Deep Learning - 04/2024

Deep Learning for Computer Vision

Universidad Politécnica de Madrid

Federico Paschetta, Cecilia Peccolo

federico.paschetta@alumnos.upm.es
cecilia.peccolo@alumnos.upm.es

Contents

Chapter 1

Introduction

In an era characterized by the proliferation of geospatial data, the demand for advanced techniques in satellite imagery analysis has become increasingly prominent. Among the myriad applications of such analysis, object classification within satellite imagery stands out as a critical task with far-reaching implications across various domains.
This report presents an in-depth exploration of object classification techniques applied to satellite imagery, focusing on the identification and categorization of transportation infrastructure and buildings.

The work is carried out for the final assignment of the *"Deep Learning" course*, which focused on developing and evaluating deep neural network models for image recognition on the xView dataset.

1.1 Goal

The main goals of this assignment were to explore different deep learning architectures and techniques for the image recognition task, using TensorFlow and Keras as the primary deep learning libraries. Specifically, we investigated three main approaches:

- **Feed-forward Neural Networks (FFNNs):** We designed and optimized FFNNs from scratch, tuning the number of layers, units per layer, optimization algorithms, and regularization techniques to achieve high performance on the image recognition task.
- **Convolutional Neural Networks (CNNs) from Scratch:** Given the success of CNNs for computer vision tasks, we implemented and trained CNN architectures from scratch, exploring different configurations of convolutional, pooling, and fully-connected layers.
- **Transfer Learning with Pre-trained CNNs:** To leverage the knowledge learned from large-scale datasets like ImageNet, we also experimented with popular CNN architectures like GoogLeNet, VGG, and ResNet, using transfer learning by fine-tuning the pre-trained models on the xView dataset.

A significant challenge in this task was the class imbalance present in the dataset, with some categories like buildings and small cars being much more prevalent than others like helicopters. This imbalance can lead to biased models that perform well on frequent classes but poorly on rare

ones. As a result, we explored various techniques to mitigate this issue, such as class-weighted loss functions and data augmentation strategies.
In the following sections, we describe the approach taken for each of the three main techniques mentioned above, including the neural network architectures, optimization processes, and results achieved. We also discuss the key insights gained, challenges faced, and potential areas for further improvement in developing robust image recognition models for satellite/aerial imagery using deep learning.

The report is structured as follows: Chapter 2 will cover the problem our project wants to solve, while Chapter 3 will focus on the dataset. Chapter 4 explains architectures code structure and implementation, Chapter 5 covers the feed-forward neural network approach, Chapter 6 details the convolutional neural networks trained from scratch, and Chapter 7 discusses the transfer learning experiments with pre-trained CNNs, followed by conclusions and future work in Chapter 8.

Chapter 2

Problem

In the rapidly evolving landscape of modern technology, data-driven decision-making has become paramount across various industries. With the exponential growth of data generation, organizations are increasingly reliant on advanced analytical techniques to extract valuable insights and drive strategic initiatives.
One such critical task is that of classification, wherein data points are categorized into distinct groups or classes based on their attributes or features. Classification tasks find widespread application in numerous domains, including but not limited to image recognition, natural language processing, financial fraud detection, and medical diagnosis.
However, the efficacy of classification models is contingent upon several factors, chief among them being the complexity and diversity of the dataset. In real-world scenarios, datasets often exhibit intricate patterns and variations, presenting formidable challenges to traditional classification approaches.

The problem we aim to address in this report revolves around the classification of data points into multiple categories. Specifically, we are tasked with assigning probabilities to each data point, indicating its likelihood of belonging to each of the predefined categories. This problem, known as multi-class classification, is inherently challenging due to the need to discern subtle differences and nuances among the various classes.
Moreover, the dataset at hand comprises complex and diverse data, posing additional hurdles to traditional classification methodologies. In particular, the data consists of high-dimensional images with intricate features, making it particularly challenging to extract meaningful patterns and information using conventional techniques. Furthermore, advancements in machine learning and artificial intelligence have opened up new avenues for tackling complex classification problems. The emergence of deep learning techniques, particularly neural networks, has shown remarkable promise in handling intricate datasets and extracting latent patterns that may elude traditional methods.

2.1 Objectives

The primary objective of this report is to propose and implement a robust classification framework using computer vision techniques, specifically deep neural networks (DNNs), convolutional neural networks (CNNs), and transfer learning. Our goals include:

- Investigating and comparing various computer vision methodologies, focusing on DNNs, CNNs, and transfer learning, to determine the most suitable approach for the given classification task.

- Designing and implementing a customized classification pipeline leveraging the strengths of computer vision techniques to effectively categorize complex image data into multiple classes.

- Evaluating the performance of the proposed framework through rigorous experimentation and comparative analysis against baseline models, showcasing the efficacy of computer vision techniques in handling intricate datasets.

- Identifying and addressing challenges encountered during the implementation process, providing insights into potential refinements and optimizations for future research and development in computer vision-based classification methodologies.

By focusing exclusively on computer vision techniques, including DNNs, CNNs, and transfer learning, we aim to harness the power of visual information to enhance classification accuracy and contribute to advancements in image-based data analysis. Through systematic experimentation and analysis, we seek to provide valuable insights into the effectiveness and applicability of these techniques in real-world classification tasks.

Chapter 3

Dataset

3.1 Description

The dataset utilized in this study originates from the **xView Recognition Challenge**, a renowned competition in the field of computer vision aimed at advancing the state-of-the-art in satellite image analysis. The xView dataset is a large publicly available object detection dataset containing approximately 1 million manually annotated objects across 60 categories captured by the WorldView-3 satellite at 0.3m ground resolution. For this assignment, we used a subset of the dataset consisting of 846 annotated images, which were divided into 761 images for training and 85 images for testing. From these images, we extracted cropped objects based on the provided bounding box annotations, resulting in 21,377 images for training and 2,635 images for testing, with all images resized to 224x224 pixels. The task was to build models to classify these objects into 12 different categories.

3.2 Data Composition

The dataset consists of two main components: a training set and a test set, each comprising satellite images showcasing a wide array of transportation vehicles and buildings. The images are annotated with bounding boxes delineating the boundaries of objects belonging to twelve distinct categories. These categories encompass different types of transportation vehicles and buildings commonly encountered in satellite imagery, including but not limited to buildings, vehicles, and industrial structures.
The twelve categories present in the dataset are the following: **Building, Small Car, Bus, Truck, Cargo Plane, Dump Truck, Excavator, Fishing Vessel, Helicopter, Motorboat, Shipping Container, Storage Tank**.
Below, in the Figure 3.1, the representation of the object distribution between the categories in the training dataset.

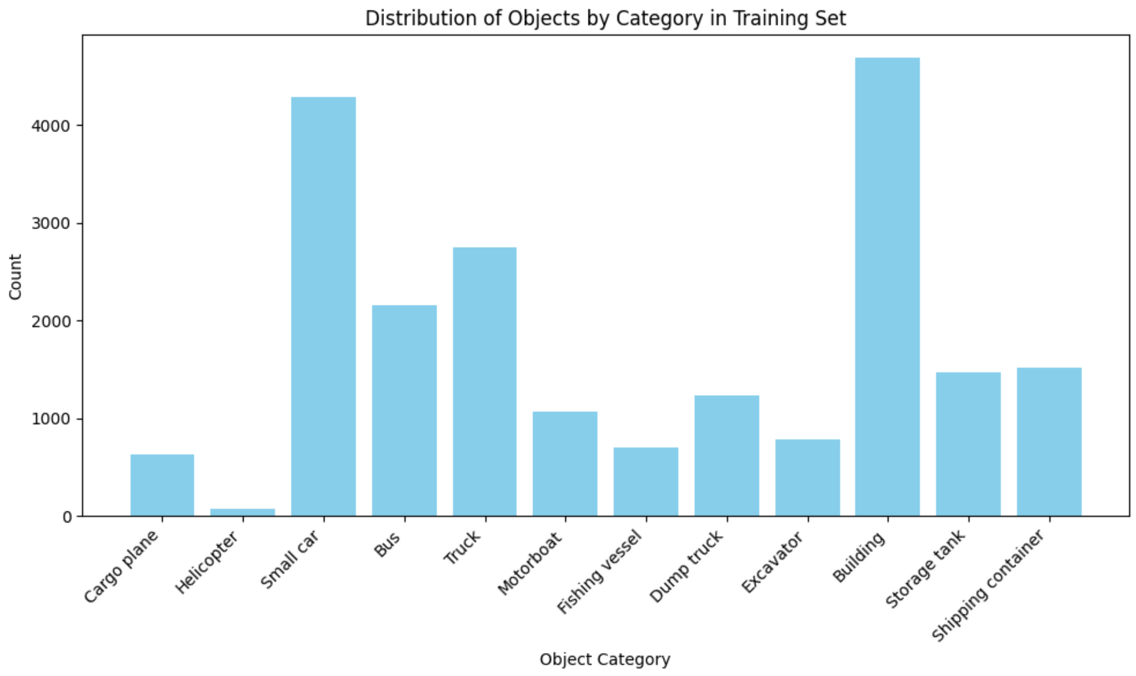


Figure 3.1: Distribution of the objects by category in the training set

Chapter 4

Implementation

4.1 Libraries Utilized

In our implementation, we leveraged several libraries to develop and evaluate our classification models. Notably, we utilized **TensorFlow** and its **Keras** API for constructing neural network architectures. TensorFlow provides a comprehensive framework for building deep learning models, offering flexibility and scalability for various machine learning tasks. Keras, as a high-level neural networks API, enabled us to quickly prototype and experiment with different architectures. Additionally, we employed NumPy for efficient numerical computations, Matplotlib for data visualization, and scikit-learn for evaluation metrics and data preprocessing. These libraries collectively facilitated the development, training, and evaluation of our classification framework.

