Team AB

CE 784A Project Report: Semester 2024-25 (II)

**Aerial Scene Classification**

Anshuman Kumar Yadav (241030018), Harish Babu Kumara (241030038)

Rupak Kumar (241030068), Srashti Singh (241030068)

Abstract.This study explores the challenge of aerial scene classification using the high - resolution images from Google Earth Aerial Image Dataset (AID), focusing on developing an efficient and accurate approach to categorize 30 distinct scene classes. The classification task is particularly complex due to significant intra-class variations, where images within the same category exhibit diverse appearances, and subtle inter-class differences, making it difficult to distinguish between visually similar classes. By addressing these challenges, the research aims to enhance scene recognition performance and contribute to the advancement of aerial image analysis.

Keywords: Model training; Transfer learning; Attention mechanisms; Feature extraction; Satellite image classification; Convolutional Neural Networks (CNNs); Remote sensing; Data augmentation

1. INTRODUCTION

Classifying aerial imagery plays a crucial role in remote sensing and geospatial applications, significantly contributing to fields such as urban development, environmental assessment, agriculture, and disaster mitigation. This process involves interpreting high-resolution aerial photographs to derive meaningful patterns and categorize them into semantic groups, transforming raw image data into practical insights.

Aerial scene classification involves recognizing and assigning labels to entire scenes captured from an overhead viewpoint. This capability is essential for producing expansive geospatial datasets that assist researchers, policymakers, and industries in making informed decisions. Distinguishing between similar landscapes, such as industrial and residential zones or forests and shrublands, is vital for accurate spatial analysis.

However, the task is complicated by various factors inherent to aerial images. High intra-class variation—caused by seasonal shifts, weather changes, or differences in sensor quality—makes it difficult to group similar categories. At the same time, scenes from different categories often appear visually similar, which increases the challenge of classification. The size of the dataset and the presence of overlapping scene elements within a single image add further complexity.

In recent years, deep learning models especially Convolutional Neural Networks (CNNs), Vision Transformers (ViTs), and their hybrid variants have significantly advanced aerial image analysis. These models can capture complex spatial relationships and patterns within images. Enhancements like attention mechanisms and transfer learning have further improved classification accuracy. Nevertheless, key challenges remain in maintaining computational efficiency, ensuring generalization across datasets, and improving interpretability.

To meet rising demands for scalable and precise aerial classification, research continues to explore newer frontiers—such as the integration of multi-source data (e.g., LiDAR or hyperspectral imagery) and the use of unsupervised or semi-supervised learning strategies. These efforts aim to build robust systems capable of handling real-world geospatial challenges both now and in the future.

**2. STUDY OBJECTIVE**

This research aims to build an effective and accurate multi-class classification pipeline for aerial scenes using high-resolution satellite imagery. The project leverages the AID dataset, which contains 30 diverse scene categories with 200–400 samples each, to design a reliable classification model that can discern complex terrain and land use types.

The focus is on utilizing state-of-the-art machine learning methods to extract features and assign labels to aerial scenes. Special attention is given to addressing issues such as large intra-class variation and subtle inter-class distinctions that are present in the dataset. The study also incorporates recent developments in the field, including the exploration of Vision-Language Models, to test their potential in improving classification outcomes.

Ultimately, this work contributes to the growing field of geospatial data analysis by proposing a scalable, data-driven solution applicable to real-world tasks like city planning, ecological monitoring, and emergency response.

**3. LITERATURE REVIEW**

The foundational work in [[2](file:///C:\Users\Hp\Downloads\2.docx)] presents the Aerial Image Dataset (AID), a large and diverse benchmark designed to test aerial scene classification methods. This dataset comprises high-resolution imagery across 30 categories, sourced from Google Earth, with 200–400 samples per class. The authors underscore challenges like substantial intra-class variation and nuanced differences between scene types, providing a benchmark for evaluating different classification algorithms.

The study in [3] examines how Google Earth imagery combined with Object-Based Image Analysis (OBIA) can improve land use and land cover classification. OBIA works by segmenting images into spatially and spectrally coherent objects, offering higher accuracy than traditional pixel-level classification, particularly for high-resolution satellite images.

In [4], the authors explore unsupervised learning strategies to extract key features from aerial scenes without relying on labelled data. Their results show that these unsupervised methods can effectively identify complex spatial patterns, outperforming handcrafted feature extraction techniques in several classification tasks.

Reference [5] investigates how Convolutional Neural Networks (CNNs) can be used to classify land use in satellite imagery. By employing pre-trained CNNs and fine-tuning them on remote sensing data, the study demonstrates that deep learning significantly improves performance compared to traditional methods.

The approach proposed in [6] introduces part-based object detection for handling multi-class classification in geospatial imagery. This method leverages both spatial structure and object parts to accurately identify and classify features in aerial scenes, contributing valuable insights to photogrammetry and remote sensing workflows.

In [7], a sparse coding framework is used to combine multiple image features—such as spectral, texture, and structural cues—for improved scene classification. This technique enhances feature differentiation and performs well on complex satellite images with high scene diversity.

Study [8] presents an innovative multiscale Bag-of-Visual-Words (BoVW) technique using a concentric circle layout to capture spatial and contextual patterns in remote sensing data. This structure allows better modelling of scene layouts and leads to improved classification, especially for diverse land-use types.

The research in [9] emphasizes CNNs' strengths in remote sensing tasks and offers techniques like pre-training, fine-tuning, and data augmentation to further enhance model accuracy. These strategies show how deep learning can effectively adapt to the unique characteristics of aerial imagery, offering a strong foundation for future exploration.

**4. DATASET DETAILS**

**Dataset Name:**

The dataset used in this study is the Aerial Image Dataset (AID), sourced from post-processed Google Earth imagery. It consists of over 10,000 high-resolution images divided into 30 distinct scene categories in which 7000 images has been used for training,1500 for validation and 1500 for testing. Each image measures 600 × 600 pixels, with each category represented by 200 to 400 samples.

**Source**:

Google Earth imagery (post-processed using RGB renderings from the original optical aerial images).

**Dataset Size**:

30 different scene classes and about 200 to 400 samples. In all, the AID dataset has several 10,000 images within 30 classes.

**Image Resolution**:

Each image has a size of 600x600 pixels in each class.

**Classes include:** 30 different scene types.

As part of the Exploratory Data Analysis (EDA), we analyzed the distribution of image classes in the Aerial Image Dataset (AID). The dataset comprises a wide range of land-use and land-cover classes, which vary significantly in frequency.

The bar chart below visualizes the number of images per class, revealing an imbalanced distribution. Classes such as Pond, Viaduct, and DenseResidential are the most represented, each with over 400 images. On the other hand, classes like Church, Forest, and BaseballField have comparatively fewer samples, indicating potential class imbalance issues that may affect model performance.

Understanding this distribution is crucial for guiding preprocessing steps such as data augmentation, sampling techniques, and model evaluation strategies to ensure robust and fair classification performance across all categories.

**A colorful chart with numbers and symbols

AI-generated content may be incorrect.**

**Figure 1: Class Distribution in AID Dataset**

**Labelling:**

All the images are labelled by the specialists in the field of remote sensing image interpretation.

**Class Balance:**

Each class contains between 200 and 400 samples.

