Deep Learning Tasks and Land-Use/Land-Cover Classification Using EuroSAT Dataset

Pau Comas and Odysseas Kyparissis[a]

[a]*UPC, Facultad de Informatica de Barcelona , Barcelona, January 2024, Master of Data Science, Spain*

Abstract

This advanced machine learning project aims to address a diverse set of tasks encompassing custom Convolutional Neural Network (CNN) modeling, optimization techniques exploration, model interpretability, transfer learning, and an open-ended project. The latter is the classification of EuroSAT dataset (Helber et al., 2019) comprising 27,000 labeled and geo-referenced images categorized into 10 distinct land use and land cover classes.

Keywords: convolutional neural network, deep learning, modeling, multiclass classification, transfer learning, model interpretability

1. Task 1: Custom CNN on Cifar10

The Cifar-10 (Krizhevsky, 2009) dataset is a widely used benchmark dataset in the field of computer vision and machine learning. It is designed for image classification tasks and consists of 60,000 32x32 color images in 10 different classes. Each class contains 6,000 images, making it a balanced dataset. The ten classes in the CIFAR-10 dataset are: *airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck.*

In this task, we were commanded to create a CNN model on the Cifar10 dataset, to then study memory requirements and the computational load of each layer. Our CustomCNN architecture is showcased in Table 1. It is a Convolutional Neural Network (CNN) designed for image classification tasks, specifically applied to the CIFAR-10 dataset, with 10 output classes. Comprising multiple convolutional layers interspersed with batch normalization, Leaky ReLU activations, max-pooling, and dropout layers, the model aims to capture intricate hierarchical features in input images. Starting with initial convolutional layers detecting low-level features, the network then deepens, extracting more abstract features in subsequent layers. The fully connected layers towards the end of the architecture consolidate learned features and produce final predictions. In total, the model has 551466 trainable parameters. LeakyReLU is chosen for mitigating the dying neurons issue in ReLU and batch normalization (BN) and dropout are included for regularization. We aimed to strike a balance between complexity and efficiency, creating a model suitable for image classification and incorporating techniques to enhance generalization and mitigate overfitting.

As we can see in Table 1, the Custom CNN architecture displays a significant variation in memory usage and computational time across different types of layers. Convolutional layers (*Conv1*, *Conv2*, etc.) exhibit higher memory usage due to their numerous parameters for feature extraction, with the first layer (*Conv1*) being particularly memory-intensive owing to its role in initial feature detection. The memory usage then gradually decreases in subsequent convolutional layers, reflecting the reduced spatial dimensions after pooling. Max pooling layers (*Pool1*, *Pool2*, etc.) and dropout layers (*Dropout1*, *Dropout2*, etc.), being parameter-free, show markedly lower memory requirements. However, their computational times are minimal, illustrating their efficiency in downsampling and regularization, respectively. The fully connected layers (*FC1*, *FC2*) demonstrate a distinct memory footprint, significantly less than the initial convolutional layers, due to their role in classification rather than feature extraction. The relatively higher computational time in the first fully connected layer (*FC1*) can be attributed to its involvement in integrating the learned features from the entire network. Furthermore, the model size is approximately 1.5 MB.

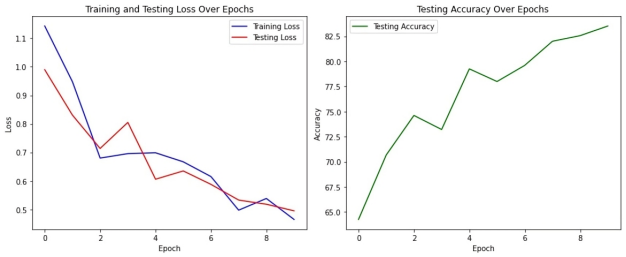


Figure 1: Custom CNN over Cifar10 loss and accuracy results

Finally, the training of CustomCNN over Cifar10 was performed with 10 epochs, batch size of 64, cross-entropy as the loss and Adam optimizer with a learning rate of 0.001. As observed in Figure 1 the top accuracy obtained was 84%, although the model could have been trained for longer as loss behaviour didn't indicate overfitting at epoch 10. Nonetheless, this was not the focus of this task.

2. Task 2: Experimentations on Custom CNN

In this task, we analyze the CustomCNN model's performance under various conditions and hyperparameters to understand their impact in the predictability and computational power

