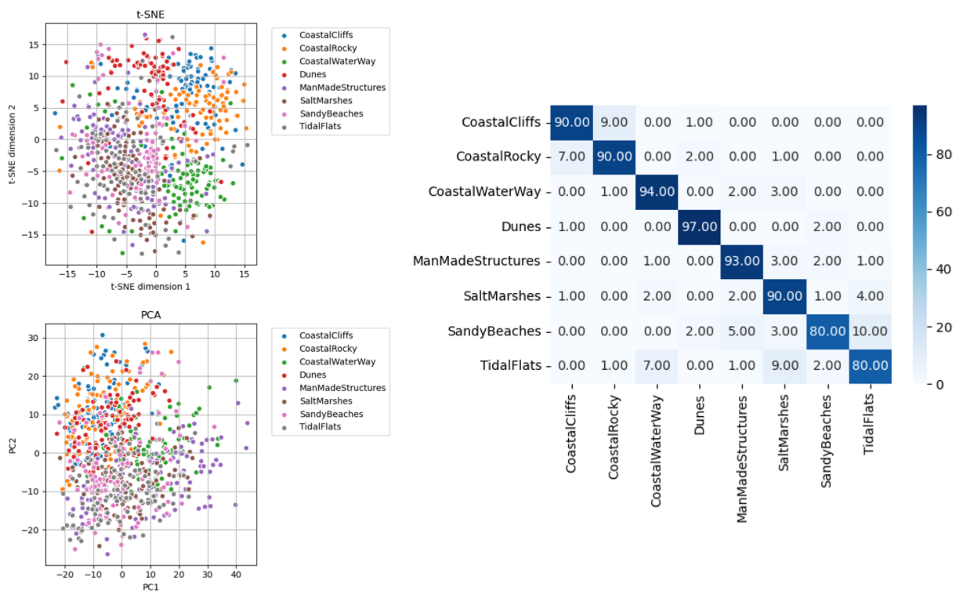
Using our three pretrained models for feature extraction, we found VGG16 yielded the best results. Table 2 shows the classification accuracy, precision, recall, and F1-score for the three fixed feature models tested.

Fixed Feature Classification				
VGG 16	Precision	Recall	F1	Accuracy
Coastal Cliffs	0.9091	0.9000	0.9045	0.8925
Coastal Rocks	0.8911	0.9000	0.8955	
Coastal Waterway	0.9038	0.9400	0.9216	
Dunes	0.9510	0.9700	0.9604	
Man-made Structures	0.9029	0.9300	0.9163	
Salt Marshes	0.8257	0.9000	0.8612	
Sandy Beaches	0.9195	0.8000	0.8556	
Tidal Flats	0.8421	0.8000	0.8205	
ResNet 50	Precision	Recall	F1	Accuracy
Coastal Cliffs	0.8208	0.8700	0.8447	0.8250
Coastal Rocks	0.8526	0.8100	0.8308	
Coastal Waterway	0.8137	0.8300	0.8212	
Dunes	0.8485	0.8400	0.8442	
Man-made Structures	0.8142	0.9200	0.8638	
Salt Marshes	0.8198	0.9100	0.8626	
Sandy Beaches	0.8243	0.6100	0.7011	
Tidal Flats	0.8100	0.8100	0.8100	
MobileNet V2	Precision	Recall	F1	Accuracy
Coastal Cliffs	0.7370	0.8500	0.8173	0.8263
Coastal Rocks	0.8778	0.7900	0.8316	
Coastal Waterway	0.8469	0.8300	0.8384	
Dunes	0.8091	0.8900	0.8476	
Man-made Structures	0.8785	0.9400	0.9082	
Salt Marshes	0.7905	0.8300	0.8098	
Sandy Beaches	0.7791	0.6700	0.7204	
Tidal Flats	0.8438	0.8100	0.8265	

Table 2: Fixed feature classification results.