**Diversity Characteristics:**

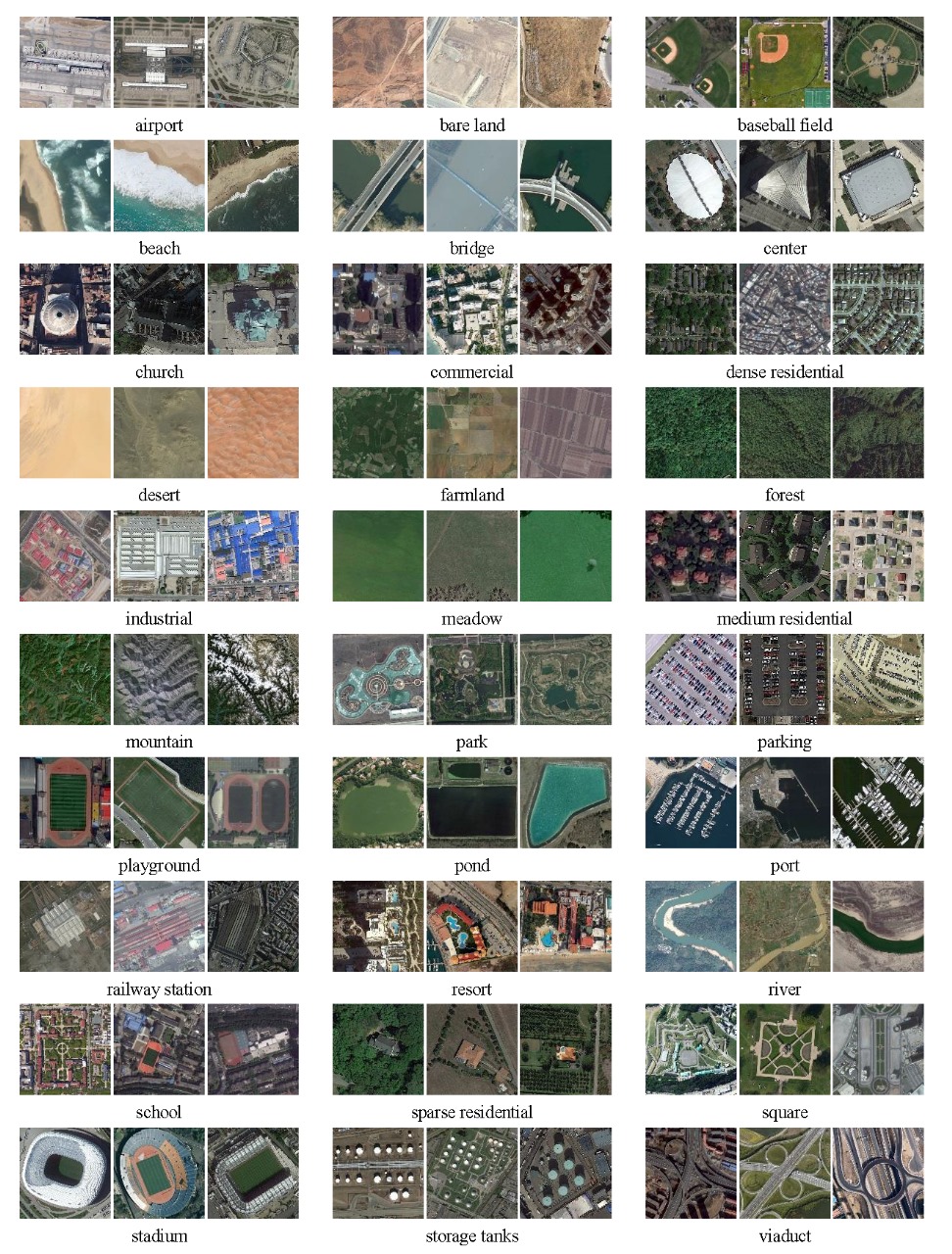
* Captured using various remote sensing platforms.
* Images originate from different geographic regions, including the U.S., China, the U.K., France, Germany, Italy, and Japan.
* The dataset captures images taken under **different seasonal conditions** (summer, winter, spring, and autumn) and times, boosting intra-class variability and **varying lighting situations** (sunny, cloudy, and shadowed environments).
* This variability helps machine learning models learn to recognize scenes despite **changes in appearance due to weather, time of day, or atmospheric conditions**.
* Furthermore, AID accounts for **differences in imaging angles and resolutions**, which are common in real-world remote sensing. This ensures that models trained on AID can generalize well to images taken from different heights, perspectives, and sensor technologies, making them more **robust and adaptable** for practical applications such as land use classification, disaster management, and environmental monitoring.

This dataset is ideal for evaluating aerial scene classification models, particularly those developed using machine learning and deep learning techniques**.**

**Dataset Access:**

The AID dataset can be accessed via Kaggle(<https://www.kaggle.com/datasets/jiayuanchengala/aid-scene-classification-datasets>).

**Purpose:** The dataset helps test and improve classification models using deep learning.



**Figure 2: Samples of the AID dataset; three examples of each semantic scene classes are shown.**

**5. GENERAL IMPLEMENTATION**

**5.1 Data Visualization**

* Sample images from all 30 categories were examined to understand visual differences.
* Some images of the same category vary due to angle, lighting, and seasonal differences.

**5.2 Class Distribution**

* The dataset is well-balanced, with each category containing 200-400 images.
* This prevents any single class from dominating the model training process.

**5.3 Feature Analysis**

* Images were analyzed for key features like texture, colour distribution, and spatial patterns.
* Edge detection techniques were used to highlight boundaries and structures within images.
* Histograms were plotted to compare pixel intensity distributions across different classes.

**5.4 Data Augmentation**

To improve model generalization, the following augmentation techniques were applied:

* Resizing images to 256×256 pixels
* Cropping to 224×224 pixels
* Random flipping (horizontal and vertical)
* Normalizing pixel values to scale between 0 and 1

**5.5 Model Readiness**

* The dataset was split into training, validation, and test sets to ensure reliable model evaluation.
* Initial tests with simple classifiers showed the need for deep learning models due to complex patterns in aerial images.

**5.6 Data Augmentation Techniques:**

The training procedure is built on curriculum learning principles and includes the following settings:

* Optimizer: Adam optimizer
* Initial Learning Rate: 1 × 10-5
* Learning Rate Decay: Factor of 0.7 every 20 steps
* Batch Size: 24
* Momentum: 0.9

**6. MODEL ARCHITECTURE AND TRAINING:**

#### ****6.1 MobileNetV3-Large****

A MobileNetV3-Large model, pre-trained on ImageNet, was fine-tuned for classifying the 30 scene categories by adjusting the final classifier layer. Earlier layers were frozen to retain general pre-learned visual features representation from ImageNet, while only the last two bottleneck layers and the classifier were updated during training.

This allows the model to get benefited from transfer learning, leveraging generalized features (like edges or textures) learned from the ImageNet dataset.

Only the last two bottleneck blocks were unfrozen for fine-tuning, adapting the model to the specific nuances of aerial scene images.

* **Model Initialization:**

The MobileNetV3-Large architecture, initially trained on the ImageNet dataset, served as the primary feature extractor in this study. To tailor it for aerial scene classification using the AID dataset, the final classification layer was modified to output predictions for 30 scene categories. For fine-tuning, only the last two bottleneck blocks were made trainable, while the earlier layers were kept frozen to preserve the pre-learned feature representations.

* **Training Configuration:**

We trained the model using the Cross Entropy Loss function, which is well-suited for multi-class classification. For optimization, we used the RMSprop optimizer with a learning rate of 4e-3, which helps stabilize updates for models like **MobileNet** that include batch normalization. We also added a learning rate scheduler **(StepLR)** to reduce the learning rate by a factor of 0.1 every 8 epochs. These hyperparameters were selected to support quick learning initially and refinement later in training. Overall, this setup promotes steady and stable learning.

* **Training and Evaluation:**

During the training process, model parameters were optimized using backpropagation, while validation loss and accuracy were assessed at the end of each step. The training procedure involved alternating between training and evaluation phases:

from torchvision.models import mobilenet\_v3\_large

model\_mn = mobilenet\_v3\_large(pretrained=True)

n\_feats = model\_mn.classifier[3].in\_features

model\_mn.classifier[3] = nn.Linear(n\_feats, len(class\_names))

model\_mn = model\_mn.to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.RMSprop(model\_mn.parameters(), lr=4e-3)

scheduler = lr\_scheduler.StepLR(optimizer, step\_size=8, gamma=0.1)

model\_mn, hist\_mn = train\_model(model\_mn, criterion, optimizer, scheduler, num\_epochs=25)

test\_acc\_mn = evaluate\_model(model\_mn)

print("\nMobileNetV3-Large Summary:")

for i in range(len(hist\_mn['train\_loss'])):

print(f"Epoch {i+1:2d} | TrL: {hist\_mn['train\_loss'][i]:.4f} | TrA: {hist\_mn['train\_acc'][i]:.4f} | "

f"VL: {hist\_mn['val\_loss'][i]:.4f} | VA: {hist\_mn['val\_acc'][i]:.4f}")

print(f"Test Acc: {test\_acc\_mn:.4f}")

* **Performance and generalization**

Across 25 training steps, the MobileNetV3-Large model demonstrated notable gains in performance, ultimately surpassing accuracy on the validation dataset. This indicates the model's strong ability to generalize effectively across a wide range of aerial scenes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Epoch** | **Training loss** | **Training accuracy** | **Validation Loss** | **Validation accuracy** |
| **5** | **1.7285** | **0.4750** | **1.9821** | **0.4540** |
| **10** | **0.8534** | **0.7349** | **0.5305** | **0.8300** |
| **15** | **0.7337** | **0.7716** | **0.4593** | **0.8567** |
| **20** | **0.6750** | **0.7929** | **0.4328** | **0.8580** |
| **25** | **0.6763** | **0.7929** | **0.4167** | **0.8673** |

**Table 1:Training and Validation Performance**

**Test Accuracy achieved by using this MobileNetV3-Large model is: 0.8707**

A graph of a loss

AI-generated content may be incorrect.