4.2 Model Architectures and Experiments

4.2.1 Data Handling with Generic Classes

At the outset of our implementation, we recognized the importance of structured data representation for efficient data handling and manipulation. To achieve this, we defined two fundamental classes: *GenericObject* and *GenericImage*.
The *GenericObject* class represents individual objects within an image, encapsulating attributes such as unique identifiers, bounding box coordinates, category labels, and prediction confidence scores. With this class, we could organize and manage object data effectively, facilitating tasks such as data preprocessing, augmentation, and model evaluation.
On the other hand, the *GenericImage* class serves as a container for image data and associated objects. It stores essential information such as image filenames, spatial extents, and lists of objects contained within each image. By utilizing this class, we could structure and organize our dataset into a format suitable for model training and evaluation.
Together, these classes provided a solid foundation for handling geospatial data, enabling us to parse annotation files, extract object information, and construct labeled datasets for experimentation with different architectures.

4.2.2 Experimentation with Model Architectures

Building upon the structured dataset obtained through the *Generic* classes, we embarked on experimenting with various deep learning architectures. Our focus was primarily on convolutional neural networks (CNNs) and transfer learning techniques, leveraging established models and custom architectures tailored to our specific classification task.
In Chapter 5, we delve deeper into the specifics of each architecture, detailing the layer configurations, activation functions, and optimization algorithms employed in our experiments. Through rigorous experimentation and evaluation, we aimed to identify the most effective architecture for our object classification task, considering factors such as accuracy, computational efficiency, and generalization capabilities.
By incorporating the *Generic* classes into our experimentation pipeline, we ensured consistency and efficiency in data handling, enabling seamless integration with different model architectures and facilitating the exploration of diverse design choices.

4.3 Evaluation Techniques

To evaluate the performance of our classification models, we employed various techniques:

- **Confusion Matrix:** We computed the confusion matrix to visualize the model's predictions across different categories. The confusion matrix provided insights into the model's classification accuracy and potential misclassifications.

- **Accuracy, Precision, Recall, and Specificity:** We calculated metrics such as accuracy, precision, recall, and specificity to quantitatively assess the models' performance. These metrics offered a comprehensive understanding of the models' classification abilities and their ability to correctly identify objects across different categories.

- **F1 Score and Dice Coefficient:** Additionally, we computed the F1 score and Dice coefficient, which are harmonic means of precision and recall. These metrics provided a balanced assessment of the models' performance, particularly in scenarios with imbalanced class distributions.

Chapter 5

Feedforward Neural Network

The first approach explored in this project was the use of feed-forward neural networks (FFNNs) for the image recognition task on the xView dataset. FFNNs are a type of artificial neural network where information flows in a single direction, from the input layer through one or more hidden layers to the output layer. In this chapter, we discuss the process of designing and optimizing the FFNN architecture, including the selection of hyperparameters such as the number of layers, neurons per layer, dropout rates, and optimization algorithms. By exploring FFNNs as a baseline, we aim to understand their strengths and limitations for image recognition, before comparing their performance to more advanced architectures like convolutional neural networks.

5.1 Starting Neural Network

The following was the architecture the professors gave us as starting point for our investigation. It is a simple shallow neural network, surely not optimized for image recognition, as the 38% of accuracy in the test set can witness.

5.1.1 Architecture

Listing 5.1: CNN Architecture

```
Input (224 x 224 x 3)
           |
      Flatten Layer
           |
    Activation (ReLU)
           |
       Dropout (0.2)
           |
     Fully Connected Layer (Size: 12)
           |
    Activation (Softmax)
```

```
           |
         Output
```

5.1.2 Hyperparameters

- **Epochs**: 20
- **Batch size**: 16
- **Optimizer**: Adam (learning rate: 0.001)

5.1.3 Results

Validation Set

- Validation Accuracy: 40.36%

Test Set

- Mean Accuracy: 37.799%
- Mean Recall: 25.428%
- Mean Precision: 27.587%

5.2 Experiment 1

This was the first experiment our team tried in order to beat professors one. We just expanded the network, bringing hidden layers to two and adding a Dropout rate of 0.2, in order to minimize overfitting risk. The validation results were better than starting ones, but we were sure we could get better.

5.2.1 Architecture

Listing 5.2: CNN Architecture

```
Input (224 x 224 x 3)
           |
      Flatten Layer
           |
    Fully Connected Layer (Size: 64)
           |
    Activation (ReLU)
           |
       Dropout (0.2)
```

```
            |
    Fully Connected Layer (Size: 64)
            |
    Activation (ReLU)
            |
       Dropout (0.2)
            |
    Fully Connected Layer (Size: 12)
            |
    Activation (Softmax)
            |
         Output
```

5.2.2 Hyperparameters

- **Epochs**: 20
- **Batch size**: 16
- **Optimizer**: Adam (learning rate: 0.001)

5.2.3 Results

Validation Set

- Validation Accuracy: 42.47%

5.3 Experiment 2

In order to get better results, we increased the complexity of the architecture bringing number of neurons in hidden layers, from 64 of previous model, to 512 and 256 of this one and adding batch normalization after hidden layers activation function. Then, we increased Dropout rate to 0.5, to cope with the increasing model complexity. Results showed that adding more complexity was a good point, but it wasn't enough.

5.3.1 Architecture

Listing 5.3: CNN Architecture

```
Input (224 x 224 x 3)
            |
      Flatten Layer
            |
    Fully Connected Layer (Size: 512)
            |
     Activation (ReLU)
```

```
            |
Batch Normalization Layer
            |
    Dropout (0.5)
            |
   Fully Connected Layer (Size: 256)
            |
   Activation (ReLU)
            |
Batch Normalization Layer
            |
       Dropout (0.5)
            |
   Fully Connected Layer (Size: 12)
            |
     Softmax Activation
            |
        Output
```

5.3.2 Hyperparameters

- **Epochs**: 20
- **Batch size**: 16
- **Optimizer**: Adam (learning rate: 0.001)

5.3.3 Results

Validation Set

- Validation Accuracy: 49.02%

5.4 Experiment 3

This model increased again complexity, compared to previous one and, following that, got better results on the validation set. We lowered Dropout rate back to 0.2, but we changed activation function to LeakyReLu and we added two more hidden layers, while reducing again the number of hidden neurons to 60-48-36-24.

5.4.1 Architecture

Listing 5.4: CNN Architecture

```
Input (224 x 224 x 3)
```

```
         |
      Flatten Layer
         |
    Fully Connected Layer (Size: 60)
         |
    Activation (LeakyReLU, alpha=0.1)
         |
        Dropout (0.2)
         |
    Fully Connected Layer (Size: 48)
         |
    Activation (LeakyReLU, alpha=0.1)
         |
        Dropout (0.2)
         |
    Fully Connected Layer (Size: 36)
         |
    Activation (LeakyReLU, alpha=0.1)
         |
    Fully Connected Layer (Size: 24)
         |
    Activation (LeakyReLU, alpha=0.1)
         |
    Fully Connected Layer (Size: 12)
         |
    Activation (LeakyReLU, alpha=0.1)
         |
         Output
```

5.4.2 Hyperparameters

- **Epochs**: 20
- **Batch size**: 16
- **Optimizer**: Adam (learning rate: 0.001)

5.4.3 Results

Validation Set

- Validation Accuracy: 51.72%

5.5 Final Architecture

Then, in order to get our best performance possible, we took the best from each model we ran and we put them together. We used a Dropout rate of 0.25, we used Batch Normalization and

LeakyReLu activation function.