Table 1: Custom CNN architecture with memory usage (KB) and time elapsed

Layer	Output Size	Parameters	Operation	Memory Usage (KB)	Time Elapsed (s)
Conv1	$32 \times 32 \times 32$	$3 \times 32 \times 3 \times 3 + 32$	Conv + BN + LeakyReLU	131.07	0.040009
Conv2	$32 \times 32 \times 32$	$32 \times 32 \times 3 \times 3 + 32$	Conv + BN + LeakyReLU	131.07	0.000641
Pool1	$16 \times 16 \times 32$	N/A	Max Pooling	32.77	0.000171
Dropout1	$16 \times 16 \times 32$	N/A	Dropout	32.77	0.001837
Conv3	$16 \times 16 \times 64$	$32 \times 64 \times 3 \times 3 + 64$	Conv + BN + LeakyReLU	65.54	0.000521
Conv4	$16 \times 16 \times 64$	$64 \times 64 \times 3 \times 3 + 64$	Conv + BN + LeakyReLU	65.54	0.000265
Pool2	$8 \times 8 \times 64$	N/A	Max Pooling	16.38	0.000041
Dropout2	$8 \times 8 \times 64$	N/A	Dropout	16.38	0.000051
Conv5	$8 \times 8 \times 128$	$64 \times 128 \times 3 \times 3 + 128$	Conv + BN + LeakyReLU	32.77	0.000566
Conv6	$8 \times 8 \times 128$	$128 \times 128 \times 3 \times 3 + 128$	Conv + BN + LeakyReLU	32.77	0.000922
Pool3	$4 \times 4 \times 128$	N/A	Max Pooling	8.19	0.000044
Dropout3	$4 \times 4 \times 128$	N/A	Dropout	8.19	0.000052
Flatten	2048	N/A	Flatten	8.19	0.000091
FC1	128	$128 \times 2048 + 128$	Linear + LeakyReLU	0.51	0.010467
Dropout4	128	N/A	Dropout	0.51	0.000073
FC2	10	$128 \times 10 + 10$	Linear	0.04	0.000531

of the model. We focus on exploring the effects of data augmentation, varying training batch sizes, the role of batch normalization, and strategies to tackle overfitting. To do that, we edited the CustomCNN design as it was necessary starting from our work done in Task 1 (Section 1).

When we employed data augmentation to create new images we did it through a set of transformations. These augmentations include random horizontal flips, rotations up to 10 degrees, affine transformations with shear up to 10 degrees and scaling between 0.8 to 1.2, as well as color jittering for variations in brightness, contrast, and saturation. By applying these transformations, we introduced variability to the training images, enriching the dataset aiming to reduce overfitting risks.

However, a crucial decision rose regarding the utilization of the augmented dataset: whether to use it exclusively or in conjunction with the original dataset. After thorough consideration, we concluded that combining the augmented and original datasets offers a more balanced approach. This decision was driven by several factors. Firstly, it ensures that the model is exposed to both the real data distribution and its augmented variations, potentially leading to enhanced robustness and generalization capabilities. Secondly, while exclusive reliance on augmented data can lead to learning features that may not generalize well to the original data distribution, the concatenated approach mitigates this risk. The combined dataset includes a broader spectrum of visual scenarios, supporting the model in capturing essential features from both original and transformed images.

Therefore, in our revised strategy, we concatenate the original CIFAR10 dataset with its augmented counterpart, forming an extended training set. This approach aligns with our goal of creating a model that is not only accurate but also robust and capable of generalizing well across diverse visual inputs. The concatenated augmented dataset is then used in the training pipeline (only in the experiments including data augmentation), replacing the sole reliance on either the original or purely augmented datasets. This method is particularly advantageous for our scenario, where the original dataset's size and diversity are limited, necessitating a more comprehensive training set to achieve better model performance and predictability.

In Table 2 we can check the hyperparameters of the five different experiments we conducted (1,2,3,4 and 5). Figure 2, 3, 4, 5 and 6 depict the training and test loss behaviour along with the classification accuracy of every experiment respectively. Experiment 1 performs no data augmentation, uses a small training batch, has no batch normalization, no dropout and no L2 regularization. The model manages to achieve 78% of accuracy, and we can clearly observe that starts to heavily overfit from epoch 5 onwards, so testing accuracy doesn't improve at-all during the last epochs. In Experiment 2 we test the effect of data augmentation, and immediately see that the training time raises from $3m12s$ to $8m12s$, which is normal given the new data addition, and the decision dicsussed previously. These 5 minutes of extra training allowed the model to reach 82% accuracy. We still see the effect of overfitting starting around epoch 5, so we believe that training for more epochs wouldn't have allowed for better accuracy results. Afterwards, Experiment 3 was performed to assess the effect of the batch size going from 32 to 512. This increment entailed the reduction of training time, as now we use much more images for one training gradient batch calculation and therefore allows for a faster epoch. In terms of performance, the accuracy obtained was 80% and an interesting effect was to see the retardment of the overfitting effect, which means that there is a possibility that training for more epochs would have allowed to overcome the results of Experiment 2.