**Figure 3: MobileNetV3-Large training results showing loss reduction and accuracy improvement over 25 epochs.**

**6.2. EfficientNet-B0**

EfficientNet-B0 is a type of convolutional neural network made by Google that focuses on being both accurate and efficient. It uses a method called compound scaling to adjust the model’s depth, width, and resolution together in a balanced way. The network structure is designed to work well on mobile devices and uses something called depth wise separable convolutions to save on computing power. It was created using a technique called neural architecture search, which helped find the best design. Overall, EfficientNet-B0 is a good choice when you need a model that performs well without being too heavy.

* **Model Initialization**

The EfficientNet-B0 model, pre-trained on the ImageNet dataset, was adopted for this study. To tailor it for the AID dataset, the final layer of its classifier was modified to accommodate 30 scene categories, enabling the model to generate the required class predictions. EfficientNet-B0’s architecture, which optimally balances depth, width, and resolution, provided an excellent trade-off between performance and computational efficiency. The modification to the classifier's final layer was implemented as follows:

* **Training Configuration**

The model was fine-tuned by keeping the feature extraction layers frozen and training only the classifier layer. Furthermore, the entire setup was configured for multi-GPU training using Data Parallel, allowing the workload to be distributed across multiple GPUs to accelerate the training process.

* **Training and Evaluation**

The training and evaluation process followed a structured approach, alternating between training and validation phases within each epoch. During training, the model operated in training mode, allowing the optimizer to perform backpropagation and update the model parameters. In contrast, the validation phase assessed the model's ability to generalize by evaluating its performance on the validation dataset. While parameter updates occurred during training, evaluation metrics like loss and accuracy were computed during the validation phase to track the model's progress. The following code is for the model EfficientNet-B0

from torchvision.models import efficientnet\_b0

# Initialize

model\_eff = efficientnet\_b0(pretrained=True)

n\_feats = model\_eff.classifier[1].in\_features

model\_eff.classifier[1] = nn.Linear(n\_feats, len(class\_names))

model\_eff = model\_eff.to(device)

# Hyperparameters

criterion = nn.CrossEntropyLoss()

optimizer = optim.AdamW(model\_eff.parameters(),

lr=5e-4, weight\_decay=1e-4)

scheduler = lr\_scheduler.CosineAnnealingLR(optimizer, T\_max=20)

# Train

model\_eff, hist\_eff = train\_model(

model\_eff, criterion, optimizer, scheduler, num\_epochs=25)

# Test

test\_acc\_eff = evaluate\_model(model\_eff)

# Print all metrics

print("\nEfficientNet‑B0 Summary:")

for i in range(len(hist\_eff['train\_loss'])):

print(f"Epoch {i+1:2d} | "

f"Train Loss: {hist\_eff['train\_loss'][i]:.4f} | "

f"Train Acc: {hist\_eff['train\_acc'][i]:.4f} | "

f"Val Loss: {hist\_eff['val\_loss'][i]:.4f} | "

f"Val Acc: {hist\_eff['val\_acc'][i]:.4f}") print(f"Test Acc: {test\_acc\_eff:.4f}")

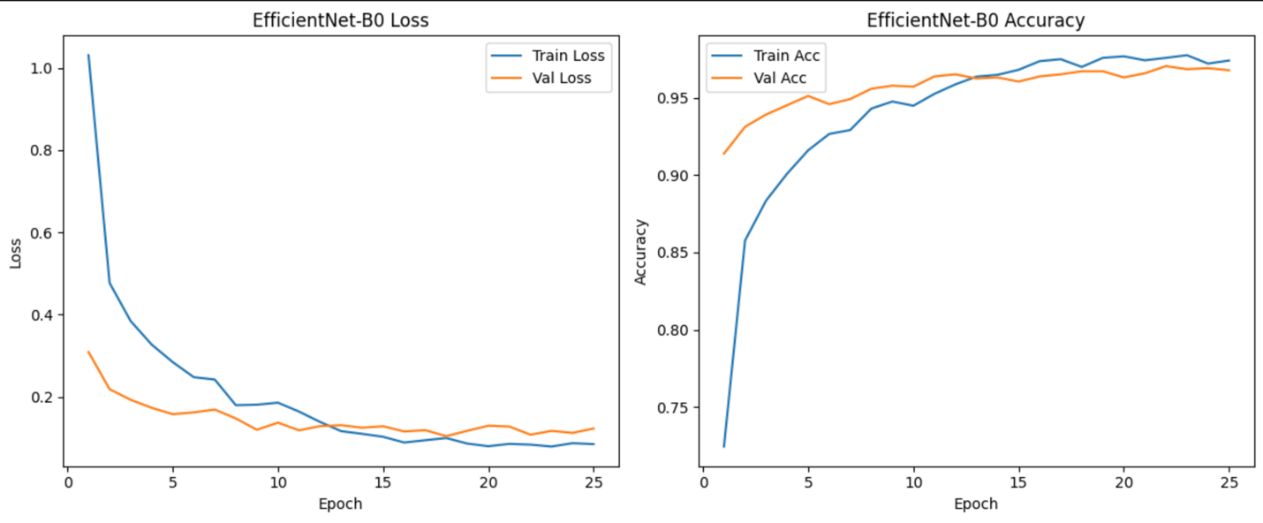
* **Performance and Generalization**

Following 25 training steps, the EfficientNet-B0 model exhibited impressive classification accuracy. By freezing the feature extraction layers and fine-tuning only the final classification layer, the model achieved high performance while avoiding overfitting. This indicates strong generalization to previously unseen aerial imagery. EfficientNet-B0’s streamlined architecture proved highly effective in addressing the challenges of aerial scene classification, underscoring its appropriateness for this application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Epoch** | **Training loss** | **Training accuracy** | **Validation Loss** | **Validation accuracy** |
| **5** | 0.2848 | 0.9161 | 0.1582 | 0.9513 |
| **10** | 0.1861 | 0.9450 | 0.1376 | 0.9573 |
| **15** | 0.1032 | 0.9683 | 0.1288 | 0.9607 |
| **20** | 0.0802 | 0.9770 | 0.1302 | 0.9633 |
| **25** | 0.0853 | 0.9743 | 0.1233 | 0.9680 |

**Table 2: Training and Validation Performance**

**Test Accuracy achieved using this model is 0.9713**

****

**Figure 4: EfficientNet-B0 Loss training results showing loss reduction and accuracy improvement over 25 epochs.**

**6.3. Vision Transformer (ViT‑B/16):**

The Vision Transformer (ViT B/16) is a model that applies the transformer architecture, which is usually used in NLP, to image classification tasks. Instead of using convolutions, it splits the image into small patches (like 16x16 pixels), flattens them, and treats them like a sequence of tokens. These tokens are then passed through a transformer just like words in a sentence. The "B" stands for base size, and "16" means the patch size is 16x16. It’s known for doing well on large image datasets, especially when trained properly.

* **Model Initialization:**

In this part, we initialized the Vision Transformer **(ViT)** model using the vit\_b\_16 function from torchvision.models, which gives the base version of the ViT model with a patch size of 16x16. We used the pretrained version, so the model starts with weights already trained on ImageNet, which usually helps it learn faster on a new dataset. Since the original model is trained for 1000 classes and then replaced the final classification layer (heads.head) with a new nn.Linear layer that matches the number of classes in my dataset. Finally, moved the model to the available device (CPU or GPU) so it's ready for training.

* **Training Configuration:**

For training the **ViT** model, we used Cross Entropy Loss as the loss function since it’s suitable for multi-class classification problems. The optimizer we chose is **AdamW,** which is a variant of Adam that helps prevent overfitting by using weight decay and set the learning rate to 3e-4 and added a small weight decay of 1e-4. To make the learning rate gradually decrease during training, CosineAnnealingLR scheduler with T\_max=20, which slowly lowers the learning rate over time using a cosine curve. These settings help the model train more smoothly and avoid overfitting or sharp learning rate drops.