5.5.1 Architecture

Listing 5.5: CNN Architecture

```
Input (224 x 224 x 3)
           |
      Flatten Layer
           |
    Activation (ReLU)
           |
         Dropout (0.25)
           |
Fully Connected Layer (Size: 256)
           |
 Batch Normalization Layer
           |
    Activation (LeakyReLU, alpha=0.1)
           |
         Dropout (0.25)
           |
Fully Connected Layer (Size: 128)
           |
 Batch Normalization Layer
           |
    Activation (LeakyReLU, alpha=0.1)
           |
         Dropout (0.25)
           |
Fully Connected Layer (Size: 64)
           |
 Batch Normalization Layer
           |
    Activation (LeakyReLU, alpha=0.1)
           |
         Dropout (0.25)
           |
Fully Connected Layer (Size: 12)
           |
      Softmax Activation
           |
         Output
```

5.5.2 Hyperparameters

- **Epochs**: 20
- **Batch size**: 16
- **Optimizer**: Adam (learning rate: 0.001)

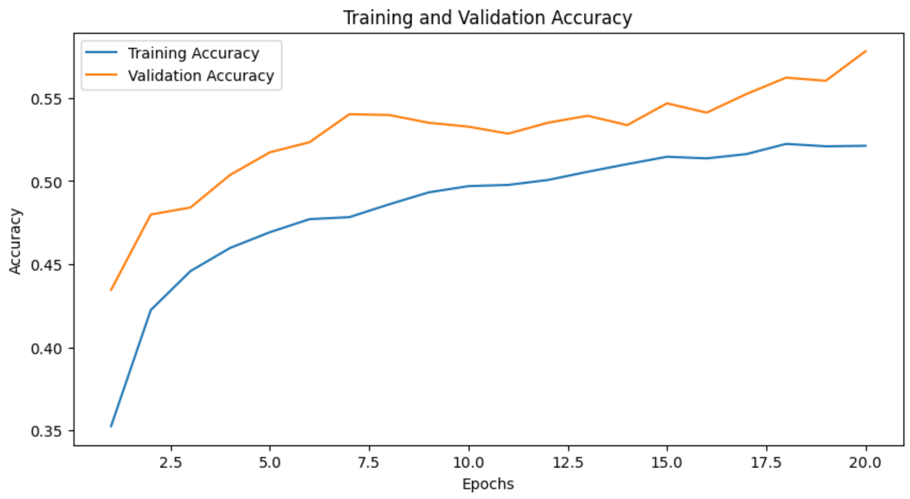
5.5.3 Results

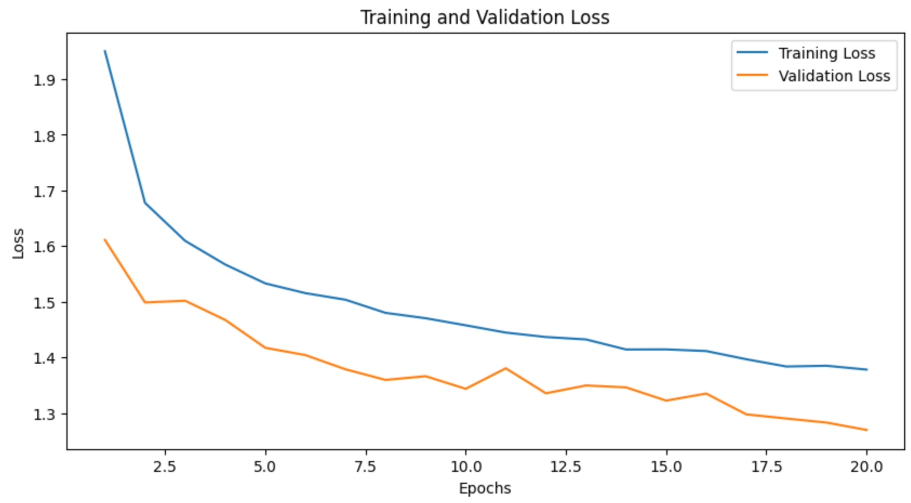
Validation Set

- Validation Accuracy: 57.81%

Test Set

- Mean Accuracy: 49.526%
- Mean Recall: 40.355%
- Mean Precision: 43.793%





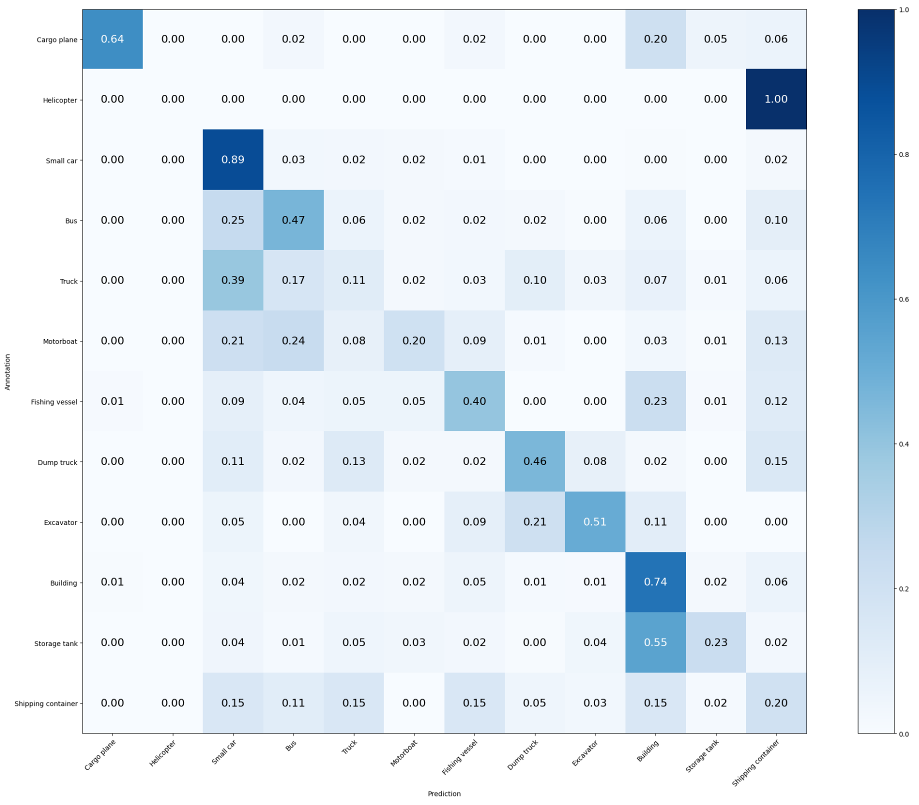


Figure 5.1: Confusion Matrix of the final FFNN model.

Chapter 6

Convolutional Neural Network

Extending beyond feed-forward neural networks, we explored convolutional neural networks (CNNs), architectures specifically designed for image data. CNNs leverage convolutional filters to extract low and high-level features, capturing spatial and local correlations in images. This chapter covers our implementation and evaluation of CNN models trained from scratch on the xView dataset. We discuss the CNN architectures, including convolutional, pooling, and fully connected layers, as well as hyperparameters like filter sizes and activation functions. We compare CNNs to the baseline FFNNs, highlighting their strengths for image recognition.

6.1 Architecture 1

This first architecture became our starting point, as none in the team had ever experienced with Convolutional Neural Networks. For the convolutional network part, we used the most common kernel size for four layers, with filters doubling each time, followed by Max Pooling layers, while for the fully connected network we kept it simple, to avoid overfitting risks, only with one hidden layer, but with quite every technique we used previously to optimize performance. This model already improved our best FFNN validation accuracy of 10% but, as we knew they could improve more, we kept building another model.

6.1.1 Architecture

Listing 6.1: CNN Architecture

```
Input (224 x 224 x 3)
              |
     Conv2D Layer (Filters: 32, Kernel Size: 3x3, Activation: ReLU)
              |
MaxPooling2D Layer (Pool Size: 2x2)
              |
     Conv2D Layer (Filters: 64, Kernel Size: 3x3, Activation: ReLU)
```

```
            |
MaxPooling2D Layer (Pool Size: 2x2)
            |
    Conv2D Layer (Filters: 128, Kernel Size: 3x3, Activation: ReLU)
            |
MaxPooling2D Layer (Pool Size: 2x2)
            |
    Conv2D Layer (Filters: 128, Kernel Size: 3x3, Activation: ReLU)
            |
MaxPooling2D Layer (Pool Size: 2x2)
            |
       Dropout (0.5)
            |
      Flatten Layer
            |
    Fully Connected Layer (Size: 512)
            |
Batch Normalization Layer
            |
      Activation (ReLu)
            |
       Dropout (0.5)
            |
    Fully Connected Layer (Size: 12)
            |
      Activation (Softmax)
            |
         Output
```

6.1.2 Hyperparameters

- **Epochs**: 15
- **Batch size**: 32
- **Optimizer**: Adam (learning rate: 0.001)

6.1.3 Results

Validation Set

- Validation Accuracy: 67.39%

6.2 Final Architecture

After analysis of best CNNs architectures in class, we understood that depth in convolutional network could have improved our overall performance. For this reason we tried to build a deeper network, with a total of 5 convolutional layers, with filters doubled each time, each followed by a MaxPooling layer. We inserted a series of Batch Normalization and ReLu activation function after convolutional layers and a 0.3 Dropout rate after MaxPooling ones. Then, the fully connected neural network is composed by two hidden neurons with 256-128 neurons, with HeNormal initializer and L2 (0.001) regularizer, to reduce overfitting risks due to overall network complexity increase. Each layer has a Batch Normalization, a LeakyReLu activation function and a 0.3 Dropout rate. After testing the CNN, we were quite surprised of the results, as it was already outscoring professors' weekly model, so we considered our intermediate goal reached.

