Image Recognition Assignment

Deep Learning

Course 2022/23

Víctor Morcuende Castell

Guillermo Nájera Lavid

TABLE OF CONTENTS

1. Introduction

In the rapidly evolving fields of deep learning and computer vision, image recognition has emerged as a crucial component with a wide range of applications, such as autonomous vehicles, satellite imagery analysis, and surveillance systems. Deep learning methods, particularly neural networks, have demonstrated remarkable performance in tackling image recognition challenges. This report aims to present our efforts to develop and optimize deep learning models, specifically feed-forward Neural Networks (ffNNs) and Convolutional Neural Networks (CNNs), to address an image recognition problem using the xView dataset.

The xView dataset is a comprehensive collection of high-resolution satellite images, comprising approximately one million annotated objects across 60 categories. Captured using the WorldView-3 satellite, the dataset features a 0.3m ground sample distance, providing rich and detailed images. In total, 846 annotated images are available, which have been divided into 761 and 85 images for training and testing, respectively. For the purpose of this assignment, we were told to focus on the 12 most represented classes ("Cargo plane", "Helicopter", "Small car", "Bus", "Truck", "Motorboat", "Fishing vessel", "Dump truck", "Excavator", "Building", "Storage tank", "Shipping container"), resulting in 717 training images and 79 testing images.

To address the image recognition task, we were given an already implemented generator that extracts objects of interest from the images by cropping them according to their annotated bounding boxes. This procedure led to the selection of 40,186 objects for training and 4,777 objects for testing. The cropped images were then resized to a dimension of 224x224 pixels, which we think was done to facilitate uniformity in training and evaluation.

One challenge in utilizing the xView dataset is the presence of significant class imbalance, particularly for the "Building" and "Small Car" categories. This disparity in class frequencies will likely impact negatively the overall accuracy of the model, requiring the implementation of strategies to mitigate its effects during model training.

In this report, we will detail the techniques and methodologies employed to develop and optimize ffNN and CNN models for the image recognition task. Additionally, we will explore the process of training popular architectures from scratch and compare their performance with pre-trained versions utilizing transfer learning. Our goal is to demonstrate proficiency in using Tensorflow/Keras for training Neural Nets and to apply the acquired knowledge in the context of an image recognition problem.

2. Intermediate Results

2.1. First Results

For this part, we were only allowed to use feed-forward Neural Networks (ffNNs), for the purpose of putting into practice the knowledge acquired to optimize the parameters and architecture of this type of networks.

Therefore, the model created in this first part consisted of a sequential architecture, starting with a Flatten layer to convert the input images of size 224x224x3 into 1D arrays. After that, subsequent layers included a series of Dense layers, each followed by Batch Normalization, a Leaky ReLU activation function with an alpha of 0.01, and a Dropout with varying rates ranging from 0.3 to 0.4. Furthermore, the number of neurons in the Dense layers increases from 64 to 2048 (by a power of 2) before decreasing again to 512 and 256. Moreover, the Dense layers from 2048 neurons until the end included L1 and L2 regularization for the purpose of reducing any possible overfitting. Finally, the last Dense layer had as many neurons as categories existed (so, 12 neurons), followed by a SoftMax activation function to output probabilities for each class. This deep architecture that we created aimed to learn complex representations of the input data while enhancing the model's classification performance. Finally, it should be noted that the only optimization algorithm used at this stage is the Adam optimizer.

Considering all this, the first model had the following parameters:

- Total params: 13,633,996
- Trainable params: 13,624,396
- Non-trainable params: 9,600

Moreover, the training process reached 60 epochs, with the best validation accuracy of 0.5934 achieved at epoch 56 and the best accuracy of 0.5734 achieved at epoch 60. Next, below it can be seen the model's performance on the train and validation:

