DEEP LEARNING (GRS-34806)

MGI PROJECT REPORT - GROUP 5

Liam Adam & Francesco Pasanisi

1. Introduction

Satellite imagery is nowadays used for a wide range of applications, ranging from environmental monitoring, disaster response, and land planning. Thanks to advancements in deep learning during the last two decades, the results obtained for object detection and image classification using convolutional neural networks (CNNs) are impressive. However, the application of deep learning on satellite imagery can be affected by some limitations that make it challenging to use it to train CNNs. For instance, Pritt and Chern (2017) mentioned that one reason for this difficulty is the lack of large labeled dataset to use in training algorithms. In fact, neural networks usually require thousands or millions of input images before they learn to recognize objects and classes. Until recent times, these datasets did not exist.

Yang and Shawn (2010) proposed a satellite imagery dataset called UC Merced (UCM) Land Use Dataset. This dataset was used in this project to train a CNN for image classification. Two distinct cases were considered: single and multi-label classification.
For the first, the original UCM dataset with 21 classes was used. This dataset is made of 100 images for each class, thus consisting of 2100 images in total. Each image size is 256x256 pixels, since they were extracted from larger images of the USGS National Map Urban Area Imagery collection.
For the multi-label scenario, a modification of the UCM dataset published by Chaudhuri et al. (2018) was used. The authors argued that remote sensing images might have more than one land use, so multi-labels are needed for each image. They proposed a semi-supervised graph-theoretic method to retrieve multi-labeled images from the UCM dataset. Compared to the original dataset, the multi-label one includes 17 different classes. Table 1 shows the classes of the multi-label dataset, together with the distribution of each class.
In Section 2 the implementation of the models is explained in terms of architecture and data structure used to manage and load the dataset. In Section 3 the results are shown.

2. Methods

The main tool used for this project was PyTorch Lightning (PL) (W. Falcon, 2023), a high-level lightweight PyTorch wrapper intended to achieve higher performances and ease to use. There are several advantages of using PL instead of the conventional PyTorch: it is easy to install and allows writing more concise and cleaner code, since developers do not need to define functions for all the steps of the training loop of a neural network. For example, evaluation steps and training status visualization are very simple to implement with PL. Many steps are already wrapped in PL

classes and methods, and it is sufficient to change some arguments to tune the training loop. Thanks to PL, it was possible to create a standard template to be applied to different models, as well as a main training loop template. It allowed for a simple switch of the desired model for training, without the necessity for further modifications.

The UCM dataset was split into three subsets for training, validation and testing. The default proportions are 0.60, 0.2 and 0.2, but the user can customize it and try different splits. During training, validation and training loss are monitored. After training, a one-shot test prediction with the trained model was carried out to compute accuracy metrics and visualize some classification predictions.

2.1 Data Structure

A dataset class named UCMDataModule was defined as a subclass of the PL LightningDataModule, inheriting its structure. A PL data module is a class organized in a way to be reusable and to encapsulate all of the most important steps to process the input data. Thanks to this data module it was possible to wrap up all the training and validation steps, as well as data transformations and augmentations. This custom data structure is designed to manage the loading and processing of image datasets for a neural network. It allowed for easy and efficient data management, including multi-label classification and data augmentation.

2.2 Models

In this section, the implemented models are described. In the first subsection, the model template that was used to train all the other implemented models is explained. In the other subsections, details about the specific models are provided.

2.2.1 LitModel Template

The model template, named LitModel, was defined to take a model as an argument and use a general set of training, validation and test steps for the chosen model. Together with the model, also the number of classes, the learning rate, the multi-label boolean flag, and the tensor with the multi-label class weights were passed as arguments. LitModel was defined as a subclass of the PL LightningModule, so every model implemented inherits the features of this class.