6.2.1 Architecture

Listing 6.2: CNN Architecture

```
Input (224 x 224 x 3)
           |
    Conv2D Layer (Filters: 32, Kernel Size: 3x3)
 Batch Normalization Layer
    Activation Layer (ReLU)
           |
MaxPooling2D Layer (Pool Size: 2x2)
       Dropout (0.3)
           |
    Conv2D Layer (Filters: 64, Kernel Size: 3x3)
 Batch Normalization Layer
    Activation Layer (ReLU)
           |
MaxPooling2D Layer (Pool Size: 2x2)
       Dropout (0.3)
           |
    Conv2D Layer (Filters: 128, Kernel Size: 3x3)
 Batch Normalization Layer
    Activation Layer (ReLU)
           |
MaxPooling2D Layer (Pool Size: 2x2)
       Dropout (0.3)
           |
    Conv2D Layer (Filters: 256, Kernel Size: 3x3)
 Batch Normalization Layer
    Activation Layer (ReLU)
           |
MaxPooling2D Layer (Pool Size: 2x2)
       Dropout (0.3)
           |
    Conv2D Layer (Filters: 512, Kernel Size: 3x3)
 Batch Normalization Layer
    Activation Layer (ReLU)
```

```
           |
MaxPooling2D Layer (Pool Size: 2x2)
       Dropout (0.3)
      Flatten Layer
           |
    Fully Connected Layer (Size: 256)
 Batch Normalization Layer
LeakyReLU Activation Layer (Alpha: 0.1)
       Dropout (0.3)
           |
    Fully Connected Layer (Size: 128)
 Batch Normalization Layer
LeakyReLU Activation Layer (Alpha: 0.1)
       Dropout (0.3)
           |
    Fully Connected Layer (Size: 12)
      Softmax Activation
           |
         Output
```

6.2.2 Hyperparameters

- **Epochs**: 40
- **Batch size**: 32
- **Optimizer**: Adam (learning rate: 0.001)

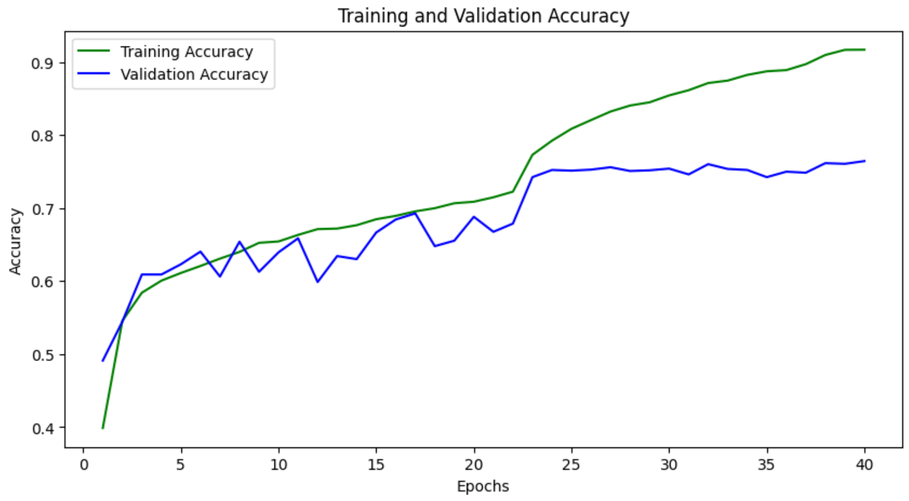
6.2.3 Results

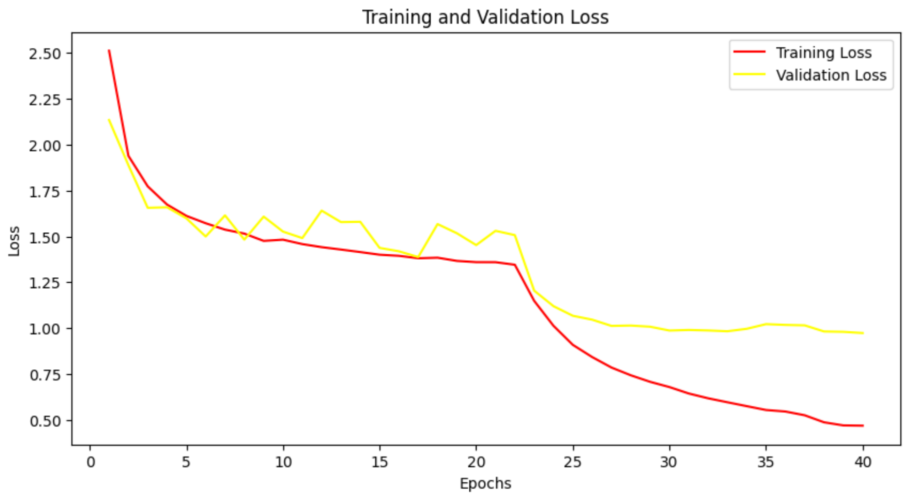
Validation Set

- Validation Accuracy: 76.42%

Test Set

- Mean Accuracy: 67.14%
- Mean Recall: 59.19%
- Mean Precision: 59.73%


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epochs


Training and Validation Loss
Training Loss
Validation Loss
Loss
Epochs

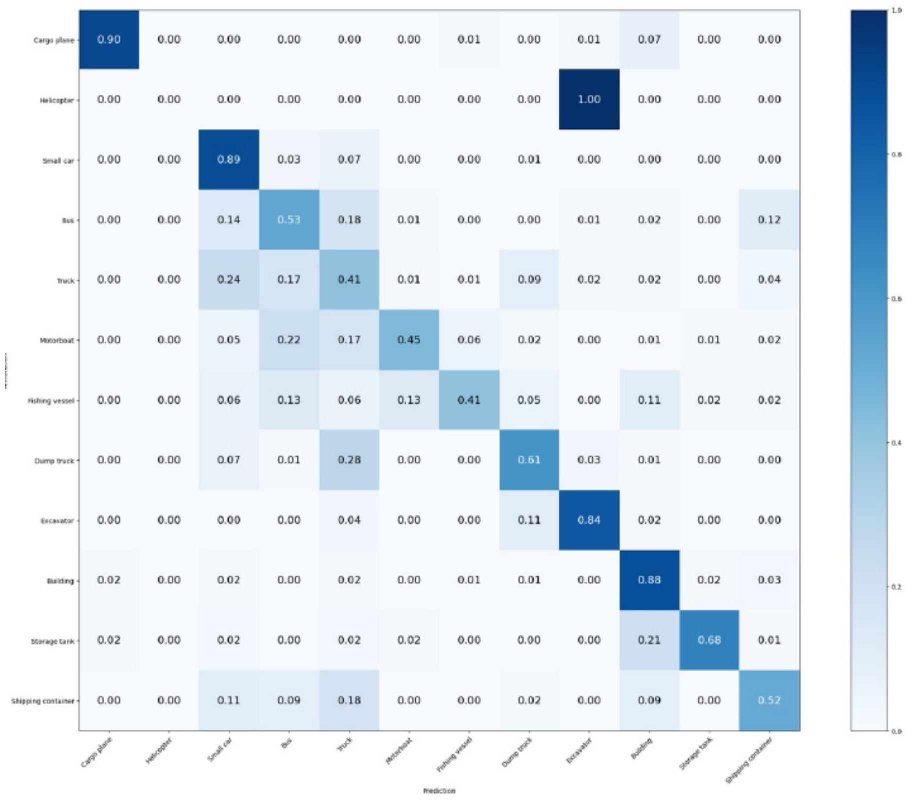


Figure 6.1: Confusion Matrix of the final CNN model.

Chapter 7

Transfer Learning

After training models from scratch, we explored transfer learning, leveraging pre-trained models on large datasets like ImageNet and fine-tuning them on the xView task. This chapter covers our experiments with transfer learning using popular CNN architectures like VGGNet, ResNet, and Inception. We discuss selecting suitable pre-trained models and mitigating class imbalance effects. Strategies for optimizing hyperparameters like learning rates and fine-tuning schedules are also explored. The fine-tuned models' performance is evaluated on the test set and compared to models trained from scratch, analyzing trade-offs between complexity, training time, and accuracy.

7.1 Architecture 1

For the first architecture, we started with one of the best model we saw during lesson, VGG, followed by a simple fully connected neural network, with only one hidden layer, with ReLu function and Dropout, at 0.5 rate. Results were good, even more considering it was only the initial architecture.

Listing 7.1: CNN Architecture

```
Input (224 x 224 x 3)
           |
    VGG16 Base Model (Pre-trained on ImageNet)
           |
      Flatten Layer
           |
    Fully Connected Layer (Size: 256)
           |
      Activation (ReLu)
           |
        Dropout (0.5)
           |
    Fully Connected Layer (Size: 12)
           |
      Activation (Softmax)
           |
         Output
```

7.1.1 Hyperparameters

- **Epochs**: 36 (Early Stopping)
- **Batch size**: 32
- **Optimizer**: Adam (learning rate: 0.001)