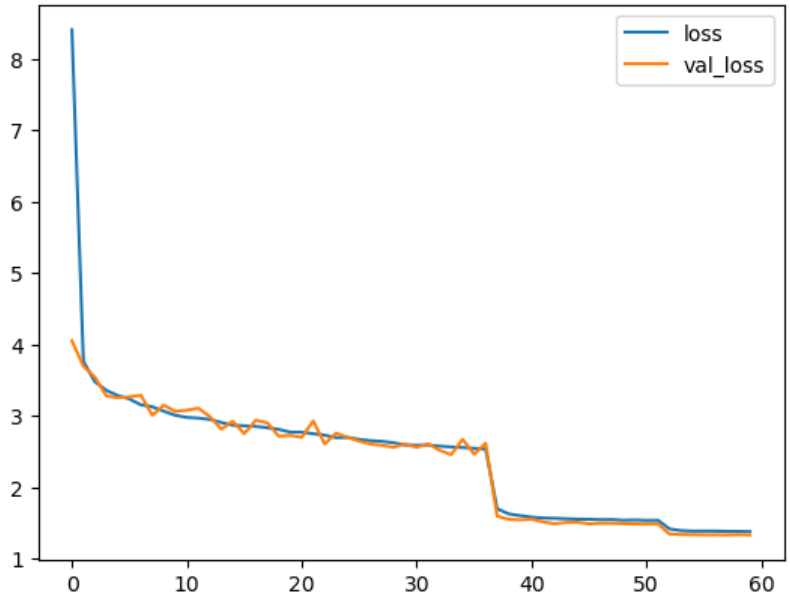

Figure 1: Loss vs Validation Loss for Model 1

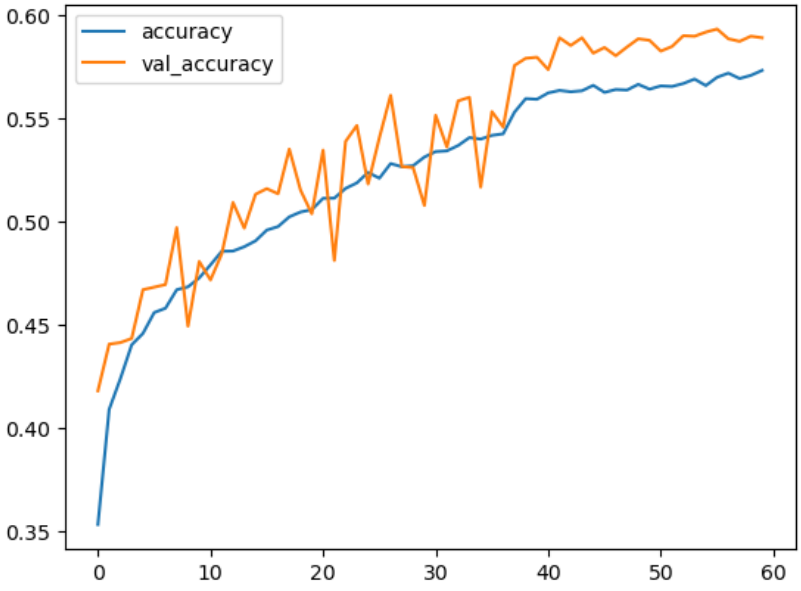

Figure 2: Accuracy vs Validation Accuracy for Model 1

Regarding the test dataset, the results provided an evaluation of the model's performance on unseen data, which can help determine how well the model generalizes. In this case, the model achieved a mean accuracy of 58.007% of the test samples, which is not acceptable, clearly indicating room for improvement.

The 3 metrics used to measure the test results are accuracy, recall and precision. Recall measures the proportion of actual positive cases that the model correctly identified, while precision assesses the proportion of predicted positive cases that were actually positive. Also, the Dice score (used at the class-specific metrics) is a metric that

combines both recall and precision to provide a single measure of the model's performance, ranging from 0 (worst) to 1 (best).

In this case, the model demonstrates a significant imbalance in its ability to recognize different object classes, as indicated by the mean recall (22.293%), the mean precision (20.072%), and Dice scores, which was not a surprise for us since we already acknowledged the severe class imbalances that the dataset possesses. Therefore, looking at the class-specific metrics, the model performs well on classes such as "Small Car", "Bus", and "Building", with relatively high recall and precision values. However, it performs poorly on classes such as "Cargo Plane", "Helicopter", "Motorboat", "Fishing Vessel", and "Dump Truck", with recall and precision values of 0%. This suggests that the model is not able to identify these classes effectively, something understandable given how small are them compared to the others and considering the fact that we are only using dense layers.

Finally, here it can be appreciated the confusion matrix of this model, which contrasts the explanation just provided:

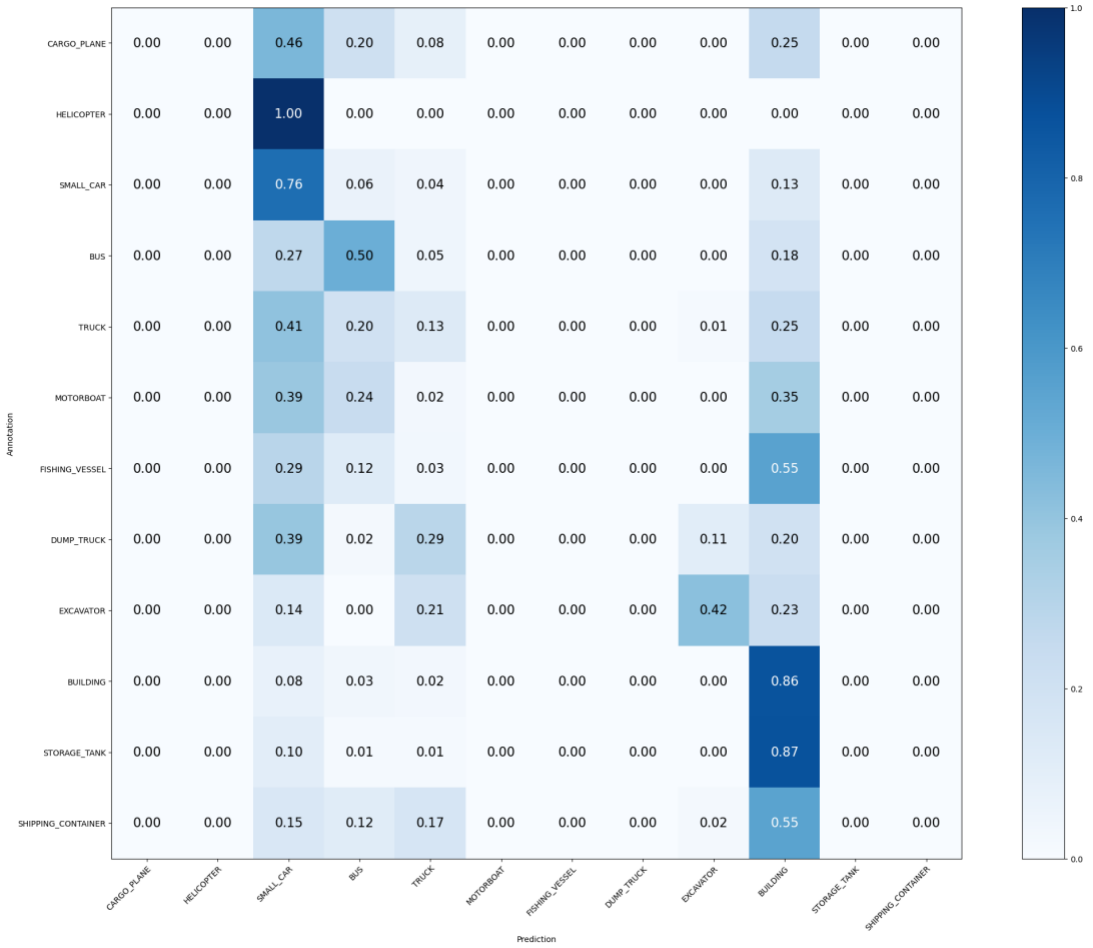

Figure 3: Confusion Matrix for Model 1

2.2. Second Results

For this part, we were finally able to use convolutional neural networks (CNNs). This was relieving since trying to achieve decent results merely using dense layers was as challenging as inefficient. Now we would be able to explore the full potential of the dataset by implementing our own model based on convolutional networks, which are way more effective at the process of extracting features from the data and retrieve the necessary patterns and information to effectively train a proper classifier.

As a result, the model employed in this part was a deep convolutional neural network (CNN) with several layers. It began with 6 Conv2D layers, each with different numbers of filters (32, 64, 128, 256, 512, and 1024) and a kernel size of (3,3), being He Normal the kernel initializer used, which is particularly suitable for networks with ReLU or GELU activation functions. This initialization method ensures that the weights have a proper scale, which can help prevent vanishing or exploding gradients, leading to a more efficient training and better convergence. Regarding the convolutional layers, they were designed to detect and extract features from the input images. Following each Conv2D layer, a MaxPooling2D layer was applied with a pool size of (2,2) to reduce the spatial dimensions, which helps in reducing the computational cost and controlling overfitting. Batch Normalization was employed after each pooling layer to stabilize and accelerate the training process by normalizing the inputs. The activation function used throughout the model is GeLU, which is known to work well in deep networks.

To prevent overfitting and improve generalization, we implemented L1 and L2 regularization techniques in the 4 Dense layers and the last 2 Conv2D layers. Additionally, Dropout layers with varying rates (0.3 and 0.4) were incorporated to randomly drop some of the neurons during training, thereby reducing the reliance on any single neuron and promoting a more robust model.

Finally, the last Dense layer had as many neurons as categories existed. The SoftMax activation function was then applied to output probabilities for each class. As previously, the optimization algorithm used was the Adam optimizer.

Considering all this, this new model had the following parameters:

- Total params: 6,990,412
- Trainable params: 6,984,588
- Non-trainable params: 5,824

Moreover, the training process reached 100 epochs, with the best validation accuracy of 0.8151 achieved at epoch 89 and the best accuracy of 0.9985 achieved at epoch 100. Next, below it can be seen the model's performance on the train and validation:

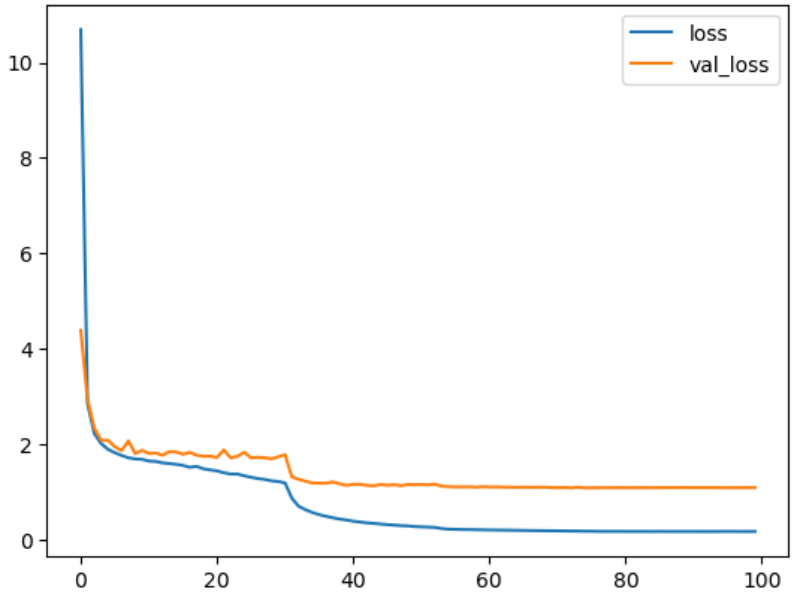

Figure 4: Loss vs Validation Loss for Model 2

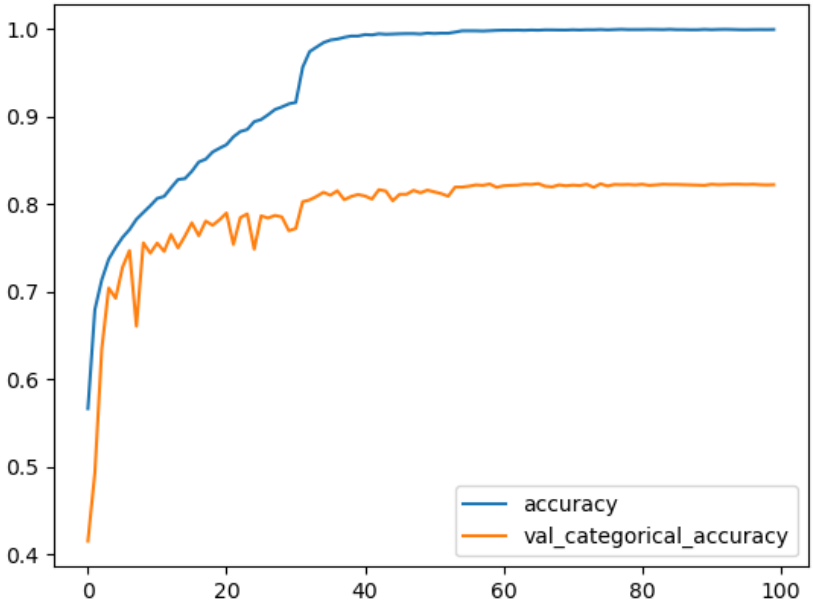

Figure 5: Accuracy vs Validation Accuracy for Model 2

With respect to the test dataset results, in this case they indicated varying performance across different object categories. With a mean accuracy of 78.731%, the model correctly classified most instances. However, the mean recall and precision values of 52.738% and 69.430% respectively, suggest that, again, there is room for improvement.

Looking into the class-specific metrics, some categories like “Cargo Plane”, “Small Car”, “Excavator”, and “Building”, showed strong performance with high recall and precision values. For instance, “Cargo Plane” had a recall of 80.723% and a precision

of 93.056%, indicating that the model is highly capable of identifying and correctly classifying cargo planes. The “Building” category had the highest recall (96.119%) and a high precision (89.035%), demonstrating the model’s robustness in detecting and classifying buildings.

However, the model struggled greatly with certain categories, like: “Helicopter”, “Fishing Vessel”, “Dump Truck”, and “Storage Tank”, which exhibit lower recall values. The model completely failed to identify any “Helicopter” instance, as indicated by the recall and precision of 0%, which again, did not surprise us since “Helicopter” has an appearance frequency of 0.17%. In contrast, the model has a high precision for “Storage Tank” (93.902%) but a low recall (53.472%), suggesting that it can correctly classify storage tanks when detected but often misses them.

The specificity scores are generally high for most categories, indicating that the model rarely misclassifies other objects as the given category. The Dice coefficients also vary across categories, reflecting the discrepancies in recall and precision.

In summary, while the model performs well for some categories, it struggles with others.

Finally, it can be seen below the confusion matrix for this second part:

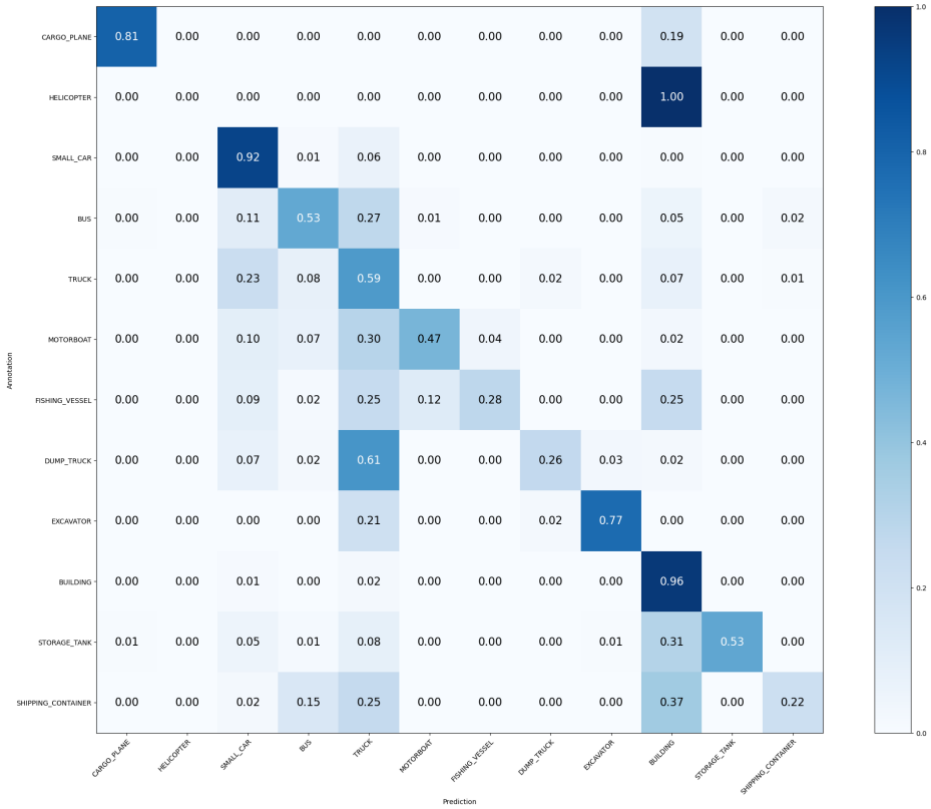

Figure 6: Confusion Matrix for Model 2

2.3. Third Results

For this phase, we were able to implement transfer learning, which we thought would be very interesting since now we would be able to not only use an already implemented network, such as the EfficientNet, but also, we were able to use pretrained weights for our model. This new scenario left great possibilities for improvement, considering the incredible enhancement in performance we would see by loading pretrained weights from giant databases such as “imagenet”.