For different classification scenarios, distinct loss functions were applied. For the single-label case, since it was necessary to only find one class with the highest probability for each image, we used the cross-entropy loss (L_{ce}):

$$L_{ce} = -\sum_{i}^{t_i} log(p_i), \text{ for n classes,}$$

where t_i is ground truth label and p_i is the Softmax probability for the i^{th} class.
However, this function would not work properly for the multi-label scenario, where more classes could be present in a single image. For this reason, we chose to use binary cross entropy with logit loss. This loss function combines binary cross entropy with a Sigmoid function in the same class. In a multi-label scenario, this function is described as:

$$\boldsymbol{\ell_c(x,y) = L_c = \{l_{1,c}, \dots, l_{N,c}\}, l_{n,c} = -w_{n,c}[p_c y_{n,c} \cdot log_\sigma(x_{n,c}) + (1 - y_{n,c}) \cdot log(1 - \sigma(x_{n,c}))],}$$

where c is the class index, *N* is the batch size n is the sample index in a batch and p_c is the weight of the positive answer for the class c. The tensor $\boldsymbol{W_{n,c}}$ was used to correct for the imbalance in class frequency, shown in table 1 in section 7 (Tables).

In the LitModel we also defined the classification metrics to use in the test step. For the single-label scenario, we only used the multiclass accuracy, defined as:

$$Accuracy = \frac{1}{N}\sum_{i}^{N} 1(y_i = \hat{y}_i)$$

where y is a tensor of target values, and $\hat{y}$ is a tensor of predictions.

For the multi-label scenario, we used a set of metrics:

- Multi-label accuracy, defined the same way as the multi-class accuracy.
- Recall, which quantifies the number of true positive (TP) predictions out of all the possible positive predictions that could have been made (TP and false negatives - FN). It is defined as:

$$Recall = \frac{TP}{TP + FN}$$

- Precision, which defines the quantity of positive observation correctly predicted. It is defined as:

$$Precision = \frac{TP}{TP + FP}$$

The choice of using these additional metrics for the multi-label scenario was crucial to correctly assess the classification performance of our model. In fact, in some situations a high accuracy can be achieved, while the recall and/or precision might be poor. This represents a relevant flag alert concerning the overall classification accuracy of a model.

In LitModel the optimization algorithm to implement was also defined. Specifically, the Stochastic Gradient Descent (SGD) was implemented. This optimization algorithm is widely used in deep learning due to its effectiveness and ease of implementation. Its main feature is the calculation of the gradient using a mini batch, leading to a reduction of computation time per iteration and faster

convergence, as also shown by Vaswani et al. (2019). The choice of optimization algorithm depends on the specific problem and dataset, but SGD usually represents a good starting point.

2.2.2 LeNet

As the first model, the simplest CNN considered during the course was implemented: LeNet-5, as proposed by LeCun et al. (1998). This simple architecture was chosen to familiarize us with the PL syntax and the defined data structure, serving as a starting point for the project experiments. The primary goal was to ensure the proper functioning of the training loop and data pipeline, without anticipating high accuracy from this model due to its simplicity in relation to the complex problem.
The original LeNet-5 was designed for grayscale images of size 28x28. For this reason, the UCM dataset was adapted to match this requirement. In particular, the input images were resized to 28x28, and center cropping was applied. The original architecture was then modified to accept RGB inputs (with 3 channels) instead of grayscale images.

2.2.3 AlexNet

The next step in the experimentation was to increase the model complexity to try achieving better accuracy on the UCM dataset. To this end, the architecture proposed by Krizhevsky et al. (2012), known as AlexNet, was implemented adapting to the PL syntax.
As with the previous model, it was necessary to resize the input images to fit the requirements of AlexNet, which accepts images of size 227x227x3. The decision to implement this architecture was driven by the need of trying a greater complexity in the architecture compared to LeNet-5. In fact, AlexNet consists of five convolutional layers, as opposed to LeNet's two, allowing it to extract and learn more meaningful features from the dataset.
Furthermore, AlexNet applies Rectified Linear Units (ReLU) as activation function, instead of the Sigmoid used in LeNet. This activation function offers several advantages, such as faster training times compared to other activation functions like sigmoid and tanh and mitigate the vanishing gradient problem that could take place in deep networks.
Obviously, it was expected to have a higher accuracy of AlexNet compared to LeNet, due to its more complex structure. However, it is essential to consider that AlexNet was originally designed to work on a dataset containing 1.2 million high-resolution images, whereas the UCM dataset is made of just 2100 images. The significant difference in dataset size makes it challenging to train the network from scratch, as it may not have enough training data to learn the patterns and features for an accurate classification.

