BENCHMARKING RESNET MODELS ON THE AERIAL IMAGE DATASET AND THE VALUE OF DATASET-SPECIFIC STANDARDIZATION

BENEDIKT REIN
MATRICULATION NUMBER: 565136

July 17, 2023

1 Introduction

The utilization of Artificial Intelligence (AI) techniques in the domain of Remote Sensing has witnessed substantial advancements in recent years. Developments in the field of medical image recognition have contributed to the notable improvement of aerial image recognition (AIR). AIR focuses on developing algorithms to analyse and interpret aerial imagery. It involves a wide range of tasks, including scene classification, object detection, change detection, and semantic segmentation, with the goal of extracting meaningful information from aerial images. Use cases of aerial image recognition include Land Cover Mapping, Object Detection or Semantic Segmentation.

The goal of this research is to apply taught principles of the course 'AI4RS' to a different benchmark dataset and evaluate the performance of one or multiple selected deep-learning algorithms. In this research, we intend to evaluate the performance of two ResNet models and put the performance in perspective of the needed training time and resources. Additionally, we are interested in the value of dataset-specific normalization. As discussed in the official PyTorch forum by high-level PyTorch developers, the generic solution is to reuse ImageNet dataset specific values for normalization [1]. A key argument for generic normalization is, that most models have been pre-trained with those values. Other practitioners, however, argue for the value of dataset-specific normalization. We intend to benchmark both methods on the Aerial Image Dataset[2], which is especially interesting, because its images are taken from Google Earth, thus having images from different sensors, different parts of the world and most likely also different seasons and times of day. Making it one of the more realistic benchmarks for what can be expected in a real-world application of the algorithms.

2 Related Work

In the realm of aerial image classification, previous literature has extensively explored the application of pre-trained ResNet models for scene classification[3]. Researchers have recognized the effectiveness of transfer learning, utilizing deep neural networks pre-trained on large-scale datasets, such as ImageNet, to tackle the challenges associated with limited labelled aerial data. Many studies showcase the potential of pre-trained ResNet models for scene classification in aerial imagery, highlighting the ability to achieve high accuracy levels even with limited training data. However, there remains a need to further investigate and explore the applicability of such models in the context of specific aerial image classification tasks, considering the unique characteristics and challenges posed by remote sensing data.

Methods	UC-Merced (50%)	UC-Merced (80%)	Our dataset (20%)	Our dataset (50%)
CaffeNet	93.98±0.67	95.02±0.81	86.86±0.47	89.53±0.31
VGG-VD-16	94.14±0.69	95.21±1.20	86.59±0.29	89.64±0.36
GoogLeNet	92.70±0.60	94.31±0.89	83.44±0.40	86.39±0.55

Figure 1: Performance of high-level models on multiple datasets [2]

For the dataset, multiple benchmarks have been published already. The paper introducing the Aerial Image Dataset implements multiple low-level, mid-level and high-level algorithms [2]. The best performing models are CaffeeNet, VGG-VD-16 and GoogleLeNet, which achieve an accuracy of 89.5%, 89.6%, and 86.4% respectively. It should be noted, that for this performance only 50% of the dataset has been used for the training phase, while our implementation is using 80%. Some more recent implementations can be found on paperswithcode.com, the top 5 models achieve an accuracy between 98% and 98.5% with 50% training data [4].

The value of different normalization techniques has been discussed in many papers. Pal et al. (2016) are able to show the added value of standardization over other techniques in CNNs, with standardization just being outperformed by Zero Component Analysis (ZCA) in some cases[5].

3 Research Objective

As every ResNet model's size increase also brings higher computational demand with it, we are especially interested in exploring what accuracy gains can be expected with a comparable training load. While smaller models usually have a significantly lower demand for training data, they also need fewer computational resources for inference in a production environment. The selected models will be benchmarked on a scene classification task.

For this research, we are interested in the potential difference in training time and the difference in the final performance of ResNet-18 and ResNet-34. After deciding on appropriate training parameters, both models will be compared with different normalization implementations. As there has been a discussion on the PyTorch forum[1], and entries about this topic seem to have been removed from the documentation, we are going to evaluate the difference in performance when using dataset-specific standard deviation and mean values for the RGB band normalization or using generic values, usually taken from the ImageNet dataset. We are interested, if a more accurate transformation has a positive impact on accuracy convergence or if it has a negative impact, especially with the assumption, that the ResNet models have been pretrained with ImageNet normalization values.