Consequently, we employed the EfficientNetB0 model as the base architecture. EfficientNetB0 is a pre-trained CNN that has shown excellent performance in various computer vision tasks. Furthermore, to fine-tune the model’s architecture, we added a GlobalAveragePooling2D layer after the base model’s output to reduce the spatial dimensions of the feature maps. This is later followed by a Dense layer with 12 output units, each corresponding to one of the target categories. Then, we used the SoftMax activation function in the final Dense layer to output probabilities for each category. Also, to prevent overfitting and improve the model’s generalization, we utilized L1 and L2 regularization techniques in the Dense layer, which added penalties to the loss function proportional to the magnitude of the weights.

Regarding the model’s training, we used the Adam optimizer with a learning rate of 0.001, which is an adaptive learning rate optimization algorithm combining the advantages of both AdaGrad and RMSProp. The model was compiled with a categorical cross-entropy loss function, which is suitable for multi-class classification problems, and a categorical accuracy metric to monitor the model’s performance during training.

Finally, we used data augmentation to increase the diversity and size of the training dataset, which in turn enhances the model’s performance and generalization capabilities. By applying various transformations to the original images, such as rotations, flips and zooms, we generated new training examples that mimic the variations that might be encountered in real-world scenarios. Utilizing data augmentation helps the model become more robust to these variations, as it exposes the model to a broader range of input data during training. This increased diversity reduces the risk of overfitting, as the model learns to identify the essential features of each object category across different conditions rather than memorizing specific examples.

In our case, we incorporated data augmentation into the training pipeline using Keras’ ImageDataGenerator. This function allows for real-time augmentation, meaning that new transformed images are generated on-the-fly during training, reducing the memory requirements and computational cost. By employing data augmentation, we effectively increased the size and variety of the training dataset, leading to a more robust and accurate model capable of handling a wide range of input images with minimal performance degradation.

Considering all this, the third model had the following parameters:

- Total params: 4,064,943
- Trainable params: 4,022,920
- Non-trainable params: 42,023

Moreover, the training process reached 10 epochs, with the best validation accuracy of 0.8544 and the best accuracy of 0.8992 both achieved at epoch 10. Next, below it can be seen the model's performance on the train and validation:

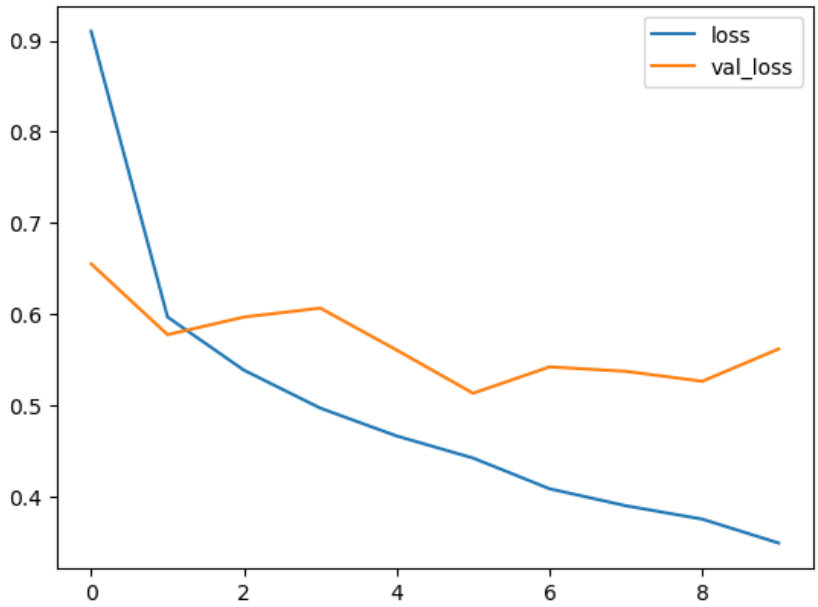

Figure 7: Loss vs Validation Loss for Model 3

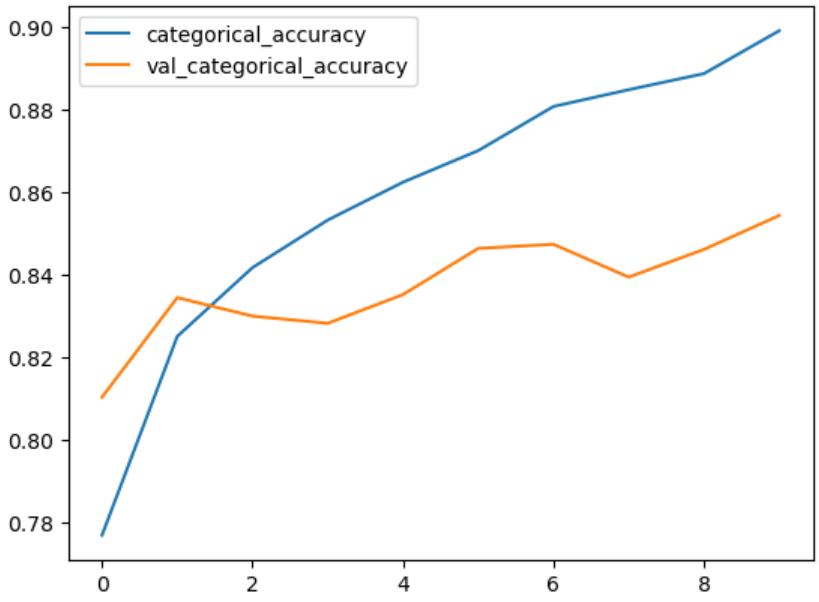

Figure 8: Accuracy vs Validation Accuracy for Model 3

With respect to the test dataset, finally we could say that we started to get great results: they demonstrated the model's excellent performance across various object categories. In this sense, the model achieved a mean accuracy of 90.639%, a mean recall of 87.605%, and a mean precision of 91.359%. These metrics indicate that the model can correctly identify and classify objects with high accuracy, while minimizing false positives and false negatives.

Examining individual object categories, the model performed exceptionally well for some classes, such as "Cargo Plane" (Dice: 98.491%), "Helicopter" (Dice: 96.296%), and "Building" (Dice: 98.220%). These high scores for recall, precision, specificity, and Dice coefficient suggested that the model is proficient at detecting and classifying these objects with minimal errors.

However, for some categories like "Truck" (Dice: 75.382%), "Dump Truck" (Dice: 78.413%), and "Shipping Container" (Dice: 82.127%), the model's performance was relatively lower. The recall scores for these categories (65.164%, 66.877%, and 91.789%, respectively) indicated that the model could improve slightly in identifying these objects. Despite this, the high precision scores (89.398%, 94.757%, and 74.306%, respectively) demonstrated that when the model does detect these objects, it is most likely correct.

Overall, the model exhibited strong performance on the test dataset, showcasing its ability to generalize well to unseen data. However, the varying results across different object categories indicated that further fine-tuning and optimization may be required to improve the model's performance, if that was possible, especially for categories where the recall scores are relatively lower.

Finally, it can be observed below the confusion matrix for this part:

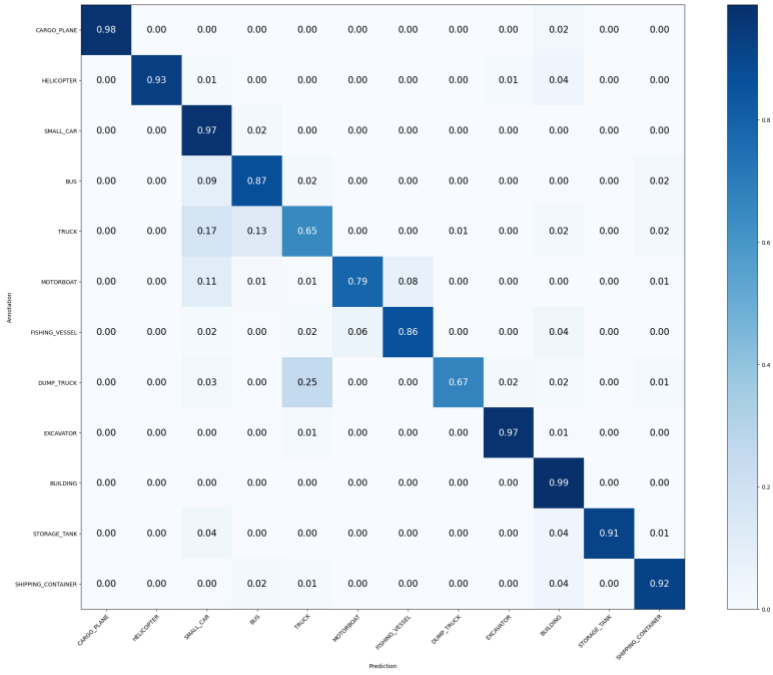

Figure 9: Confusion Matrix for Model 3

2.4. Final Results

As for our final model, we believed we already had a powerful classifier, being able to classify mostly every class correctly and precisely. Therefore, our main aim in this last part was to reduce the slight overfitting the model was presenting.

To do this, we decided to implement a fair level of Dropout in the model, by adding two more layers, one right before the global average pooling layer and another right after it. We chose to go with a level of 0.4 for both Dropout layers, since we thought it would be the best value to keep a balance between dropping the neurons drastically and dropping the neurons in a conservative manner.