Thereupon, Experiment 4 assessed the effect of batch normalization in our configuration. The conclusion is that it led to a bit faster generalization in the same training time as the previous run (84% accuracy) without being enough regularization for solving the overfitting problem. Finally. Experiment

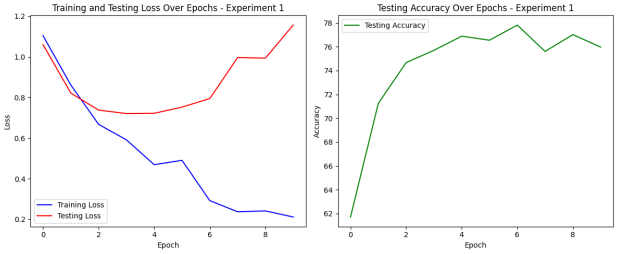


Figure 2: Experiment 1 loss and accuracy results

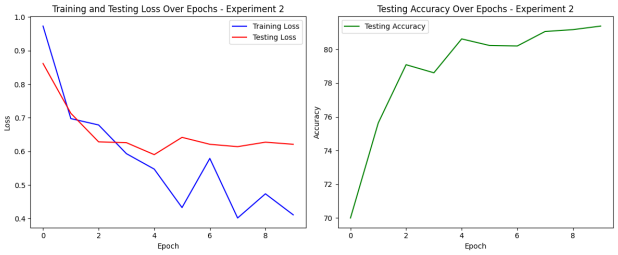


Figure 3: Experiment 2 loss and accuracy results

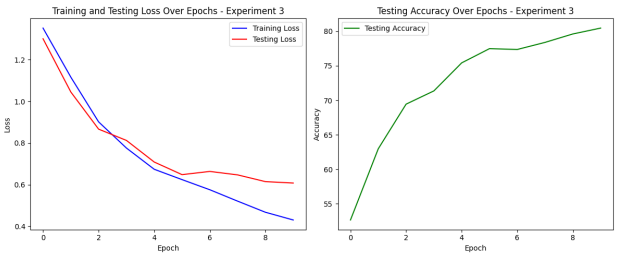


Figure 4: Experiment 3 loss and accuracy results

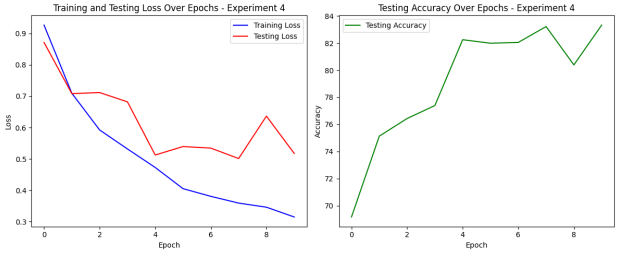


Figure 5: Experiment 4 loss and accuracy results

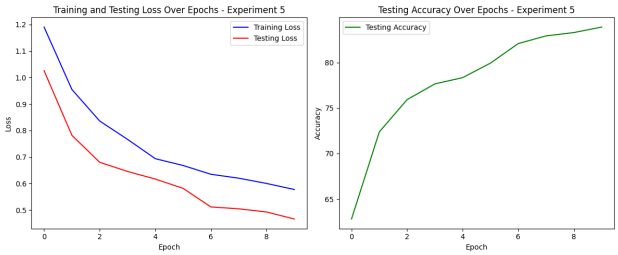


Figure 6: Experiment 5 loss and accuracy results

5 entails applying together our three regularization techniques: batch normalization, L2 regularization and dropout. The latter was applied with a 25% of probability of zeroing an input element during training and placed every two convolutional operations and after all fully connected layers. The result is that Experiment 5 achieved the highest accuracy with an 84% and as observed in the training/test loss functions of Figure 6 the testing loss never stops decreasing anymore and we can state that we solved our previous overfitting issues; which means that as opposite to Experiment 4 training for more epochs has better chances of retrieving a better model. Moreover, a good note to make is that in this run the testing loss is always lower than the training loss, which would be unnatural if it wasn't because it's due to the dropout effect: activating it during training allows for its effect to take place and randomly zero out some neurons, which complicates predictions, but when we test we deactivate it and allows for reduced losses.

3. Task 3: Interpretability

Interpretability in machine learning is paramount for understanding model decisions and ensuring trust in automated systems. In this task, we explore the interpretability of convolutional neural networks (CNNs) using our custom CNN from Task 1 (Section 1) and a pre-trained VGG16 model (Simonyan and Zisserman, 2014), both applied to the CIFAR 10 dataset. Our objective is to visualize and interpret the internal workings of these models to gain insights into their decision-making processes.

The methodology used includes several techniques to understand the models' behaviors. We visualize the filters and activations within the networks to comprehend the features they learn at different layers. Filters reveal the patterns and textures that the network seeks in the input images, while activations showcase the responses from various layers to these patterns.

Additionally, we retrieve the top k-samples that maximally activate each unit in a layer. These samples are indicative of the features that the network perceives as significant. To facilitate a deeper understanding, we employ the t-Distributed Stochastic Neighbor Embedding (t-SNE) visualization technique on the fully connected layers of the networks. t-SNE is a powerful tool for reducing high-dimensional data to two or three dimensions, thereby enabling us to observe the clustering of activations and understand the models' learned representations.

To implement these methodologies, we adopted several techniques to analyze the neural network models. Initially, we focused on filter visualization. This involved extracting weights from all the convolutional layers of each model and processing them to generate interpretable images. Subsequently, to obtain activation maps, we forwarded a subset of the CIFAR 10 images through the networks and captured the outputs from each layer.