3.1 Dataset Description

The Aerial Image Dataset (AID) [2] is a large-scale aerial image dataset collected from Google Earth imagery. It consists of 30 aerial scene types, including airport, beach, forest, and residential areas, among others. Examples of all classes can be seen in Figure 2. The dataset contains 10,000 images labelled by remote sensing specialists, showcasing various regions and seasons. The provided bands are RGB and the images have a native resolution of 600×600 pixels. Despite being pre-processed, the Google Earth images exhibit similar characteristics to real optical aerial images, making them suitable for evaluating scene classification algorithms. AID presents the challenge of multi-source imagery from different remote imaging sensors, providing greater complexity compared to single-source datasets. The dataset is divided into an 80:10:10 split for training, validation and finally testing.

The different sensors, locations and potentially pre-processing, make the AID one of the more realistic datasets. Evaluating algorithms on this benchmark gives a realistic estimate, on how those algorithms could perform on data that is easy to attain, without investing a lot of time into manual evaluation and image selection.

3.1.1 Data Preparation

We can take multiple steps to ensure consistent data input into our model, to make sure it can learn from all the input and not have any training data, that is skewing the model weights. Additionally, we can use the data to create synthetic input, so the model can learn from even more diverse data. This way, we can ensure, that the model is getting a big variety of inputs. Typical steps would be rotating, flipping, and cropping the images or inserting some colour jitter.

For normalization, firstly, we reuse generic values provided in the course 'AI4RS', which are taken from the ImageNet dataset. Secondly, we need to calculate mean and standard deviation values of the AID for the red, green, and blue bands, provided in the train dataset. As mentioned in the research objective, both implementations will be evaluated.

The mean values calculated on the Aerial Image Dataset are 0.5502 for the red band, 0.5468 for the green band and 0.5135 for the blue one. Respective standard deviations are: 0.1665, 0.1651, and 0.1737. Values taken from the ImageNet dataset are: 0.485, 0.456, 0.406 and 0.229, 0.224, 0.225, respectively. The selected formula, also provided by PyTorch [6], is the z-transformation, which is defined as:

$$output[channel] = (input[channel] - mean[channel]) / std[channel]$$

This form of normalization would be called standardization, however this is just true when using dataset specific values. Thus, we will keep the name normalization[5]. Normalization