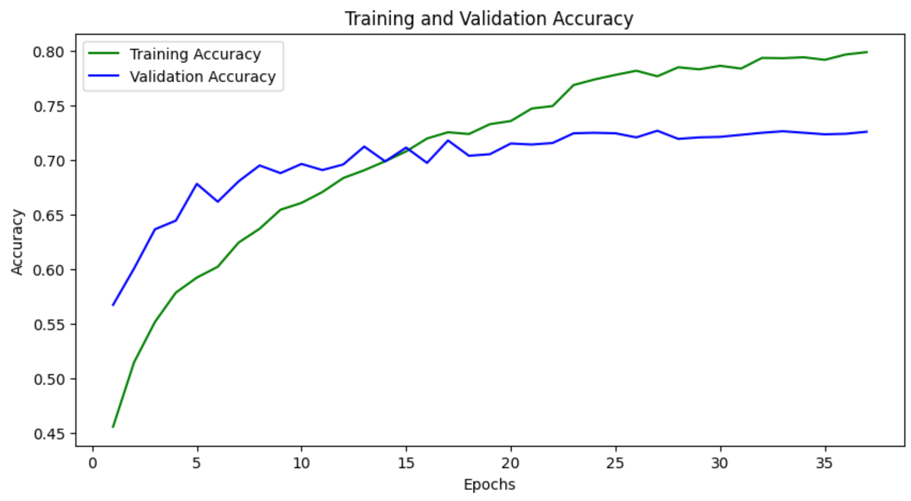
7.1.2 Results

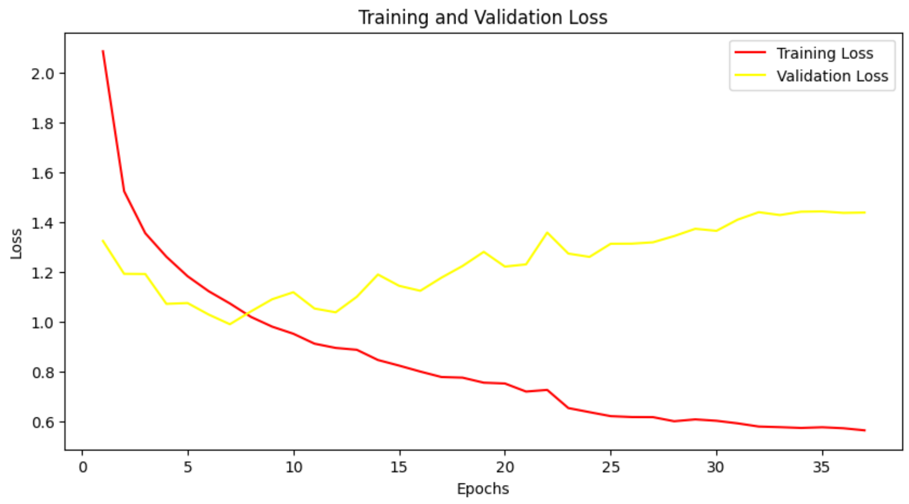
Validation Set

- Validation Accuracy: 73.27%

Test Set

- Mean Accuracy: 69.943%
- Mean Recall: 57.422%
- Mean Precision: 62.937%





7.2 Architecture 2

Then, after looking at Keras documentation, we found that EfficientNet could be the most suitable net for transfer learning, due to its efficiency, combined with the light weight it has. In order to see if our thoughts were right, we kept the rest of the network the same and, seeing our results, we were right, as the validation accuracy increased of more than 2%.

Listing 7.2: CNN Architecture

```
Input (224 x 224 x 3)
           |
EfficientNetB0 Base Model (Pre-trained on ImageNet)
           |
Global Average Pooling Layer
           |
    Fully Connected Layer (Size: 256)
           |
    Batch Normalization Layer
           |
     Activation (Leaky ReLu, alpha=0.1)
           |
          Dropout (0.5)
           |
    Fully Connected Layer (Size: 12)
           |
      Activation (Softmax)
           |
         Output
```

7.2.1 Hyperparameters

- **Epochs**: 40
- **Batch size**: 32
- **Optimizer**: Adam (learning rate: 0.001)

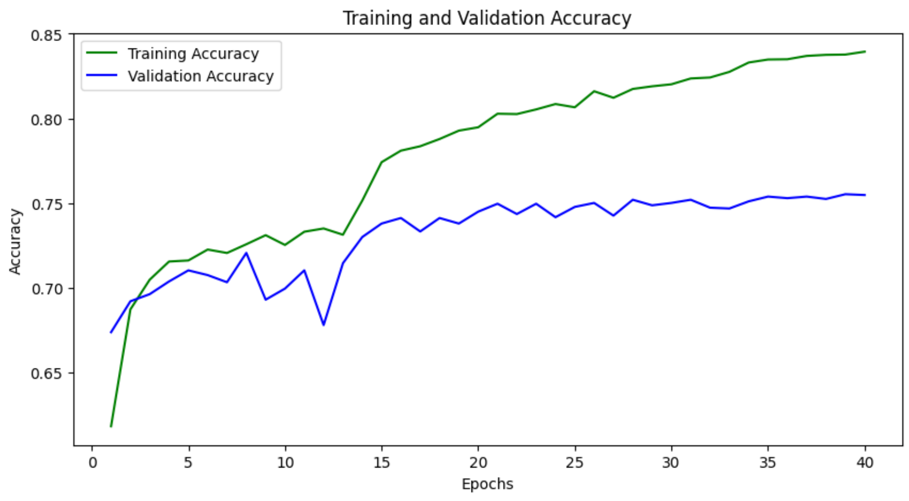
7.2.2 Results

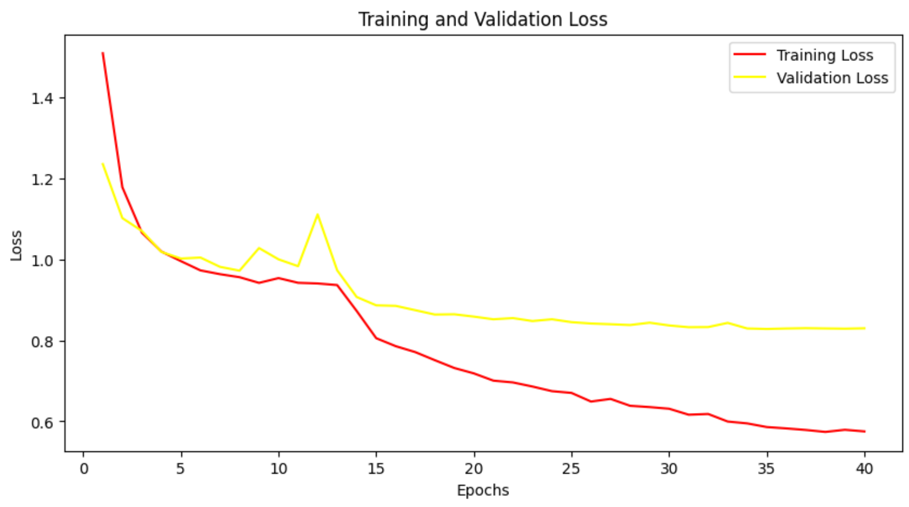
Validation Set

- Validation Accuracy: 75.43%

Test Set

- Mean Accuracy: 72.182%
- Mean Recall: 61.109%
- Mean Precision: 63.001%





7.3 Architecture 3

We continued our investigation changing increasing the complexity in the network. We started by taking a more efficient (and heavier) model for the transfer learning, switching from EfficientNetB0 to EfficientNetB1, which can be the most suitable to us among all EfficientNet models (due to our limited resources), then we doubled neurons in the hidden layer. We increased also the batch size to 64 and, seeing colleagues results, we started using Nadam optimizer, instead of Adam and we saw our network achieving better results.

Listing 7.3: CNN Architecture

```
Input (224 x 224 x 3)
           |
EfficientNetB1 Base Model (Pre-trained on ImageNet)
           |
Global Average Pooling Layer
           |
    Fully Connected Layer (Size: 512)
           |
    Batch Normalization Layer
           |
     Activation (Leaky ReLu, alpha=0.1)
           |
          Dropout (0.5)
           |
    Fully Connected Layer (Size: 12)
           |
      Activation (Softmax)
```

```
            |
          Output
```

7.3.1 Hyperparameters

- **Epochs**: 20
- **Batch size**: 64
- **Optimizer**: Nadam (learning rate: 0.001)