Furthermore, we embarked on the calculation of the top k-samples for each unit. This was achieved by ranking the activations and selecting the images that resulted in the highest activation values. Lastly, for the purpose of visualizing high-dimensional data, we generated t-SNE plots. This process en-

Table 2: Task 2: Experiment hyperparameters

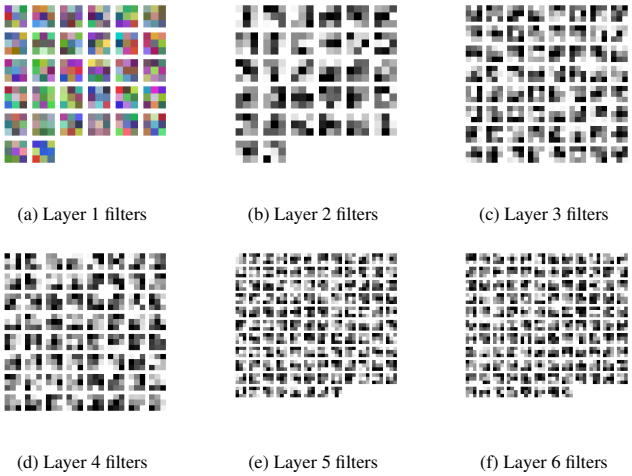
Experiment	Batch Size	Data Augmentation	Batch Normalization	Dropout	L2 Reg.	Epochs	Time
1	32	No	No	No	No	10	3m 12s
2	32	Yes	No	No	No	10	8m 12s
3	512	Yes	No	No	No	10	6m 12s
4	512	Yes	Yes	No	No	10	6m 11s
5	512	Yes	Yes	Yes	Yes	10	5m 58s

tailed extracting the high-dimensional activation data from the fully-connected layers of interest. We then applied the t-SNE algorithm, which aided in reducing dimensionality while preserving the relative distances between data points.

Each step of the methodology is designed to peel back the layers of complexity within the CNNs, offering us a view into their feature extraction and classification strategies. The subsequent sections will detail the results and interpretations derived from these techniques, providing a semantic understanding of our models' functionality.

3.1. Filters Visualization

Convolutional layers are instrumental in feature detection within CNNs. Each layer's filters capture different aspects of the input data, which are progressively abstracted into higher-level features. In this subsection, we present the visualizations of filters from the six convolutional layers of the CustomCNN model (see architecture in Table 1). The visualizations in Figure 7 help us understand what features each layer is extracting from the CIFAR-10 dataset images.



(a) Layer 1 filters (b) Layer 2 filters (c) Layer 3 filters

(d) Layer 4 filters (e) Layer 5 filters (f) Layer 6 filters

Figure 7: Visualization of filters from all convolutional layers of the CustomCNN model. Each subfigure illustrates the filters from a different convolutional layer, showing the progression from low-level feature detection in the earlier layers to more abstract feature representation in the deeper layers.

The analysis of the convolutional layers in the CustomCNN model reveals a hierarchical nature of feature extraction. **Layer 1** (Figure 7a) filters are primarily concerned with detecting low-level features such as edges and simple textures from the raw pixel data. The result is in colored format, reflecting the first layer's handling of a 3-channel image. This characteristic diminishes in subsequent layers due to flattening processes. Moving to **Layer 2** (Figure 7b), the filters capture more complex patterns, including corners and color gradients, indicating a progression towards more sophisticated feature extraction. In **Layer 3** (Figure 7c), the filters seem to combine simple features from previous layers into compound textures and parts of objects within the images. The filters of **Layer 4** (Figure 7d) show increased complexity, capturing higher-order features that may correspond to parts of objects, such as the wheels of vehicles or eyes of animals. By **Layer 5** (Figure 7e), the filters are more abstract, indicating an aggregation of features representing more significant aspects of the image's content. Finally, **Layer 6** (Figure 7f) displays a level of abstraction that suggests the network's capability to detect and represent high-level features necessary for class discrimination.

These visualizations illustrate the hierarchical nature of feature extraction performed by the CustomCNN. Initially focusing on simple edges and textures, the network progresses towards recognizing more complex and abstract representations as we advance through the layers. This hierarchical processing is fundamental to the CNN's ability to learn from visual data and make accurate predictions. Finally, the visualizations of the VGG16's filters is not included in the report, for space-saving reasons, since the model includes 13 distinct convolutional layers. Although, the results obtained present a similar behavior for VGG16, meaning that the first convolutional layers seem to extract low-level features that are relevant with edges, and as the model progresses, the filters get more complex and abstract, leading to the result, that VGG16 is capable of extracting very abstract patterns that are able to help the last layers identify better the class of the image received in the input (results of VGG16 are included in the project's deliverable).

3.2. Analysis of Top-k Activations in Dense Layers

The top-k activations for each unit in the dense layers of the CustomCNN and VGG16 models, where $k = 10$, reveal insights into the feature extraction and classification process of each model. Figures 8a to 8e illustrate the activations for the three dense layers of both models, providing a visual representation of the models' internal state in response to the input data.