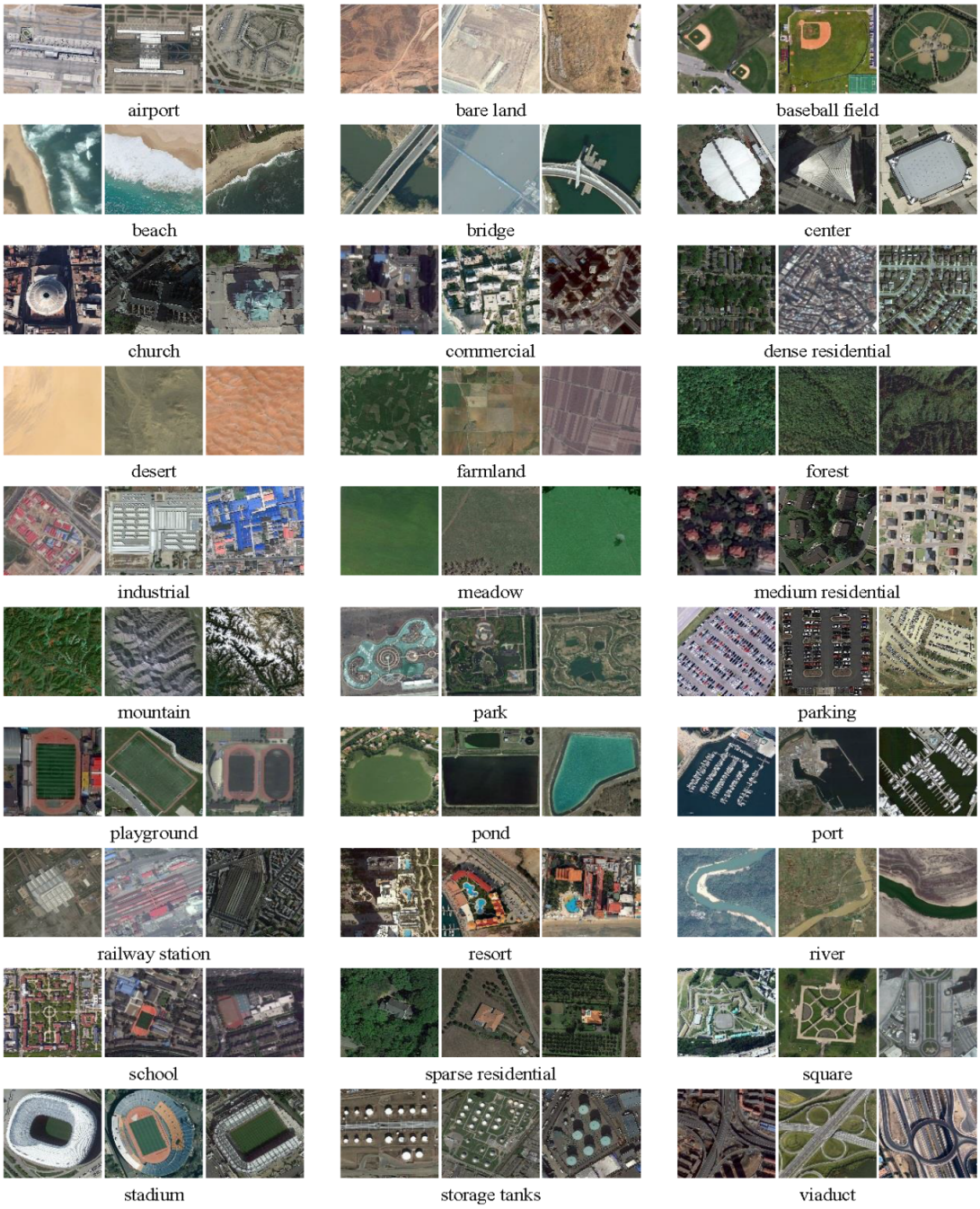

Figure 2: Examples for all dataset classes[2]

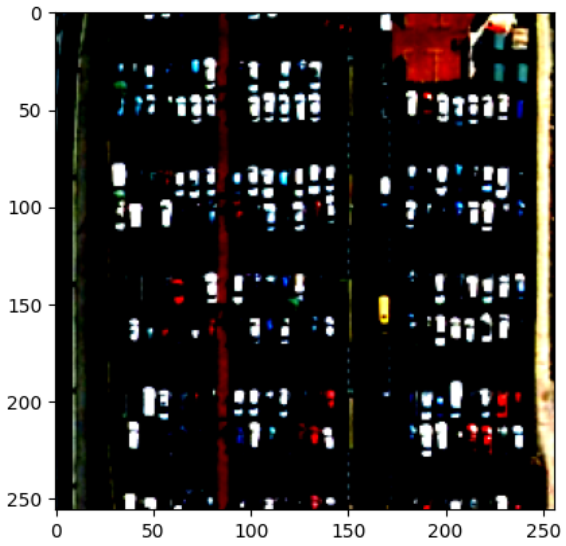

Figure 3: Example for a normalized image for *Parking*

steps are taken for the training, validation, and test datasets. The augmentation steps are only implemented for the training dataset. This is according to general deep-learning best practices and not image recognition specific. When looking at some example images after transformation, most images are still high quality, contrasted, and easy to recognize with the human eye. However, some images are very dark and are missing some higher contrast to distinguish transitions more clearly. This leads to the assumption, that the differences between some images are quite high, leading to mostly improvements but also degradation of the clarity of some images. A possible solution for this would be putting images in their respective sensor, region, or target category and do category-specific calculation of mean and standard deviation. This step would, however, have to be taken in the data collection phase.

3.2 Model Description

ResNet in general is a Convolutional Neural Network (CNN) architecture that was introduced by Microsoft in 2015 [3]. It was designed to address the problem of vanishing gradients in very deep neural networks. The initial model has 18 layers, more recent models are scaling up to 152 or even more than 1000 layers. The key innovation is the introduction of residual connections or skip-connections. They enable the network to bypass one or multiple layers, allowing the information to flow directly from one layer to another without undergoing transformations. This helps in retaining and propagating gradients effectively, facilitating the training of deeper networks. The architecture consists of several repeated building blocks, each containing multiple

convolutional layers, batch normalization and ReLU activation functions. The number of filters in each stage gradually increases. The output layers consist of a global average pooling layer that reduces the dimensions of the features to a fixed size. Finally, a fully connected layer and Softmax activation function are used to predict probabilities for each class. Input and output layers are matched to the provided data. The output size is 30, matching the different target classes provided in the dataset. We are using categorical cross-entropy as a loss function.