2.2.4 PreTrainedAlexNet

Given the expected unremarkable accuracy of the AlexNet implementation from scratch, also a pre-trained version of this model was applied. This was done thanks to the PyTorch

implementation based on Krizhevsky (2014), and a model class called PreTrainedAlexNet was defined. The pre-trained model was used as a feature extractor, disabling the gradient calculation for this portion of the network to preserve the pretrained weights. Subsequently, a classifier was defined consisting of only a single fully connected (FC) layer, which allowed it to set the desired number of output classes (different from the original 1000 classes that AlexNet was trained on). By using this pretrained model, the goal was to benefit from the weights that had already been learned on a much larger dataset. Therefore, the network would only need to learn the weights of the single FC in the classifier, reducing training time.

2.2.5 TrimoNet

Lastly, a custom CNN architecture named TrimoNet was designed. During the designing process of this neural network, all the knowledge regarding image classification acquired during the course was considered to properly analyze the model structure.
This architecture is very similar to AlexNet in terms of number of layers: in fact, it is made of a feature extractor which involves five convolutional layers, but the classifier has only two FC layers. The two main relevant differences of TrimoNet compared to AlexNet are:

- The use of average pooling instead of max pooling to average adjacent pixel values and obtain images with better signal to noise ratio.
- The application of batch normalization (Ioffe and Szegedy, 2015). This data regularization technique allows to normalize each mini batch, making it possible to use a higher learning rate.

The network has in total 3.1 million trainable parameters. With this model the goal was to try balancing model complexity with the considered dataset. Since the UCM data set is small, using a deeper network with a higher number of trainable parameters would lead to decreased accuracy and lower ability for the network to learn the data. The architecture of TrimoNet is shown in Figure 2.

When applying this model, the images were all resized to 242x242. This was done because the original dataset should have been 256x256, but some images with different sizes were found when exploring the dataset at the beginning. For this reason, it was decided to resize to a common size and center cropped. This led to the loss of some information along the edges of the images, but it was crucial to generate a dataset with the same image size.

2.3 Training loop

The training loop was organized in a manner that the user can easily switch from the single to the multi-label classification. When training, due to the small size of the dataset, data augmentation was applied. This approach allows the model to rely less on specific characteristics of the train data, increasing its generalization ability. The data augmentation is carried out at each epoch, instead of inflating the original dataset. Abdelhack (2020) reported the most effective data

augmentation methods for satellite imagery classification. Following this paper results, these data augmentations were applied to the dataset:

- Random horizontal and vertical flip;
- Random affine with rotation and shear.

A relevant feature of our training loop is the PL Tuner class implemented to find the optimal learning rate for the specific model. This class executes some initial tests to find a good initial learning rate before the training starts.

3. Results and discussion

In this section the most important results are reported and discussed. In Section 7 (Tables) in table 2 a comparison between model accuracies is shown.

3.1 Single-label classification

3.1.1 LeNet

With LeNet-5 trained from scratch, results were predictably poor due to the model's limited capacity. The achieved test accuracy was 3.6%. The learning rate found by the tuner was 0.6309, and the training epochs were 20.
Clearly, this model architecture is too simple and was not able to learn from the dataset.
Despite this low classification performance, implementing and testing LeNet-5 was an important experiment to understand the data structure, the model template and overall workflow of the project. With a solid foundation in place, it was possible to implement different models and easily explore different architectures, as well as carrying out fine-tuning of hyperparameters.

3.1.2 AlexNet from scratch

AlexNet, trained from scratch, gave unsatisfactory results. The achieved test accuracy was 3.2%. The learning rate found by the tuner was 0.33 and the training epochs were again 20. Due to its results, also AlexNet showed to be too simple and not able to generalize well for the given dataset. It was noticed that when re-running AlexNet and LeNet, different test accuracies are obtained, but always very low. This is because the models' classification capability is poor on the UCM dataset, thus when they are tested the results are just random and there is no consistency.
This highlights the importance of exploring alternative techniques such as fine-tuning pretrained models to improve performance on dataset with limited samples.