In a comparative analysis, the first dense layer of the CustomCNN model (8a) and the VGG16 model (8b) exhibit distinct patterns of activation. CustomCNN's first dense layer shows a propensity for identifying specific classes in some of its neurons, although in this phase it stills confuses most of them. For

(a) CustomCNN Dense Layer 1 (b) VGG16 Dense Layer 1 (c) CustomCNN Output Layer

(d) VGG16 Dense Layer 2 (e) VGG16 Dense Output Layer

Figure 8: Top-k activations for each unit in the dense layers of CustomCNN and VGG16 models.

example, *unit 6* is clear that it tries to identify *birds*, but sometimes *airplanes* are included as well in the top activated images. On the other hand, it can be seen that in *unit 30* of CustomCNN, the model is performing extremely good in identifying *cars*, same for *unit 54* about horses. Although, an example of confusion is present in *unit 66* where deers are confused with some airplane images (which it makes sense, if you closely take a loot at the images), because the horms of the animal are similar to the shapes of the airplane wings, leading the model to confusion. However, in general it can be seen that, even from the first densed layer, some classes can be identified in a satisfactory level. On the other hand, VGG16 which is trained to identify 1000 distinct classes, in the first dense layer demonstrates moderate results for the identification of the classes of CIFAR 10, except for dogs and cats. Although, its first fully-connected layer has 4096 units, we only present here some of them up the 128*th*, and this might be a reason of not selecting units that perform classification of specific classes in correctly. For example, in *unit 72* we can see a clear confusion of *horses, cats, dogs, cars and boats*, while in *unit 42* it purely includes images of *dogs*. Since, VGG16 was trained in a huge number of different examples and 1000 distinct classes, as mentioned before, it makes sense to have difficulties on identifying correctly the 10 classes of a different dataset, in its first dense layer.

Moving to the second dense layer, CustomCNN (8c) and VGG16 (8d) present interesting results. Since for the CustomCNN architecture, the activations at this layer belong to the output layer, it can be seen that each unit (10 in total, same as the number of classes of CIFAR 10) has the maximum activations for a specific class, indicating very good results on the performance of the model. Meaning that each unit is responsible for distinguishing one class from the other (e.g. *unit 0* is responsible for identifying planes while *unit 9* for trucks).

On the other hand, the second dense layer for VGG16 (still having 4096 units, but we present random ones up to the first 128), presents better results compared to its first dense layer. For example, it can be seen now that *unit 36* mainly identifies vehicles, such as *cars, trucks and boats*, while *unit 127* identifies perfectly dogs. It is interesting to mention that *unit 98* for instance identifies *cars and trucks* of color red, with just a single ”mistake” of a yellow car.

Finally, in the output layer of VGG16 (8e), which still includes 1000 units, the activations start to have a more clear pattern of identification of objects. However, still the results are not very satisfying for the classification task of CIFAR 10 dataset, since only 10 distinct classes are present in it, while VGG16 identify more abstract and general patterns in this step due to the training dataset. It is interesting that the top activations of each unit, include images, that are similar in the matter of colors, edges, and shapes, indicating the details we mentioned before, about the learnable features of the CNNs as we dive into deeper layers.

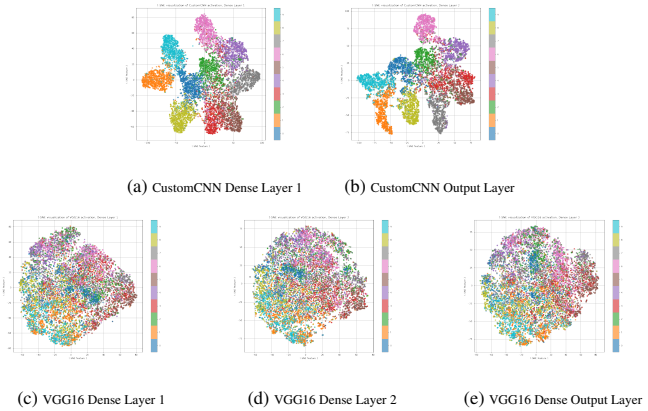
These visual comparisons underscore the unique feature hierarchies developed by the CustomCNN and VGG16 models and how they differently interpret the input data to arrive at their classifications.

3.3. t-SNE Visualization of Dense Layer Activations

The t-SNE technique provides a powerful method for visualizing the high-dimensional activation data produced by the dense layers of neural networks. By mapping these activations to a two-dimensional plane, t-SNE plots allow us to observe the clustering of activations and how well the network separates features corresponding to different classes, but at the same time, it allows us to cross-check with a clearer visualization the results obtained in the previous sub-section.

Figures 9a to 9e display the t-SNE visualizations for the dense layers of the CustomCNN and VGG16 models. These plots complement our top-k activation analysis by providing a

spatial representation of how the models group and differentiate between features.