3.1.1 VGG16 Fixed Feature Model

Figure 8 shows the confusion matrix, t-SNE plot, and PCA plot for VGG16 with fixed feature layers. From this visualization, we can see that while the model performs well, it has more difficulty distinguishing tidal flats, salt marshes, and sandy beaches. The t-SNE plot indicates this model did reasonably well in separating the different classes in the data, particularly dunes, coastal waterways, and coastal rocks, but also shows difficulty separating tidal flats from sandy beaches and salt marshes, supporting the information from the confusion matrix.

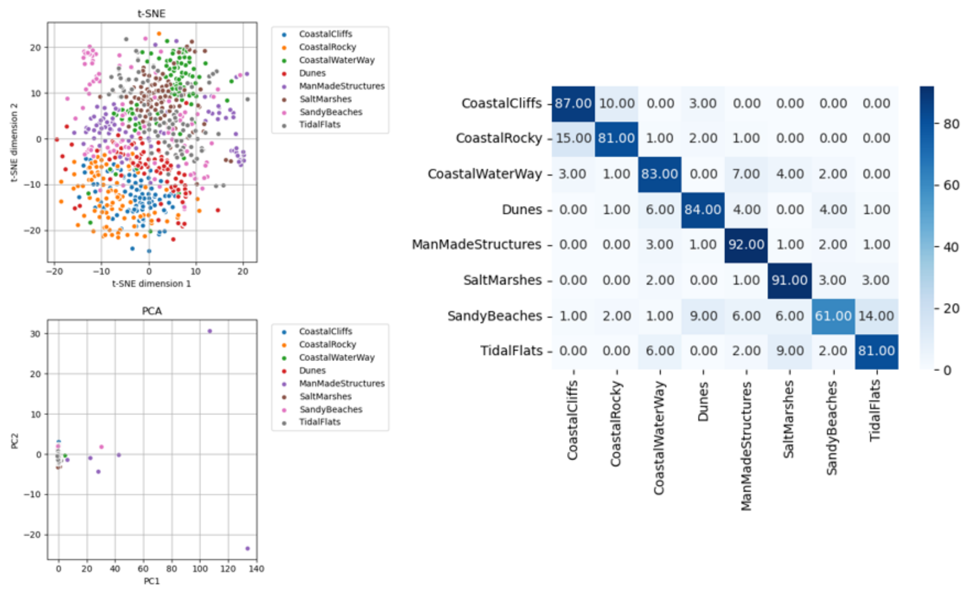


	CoastalCliffs	CoastalRocky	CoastalWaterWay	Dunes	ManMadeStructures	SaltMarshes	SandyBeaches	TidalFlats
CoastalCliffs	90.00	9.00	0.00	1.00	0.00	0.00	0.00	0.00
CoastalRocky	7.00	90.00	0.00	2.00	0.00	1.00	0.00	0.00
CoastalWaterWay	0.00	1.00	94.00	0.00	2.00	3.00	0.00	0.00
Dunes	1.00	0.00	0.00	97.00	0.00	0.00	2.00	0.00
ManMadeStructures	0.00	0.00	1.00	0.00	93.00	3.00	2.00	1.00
SaltMarshes	1.00	0.00	2.00	0.00	2.00	90.00	1.00	4.00
SandyBeaches	0.00	0.00	0.00	2.00	5.00	3.00	80.00	10.00
TidalFlats	0.00	1.00	7.00	0.00	1.00	9.00	2.00	80.00

Figure 8: VGG16 fixed feature classifier confusion matrix, t-SNE plot, and PCA plot.

3.1.2 ResNet50 Fixed Feature Model

Figure 9 shows the confusion matrix, t-SNE plot, and PCA plot for ResNet50 with fixed feature layers. We can see ResNet50 struggles with classifying between tidal flats, salt marshes, and sandy beaches to a greater extent than VGG16, and additionally has difficulty with classifying between coastal cliffs or coastal rocks and sandy beaches or dunes.