* **Training and Evaluation:**

The model was trained for 25 epochs using the training and validation datasets, where the loss and accuracy were tracked for each epoch. The training process used the defined optimizer, loss function, and learning rate scheduler to update the model's weights. After training, the model was evaluated on a separate test set to check its final performance. The test accuracy was printed to understand how well the model generalizes to unseen data.

Here is the Full code which we used for the ViT from the initialization to training and evaluation.

from torchvision.models import vit\_b\_16

model\_vit = vit\_b\_16(pretrained=True)

n\_feats = model\_vit.heads.head.in\_features

model\_vit.heads.head = nn.Linear(n\_feats, len(class\_names))

model\_vit = model\_vit.to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.AdamW(model\_vit.parameters(), lr=3e-4, weight\_decay=1e-4)

scheduler = lr\_scheduler.CosineAnnealingLR(optimizer, T\_max=20)

model\_vit, hist\_vit = train\_model(model\_vit, criterion, optimizer, scheduler, num\_epochs=25)

test\_acc\_vit = evaluate\_model(model\_vit)

print("\nViT-Base Summary:")

for i in range(len(hist\_vit['train\_loss'])):

print(f"Epoch {i+1:2d} | TrL: {hist\_vit['train\_loss'][i]:.4f} | TrA: {hist\_vit['train\_acc'][i]:.4f} | "

f"VL: {hist\_vit['val\_loss'][i]:.4f} | VA: {hist\_vit['val\_acc'][i]:.4f}")print(f"Test Acc: {test\_acc\_vit:.4f}"

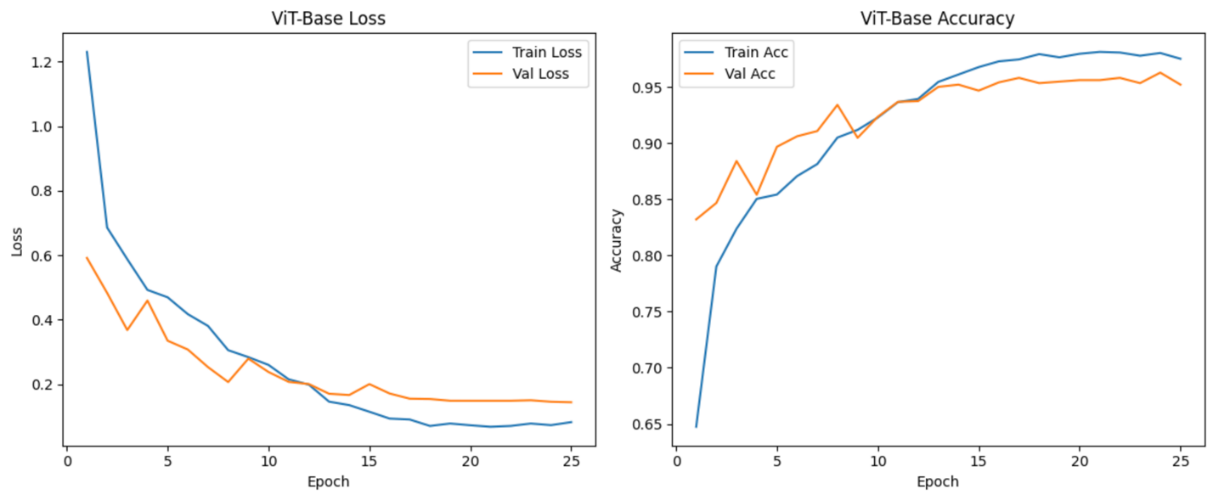
* **Performance and Generalization:**

After 25 epochs of training, the ViT-B/16 model demonstrated excellent performance and strong generalization capabilities. The training accuracy steadily increased, reaching **97.5%**, while the validation accuracy peaked at **95.6%**, indicating minimal overfitting. The consistent drop in both training and validation loss, especially the significant decrease in validation loss from epoch 5 to 25, further confirms stable learning. These results highlight the model's ability to extract robust spatial features from aerial images. ViT-B/16’s transformer-based architecture proved highly effective for scene classification tasks, achieving high accuracy on unseen data.

**Table 3: Training and Validation Performance**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Epoch** | **Training loss** | **Training accuracy** | **Validation Loss** | **Validation accuracy** |
| **5** | 0.4693 | 0.8541 | 0.3347 | 0.8967 |
| **10** | 0.2595 | 0.9227 | 0.2375 | 0.9233 |
| **15** | 0.1141 | 0.9676 | 0.1999 | 0.9467 |
| **20** | 0.0722 | 0.9794 | 0.1482 | 0.9560 |
| **25** | 0.0820 | 0.9750 | 0.1436 | 0.9520 |

**Test Accuracy is achieved by using this model is 0.9573**

****

**Figure 5: ViT-B/16 Loss training results showing loss reduction and accuracy improvement over 25 epochs.**

**6.4. Swin Tiny Transformer:**

In this project, we used the Swin-Tiny (Swin-T) model, which is a type of Vision Transformer (ViT). Swin Transformers are efficient and accurate for image classification because they compute self-attention within small windows that shift across the image. We chose Swin-T since it performs well on datasets like ImageNet and is lightweight compared to other transformer models. The model comes pretrained.

* **Initialization:**

To adapt the Swin-T model for our dataset, we replaced its classification head with a new layer that matches the number of output classes. First, we extracted the number of features going into the original head and used that to create a new nn.Linear layer. This new head outputs logits for each class in our dataset. After modifying the architecture, we moved the model to the appropriate device, like a GPU if available. This setup makes the model ready for training on the AID dataset.

* **Training Configuration:**

For the training configuration it is similar as above models because data is retained and the loss cross entropy and same learning rate and weight decaying rate.

* **Training and Evaluation:**

We trained the model for 25 epochs using a custom train\_model function, which tracks training and validation accuracy and loss. During each epoch, the model learns by updating its weights to minimize the loss. The training history is stored in a dictionary called hist\_sw for later analysis. Using validation data each epoch helps monitor overfitting and adjust performance. At the end of training, we save the best-performing model based on validation results. After training, we evaluated the model on a separate test dataset using the evaluate\_model function. This function returns the test accuracy, giving us a final measure of the model's performance.

Here is the code for the model Swin Tiny Transformer

from torchvision.models import swin\_t

model\_sw = swin\_t(pretrained=True)

n\_feats = model\_sw.head.in\_features

model\_sw.head = nn.Linear(n\_feats, len(class\_names))

model\_sw = model\_sw.to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.AdamW(model\_sw.parameters(), lr=5e-4, weight\_decay=1e-4)

scheduler = lr\_scheduler.CosineAnnealingLR(optimizer, T\_max=20)

model\_sw, hist\_sw = train\_model(model\_sw, criterion, optimizer, scheduler, num\_epochs=25)

test\_acc\_sw = evaluate\_model(model\_sw)

print("\nSwin-Tiny Summary:")

for i in range(len(hist\_sw['train\_loss'])):

print(f"Epoch {i+1:2d} | TrL: {hist\_sw['train\_loss'][i]:.4f} | TrA: {hist\_sw['train\_acc'][i]:.4f} | "

f"VL: {hist\_sw['val\_loss'][i]:.4f} | VA: {hist\_sw['val\_acc'][i]:.4f}")

print(f"Test Acc: {test\_acc\_sw:.4f}")