3.3 Experimental Setup

3.3.1 Hardware

Due to limited access to GPUs, and the high capability of pre-trained image recognition models, the initial setup is run on a CPU. The final model is trained on an *AMD Ryzen 7 PRO 4750U CPU*. For now, multithreading is not implemented.

3.3.2 Model selection

We are using pre-trained ResNet-18 and ResNet-34 models, provided through PyTorch. They are initialized with the provided weights and trained with the previously described dataset. Due to the mentioned computational limitations, ResNet-18 is used as an initial baseline, for further comparison, we are also running the ResNet-34. The predictive performance of those models will be compared, based on different normalization methods.

3.3.3 Training

As an initial baseline, we are running the model with a 256×256 resolution and no specified hyperparameters. After a first evaluation of the models' performance, we decide to keep this resolution, as a 600×600 resolution increases the training time 5-fold, without giving meaningful improvements when looking at the evaluation metrics. Every epoch using the training dataset takes around 15 minutes, making fine-tuning on a laptop CPU feasible. The model will get the chance to look at the full training data 25 times, taking around 8 hours. The ResNet-34 model takes slightly less than 30 minutes for one epoch, making the final runtime for 20 epochs around 10 hours and comparable with the training time of the smaller model. Additionally, we are implementing a learning rate scheduler, which decays the learning rate by a factor of 0.1, if the validation accuracy has not increased for 2 epochs.

Table 1: ResNet-18 category-specific results on test dataset with specific normalization

Numeric Category	Category	F1-Score	Accuracy	Precission	Recall
0	Airport	0.96	0.97	0.95	0.97
1	BareLand	0.92	0.9	0.93	0.9
2	BaseballField	1	1	1	1
3	Beach	1	1	1	1
4	Bridge	0.99	0.97	1	0.97
5	Center	0.86	0.92	0.8	0.92
6	Church	0.96	0.96	0.96	0.96
7	Commercial	0.9	1	0.81	1
8	DenseResidential	0.95	0.98	0.93	0.98
9	Desert	0.9	0.9	0.9	0.9
10	Farmland	1	1	1	1
11	Forest	1	1	1	1
12	Industrial	0.95	0.97	0.93	0.97
13	Meadow	1	1	1	1
14	MediumResidential	1	1	1	1
15	Mountain	0.99	0.97	1	0.97
16	Park	0.9	0.86	0.94	0.86
17	Parking	0.99	0.97	1	0.97
18	Playground	0.99	1	0.97	1
19	Pond	0.96	0.95	0.98	0.95
20	Port	0.99	0.97	1	0.97
21	RailwayStation	0.94	0.96	0.93	0.96
22	Resort	0.83	0.83	0.83	0.83
23	River	0.99	1	0.98	1
24	School	0.83	0.73	0.96	0.73
25	SparseResidential	0.98	0.97	1	0.97
26	Square	0.8	0.79	0.81	0.79
27	Stadium	0.93	0.86	1	0.86
28	StorageTanks	0.93	0.94	0.92	0.94
29	Viaduct	0.99	1	0.98	1

3.3.4 Hyperparameter optimization

We decided against doing full hyperparameter optimization runs. Because we are more interested in the value of the dataset specific normalization, the intention is to keep as many parameters the same as possible, to make sure our metrics are still comparable.

One crucial parameter, that cannot be ignored, is the learning rate. Especially when working with pre-trained models. From our experience, small learning rates tend to be the optimal choice. However, for multiple exploratory training runs, we still selected a learning rate, that was too high, which in turn degraded the models parameters and made the training start off with an accuracy of around 8% after epoch 1, reaching a final performance of around 50% accuracy, even when decaying the learning rate significantly. This was an interesting learning, how easy it can be to remove a lot of pre-trained knowledge from a model when using unsuitable parameters. After some more trial runs, starting at a learning rate of 0.005, we finally decided on an optimal learning rate of 0.0005.

4 Experimental Results

First off, we are able to show, that all models did learn meaningful information from the selected dataset. Our results are not fully comparable with results provided in other benchmarks, because we did use a different ratio of training and testing data. However, the results are in a comparable ballpark, thus leading to the first assumption, that no major mistakes or target leakage are present. Another interesting takeaway is, that even an 'old' ResNet-18 can perform on par with state-of-the-art models when being trained on 80% instead of 50% of the data.