3.1.3 PreTrainedAlexNet

As expected, PreTrainedAlexNet showed better results compared to the model trained from scratch. The achieved test accuracy was 75.6%. The model was trained again for 50 epochs and the optimal learning rate found by the tuner was 0.003.
This result pointed out how pretrained models represent a valuable resource when the amount of training data is limited. Even though this model was trained for a different classification task, on a different dataset and to classify a different number of classes, it was sufficient to just modify the classification head to exploit its pretrained weights. They have proven to be suitable to considerably increase the classification accuracy on the UCM dataset.
It is important to mention that the test loss calculated (shown in Table 2) for this model is referred only to the parameters of the classification head, since they are the only parameters that are trained.

3.1.4 TrimoNet

The result obtained with TrimoNet on the single-label scenario was better compared to AlexNet. In fact, the obtained test accuracy was 71.52%, and the best learning rate found by the tuner is 0.1. Differently from earlier, this model was trained for 100 epochs. The decision came from the fact that, monitoring the validation and training loss during training with fewer epochs, the model was still learning. Therefore, it was possible to train it for longer and make it learn more. The confusion matrix obtained for TrimoNet in the single-label scenario is shown in Figure 3: it is possible to notice that the ‘agriculture’ class was the easiest to classify, as 100% of the images are classified correctly. This is because the images of this class contain only agricultural fields, hence there is no presence of other objects that can confound the classification process. On the other hand, “tennis court” is the class with the poorest classification accuracy, since only 53% of the images of this class are classified correctly. This comes from the fact that tennis courts are usually surrounded by different objects, such as buildings, roads, residential areas, making it more challenging to classify this class correctly. Figure 4 shows a few predictions obtained with this model.
This result highlights the importance of batch normalization and average pooling on the learning capability of the model, which is now significantly better than AlexNet from scratch.

3.2 Multi-label classification

An important note before discussing the multi-label results is the difference in accuracy interpretation. As discussed in the methodology, the classes’ frequency distribution was significantly imbalanced. Thus, the accuracy was a poor indicator of model performance as the model could achieve high accuracy results by predicting the most frequent classes (pavement, trees, cars) for each image. It should be evaluated with respect to the model’s recall and precision. In Section 7 (Tables) in table 3 a comparison between model accuracies is shown.

3.2.1 LeNet

With LeNet-5 again trained from scratch but this time for the multi-label task, results were unsurprisingly poor due to the model's limited capacity and single-label design. The achieved test accuracy was 44.85%, recall 59.5%, and precision 14.2%. The learning rate found by the tuner was 0.0001, and the training epochs were 20.

These results were expected as LeNet-5 was originally designed to classify grayscale images with dimensions 28x28 with a single-label. Nonetheless, its recall is remarkable as it means that each class has more correct predictions than false positives. However, the low precision means that few samples of each class are identified.

3.2.2 AlexNet

With its higher complexity and greater depth compared to LeNet-5 we expected better results from AlexNet. Trained for 20 epochs with an optimal learning rate found by the tuner of $9.12*10^{-8}$ it did indeed produce a higher test accuracy than LeNet-5: 53.04%. However, the recall (35.18%) and precision (6.933%) were worse which means that the model was predicting none or many classes for each image. These poor results can be explained by the huge number of parameters relative to the size of the data set and its single-label design.

3.2.3 PreTrainedAlexNet

The AlexNet with pre-trained weights for its feature extractor did indeed perform significantly better due to its smaller number of parameters. We obtained a test accuracy of 91.82% after training for 20 epochs with an optimal tuned learning rate of $2.754*10^{-8}$. The model's recall and precision were impressive as well: 83.7% and 73.12% respectively. These results come closer to the results reported with state-of-the-art models, which will be investigated in section 3.3.

3.2.4 TrimoNet

With the pre-trained version of AlexNet producing remarkable results, we were aiming to beat these results with our own network; TrimoNet. Unfortunately, TrimoNet did not come close to beating pre-trained AlexNet in all metrics. Its test accuracy was 71.81%, its recall 35.94%, and precision 24.48% after training for 20 epochs with an optimal tuned learning rate of $6.31*10^{-8}$. Even though these results seem poor compared to pre-trained AlexNet's results they were better than LeNet's and AlexNet's ones as the three accuracy metrics were more balanced. For LeNet and AlexNet the recall was high, but the precision was low. In contrast, TrimoNet produced more balanced results where they were in the same range.

A few multi-label predictions of TrimoNet are shown in figure 5 in section 6 (Figures).