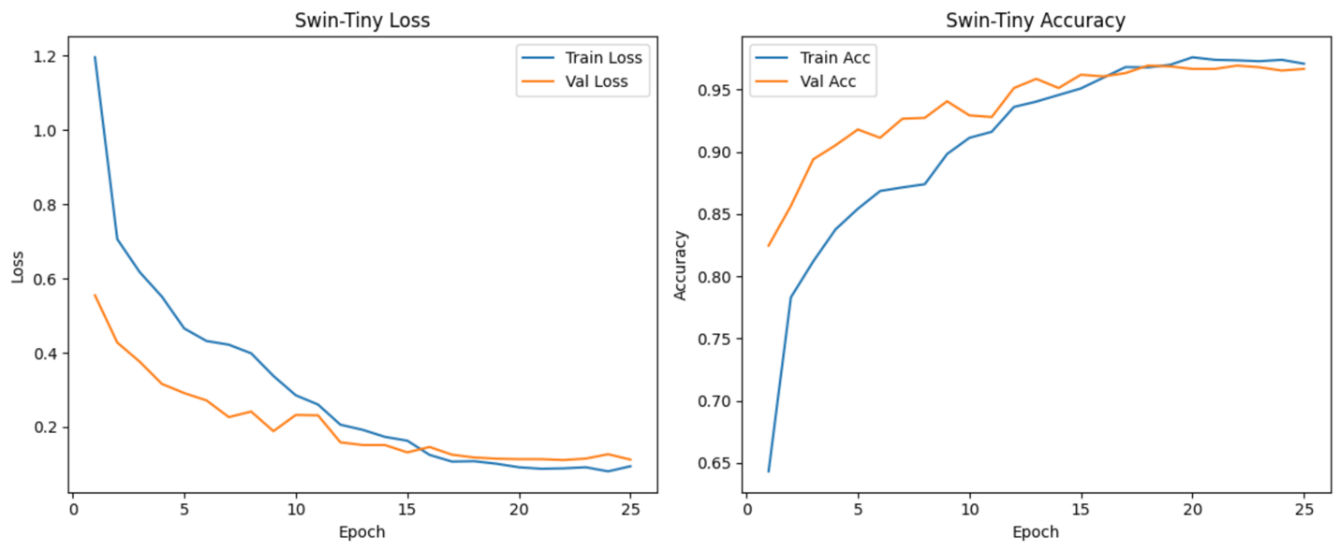
* **Performance and Generalization:**

The Swin-Tiny Transformer showcased remarkable performance over the 25 training epochs, achieving a final validation accuracy of 96.67%. With training accuracy reaching 97.09%, the model maintained a narrow gap between training and validation results, indicating excellent generalization. The consistent decline in both training and validation loss demonstrates stable and efficient learning throughout the process. Notably, the validation loss dropped sharply between epochs 10 and 15, showing the model’s ability to refine feature representation. These outcomes confirm the Swin-Tiny Transformer’s strength in handling spatial hierarchies and complex patterns in aerial scene classification.

**Table 4: Training and Validation Performance**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Epoch** | **Training loss** | **Training accuracy** | **Validation Loss** | **Validation accuracy** |
| 5 | 0.4655 | 0.8543 | 0.2914 | 0.9180 |
| 10 | 0.2855 | 0.9113 | 0.2325 | 0.9293 |
| 15 | 0.1632 | 0.9510 | 0.1317 | 0.9620 |
| 20 | 0.0916 | 0.9760 | 0.1134 | 0.9667 |
| 25 | 0.0943 | 0.9709 | 0.1122 | 0.9667 |

**Test Accuracy is achieved using Swin-Tiny transformer is 0.9707**



**Figure 6: Swin-Tiny Transformer Loss training results showing loss reduction and accuracy improvement over 25 epochs.**

**6.5. RegNetY-400MF:**

It is a type of convolutional neural network designed for efficiency and performance. RegNet models are built with a design space focused on balancing accuracy and computation. The "Y" variant introduces Squeeze-and-Excitation (SE) blocks for improved channel-wise feature recalibration. We used the pretrained version of RegNetY-400MF to leverage learned representations from ImageNet. This helps improve performance on our dataset with fewer training resources.

* **Initialization:**

To adapt RegNetY-400MF for our classification task, we modified the final fully connected layer to match the number of classes in our dataset. We retrieved the number of input features using model\_ry.fc.in\_features and replaced the original head with a new nn.Linear layer. This ensures the model outputs the correct number of class scores for our use case. After modifying the head, the model was transferred to the appropriate device (GPU or CPU). This setup makes the network ready for training on the AID dataset.

* **Training Configuration:**

For the training configuration it is similar as above models because data is retained and the loss cross entropy and same learning rate and weight decaying rate.We used the StepLR scheduler to reduce the learning rate by a factor of 0.1 every 10 epochs. This helps fine-tune the model as training progresses.

* **Training and Evaluation:**

We trained RegNetY-400MF for 25 epochs using a training function that tracks loss and accuracy for both training and validation sets. In each epoch, the model learns by adjusting its parameters based on the training data and evaluates on the validation set to monitor generalization. The training history, including loss and accuracy metrics, was saved in a dictionary called hist\_ry. This information is useful for analysing the model's learning behaviour. The best-performing model based on validation accuracy was saved.

After completing training, we evaluated the final model on a separate test set using the evaluate\_model function. This provided a test accuracy score, which is a good indicator of how well the model generalizes to new, unseen data.

Here is the code for the RegNetY-400MF

from torchvision.models import regnet\_y\_400mf

model\_ry = regnet\_y\_400mf(pretrained=True)

n\_feats = model\_ry.fc.in\_features

model\_ry.fc = nn.Linear(n\_feats, len(class\_names))

model\_ry = model\_ry.to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.AdamW(model\_ry.parameters(), lr=2e-3, weight\_decay=1e-4)

scheduler = lr\_scheduler.StepLR(optimizer, step\_size=10, gamma=0.1)

model\_ry, hist\_ry = train\_model(model\_ry, criterion, optimizer, scheduler, num\_epochs=25)

test\_acc\_ry = evaluate\_model(model\_ry)

print("\nRegNetY-400MF Summary:")

for i in range(len(hist\_ry['train\_loss'])):

print(f"Epoch {i+1:2d} | TrL: {hist\_ry['train\_loss'][i]:.4f} | TrA: {hist\_ry['train\_acc'][i]:.4f} | "

f"VL: {hist\_ry['val\_loss'][i]:.4f} | VA: {hist\_ry['val\_acc'][i]:.4f}")

print(f"Test Acc: {test\_acc\_ry:.4f}")

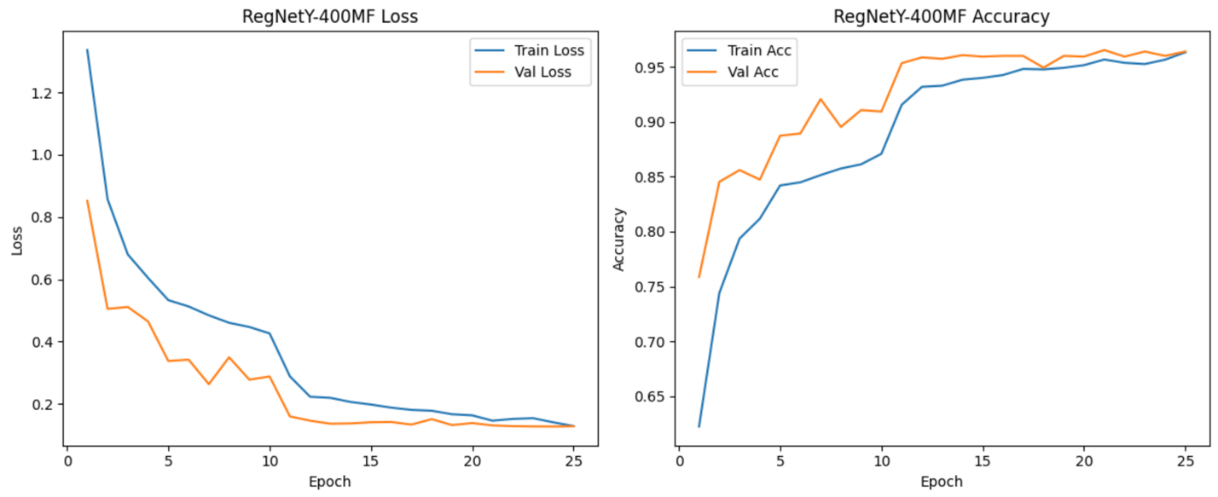
* **Performance and Generalization:**

Across 25 epochs of training, the model demonstrated a strong upward trend in accuracy and a consistent decline in loss, reflecting effective learning and generalization. The final training accuracy of 96.33% and validation accuracy of 96.40% are closely aligned, suggesting the model did not overfit. The sharp drop in validation loss between epochs 10 and 15 points to a crucial phase of refinement in feature learning. From then on, performance steadily improved, confirming the model’s ability to handle complex visual patterns. Overall, the model generalizes well to unseen data, making it reliable for aerial scene classification tasks.