Now, we will first have a look at the class-specific results from the ResNet-18 model with dataset-specific normalization in Table 1. You can see, that some categories have been predicted with an accuracy of 100%. However, some others cannot be predicted as accurately by our model. The classes with the worst performance are: *Resort, School, Square, Center, Park, and Stadium*. On the other side, the classes with an accuracy of 100% are: *Baseball, Beach, Farmland, Forest, Meadow, and MediumResidential*. Exemplary images can be checked in the notebook provided. When looking at the overall performance of all the models, only small differences can be observed. This is to be expected, because the pre-trained models are already performing on a high level and even the best ones are just able to increase the accuracy by less than one percent point. With 25% higher training time and 20% less epochs, the ResNet-34 models are capable to achieve similar accuracy as the ResNet-18 models. However, the ResNet-34 model with generic normalization performs the worst overall. With limited training time, the ResNet-34 models do

Table 2: Trials and their respective performance on the test set

Trial	Overall Accuracy	F1-Score	Accuracy	Precission	Recall
ResNet-34 generic	94.2	93.86	93.97	94.26	93.97
ResNet-34 z-score	94.7	94.56	94.55	94.91	94.55
ResNet-18 generic	94.7	94.41	94.35	94.74	94.35
ResNet-18 z-score	94.9	94.67	94.64	94.97	94.64

not manage to outperform the ResNet-18 models significantly. When excluding the ResNet-34 model with generic normalization, variance in all performance metrics are a maximum of 0.5 percent points. Because the experimental trials are missing a consistent random state, those results might even just be due to a differently selected test subset. Especially, because this trial was run at a different time, which included reloading the whole script.

When just comparing for the impact of the different normalization methods, the smaller model is 0.2 percent points better with z-score transformation, the bigger model is 0.5 percent points better on overall accuracy. Comparable results for F1-score, precision and recall can be found in Table 2. In the scope of pre-trained models, those improvements can be seen as significant, leading to the conclusion, that dataset specific normalization can be a key driver for reaching some final, small improvements in predictive performance. As discussed in the PyTorch forum [1] this value-add is very dependent on the difference between the datasets. Using the generic normalization on medical images, would most likely lead to sub par performance.

Finally, we can have a look at the confusion matrixes provided in Figure 4 and 5. The corresponding classes to the numeric labels are provided in Table 1. All matrixes give us a fast and comprehensive idea of the predictive performance of our models. On the left we have the models with dataset-specific normalization, on the right we can see the performances with generic normalization. Because the overall performance of all models is quite good, it is difficult to get insightful information from this visualization. We can see a clear overlap of predicted labels and true labels. For the ResNet-18 models, the most common mistake (three times) is wrongly predicting *Commercial (7)* to be *School (24)*. The ResNet-34 model is predicting *Resort (22)* to be a *Park (16)* 5 times, the generic normalization is just doing 4 misclassifications. When looking at those categories, even classification by a human seems to be difficult.

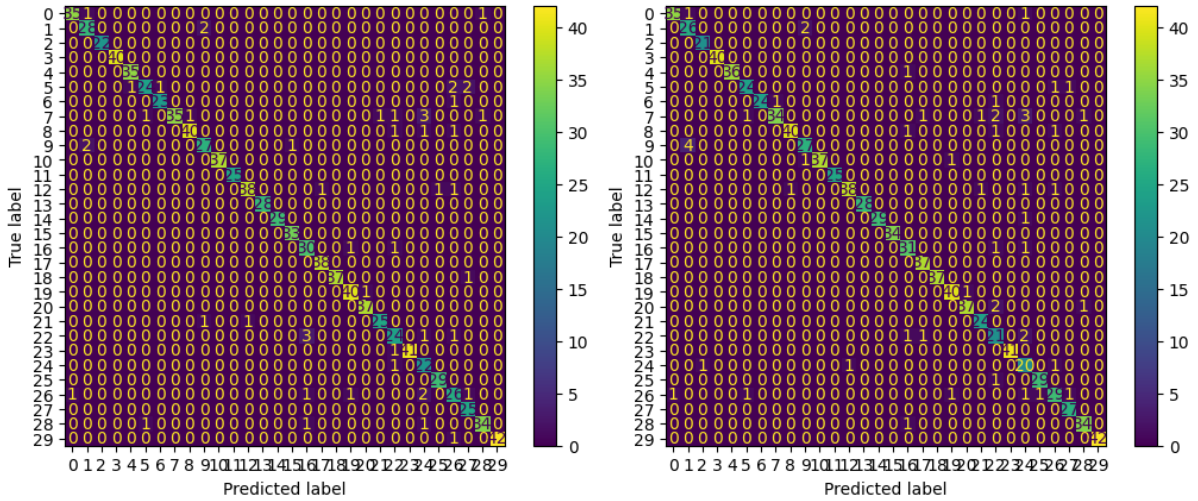

Figure 4: Confusion matrix for best ResNet-18 models on test dataset with left: dataset specific and right: generic normalization