(a) CustomCNN Dense Layer 1 (b) CustomCNN Output Layer

(c) VGG16 Dense Layer 1 (d) VGG16 Dense Layer 2 (e) VGG16 Dense Output Layer

Figure 9: t-SNE visualizations of the activations in the dense layers of the CustomCNN and VGG16 models.

The t-SNE plot for CustomCNN's first dense layer (9a) shows distinct clusters corresponding to different classes, which aligns with our earlier observation of the model's ability to discern specific classes at an early stage, although minor confusion between classes is present. This clustering becomes more pronounced in the output layer (9b), demonstrating a clear separation between classes and indicating the model's effective learning of discriminative features.

In contrast, the t-SNE visualizations for VGG16's dense layers reveal a more diffuse clustering at the first dense layer (9c), which is consistent with the model's initial difficulty in distinguishing the CIFAR 10 classes. However, as we move deeper into the network, the second (9d) and third (9e) dense layer visualizations show a gradual improvement in class separation, though not as distinct as CustomCNN's. This suggests that while VGG16 has learned generalizable features from its extensive training on a large dataset with numerous classes, its adaptation to the CIFAR 10 dataset is not as immediate or clear-cut as the CustomCNN model specifically trained on this dataset.

4. Task 4: Transfer learning

In this task different training configurations involving the transfer of pretrained layers and fine-tuning are tested for interpretation and comparison of outcomes. The goal was to delve into possible impacts of the design choices regarding training a whole model from scratch to utilizing pretrained layers in other tasks, which leads to deciding to freeze or fine-tune them. We had to perform 4 subtasks, which we nextly introduce here one-by-one to then focus on the results obtained.

4.1. Full training and finetune of Custom CNN over Terrassa 900 dataset

Terrassa Buildings (UPC, 2015) is a dataset of still images of buildings from the city of Terrassa. The dataset was collected and annotated by the students of the ESEIAAT school in UPC. It has 13 different types of buildings and has 2061 train split images, 820 validation split images and 1245 test split images. We use train+validation for our train set (70%) of the data and keep the test set (30%) for evaluation purposes.

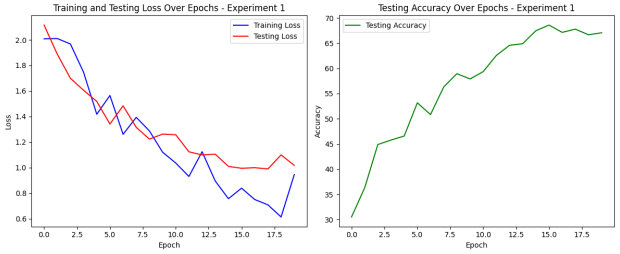


Figure 10: Transfer test 4.1: Loss and accuracy results

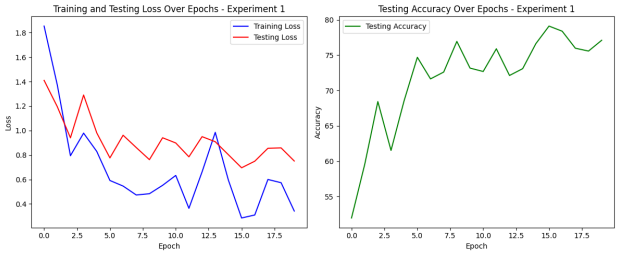


Figure 11: Transfer test 4.1.5: Loss and accuracy results

In this subtask we applied our CustomCNN, designed in previous work, consisting of 6 convolutional layers and 551853 trainable parameters in total, to training of the Terrassa 900 classification challenge from scratch. In Figure 10 we can observe the train/test losses and the accuracy over the 20 epochs of training. The total training time was 3m 3s. Starting from a 27% accuracy the model achieves a peak 68% accuracy. From epoch 12 onwards the model starts to overfit. It is important to note that because CustomCNN is designed to receive input images of 32x32x3, we had to adapt all Terrassa images, which were considerably bigger. This abrupt reducing of the input space's dimensionality may cause a loss of information for the model's potential.

In an additional test that we named 1.5, we decided to use the pretrained weights of CustomCNN over Cifar10, which achieved a 84% accuracy, as a initial version of the model and fine-tune it as a whole. The results are in Figure 11 and show how now the model starts from a 52% of accuracy and achieves a 79% peak accuracy. The starting point is a 28% more accurate than in 4.1 and the maximum accuracy is a 11% better. This proves that transferring the Cifar10 layers of our CustomCNN and using them as a base for the Terrassa 900 training helps in achieving better results faster, given that we again trained for 20 epochs and the execution time was 3m.