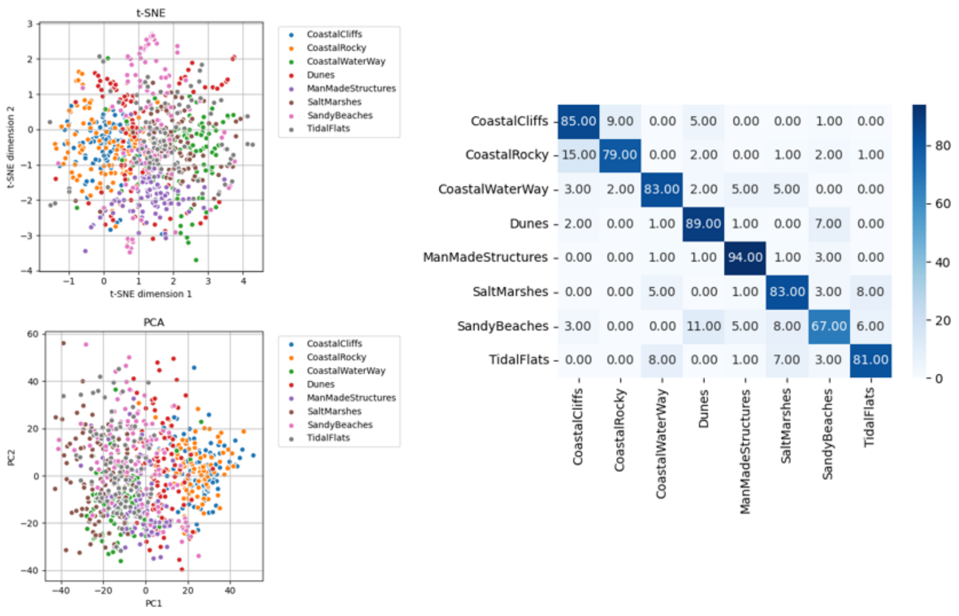


	CoastalCliffs	CoastalRocky	CoastalWaterWay	Dunes	ManMadeStructures	SaltMarshes	SandyBeaches	TidalFlats
CoastalCliffs	87.00	10.00	0.00	3.00	0.00	0.00	0.00	0.00
CoastalRocky	15.00	81.00	1.00	2.00	1.00	0.00	0.00	0.00
CoastalWaterWay	3.00	1.00	83.00	0.00	7.00	4.00	2.00	0.00
Dunes	0.00	1.00	6.00	84.00	4.00	0.00	4.00	1.00
ManMadeStructures	0.00	0.00	3.00	1.00	92.00	1.00	2.00	1.00
SaltMarshes	0.00	0.00	2.00	0.00	1.00	91.00	3.00	3.00
SandyBeaches	1.00	2.00	1.00	9.00	6.00	6.00	61.00	14.00
TidalFlats	0.00	0.00	6.00	0.00	2.00	9.00	2.00	81.00

Figure 9: ResNet50 fixed feature classifier confusion matrix, t-SNE plot, and PCA plot.

3.1.3 MobileNetV2 Fixed Feature Model

Figure 10 shows the confusion matrix, t-SNE plot, and PCA plot for MobileNetV2 with fixed feature layers. We can see that it performs similarly to ResNet50 with fixed feature layers, and struggles with generally the same categories of littoral features.



	CoastalCliffs	CoastalRocky	CoastalWaterWay	Dunes	ManMadeStructures	SaltMarshes	SandyBeaches	TidalFlats
CoastalCliffs	85.00	9.00	0.00	5.00	0.00	0.00	1.00	0.00
CoastalRocky	15.00	79.00	0.00	2.00	0.00	1.00	2.00	1.00
CoastalWaterWay	3.00	2.00	83.00	2.00	5.00	5.00	0.00	0.00
Dunes	2.00	0.00	1.00	89.00	1.00	0.00	7.00	0.00
ManMadeStructures	0.00	0.00	1.00	1.00	94.00	1.00	3.00	0.00
SaltMarshes	0.00	0.00	5.00	0.00	1.00	83.00	3.00	8.00
SandyBeaches	3.00	0.00	0.00	11.00	5.00	8.00	67.00	6.00
TidalFlats	0.00	0.00	8.00	0.00	1.00	7.00	3.00	81.00

Figure 10: MobileNetV2 fixed feature classifier confusion matrix, t-SNE plot, and PCA plot.