3.3 Results compared to literature

Our results for the single-label scenario were compared to the results obtained on the same dataset by Han et al. (2018). In this paper, the authors experimented with different pretrained

models on the UCM dataset, using the 60% of training set like in this project. The models implemented with a supervised learning approach all had a classification accuracy ranging from 93.15% to 95.52%. Obviously, these models significantly outperform TrimoNet on the UCM dataset. For instance, the model with the lower accuracy (93.15%) within the set of models used by the authors was GoogLeNet (Szegedy et al., 2015), a CNN made of 22 convolutional layers and optimized with its 'inception module' to decrease the number of parameters. Thanks to this peculiarity, the network can be deeper, hence extract and learn more features from the input images, and it is also less likely to overfit. Furthermore, it is crucial to point out that these models are pre-trained on ImageNet (Deng et al., 2009), a dataset with 3.2 million images in total. These models are clearly likely to outperform TrimoNet, which is trained from scratch using a significantly smaller dataset. Predictably, for the single-label scenario, the model implemented in this project which gets closer to the accuracies achieved by Han et al. (2018) is PreTrainedAlexNet, which was pre-trained on the same dataset as GoogLeNet. The difference of classification accuracy between these 2 models is certainly due to the greater complexity and depth of GoogLeNet, which allows it to learn more from the dataset.

Concerning the multi-label scenario, the results were compared to Li et al. (2020). In this work, the authors proposed a model called MLRSSC-CNN-GNN, which combines a convolutional neural network and graph neural network (GNN) and tested it on the multi-label UCM dataset. They evaluated the model performance using Precision and Recall as done in this project and, additionally, they used F1 and F2 score. The Precision and Recall of their model were 86.41% and 87.11%, respectively, clearly outperforming TrimoNet. Even though the authors used a model trained from scratch on the UCM dataset, their results were obviously better due to the more complex architecture compared to TrimoNet. Also in this case, PreTrainedAlexNet predictably gets closer to these values, taking advantage of its pretrained weights on a larger dataset.

3.4 Future improvements

After comparing our results with literature, some potential improvements for TrimoNet were identified:

- Implement F1 score as multi-label classification metric;
- Implement multi-scale prediction to generate multiple feature maps with different scale, increasing the capability of the model to performance inference with object at different scale;
- Increase the complexity of the neural network, such as adding more convolutional layers and make it deeper, add residual connection and regularization techniques such as weight decay.

4. Implementation

Users can access our notebook using Google Colab here.

5. References

- Abdelhack, M. (2020). A Comparison of Data Augmentation Techniques in Training Deep Neural Networks for Satellite Image Classification. Paper pre-print: https://arxiv.org/abs/2003.13502v1
- Chaudhuri, B. Demir, S. Chaudhuri and L. Bruzzone, "Multilabel Remote Sensing Image Retrieval Using a Semisupervised Graph-Theoretic Method," in IEEE Transactions on Geoscience and Remote Sensing, vol. 56, no. 2, pp. 1144-1158, Feb. 2018, doi: 10.1109/TGRS.2017.2760909.
- Deng, J., Dong, W., Socher, R., Li, L. J., Li, K. and Li, F. F. (2009). ImageNet: A large-scale hierarchical image database. IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848.
- Falcon, William, & The PyTorch Lightning team. (2023). PyTorch Lightning (2.0.2). Zenodo. https://doi.org/10.5281/zenodo.7859091
- Han, W., Feng, R., Wang, L., & Cheng, Y. (2017). A semi-supervised generative framework with deep learning features for high-resolution remote sensing image scene classification. ISPRS Journal of Photogrammetry and Remote Sensing. doi:10.1016/j.isprsjprs.2017.11.0
- Ioffe, S., & Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167
- Krizhevsky, A. (2014). One weird trick for parallelizing convolutional neural networks. https://doi.org/10.48550/arXiv.1404.5997
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems (pp. 1097–1105).
- Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. (1998). "Gradient-based learning applied to document recognition" (PDF). Proceedings of the IEEE. 86 (11): 2278–2324. doi:10.1109/5.726791. S2CID 14542261.
- Li, Y., Chen, R., Zhang, Y., Zhang, M., Chen, L. (2020). Multi-Label Remote Sensing Image Scene Classification by Combining a Convolutional Neural Network and a Graph Neural Network. Remote Sens. 2020, 12, 4003. https://doi.org/10.3390/rs12234003
- Pritt, M. and Chern, G "Satellite Image Classification with Deep Learning," 2017 IEEE Applied Imagery Pattern Recognition Workshop (AIPR), Washington, DC, USA, 2017, pp. 1-7, doi: 10.1109/AIPR.2017.8457969.
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., 2015. Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–9.