4.2. Custom CNN as a feature extractor for Terrassa 900 dataset

In this test, we obtained the pretrained CustomCNN on Cifar10 and froze all the layers for our Terrassa 900 training, such as a feature extractor of 550176 non-trainable parameters. The only trainable parameters were in a final linear layer, of

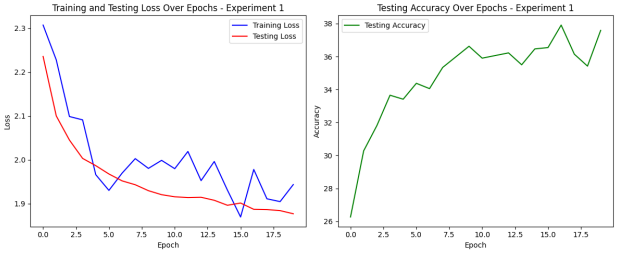


Figure 12: Transfer test 4.2: Loss and accuracy results

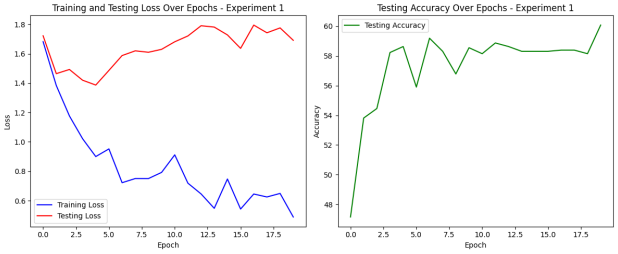


Figure 13: Transfer test 4.3: Loss and accuracy results

1677 parameters and 13 output neurons (one for every building class). Training results are shown in Figure 12. The main outcome is that freezing the layers and only training the linear classifier doesn't allow the model to achieve good results, as compared to 4.1.5, and here we only achieve a 38% accuracy. We therefore have underfitting and means that the frozen layers from Cifar10 don't capture enough information for easing up the Terrassa 900 classification without being fine-tuned (such as in 1.5). On the other hand, the good thing of training less parameters is that we sped up training (only 2 minutes now) over the course of the 20 epochs.

4.3. Finetuning and freezing over VGG16 for Terrassa 900 dataset

Now we moved on from our CustomCNN model and brought a pretrained instance of VGG16, which had been already introduced in the project. For this test, we froze the first 5 convolutional layers (the closest to the input) and substituted the VGG16 classifier part with our own sequential feed forward architecture. So at the end from the 68208461 parameters possible, 14714688 were left as non-trainable. Terrassa images are fed with 224x224x3 dimensionality, as it is the default input size of the VGG16 net. The results are shown in Figure 13. The pretrained layers effect allows us to start already from an accuracy of 47%, and then there is a swift grow until a maximum of 60%. This run clearly overfits much faster than before, as we can see in the tendency of the training and test losses. We can conclude that the pretrained weights allow us to first quickly reach moderately competitive classification, but the first frozen layers could be hindering the potential of the model and we get stuck in a suboptimal point.

Figure 14: Sample crops of *triomphe* (top), *sacrecoeur* (middle) and *notredame* (bottom) in the ParisComas dataset

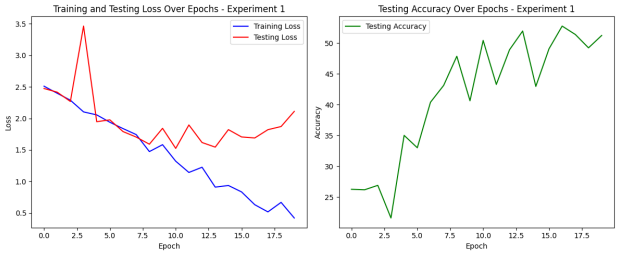


Figure 15: Transfer test 4.4: Loss and accuracy results

4.4. More finetuning and freezing with the Terrassa + ParisComas dataset

The Paris Dataset (Philbin and Zisserman, 2017) consists of 6412 images collected from Flickr by searching for 12 particular Paris landmarks, such as the Eiffel Tower, Moulin Rouge, etc. Nonetheless, this is a pretty noisy dataset, with a lot of images that do not correspond to the actual buildings/landmarks. For that reason, we decided to manually clean the dataset as well as we could, selecting between 100-110 images that identify easily the landmark for a human. This new version of the dataset, with 9 landmarks (*sacrecoeur, defense, eiffel, notredame, triomphe, invalides, moulinrouge, pantheon*), is being openly shared now in a Dropbox repository (Comas, 2023). See Figure 14 for sample images of the ParisComas dataset.

This new buildings dataset was appended to the Terrassa 900 to create a training set of 13+9 possible building classes. Test set wasn't updated, being only formed by Terrassa 900 images. Then we proceeded with the same architecture choices as in 4.3 and obtained the execution results of Figure 15. As opposed to 4.3, we can observe that using extra buildings data delays the overfitting effect in the testing loss, which now stops getting better around epoch 10. We believe this is thanks to the incorporation of extra training building images. However, the best accuracy obtained with this model is 53%, a lower classification capability than 4.3 in a similar amount of time over 20 epochs, 23 minutes of training. We can interpret that adding additional building images and classes help the model to be a more complete and complex tool, but with our Terrassa-only test set, at the end the result is worse than before when not using the ParisComas dataset.