3.2 Fine-tuned classification

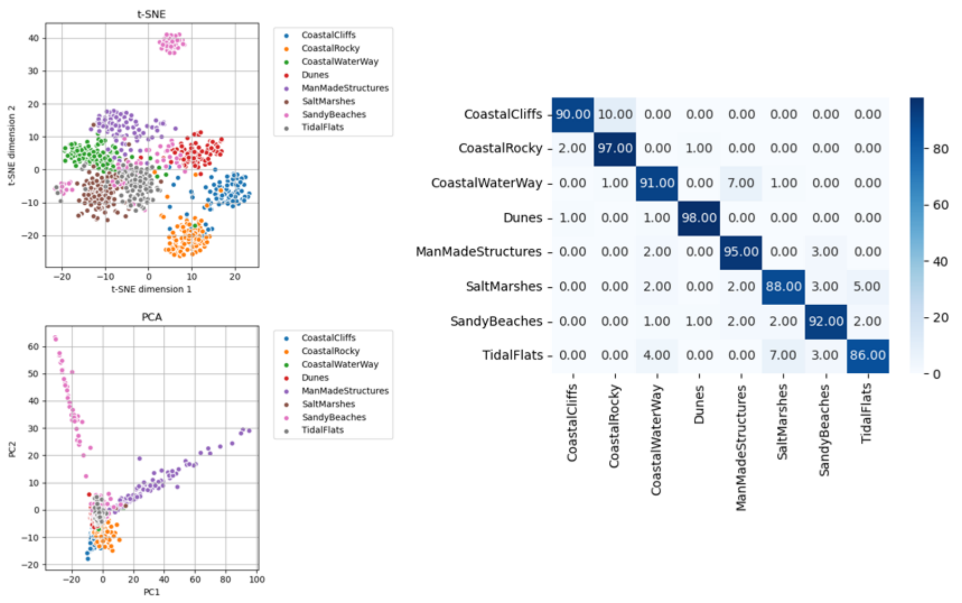
Table 3 shows the classification accuracy, precision, recall, and F1-score for the three pretrained models used, with specific layers unfrozen to produce models more fine-tuned to our dataset instead of the ImageNet dataset. Out of the experiments we conducted, our fine-tuned ResNet50 model had the highest performance by accuracy score.

Fine-tuned Classification				
VGG 16	Precision	Recall	F1	Accuracy
Coastal Cliffs	0.9677	0.9000	0.9326	0.9213
Coastal Rocks	0.8981	0.9700	0.9327	
Coastal Waterway	0.9010	0.9100	0.9055	
Dunes	0.9800	0.9800	0.9800	
Man-made Structures	0.8962	0.9500	0.9223	
Salt Marshes	0.8980	0.8800	0.8889	
Sandy Beaches	0.9109	0.9200	0.9154	
Tidal Flats	0.9247	0.8600	0.8912	
ResNet50	Precision	Recall	F1	Accuracy
Coastal Cliffs	0.9789	0.9300	0.9538	0.9275
Coastal Rocks	0.9245	0.9800	0.9515	
Coastal Waterway	0.9394	0.9300	0.9347	
Dunes	0.9709	1.0000	0.9852	
Man-made Structures	0.9579	0.9100	0.9333	
Salt Marshes	0.8519	0.9200	0.8846	
Sandy Beaches	0.8667	0.9100	0.8878	
Tidal Flats	0.9438	0.8400	0.8889	
MobileNetV2	Precision	Recall	F1	Accuracy
Coastal Cliffs	0.9247	0.8600	0.8912	0.8975
Coastal Rocks	0.8796	0.9500	0.9135	
Coastal Waterway	0.8679	0.9200	0.8932	
Dunes	0.8879	0.9500	0.9179	
Man-made Structures	0.9400	0.9400	0.9400	
Salt Marshes	0.8440	0.9200	0.8804	
Sandy Beaches	0.9390	0.7700	0.8462	
Tidal Flats	0.9158	0.8700	0.8923	

Table 3: Fine-tuned classification results.