- Vaswani, S., Bach, F., Schmidt, M. (2019). Fast and Faster Convergence of SGD for Over-Parameterized Models and an Accelerated Perceptron. https://arxiv.org/abs/1810.07288
- Yi Yang and Shawn Newsam, "Bag-Of-Visual-Words and Spatial Extensions for Land-Use Classification," ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS), 2010.

6. Figures

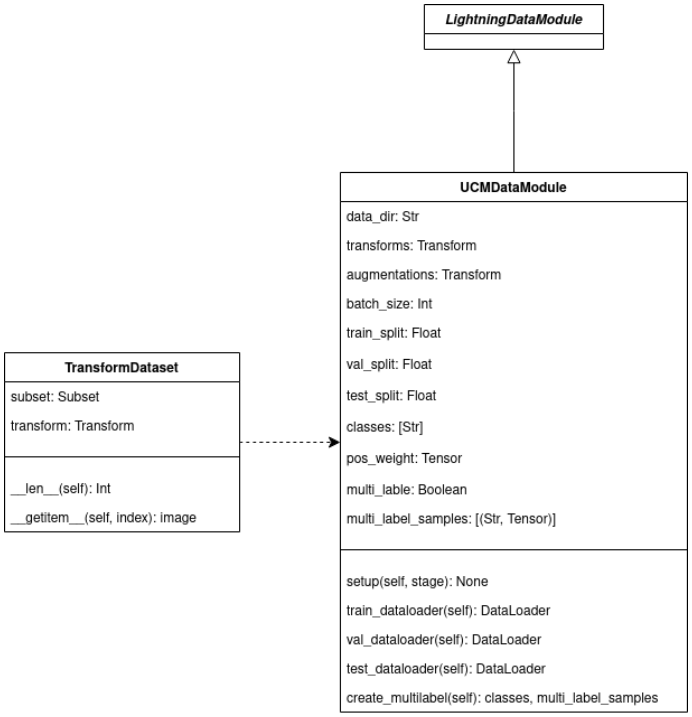

Figure 1: UML diagram showing the class structure of the data.

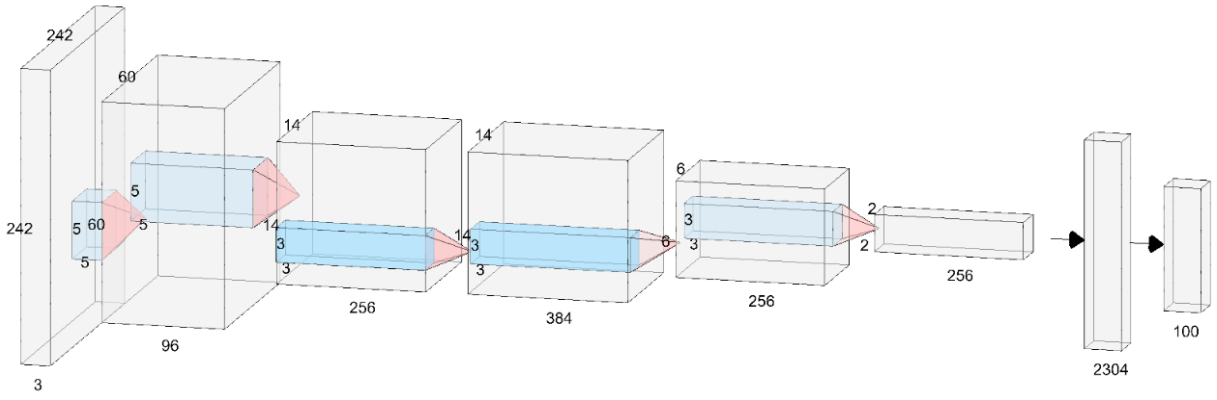

Figure 2: architecture of TrimoNet.