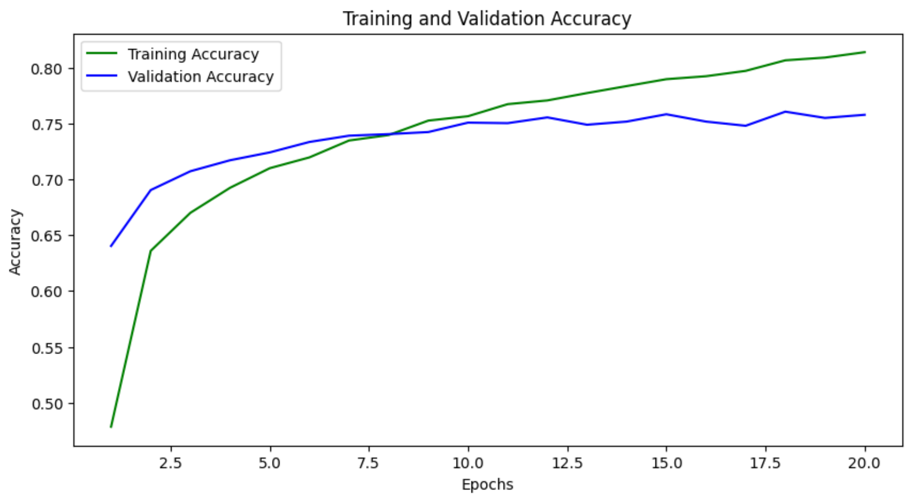
7.3.2 Results

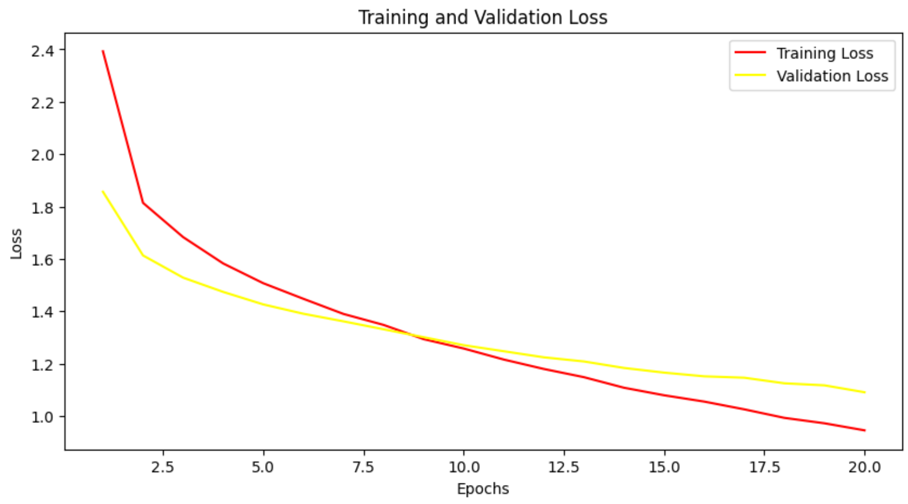
Validation Set

- Validation Accuracy: 76.05%

Test Set

- Mean Accuracy: 73.586%
- Mean Recall: 61.728%
- Mean Precision: 63.696%





7.4 Architecture 4

Changes in architecture 4 followed the same trend of increasing complexity, as we added another hidden layer in our network and we gave our model more time to train, raising epochs hyperparameters to 40. In this case we obtained really suprising results, with more than 78% of validation accuracy.

Listing 7.4: CNN Architecture

```
Input (224 x 224 x 3)
           |
EfficientNetB1 Base Model (Pre-trained on ImageNet)
           |
Global Average Pooling Layer
           |
    Fully Connected Layer (Size: 512)
           |
    Batch Normalization Layer
           |
     Activation (Leaky ReLU, alpha=0.1)
           |
        Dropout (0.5)
           |
    Fully Connected Layer (Size: 256)
           |
    Batch Normalization Layer
           |
     Activation (Leaky ReLu, alpha=0.1)
```

```
          |
       Dropout (0.5)
          |
    Fully Connected Layer (Size: 12)
          |
     Activation (Softmax)
          |
        Output
```

7.4.1 Hyperparameters

- **Epochs**: 40
- **Batch size**: 64
- **Optimizer**: Nadam (learning rate: 0.0001)

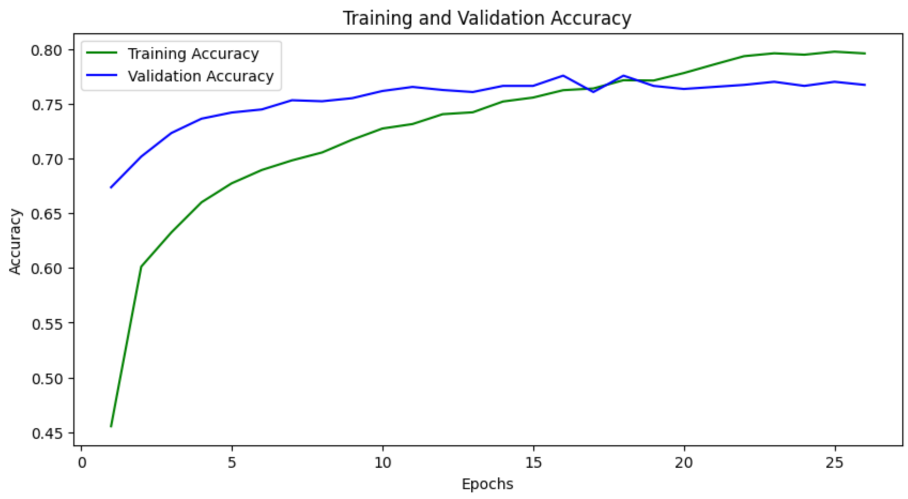
7.4.2 Results

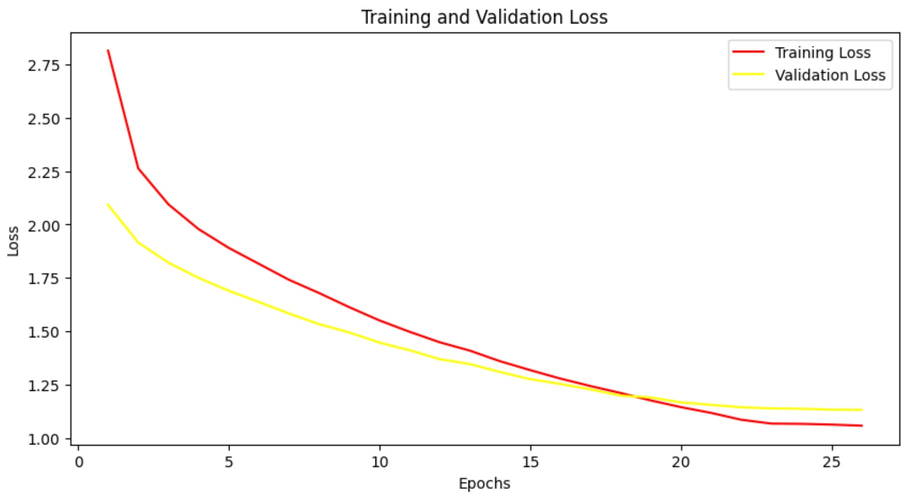
Validation Set

- Validation Accuracy: 78.13%

Test Set

- Mean Accuracy: 75.218%
- Mean Recall: 64.635%
- Mean Precision: 65.501%





7.5 Architecture 5

To enhance our model's performance, we decided to employ data augmentation techniques, recognizing the critical role of augmented data in improving training outcomes. Leveraging the Keras library's ImageDataGenerator, we augmented our dataset with diverse variations of input images. However, this approach introduced a significant drawback: a tenfold increase in training time per epoch, adversely impacting our experimentation process.

To address this computational bottleneck, given our limited resources, we opted for a simpler neural network architecture. We chose the EfficientNet B0 model as our base due to its lightweight nature and added only one hidden layer. This layer incorporated batch normalization, Leaky ReLU activation, and dropout regularization techniques to enhance the model's robustness and prevent overfitting. Additionally, we adjusted the batch size to 128 and reduced the number of epochs to 20 to accommodate computational constraints.

Despite these adjustments, our results fell short of expectations, failing to surpass the performance of the previous architecture. This outcome underscores the complexity of optimizing neural network architectures and the need for careful experimentation to achieve desired outcomes.

Listing 7.5: CNN Architecture

```
Input (224 x 224 x 3)
    |
EfficientNetB0 Base Model (Pre-trained on ImageNet)
    |
Global Average Pooling Layer
    |
Fully Connected Layer (Size: 256)
    |
    Batch Normalization Layer
       |
    Activation (Leaky ReLU, alpha=0.1)
       |
    Dropout (0.5)
       |
    Fully Connected Layer (Size: 12)
       |
    Activation (Softmax)
       |
    Output
```

7.5.1 Hyperparameters

- **Epochs**: 20
- **Batch size**: 128
- **Optimizer**: Nadam (learning rate: 0.0001)