3.2.1 VGG16 Fine-tuned Model

Figure 11 shows the confusion matrix, t-SNE plot, and PCA plot for VGG16 with fine-tuning. The fine-tuned model produced better results than the fixed feature model, but still had some difficulty distinguishing coastal cliffs or coastal rocky images and salt marshes or tidal flats. Of the three types of models tested, VGG16 attained the smallest performance gain, and ResNet50 attained the largest performance gain with fine-tuning.

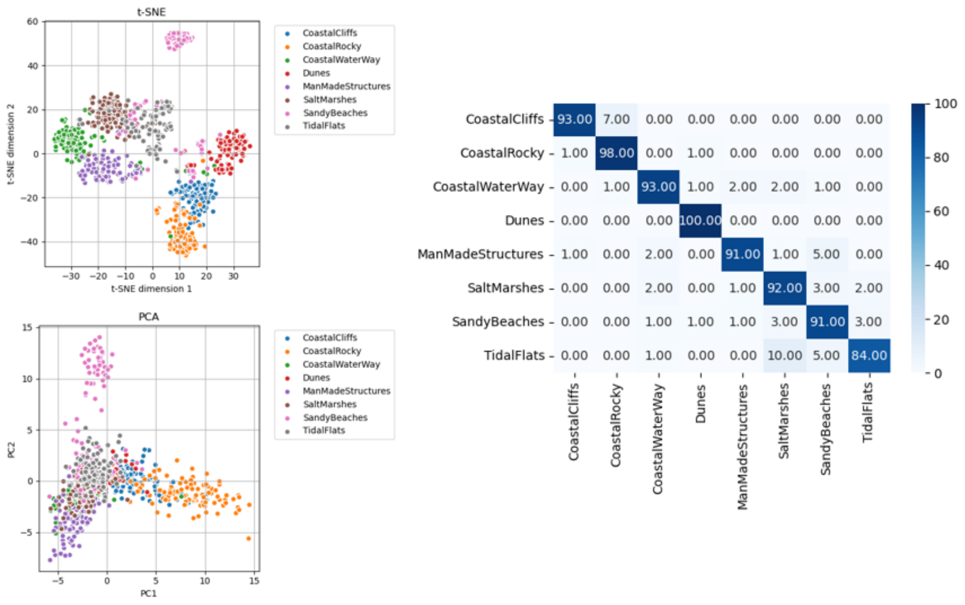


	CoastalCliffs	CoastalRocky	CoastalWaterWay	Dunes	ManMadeStructures	SaltMarshes	SandyBeaches	TidalFlats
CoastalCliffs	90.00	10.00	0.00	0.00	0.00	0.00	0.00	0.00
CoastalRocky	2.00	97.00	0.00	1.00	0.00	0.00	0.00	0.00
CoastalWaterWay	0.00	1.00	91.00	0.00	7.00	1.00	0.00	0.00
Dunes	1.00	0.00	1.00	98.00	0.00	0.00	0.00	0.00
ManMadeStructures	0.00	0.00	2.00	0.00	95.00	0.00	3.00	0.00
SaltMarshes	0.00	0.00	2.00	0.00	2.00	88.00	3.00	5.00
SandyBeaches	0.00	0.00	1.00	1.00	2.00	2.00	92.00	2.00
TidalFlats	0.00	0.00	4.00	0.00	0.00	7.00	3.00	86.00

Figure 11: VGG16 fine-tuned classifier confusion matrix, t-SNE plot, and PCA plot.

3.2.2 ResNet50 Fine-tuned Model

Figure 12 shows the confusion matrix, t-SNE plot, and PCA plot for ResNet50 with fine-tuning. Compared to the fixed-feature model, fine-tuning results in much better separation between classes in the t-SNE plot and better performance as reflected in the confusion matrix. However, the fine-tuned model still continues to have most difficulty distinguishing between salt marshes and tidal flats.