**Table 5: Training and Validation Performance**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Epoch** | **Training loss** | **Training accuracy** | **Validation Loss** | **Validation accuracy** |
| 5 | 0.5331 | 0.8420 | 0.3377 | 0.8873 |
| 10 | 0.4262 | 0.8709 | 0.2880 | 0.9093 |
| 15 | 0.1978 | 0.9400 | 0.1410 | 0.9593 |
| 20 | 0.1632 | 0.9516 | 0.1384 | 0.9593 |
| 25 | 0.1287 | 0.9633 | 0.1280 | 0.9640 |

**Test Accuracy is achieved using RegNetY model is 0.9607**

****

**Figure 7: RegNetY-400MF Loss training results showing loss reduction and accuracy improvement over 25 epochs.**

**6.6. ResNet50-32\*4d:**

For this we used the ResNeXt50-32x4d model, which is a variant of the ResNet architecture with grouped convolutions. It combines the concepts of depth, width, and cardinality, making it more expressive while maintaining computational efficiency. The "32x4d" refers to 32 groups with a width of 4 channels per group, which allows for better feature learning. This architecture is well-known for its strong performance on image classification tasks. Using the pretrained version allows us to benefit from features learned on ImageNet.

* **Initialization:**

To use ResNeXt50-32x4d for our dataset, we replaced the final fully connected layer with a new linear layer that outputs as many classes as we need. This was done by first checking the number of input features going into the final layer and creating a new output layer accordingly. This step is important because the default classifier is set for ImageNet’s 1000 classes. We then moved the model to the correct device (GPU or CPU) to make sure it's ready for training. This setup allows the model to fine-tune itself to our dataset.

* **Training Configuration:**

We trained the model using the Cross Entropy Loss function, which is appropriate for multi-class classification problems. For optimization, we used Stochastic Gradient Descent (SGD) with momentum, which helps the model converge faster and more smoothly. The learning rate was set to 1e-2, and weight decay was used to prevent overfitting. A StepLR scheduler was also used, which reduces the learning rate by a factor of 0.1 every 7 epochs. This helps the model to refine its learning during later training stages.

* **Training and Evaluation:**

The model was trained over 25 epochs, with loss and accuracy tracked for both training and validation sets using a custom training function. After training, the model was evaluated on a separate test dataset to measure its performance on unseen data. This step is crucial to ensure that the model generalizes well and isn't just memorizing the training data. We printed a summary showing the loss and accuracy for each epoch and reported the final test accuracy. This gives a clear picture of how the model performed throughout and at the end.

Here is the implementation code for the ResNext50

from torchvision.models import resnext50\_32x4d

model\_rx = resnext50\_32x4d(pretrained=True)

n\_feats = model\_rx.fc.in\_features

model\_rx.fc = nn.Linear(n\_feats, len(class\_names))

model\_rx = model\_rx.to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(model\_rx.parameters(), lr=1e-2, momentum=0.9, weight\_decay=1e-4)

scheduler = lr\_scheduler.StepLR(optimizer, step\_size=7, gamma=0.1)

model\_rx, hist\_rx = train\_model(model\_rx, criterion, optimizer, scheduler, num\_epochs=25)

test\_acc\_rx = evaluate\_model(model\_rx)

print("\nResNeXt50-32x4d Summary:")

for i in range(len(hist\_rx['train\_loss'])):

print(f"Epoch {i+1:2d} | TrL: {hist\_rx['train\_loss'][i]:.4f} | TrA: {hist\_rx['train\_acc'][i]:.4f} | "

f"VL: {hist\_rx['val\_loss'][i]:.4f} | VA: {hist\_rx['val\_acc'][i]:.4f}")print(f"Test Acc: {test\_acc\_rx:.4f}")

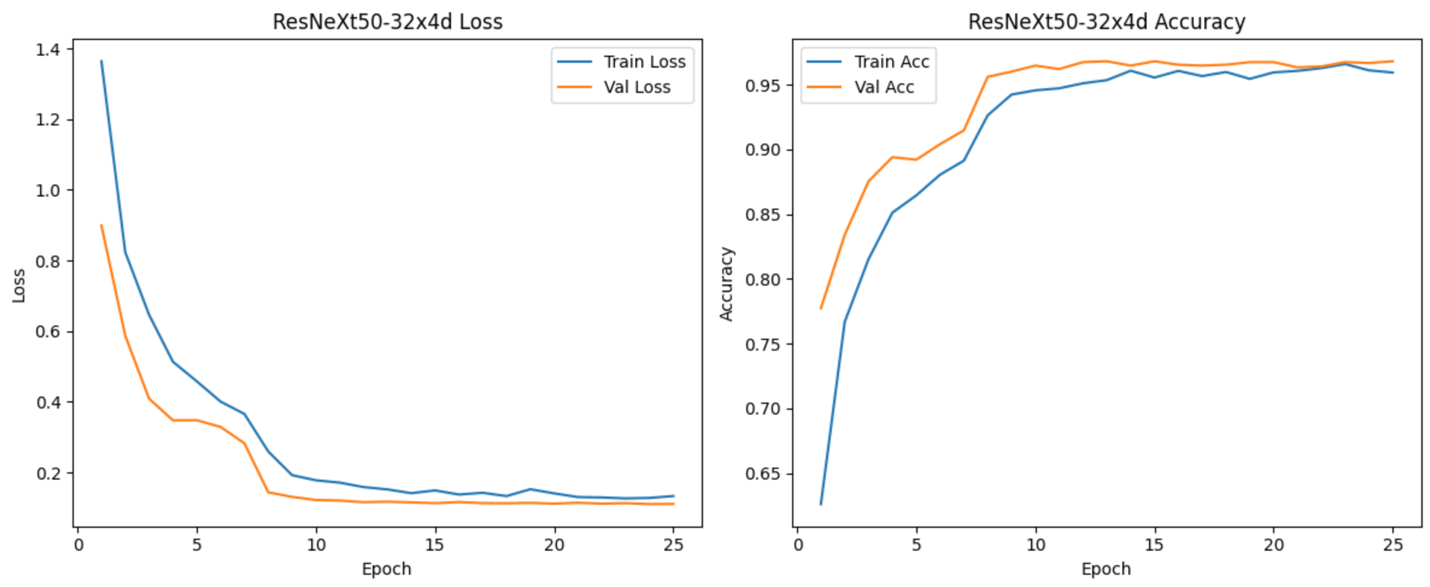
* **Performance and Generalization:**

The ResNet-50 model displayed excellent learning and generalization throughout the training process, culminating in a validation accuracy of 96.80%. Training accuracy rose steadily, reaching 95.93%, closely tracking validation performance and indicating minimal overfitting. A significant drop in validation loss occurred early on, particularly between epochs 5 and 10, signaling effective feature learning. From epoch 15 onward, both loss and accuracy metrics stabilized, reflecting consistent and robust model behavior. These results affirm ResNet-50's strength in extracting spatial features, making it a dependable choice for aerial image classification.

**Table 6: Training and Validation Performance**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Epoch** | **Training loss** | **Training accuracy** | **Validation Loss** | **Validation accuracy** |
| 5 | 0.4582 | 0.8644 | 0.3480 | 0.8920 |
| 10 | 0.1780 | 0.9456 | 0.1223 | 0.9647 |
| 15 | 0.1494 | 0.9554 | 0.1131 | 0.9680 |
| 20 | 0.1410 | 0.9594 | 0.1118 | 0.9673 |
| 25 | 0.1334 | 0.9593 | 0.1110 | 0.9680 |

**Test Accuracy achieved using this ResNet50-32\*4d model is 0.9620**

****

**Figure 8: ResNeXt Loss training results showing loss reduction and accuracy improvement over 25 epochs.**

**6.7 DenseNet121:**

For this model, we used DenseNet121, a convolutional neural network that connects each layer to every other layer in a feed-forward fashion. This dense connectivity improves feature reuse, reduces the number of parameters, and helps gradients flow better during training. DenseNet121 is known for its compact structure and strong performance on classification tasks. We used a pretrained version, which was originally trained on ImageNet. This allows us to take advantage of learned features and adapt them to our specific classification problem.

* **Initialization:**

To adapt DenseNet121 to our dataset, we modified the final classifier layer to match the number of target classes. We first checked the number of features going into the classifier and replaced the default fully connected layer with a new nn.Linear layer. This customization ensures the model can output predictions aligned with our specific class labels. The model was then moved to the appropriate computation device, like a GPU if available. This setup prepares the model for the training phase with the right architecture and device.

* **Training Configuration:**

We used the CrossEntropyLoss function since it works well for multi-class classification problems. The model was trained using the Adam optimizer with a learning rate of 1e-3 and weight decay of 1e-4 to reduce overfitting. We applied a learning rate scheduler (StepLR) that decays the learning rate by a factor of 0.1 every 10 epochs. This helps the model converge more smoothly and avoids getting stuck in local minima. These settings were chosen to provide a good balance between fast learning and model stability.