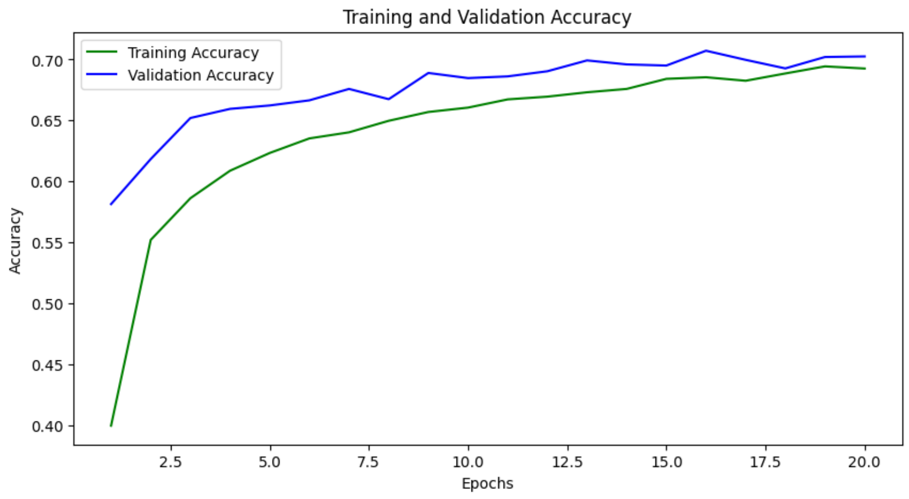
7.5.2 Results

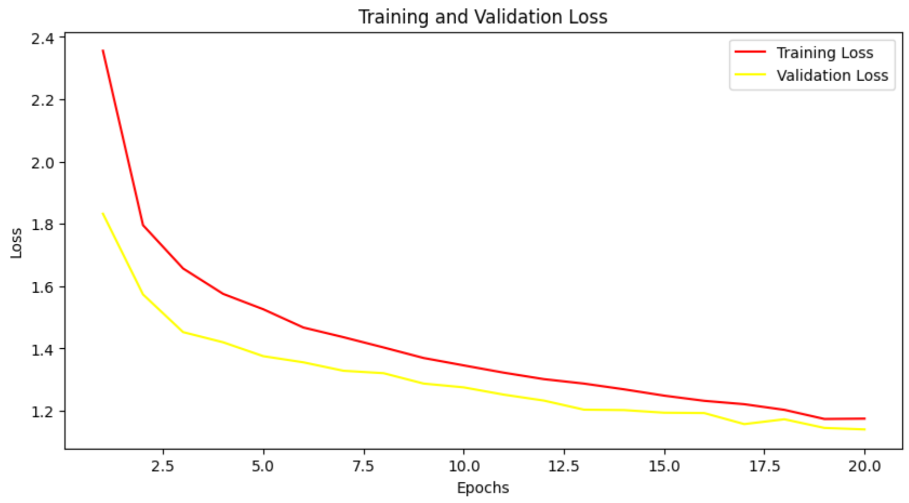
Validation Set

- Validation Accuracy: 70.72%

Test Set

- Mean Accuracy: 69.336%
- Mean Recall: 58.701%
- Mean Precision: 61.590%





7.6 Architecture 6

Following previous architecture performance, we tried to create a more complex architecture in order to take the best from our data augmentation. We added a 512 neurons hidden layer, a batch normalization, a Leaky ReLu and a dropout layer. The problems with this architecture was its training time, as said before, which led us to decrease epochs to 10. Results show the weakness or this model, strictly related to the too low epochs value. This of course leads to a underfitted model, having only 64.04% of training accuracy in the last epoch but, as visible from the graphs below, validation accuracy was still increasing. This shows that the results are quite convincing, we just need to increase epochs number (with the risk of not training the model ever).

Listing 7.6: CNN Architecture

```
Input (224 x 224 x 3)
   |
EfficientNetB0 Base Model (Pre-trained on ImageNet)
   |
Global Average Pooling Layer
   |
Fully Connected Layer (Size: 512)
   |
Batch Normalization Layer
   |
Activation (Leaky ReLU, alpha=0.1)
   |
Dropout (0.5)
   |
Fully Connected Layer (Size: 256)
   |
Batch Normalization Layer
   |
Activation (Leaky ReLU, alpha=0.1)
   |
Dropout (0.5)
   |
Fully Connected Layer (Size: 12)
   |
Activation (Softmax)
   |
Output
```

7.6.1 Hyperparameters

- **Epochs**: 10
- **Batch size**: 128
- **Optimizer**: Nadam (learning rate: 0.0001)

7.6.2 Results

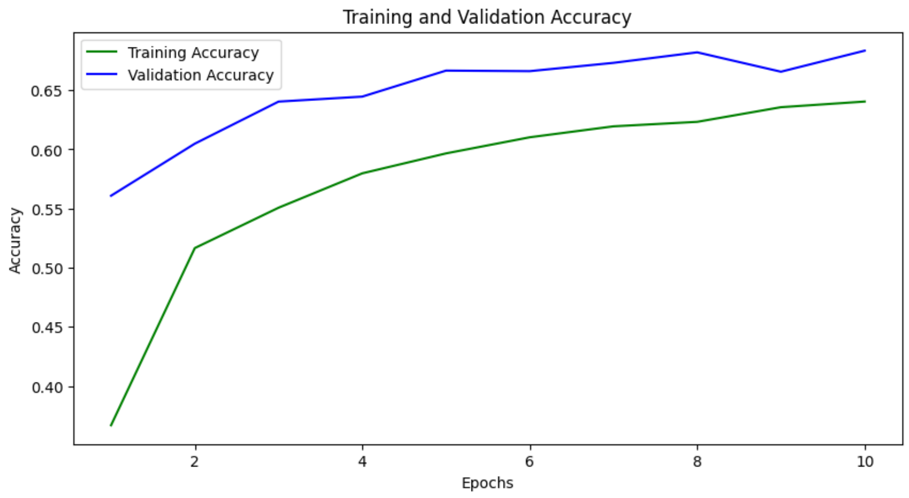
Validation Set

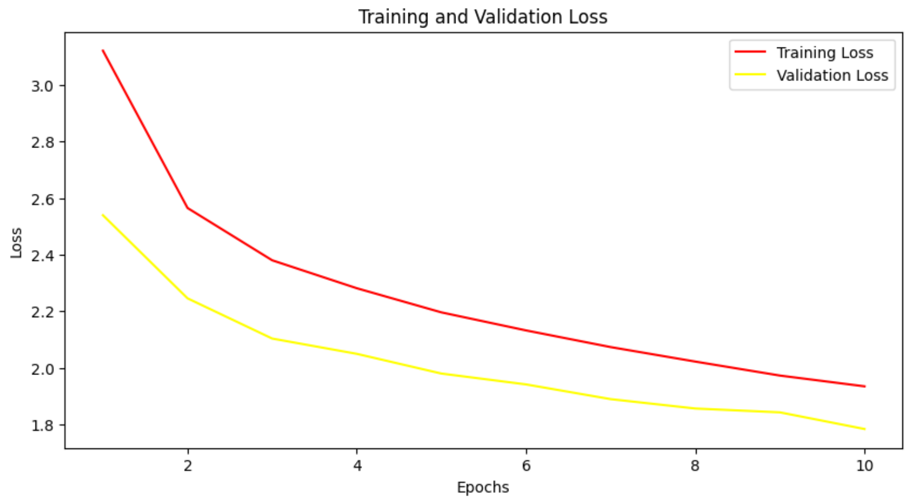
- Validation Accuracy: 68.33%

Test Set

- Mean Accuracy: 67.059%

- Mean Recall: 57.491%
- Mean Precision: 58.046%





7.7 Architecture 7

After previous architecture results, we tried to reduce data augmentation techniques, leading to impossibility of training network properly due to our resources restrictions. We limited data augmentation only to Helicopter class, which was the only one failing quite all predictions, due to the low number of images in training set. So we changed generator object, to augment only that class and we increased epochs number to 30, in a way it could try to learn more, In our opinion this value could be even set higher as in the accuracy graph validation accuracy used to grow during all training phase.

Listing 7.7: CNN Architecture

```
Input (224 x 224 x 3)
   |
EfficientNetB1 Base Model (Pre-trained on ImageNet)
   |
Global Average Pooling Layer
   |
Fully Connected Layer (Size: 512)
   |
Batch Normalization Layer
   |
Activation (Leaky ReLU, alpha=0.1)
   |
Dropout (0.5)
   |
Fully Connected Layer (Size: 256)
   |
Batch Normalization Layer
   |
Activation (Leaky ReLU, alpha=0.1)
   |
Dropout (0.5)
   |
Fully Connected Layer (Size: 12)
   |
Activation (Softmax)
   |
Output
```

7.7.1 Hyperparameters

- **Epochs**: 30
- **Batch size**: 128
- **Optimizer**: Nadam (learning rate: 0.0001)

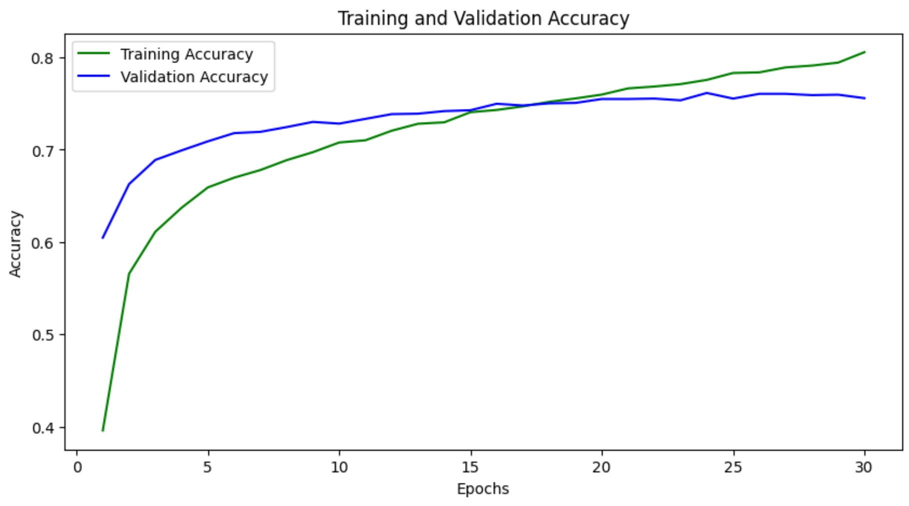
7.7.2 Results

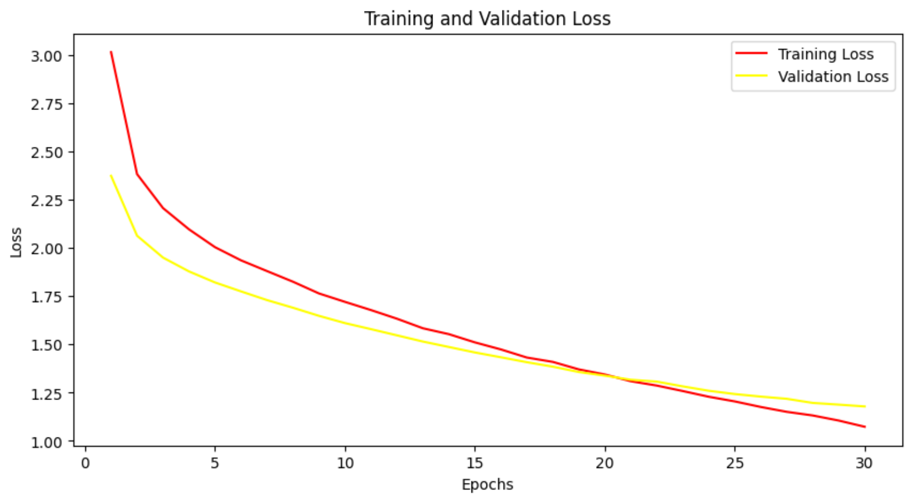
Validation Set

- Validation Accuracy: 76.15%

Test Set

- Mean Accuracy: 74.421%
- Mean Recall: 64.525%
- Mean Precision: 64.525%





7.8 Architecture 8

As some classes were consistently misclassified across all architectures used thus far, we decided to explore the application of data augmentation techniques to improve the classification of these spe-