As always, we used a training process of 30 epochs in this case, with the best validation accuracy of 0.8776 achieved at epoch 25 and the best accuracy of 0.9433 achieved at epoch 30. Next, it can be seen the model's performance on the train and validation:

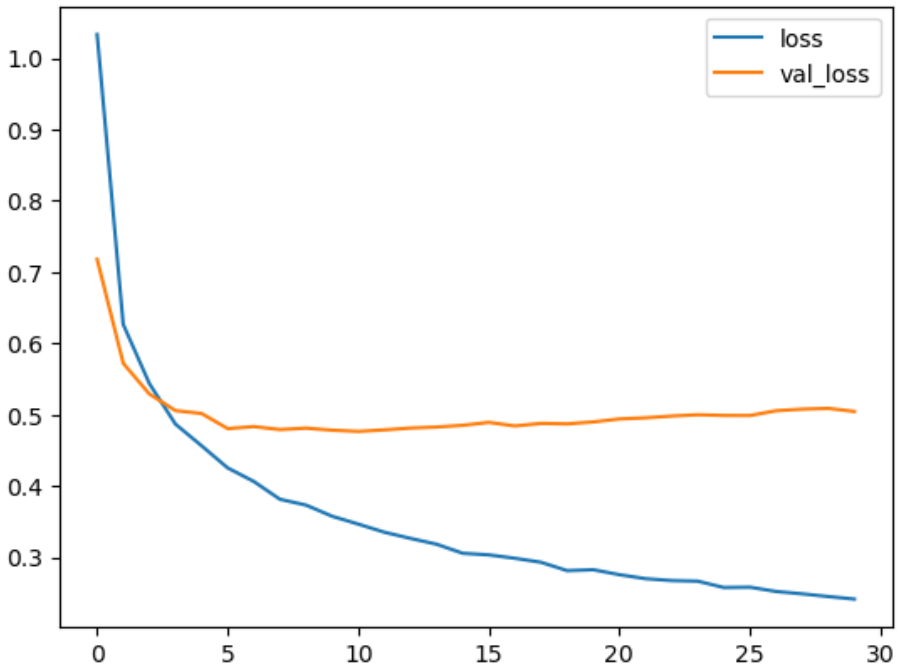

Figure 10: Loss vs Validation Loss for the last model

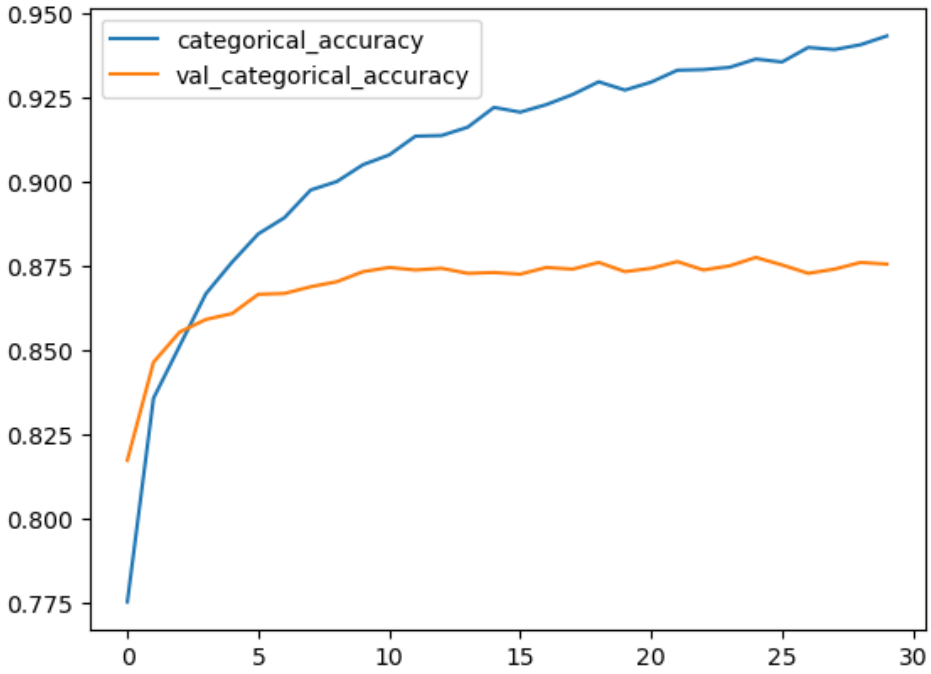

Figure 11: Accuracy vs Validation Accuracy for the last model

Given these results, we were able to appreciate an improvement in the way the model fits the data, as the mean accuracy, recall and precision obtained raised to a 96.342%, 94.530% and 96.220% respectively. Thanks to the dropout added both before and after the pooling layer, we could also see an improvement in the overall overfitting of the model, decreasing it thanks to the applied regularization. In this way, categories like "Truck", "Motorboat" and "Dump Truck" saw a substantiable improvement, going from 0.65, 0.79 and 0.67 to 0.91, 0.93 and 0.90 respectively.

As a result, we can conclude that we created a model which performed its classification tasks in a satisfactory manner while also being able to generalize well to unseen data.