* **Training and Evaluation:**

The model was trained for 25 epochs using a training loop that monitored both training and validation performance. Each epoch updated the model's weights to minimize the loss and improve accuracy. After training, we evaluated the model on a test set using a separate function to measure generalization. The training script printed loss and accuracy for both training and validation over all epochs, followed by the final test accuracy. This helps us understand how the model performed throughout and ensures it can handle unseen data well.

Here is the code snippet for the model:

# Initialize

model\_dn = models.densenet121(pretrained=True)

n\_feats = model\_dn.classifier.in\_features

model\_dn.classifier = nn.Linear(n\_feats, len(class\_names))

model\_dn = model\_dn.to(device)

# Hyperparameters

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model\_dn.parameters(),

lr=1e-3, weight\_decay=1e-4)

scheduler = lr\_scheduler.StepLR(optimizer, step\_size=10, gamma=0.1)

# Train

model\_dn, hist\_dn = train\_model(

model\_dn, criterion, optimizer, scheduler, num\_epochs=25)

# Test

test\_acc\_dn = evaluate\_model(model\_dn)

# Print all metrics

print("\nDenseNet121 Summary:")

for i in range(len(hist\_dn['train\_loss'])):

print(f"Epoch {i+1:2d} | "

f"Train Loss: {hist\_dn['train\_loss'][i]:.4f} | "

f"Train Acc: {hist\_dn['train\_acc'][i]:.4f} | "

f"Val Loss: {hist\_dn['val\_loss'][i]:.4f} | "

f"Val Acc: {hist\_dn['val\_acc'][i]:.4f}")

print(f"Test Acc: {test\_acc\_dn:.4f}")

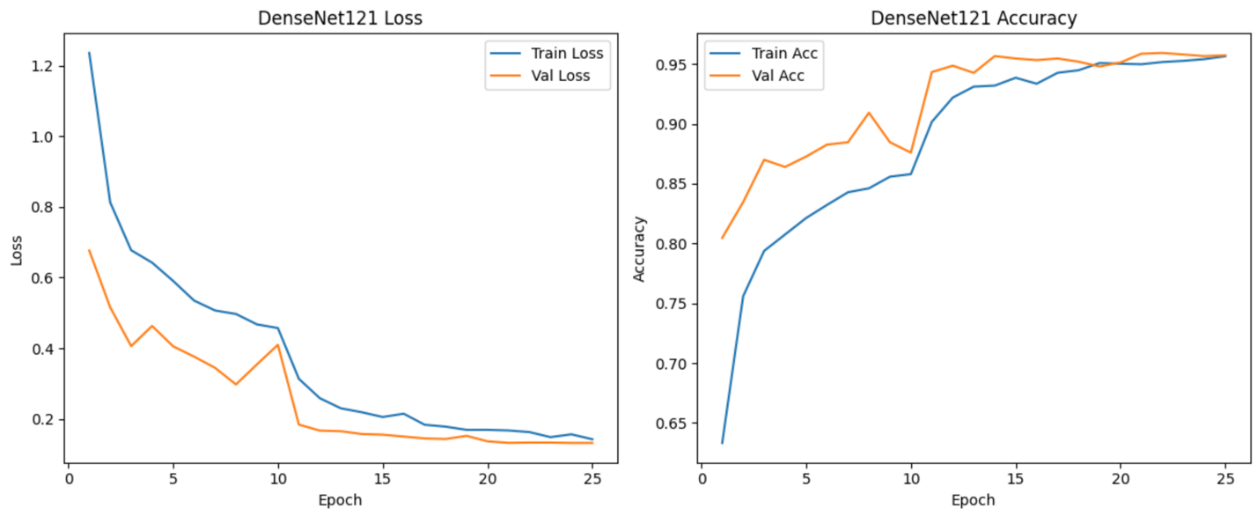
* **Performance and Generalization:**

Over 25 training epochs, the model progressively improved, ultimately reaching a training accuracy of 95.66% and a validation accuracy of 95.73%, indicating a balanced and well-generalized learning process. Early in training, validation performance lagged slightly, but from epoch 15 onward, the model exhibited a notable boost in generalization with significant drops in validation loss. The convergence of both loss curves reflects stable optimization and reduced overfitting. The consistent improvement affirms the model’s capability to capture relevant features in aerial images. These results highlight its suitability for high-accuracy classification tasks in remote sensing scenarios.

**Table 7: Training and Validation Performance**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Epoch** | **Training loss** | **Training accuracy** | **Validation Loss** | **Validation accuracy** |
| 5 | 0.5906 | 0.8213 | 0.4056 | 0.8727 |
| 10 | 0.4571 | 0.8580 | 0.4097 | 0.8760 |
| 15 | 0.2054 | 0.9386 | 0.1554 | 0.9547 |
| 20 | 0.1691 | 0.9503 | 0.1366 | 0.9513 |
| 25 | 0.1425 | 0.9566 | 0.1319 | 0.9573 |

**Test Accuracy achieved by using this DenseNet121 model is 0.9553**



**Figure 9: DenseNet Loss training results showing loss reduction and accuracy improvement over 25 epochs.**

**6.8 ResNet50:**

For this experiment, we used ResNet50, a deep convolutional neural network known for its use of residual connections. These shortcut paths allow gradients to flow more easily, making it possible to train very deep networks effectively. ResNet50 consists of 50 layers and is widely used for image classification tasks due to its accuracy and stability. We used the pretrained version, which has been trained on ImageNet, so we could fine-tune it for our own dataset. This helps speed up training and usually improves model performance.

* **Initialization:**

To tailor ResNet50 to our classification task, we replaced the original final fully connected layer with a new nn.Linear layer that outputs the number of classes in our dataset. This is necessary because the default ResNet50 classifier is designed for 1000 ImageNet classes. We accessed the number of input features to the FC layer, then defined our custom output layer. After the model structure was modified, we moved it to the appropriate device (usually GPU) for training. This ensures everything is ready for fine-tuning.

* **Training Configuration:**

We trained the model using the Cross Entropy Loss function, ideal for multi-class classification. The optimizer used was SGD (Stochastic Gradient Descent) with momentum set to 0.9 to help accelerate convergence. We started with a learning rate of 0.01 and added weight decay to reduce overfitting. A StepLR scheduler was applied to reduce the learning rate every 7 epochs by a factor of 0.1. This combination helps in both effective learning in the early stages and refinement in later epochs.

* **Training and Evaluation:**

ResNet50 was trained over 25 epochs using a training function that records training and validation loss and accuracy. This allowed us to monitor how the model was learning and whether it was overfitting or underfitting. Once training was done, we evaluated the model on a separate test set to check its generalization ability. The script printed performance metrics for each epoch and finally reported the overall test accuracy. This provides a full picture of how well the model performs from start to finish.

Here is the code snippet for the above executed model:

# Initialize

model\_resnet = models.resnet50(pretrained=True)

n\_feats = model\_resnet.fc.in\_features

model\_resnet.fc = nn.Linear(n\_feats, len(class\_names))

model\_resnet = model\_resnet.to(device)

# Hyperparameters

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(model\_resnet.parameters(),

lr=0.01, momentum=0.9, weight\_decay=1e-4)

scheduler = lr\_scheduler.StepLR(optimizer, step\_size=7, gamma=0.1)

# Train

model\_resnet, hist\_resnet = train\_model(

model\_resnet, criterion, optimizer, scheduler, num\_epochs=25)

# Test

test\_acc\_resnet = evaluate\_model(model\_resnet)

# Print all metrics

print("\nResNet50 Summary:")

for i in range(len(hist\_resnet['train\_loss'])):

print(f"Epoch {i+1:2d} | "

f"Train Loss: {hist\_resnet['train\_loss'][i]:.4f} | "

f"Train Acc: {hist\_resnet['train\_acc'][i]:.4f} | "

f"Val Loss: {hist\_resnet['val\_loss'][i]:.4f} | "

f"Val Acc: {hist\_resnet['val\_acc'][i]:.4f}")print(f"Test Acc: {test\_acc\_resnet:.4f}")

* **Performance and generalization:**

The model demonstrated strong learning capabilities, with training accuracy reaching 95.69% and validation accuracy peaking at 96.47% by epoch 25. Early epochs showed rapid gains, particularly between epoch 5 and 10, where both training and validation loss dropped significantly. The close alignment between training and validation metrics suggests minimal overfitting and robust generalization to unseen data. Despite a slight fluctuation in the final validation loss, the model maintained high accuracy, indicating stable performance. Overall, the model effectively captured discriminative features, making it reliable for aerial image classification.