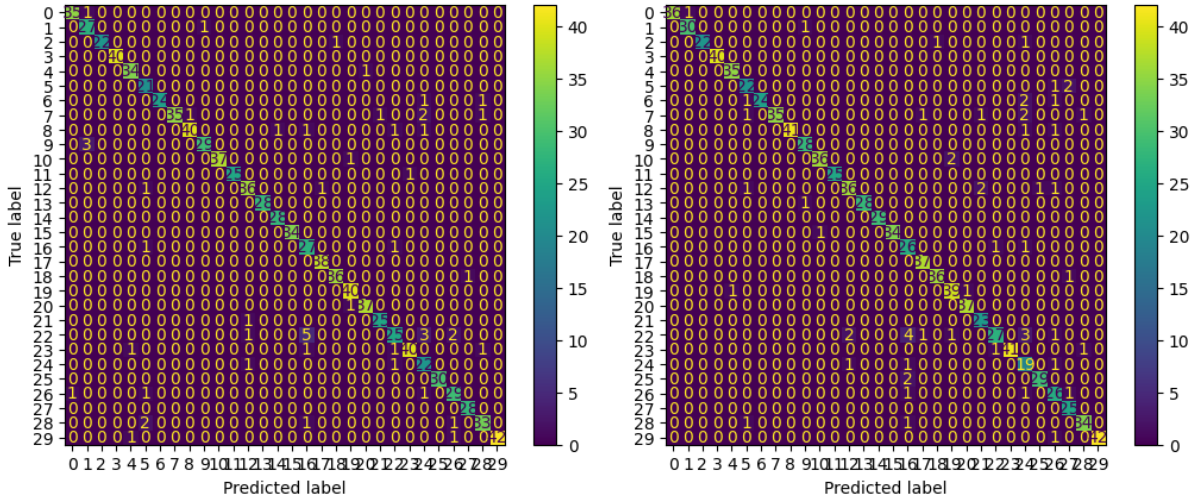

Figure 5: Confusion matrix for best ResNet-34 models on test dataset with left: dataset specific and right: generic normalization

5 Conclusion

In conclusion, pre-trained models have demonstrated their remarkable capabilities, emphasizing the importance of employing very small learning rates or freezing certain layers during training. Moreover, dataset-specific normalization has proven to be superior to more unspecific normalization methods, even when working with more advanced models. The performance of fine-tuned models on specific tasks has reached unprecedented levels, allowing for outstanding performance within hours of training on a standard CPU. Finally, the utilization of mean and standard deviation values could provide an interesting measure of differentiation among regions, times, and sensors. In the future, investigating these aspects further could yield intriguing insights in aerial image datasets.

For further improvement, this research would benefit from a predefined random state, to make all results even more comparable. Reusing train-test-ratios from previous research would also be a crucial step to improve the scientific value of this experiments. Additionally, a clear measure of convergence for our models performances would show even better scientific rigour. Even if we cannot see any meaningful improvements after a training accuracy of 95%, best practice would be to apply the same structural decision-making for all training runs.

References

[1] The PyTorch Foundation. [discussion] why normalise according to imagenet mean and std dev for transfer learning?, 2022. Available at: `https://discuss.pytorch.org/t/discussion-why-normalise-according-to-imagenet-mean-and-std-dev-for-transfer-learning/115670`.

[2] Gui-Song Xia, Jingwen Hu, Fan Hu, Baoguang Shi, Xiang Bai, Yanfei Zhong, Liangpei Zhang, and Xiaoqiang Lu. AID: A benchmark data set for performance evaluation of aerial scene classification. *IEEE Transactions on Geoscience and Remote Sensing*, 55(7):3965–3981, 2017.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 Computer Vision and Pattern Recognition*, pages 770–778. IEEE, 2016.

[4] Aerial scene classification on aid, 50% as trainset, 2023. Available at: `https://paperswithcode.com/sota/aerial-scene-classification-on-aid-50-as?tag_filter=0`.

[5] Kuntal Kumar Pal and K. S. Sudeep. Preprocessing for image classification by convolutional neural networks. In *2016 IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, pages 1778–1781, 2016.

[6] The PyTorch Foundation. Docs > transforming and augmenting images > normalize, 2022. Available at: `https://pytorch.org/vision/stable/generated/torchvision.transforms.Normalize.html`.