cific categories. The identified problematic classes included *"Helicopter," "Bus," "Fishing vessel," "Dump truck," and "Shipping container."*
To address this issue, we applied **data augmentation** techniques to images belonging to these classes. This involved generating artificial variants of the original images through rotations, translations, zooms, and other transformations. The goal of this technique was to provide the model with a greater variety of data during the training process, allowing it to learn more robust and generalizable features.
Subsequently, we applied the same structure as the previously experimented architecture in the section "Architecture 7" (see Listing 7.7) to this augmented data. We then trained the model using the augmented data and evaluated its performance against the target classes.
The results obtained were as follows.

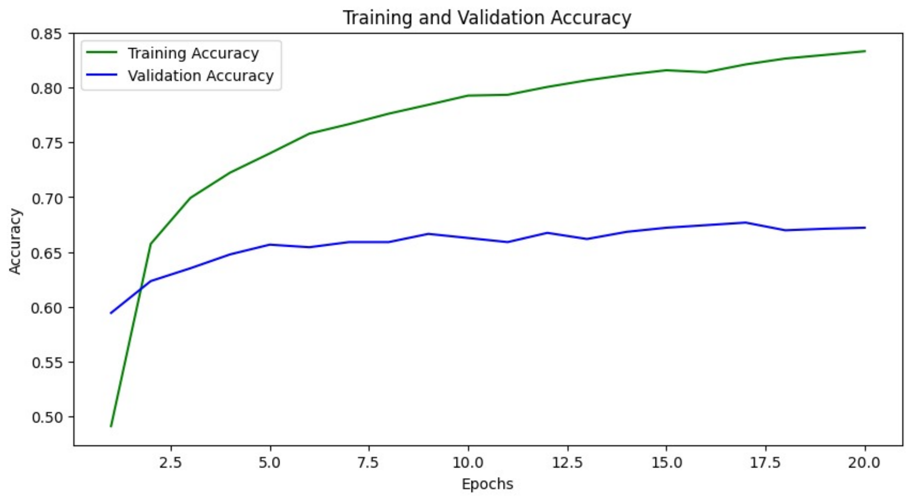
7.8.1 Results

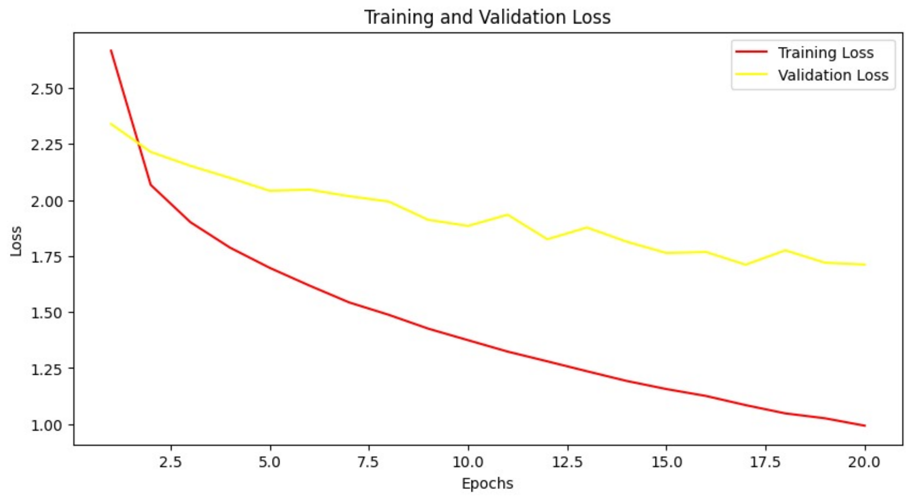
Validation Set

- Validation Accuracy: 67.68%

Test Set

- Mean Accuracy: 70.892%
- Mean Recall: 54.157%
- Mean Precision: 67.364%





7.9 Final Architecture

In our pursuit of identifying the most effective architecture, we carefully assessed several options. Ultimately, we settled on architecture number 7, which emerged as the frontrunner in terms of performance. This architecture stands out for its utilization of transfer learning with EfficientB1, followed by a CNN featuring three layers (see Listing 7.7) on 30 epochs.
While exploring various architectures, architecture 6 also caught our attention. Despite our inability to fully explore its potential due to resource limitations, its initial performance was notable. We managed to run it for only 10 epochs, but even within this short timeframe, it displayed promising results.
Our final Neural Network brought us to a final accuracy score of 74.4% and the following confusion matrix:

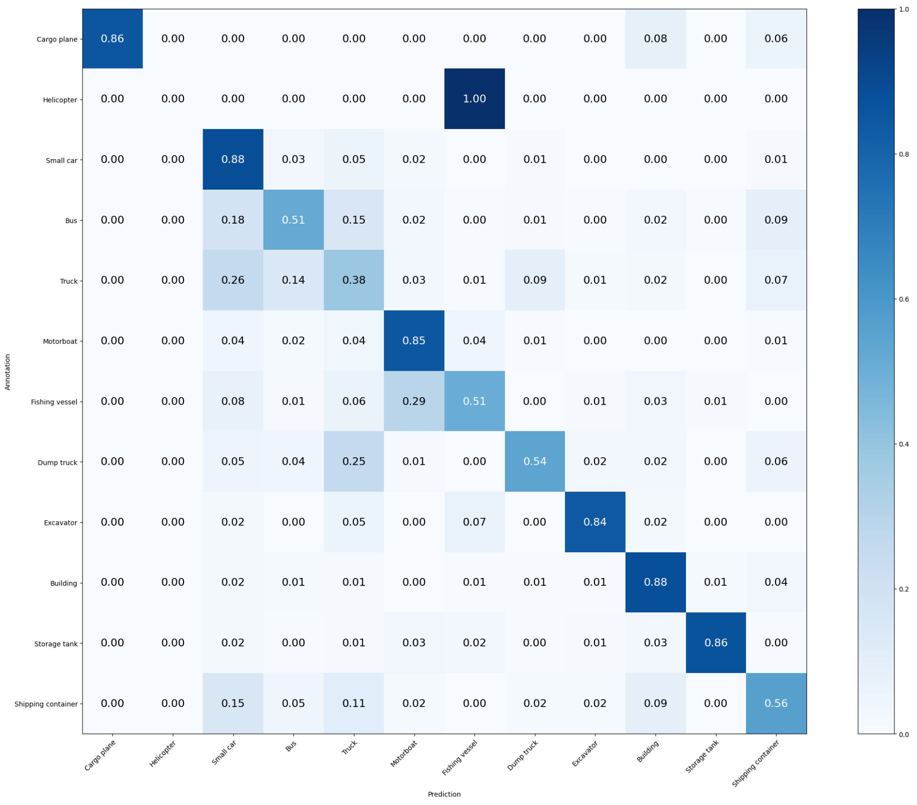


Figure 7.1: Confusion Matrix of the final model.

Chapter 8

Conclusion

Throughout our project, we conducted a series of experiments aimed at exploring the capabilities of various neural network architectures for image classification tasks. Starting with feedforward neural networks, we progressed to convolutional neural networks (CNNs) and finally experimented with transfer learning.
Our efforts yielded promising results, with our final CNN architecture achieving a maximum accuracy of 76%. We are quite satisfied with this outcome, considering the complexity of the task and the resources available to us.

However, it's important to acknowledge the limitations we encountered during the project. Due to hardware constraints, particularly limited computational resources, we faced challenges in conducting comprehensive experiments. On platforms like **Kaggle**, the runtime was prohibitively slow, while on **CESVIMA**, GPU availability was limited. Even on platforms like **Colab**, where GPU resources were more readily available, the runtime was insufficient for conducting extensive experiments.

Moving forward, to further enhance the performance of our models and explore additional avenues for improvement, we recognize the need for more extensive experimentation. This would involve leveraging more powerful computational resources to conduct a larger number of trials and fine-tune our models effectively.

In conclusion, while we have achieved promising results with our current set of experiments, there is still room for improvement. Our project serves as a foundation for future research, highlighting the potential of neural networks for image classification tasks and the importance of resource availability in conducting comprehensive experiments.