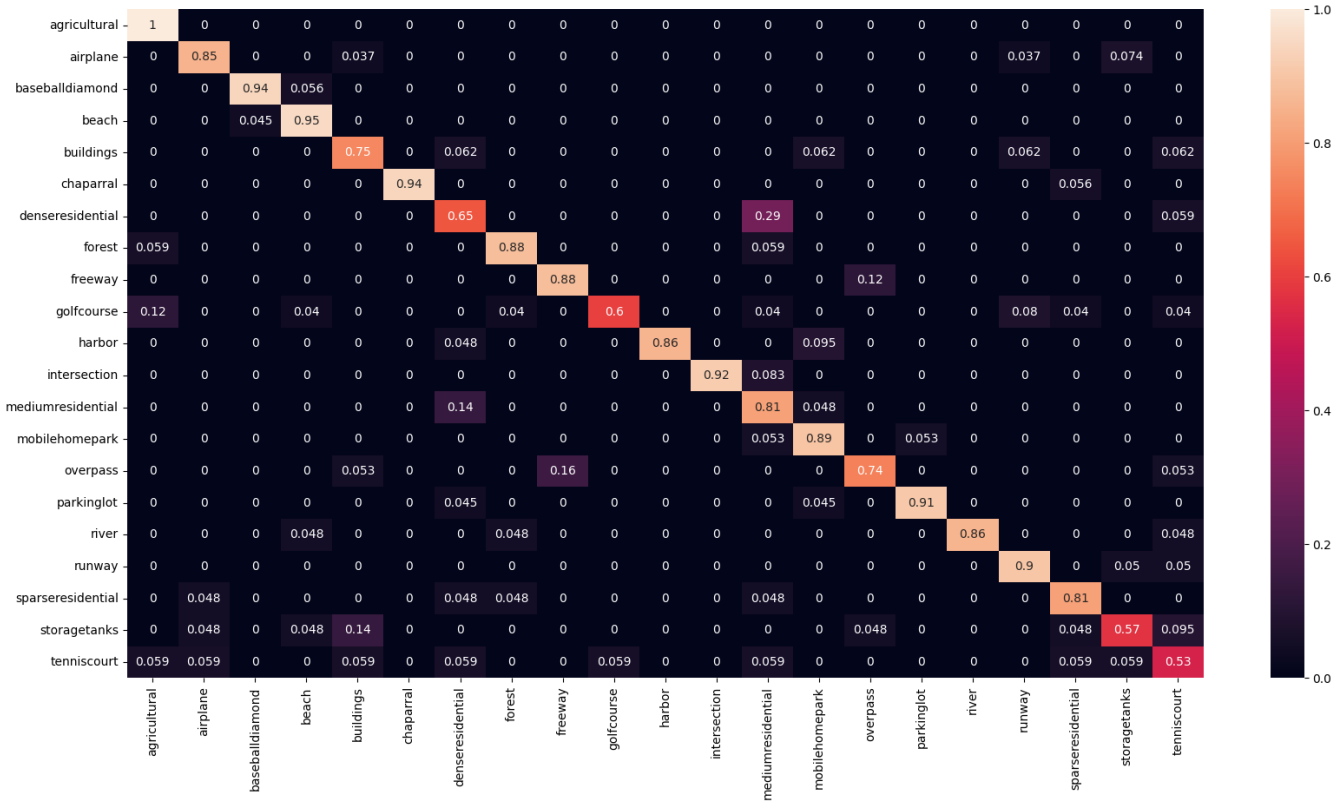

Figure 3: confusion matrix obtained with TrimoNet in the single-label scenario.