	CoastalCliffs	CoastalRocky	CoastalWaterWay	Dunes	ManMadeStructures	SaltMarshes	SandyBeaches	TidalFlats
CoastalCliffs	93.00	7.00	0.00	0.00	0.00	0.00	0.00	0.00
CoastalRocky	1.00	98.00	0.00	1.00	0.00	0.00	0.00	0.00
CoastalWaterWay	0.00	1.00	93.00	1.00	2.00	2.00	1.00	0.00
Dunes	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00
ManMadeStructures	1.00	0.00	2.00	0.00	91.00	1.00	5.00	0.00
SaltMarshes	0.00	0.00	2.00	0.00	1.00	92.00	3.00	2.00
SandyBeaches	0.00	0.00	1.00	1.00	1.00	3.00	91.00	3.00
TidalFlats	0.00	0.00	1.00	0.00	0.00	10.00	5.00	84.00

Figure 12: ResNet50 fine-tuned classifier confusion matrix, t-SNE plot, and PCA plot.

3.2.3 MobileNetV2 Fine-tuned Model

Figure 13 shows the confusion matrix, t-SNE plot, and PCA plot for MobileNetV2 with fine-tuning. MobileNetV2 had the smallest performance gain with fine-tuning, but improved in classifying man-made structures and coastal rocky areas. Dunes and sandy beaches remained relatively difficult for MobileNetV2 to classify.

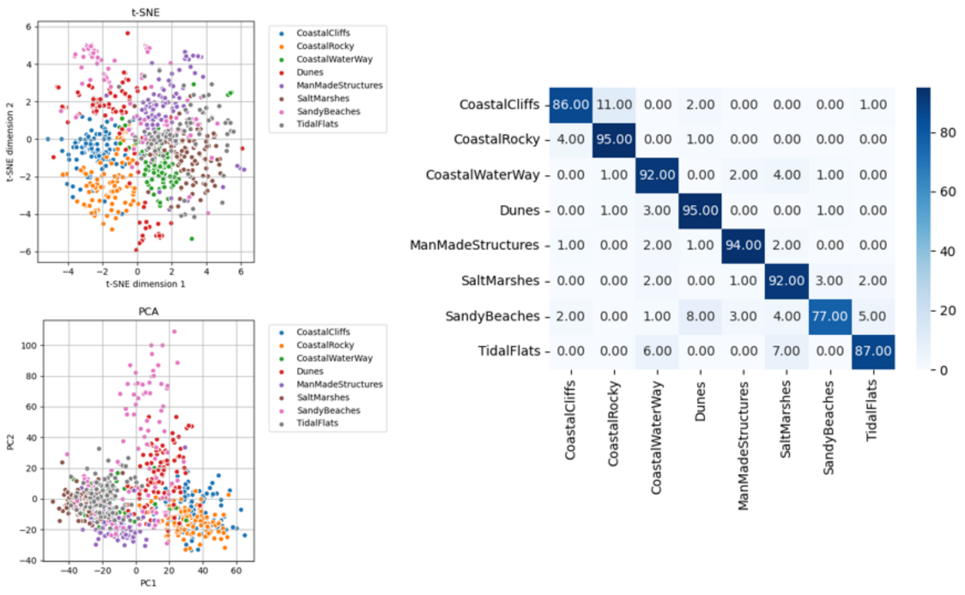


Figure 13: MobileNetV2 fine-tuned classifier confusion matrix, t-SNE plot, and PCA plot.

4 Discussion

Of the three model architectures tested (VGG16, ResNet50, and MobileNetV2) the highest-performing architecture with fixed feature extraction layers was VGG16, with about a seven percent accuracy score lead. ResNet50 was the highest-performing model architecture after fine-tuning by unfreezing all of its layers. However, considering the operational problem to be addressed by this project, it is important to note the performance gains of VGG16 and ResNet50 may not be a good tradeoff for their significantly larger size compared to MobileNetV2. A more lightweight model with lesser, but still acceptable, performance may be preferred in a tactical environment with limited storage and computational resources.

Kornblith et al. [8] investigated the relationship between ImageNet accuracy and transfer accuracy for a variety of image classification tasks. They found that there is a strong correlation between the two, with better ImageNet accuracy leading to better transfer accuracy. This suggests that ImageNet-trained networks are able to learn generalizable features that are useful for other tasks.