Finally, below we can appreciate the confusion matrix for this last model:

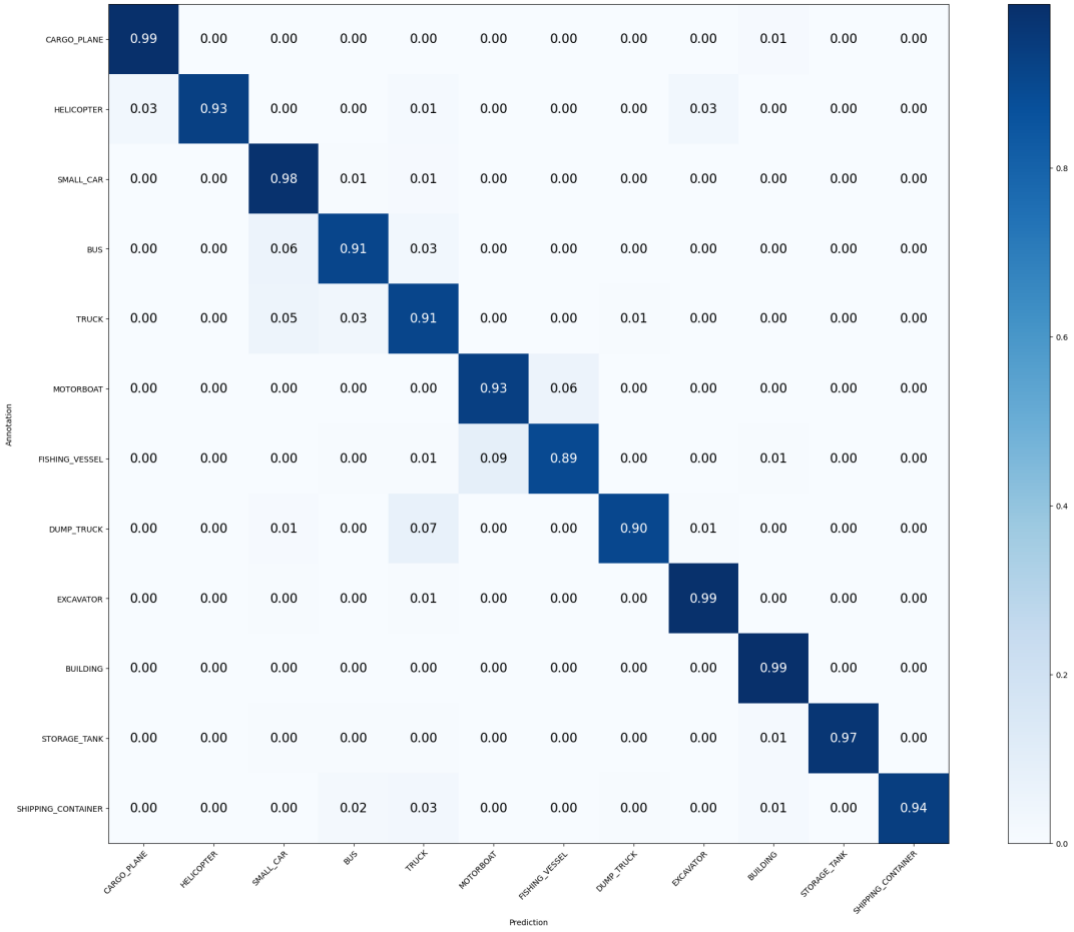

Figure 12: Confusion Matrix for the last model

3. Conclusions

Whilst doing this assignment, we learnt how to work with neural networks in a much more profound way, understanding its core foundations and applying the principles of the technology in the most efficient way possible given our background. Many obstacles have had to be climbed, and many lessons have been learnt. One of the reasons we believe this assignment has been so enriching and impactful is the way of how it has been structured: with the optional deliveries and the gradual use of the different technologies. This has set an ideal environment, which has led to a progressive and continuous learning process, giving us the possibility to keep learning in a proficient way, accumulating knowledge, and experience. Thanks to how all this has played out, we were able to understand the different aspects of the problem at hands, and consequently, identify what we had to improve to obtain better overall results.

Nonetheless, as said before, this has not been but without obstacles. Meaning how challenging it was to understand the preprocessing of the data and how to fine tune this process to substantially improve the performance of the script, which ultimately was the reason why we obtained such good results, permitting us to increase the number of epochs and batch size to make use of the brute force of the GPUs. Before, the generator feeding the data to the model was so slow, that the GPU was not able to use its full potential, causing a bottleneck in the data preprocessing and feeding process that was simply unacceptable. After optimizing this process, we were able to train each epoch in significantly less time, which greatly helped us at the time of testing new architecture variations and how they behaved in different contexts, such as different epochs, learning rates and the rest of the hyperparameters. Furthermore, we had to get familiar with several concepts and technologies which we had not touched in depth before, such as transfer learning and data augmentation for instance. Although these aspects were not as challenging, thanks to the support of the teaching material and the professors, who greatly succeeded in presenting this material and making us understand the fundamentals.

It would be imperative to also cover the CESVIMA supercomputer and the utility it has brought to the assignment completion. In this case, we did not see necessary to use it since we already possessed powerful GPUs and enough memory to fit the different models tested. However, we decided to give it a try since it would offer another machine in which to concurrently test more models, apart from our personal computers. In this case, the experience happened to be far from pleasant, something understandable given the number of users trying to access the clusters at the same time. We found it quite difficult to properly set up and the ques to access execution were outstandingly long. Nevertheless, we did not find this a troubling issue thanks to the fact that we had our personal computers, as said before.

This assignment has been extremely enlightening and enrichening, especially because of the way it has been conceived by the subject coordinators, it has fulfilled the objective of teaching us the foundations of deep learning and the many aspects that surround it. Thanks to it, we have been able to expand our horizons, and we were greatly motivated to keep pursuing a career in this discipline, giving us in fact, several business ideas that we aim to combine and implement in the ActúaUPM program.