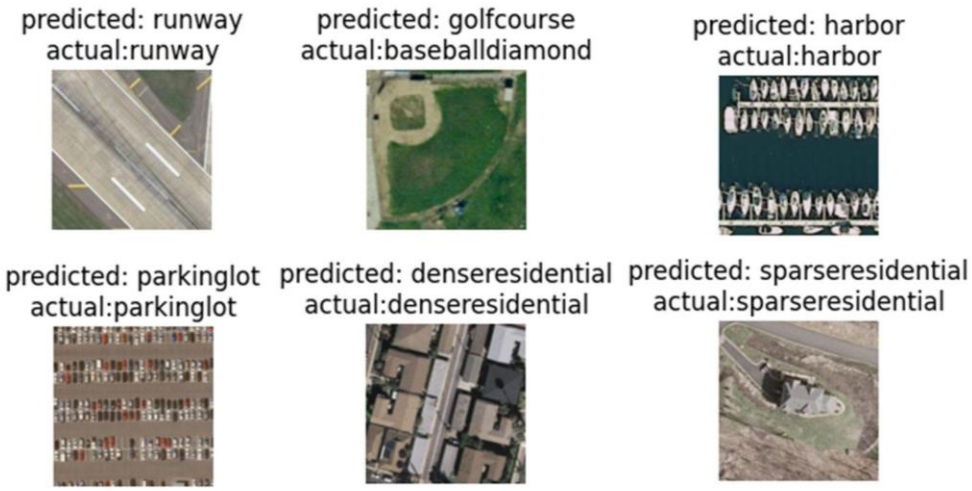

Figure 4: a few predictions made using TrimoNet in the single-label scenario.

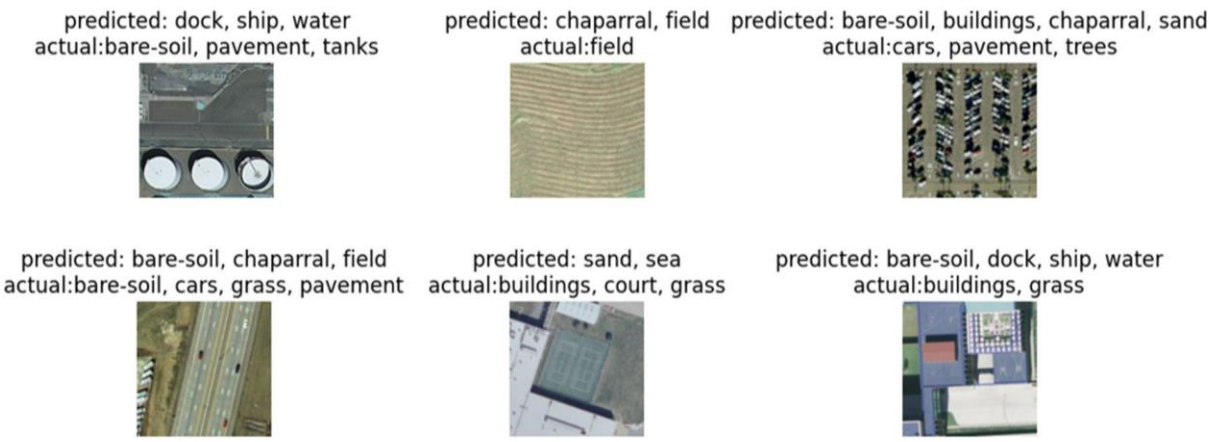

Figure 5: a few predictions made using TrimoNet in the multi-label scenario.

7. Tables

Table 1: Classes distribution in the UCM multi-label dataset

Class label	Number of Images
Airplane	100
Bare soil	718
Buildings	691
Cars	886
Chaparral	115
Court	105
Dock	100
Field	103
Grass	975
Mobile home	102
Pavement	1300
Sand	294
Sea	100
Ship	102

Tanks	100
Trees	1009
Water	203

Note: adapted from Chaudhuri et al. (2018). Our distribution is slightly different.

Table 2: Test losses and accuracies of all four models for the single-label task.

Model	**Test Loss**	**Test Accuracy (%)**
LeNet	3.047	3.6
AlexNet	3.048	3.3
PreTrainedAlexNet	1.63*	**75.6**
TrimoNet	**0.2775**	71.5

**The loss of PreTrainedAlexNet is different from the other models' test losses as the loss is only calculated for its classifier and not its feature extractor.*

Table 3: Test recall, precision, and accuracy of all four models for the multi-label task.

Model	**Test Recall (%)**	**Test Precision (%)**	**Test Accuracy (%)**
LeNet	59.5	14.47	44.85
AlexNet	35.18	6.933	53.04
PreTrainedAlexNet	**83.7**	**73.12**	**91.82**
TrimoNet	35.94	24.48	71.81