The paper also found that the way in which networks are trained on ImageNet can have a significant impact on their transferability. Networks trained with regularization techniques, such as dropout or L2 regularization, tend to have lower transfer accuracy than networks trained without regularization. This is because regularization techniques can help to prevent overfitting to the ImageNet training data, but they can also make the features learned by the network less generalizable [20].

Finally, the paper found that the size of the target dataset can also affect transfer accuracy. Networks transferred to smaller datasets tend to have lower transfer accuracy than networks transferred to larger datasets. This is because smaller datasets provide less training data for the network to learn from, which can make it more difficult for the network to generalize to new data [21].

The findings of the Kornblith paper have important implications for the use of transfer learning in real-world applications. The results suggest that choosing an ImageNet-trained network with high accuracy is important. It is also important to avoid using regularization techniques when training the network on ImageNet, as this can reduce its transferability. Finally, if the target dataset is small, it may be necessary to fine-tune the target dataset's network to achieve optimal performance.

References

[1] U.S. Naval Operations, *The Role of Environmental Information*, U.S. Navy Publication, 2020.

[2] Smith, J., *Littoral Zone Reconnaissance and Hydrographic Survey*, Naval Institute Press, 2019.

[3] Harris, A., *Machine Learning in Military Operations*, Defense Technical Information Center, 2022.

[4] He, K., Zhang, X., Ren, S., and Sun, J., *Deep Residual Learning for Image Recognition*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016.

[5] Simonyan, K. and Zisserman, A., *Very Deep Convolutional Networks for Large-Scale Image Recognition*, arXiv preprint arXiv:1409.1556, 2015.

[6] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.C., *MobileNetV2: Inverted Residuals and Linear Bottlenecks*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018.

[7] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al., *PyTorch: An imperative style, high-performance deep learning library*, Advances in Neural Information Processing Systems, 2019.

[8] Kornblith, S., Shlens, J., and Le, Q. V., *Do Better ImageNet Models Transfer Better?*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019.

[9] Naval Research Laboratory, *Importance of Machine Learning in Naval Strategies*, U.S. Navy Research Report, 2022.

[10] Goodfellow, I., Bengio, Y., and Courville, A., *Deep Learning*, MIT Press, 2016.

[11] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H., *How transferable are features in deep neural networks?*, Advances in neural information processing systems, 2014.

[12] Scikit-learn, *Manifold learning*, Accessed August 11, 2023. [Online]. Available: https://scikit-learn.org/stable/modules/manifold.html.

[13] Scikit-learn, *Decomposing signals in components (matrix factorization problems)*, Accessed August 11, 2023. [Online]. Available: https://scikit-learn.org/stable/modules/decomposition.html.

[14] Russakovsky, O., Deng, J., Su, H., et al., *ImageNet Large Scale Visual Recognition Challenge*, International Journal of Computer Vision (IJCV), Vol 115, Issue 3, 2015, pp. 211–252.

[15] Shorten, C., & Khoshgoftaar, T. M., *A survey on Image Data Augmentation for Deep Learning*, Journal of Big Data, 6(1), 2019, pp. 60.

[16] Krizhevsky, A., Sutskever, I., & Hinton, G. E., *ImageNet classification with deep convolutional neural networks*, In Advances in neural information processing systems, 2012, pp. 1097-1105.

[17] Simard, P. Y., Steinkraus, D., & Platt, J. C., *Best practices for convolutional neural networks applied to visual document analysis*, In ICDAR, Vol. 3, No. 2003.

[18] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L., *ImageNet: A large-scale hierarchical image database*, In 2009 IEEE conference on computer vision and pattern recognition, 2009, pp. 248-255.

[19] Ioffe, S., & Szegedy, C., *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, arXiv preprint, 2015. Available: arXiv:1502.03167.

[20] Morey, R. D. (2008). *Confidence intervals for adjusted means*. Psychological Methods, 13(4), 319-334.

[21] Torralba, A., & Efros, A. A. (2011). *Unbiased look at dataset bias*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1521-1528).