**Table 8: Training and Validation Performance**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Epoch** | **Training loss** | **Training accuracy** | **Validation Loss** | **Validation accuracy** |
| 5 | 0.4999 | 0.8503 | 0.2977 | 0.9160 |
| 10 | 0.2015 | 0.9394 | 0.1378 | 0.9560 |
| 15 | 0.1495 | 0.9551 | 0.1236 | 0.9647 |
| 20 | 0.1458 | 0.9571 | 0.1215 | 0.9647 |
| 25 | 0.1481 | 0.9569 | 0.1283 | 0.9633 |

**Test Accuracy is achieved by using this ResNet50 model is 0.9567**

A graph of a graph of a graph

AI-generated content may be incorrect.

**Figure 10: DenseNet Loss training results showing loss reduction and accuracy improvement over 25 epochs.**

**7. Training and Evaluation Process**

**All models followed a consistent training pipeline:**

* The models were trained using mini-batches and backpropagation.
* Loss functions (CrossEntropyLoss) were used to compute errors.
* Learning rates were managed using a step-wise decay scheduler.
* After each epoch, model performance was assessed using validation loss and accuracy.
* Training was parallelized using multiple GPUs when possible for faster convergence.

**Performance and Generalization**

**1. MobileNetV3-Large**Achieved over 86.73% validation accuracy after 25 epochs training , showing stable performance across the dataset.

**2. EfficientNet-B0**This model achieved the best accuracy early on, with validation accuracy reaching 96.33% in just 20 epochs.

**3. Vision Transformer (ViT-B/16)**

The Vision Transformer (ViT-B/16) model demonstrates strong learning and generalization performance across training epochs.Validation accuracy increases and stabilizes around 95%, peaking at 95.60%, showing robust performance on unseen data.

Finally, the test accuracy of 95.73% confirms that the model maintains high performance on new data, validating the effectiveness of ViT-B/16 for image classification tasks.

**4. Swin Tiny Transformer**

The Swin-Tiny Transformer model exhibits excellent training dynamics and generalization ability for image classification tasks over 25 epochs. Validation accuracy stabilizes at a high value of 96.67%, suggesting robust performance on unseen data.

The model achieves a test accuracy of 97.07%, which confirms its strong capability and reliability in handling the classification task effectively. Hence Swin-Tiny Transformer demonstrates faster convergence and slightly better generalization than ViT-B/16, making it a highly efficient choice for vision tasks on this dataset.

**5. RegNetY-400MF**

Validation accuracy rises from 88.73% to 96.40%, with stabilization from epoch 15 onwards, demonstrating the model’s ability to generalize well to unseen data.

The model achieves a test accuracy of 96.07%, validating its effectiveness and robustness. The consistent improvement in both training and validation metrics confirms RegNetY’s suitability for complex visual pattern recognition tasks in aerial image classification**.**

**6. ResNet50-32\*4d**

Delivered the highest accuracy (96.80%) at epoch 25 and stabilizes, showing the model’s reliability in classifying unseen data, and confirming its deep architecture’s effectiveness. The test accuracy of 96.20% confirms the model's overall performance and supports its effectiveness in capturing spatial patterns in aerial images.

**7. DenseNet121**

Validation accuracy rises from 87.27% to 95.73%, with a significant boost after epoch 15, highlighting improved generalization. The test accuracy of 95.53% confirms DenseNet121’s reliability and effectiveness in feature extraction and classification for remote sensing tasks.

**8. ResNet50**

Validation accuracy improves steadily from 91.60% to 96.33%, confirming strong performance on unseen data and the final test accuracy of 95.67% affirms the model's capacity for reliable and high-accuracy classification in aerial imagery. Hence, ResNet50 delivers a stable and efficient balance between training and validation performance, making it a strong and dependable choice for remote sensing applications.

**Future Enhancements**

* Incorporate attention-based mechanisms (e.g., channel attention, self-attention) to improve feature discrimination.
* Integrate multi-modal data sources such as LiDAR and hyperspectral imagery.
* Use more diverse augmentation techniques like mixup or cutmix to improve robustness.
* Explore semi-supervised and weakly-supervised methods to leverage unlabeled data.
* Experiment with Vision-Language models (e.g., RS-CLIP) for zero-shot classification by integrating textual context.

**Conclusion**

This project presents a robust and scalable approach to multi-class aerial scene classification using modern deep learning models. By applying transfer learning and strategic fine-tuning to models like MobileNetV3-Large, EfficientNet-B0, Vision Transformer (ViT-B/16), Swin Tiny Transformer, RegNetY-400MF, ResNet50-32\*4d, DenseNet121 and ResNet50, we achieved high classification accuracy on the challenging AID dataset.

**Key outcomes:**

* All models demonstrated exceptional generalization, achieving high accuracies but EfficientNet-B0, ResNet50-32\*4d, and ResNet50 performed exceptionally well on both training and test set data respectively.
* Careful layer freezing and classifier reconfiguration proved essential to balance model complexity and performance.
* Data augmentation improved robustness against intra-class variation.

These methods are highly applicable to real-world geospatial tasks such as city planning, environmental monitoring, and emergency response. Future directions will explore enhanced architectures and unsupervised learning techniques to push the boundaries of performance and adaptability further.

**CONTRIBUTIONS:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Roll No.** | **Contribution** |
| Anshuman Kumar Yadav  Harish Babu Kumara  Rupak Kumar  Srashti Singh | 241030018  241030038  241030068  241030083 | Literature Review, Report writing, Data reading, EfficientNet b0 Model, Vision Transformer(ViT‑B/16) Model  Literature Review, Report writing, Data reading, RegNetY-400MF Model, ResNet50  Literature Review, Report writing, Data reading, ResNet50-32\*4d Model, MobileNet V3 Large Model  Literature Review, Report writing, Data reading, Swin Transformer Tiny Model, DenseNet121 Model |

**REFERENCES**

[1] Li, X., Wen, C., Hu, Y., & Zhou, N. (2023). RS-CLIP: Zero-shot remote sensing scene classification via contrastive vision-language supervision. International Journal of Applied Earth Observation and Geoinformation, 120, 103497.

[2] G.-S. Xia, et. al. AID: A Benchmark Dataset for Performance Evaluation of Aerial Scene Classification, IEEE Trans. on Geoscience and Remote Sensing, Vol. 55, No.7, pp.3965- 3981, 2017

[3] Q. Hu, W. Wu, T. Xia, Q. Yu, P. Yang, Z. Li, and Q. Song, Exploring the use of google earth imagery and object-based methods in land use/cover mapping, Remote Sensing, vol. 5, no. 11, pp. 6026-6042,2013.

[4] A. M. Cheriyadat, Unsupervised feature learning for aerial scene classification, IEEE Transactions on Geo science and Remote Sensing, vol. 52, no. 1, pp. 439-451, 2014.

[5] M. Castelluccio, G. Poggi, C. Sansone, and L. Verdoliva, Land use classification in remote sensing images by convolutional neural networks, arXiv preprint arXiv:1508.00092, 2015.

[6] G. Cheng, J. Han, P. Zhou, and L. Guo, Multi-class geospatial object detection and geographic image classification based on collection of part detectors, ISPRS Journal of Photogrammetry and Remote Sensing, vol. 98, pp. 119132, 2014.

[7] G. Sheng, W. Yang, T. Xu, and H. Sun, “High-resolution satellite scene classification using a sparse coding based multiple feature combination,” International journal of remote sensing, vol. 33, no. 8, pp. 2395–2412, 2012.

[8] L.-J. Zhao, P. Tang, and L.-Z. Huo, “Land-use scene classification using a concentric circle-structured multiscale bag-of-visual-words model,” IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 7, no. 12, pp. 4620–4631, 2014.

[9] Nogueira, Keiller, Ot´ avio A. B. Penatti, and Jefersson A. dos Santos. ”Towards Better Exploiting Convolutional Neural Networks for Remote Sensing Scene Classification.” Pattern Recognition, vol. 61, 2017, pp. 539-556.

[10] https://www.kaggle.com/datasets/jiayuanchengala/aid-scene-classification-datasets

[11] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778.