The better classification results obtained with CustomCNN

showcases in a lesser amount of time show that downsizing the Terrassa 900 images to 32x32x3 and doing a short time training is enough for fast and reliable classification. In addition, it was proven that finetuning over the CustomCNN trained on Cifar10 indeed powered the model, being the best option to finetune the whole architecture. On the other hand, VGG16 tests outlined to us that freezing the pretrained initial layers do not allow the model to finally compete against CustomCNN, even with the fact that training is much slower because of the amount of trainable parameters. Also, overfitting in test loss after some epochs discarded the problem being that the model needed more training time. Future tests would involve training the full VGG16 network over pretrained and random weights (which will be a very long process) and seeing if that complexity boost would allow it to classify better than CustomCNN.

5. Task 5: Open project

The advent of freely accessible satellite imagery, such as the Sentinel-2 data in the Copernicus programme (ESA, 2023), has opened up new paths for earth observation applications. Land use and land cover classification is a fundamental task in utilizing this wealth of satellite data. In this project, we aim to apply advanced machine learning techniques to the EuroSAT dataset, a novel dataset comprising 27,000 labeled and geo-referenced images based on Sentinel-2 satellite imagery, which was presented in the paper titled *EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification* (Helber et al., 2019).

5.1. Description of the dataset

The original version has a multi-spectral nature, covering 13 spectral bands. However, the objective of the project is focused on the simplified color (RGB) version of the EuroSAT dataset, with an emphasis on understanding and classifying land use patterns. As mentioned before, the dataset includes 27,000 images categorized into 10 distinct land use and land cover classes, which include *Residential, Industrial, Highway, River, Pasture, Herbaceous Vegetation, Annual Crop, Permanent Crop, Forest* and *Sea/Lake*. An example of each one of the classification classes is depicted in Figure 16. Additionally, it is important to acknowledge that the size of all the images is 64 $\times$ 64 pixels.

5.2. Related work

In the context of classification, the authors of the dataset provide benchmarks for the novel dataset, which involve featuring 13 spectral bands and were conducted utilizing state-of-the-art deep Convolutional Neural Networks (CNNs). The outcomes demonstrated an impressive overall classification accuracy of 98.57% for a ResNet-50 network (He et al., 2016) and 98.18% for GoogLeNet (Szegedy et al., 2015). They performed several tries with the three different band combinations: color-infrared (CI), shortwave-infrared (SWIR) and RGB as input, with every optimal result lying between 97 and 98%.

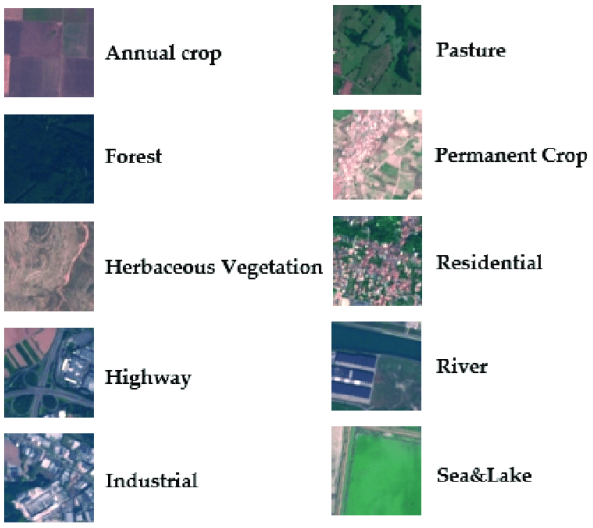


Figure 16: Sample image patches of the EuroSAT dataset

Besides, this dataset has also been used lately for other machine learning investigations. For example, researching, mapping, and conservation of wetlands is a challenging and time-consuming process. Because they produce temporal and geographical information, remote sensing and photogrammetric approaches are useful tools for analyzing and managing wetlands. In (Günen, 2022) the water areas of five different wetlands obtained with Sentinel-2 images in Turkey were classified, and the EuroSAT dataset was used in the validation process.

5.3. Data exploration and preparation

A thorough examination of the class distribution revealed a few classes that were slightly underrepresented, with Pasture standing out as having only 66% of instances compared to the most represented class (refer to Figure 17 for details). This observation prompted us to conduct in the posterior modelling part experiments considering both this imbalanced scenario and an augmented, balanced version of the dataset. Surprisingly, our results indicated that the classification performance of the underrepresented classes did not significantly suffer, as we'll see in the next sections. We attribute this resilience to the robustness of deep neural network structures in handling slight data imbalances, effectively mitigating potential issues associated with uneven class distributions.

Upon reviewing the various class previews in Figure 16, we observe certain commonalities and distinct contrasts among them. Classes depicting urban settings like *Highways*, *Residential*, and *Industrial* predominantly feature man-made structures and roadways. Both *AnnualCrops* and *PermanentCrops* classes display agricultural areas, characterized by straight lines marking different crop fields. In contrast, classes such as *HerbaceousVegetation*, *Pasture*, and *Forest* represent natural landscapes. *River*, also a part of the natural landscape category, are somewhat more distinguishable from these other natural classes. Analyzing the imagery content might give insights into

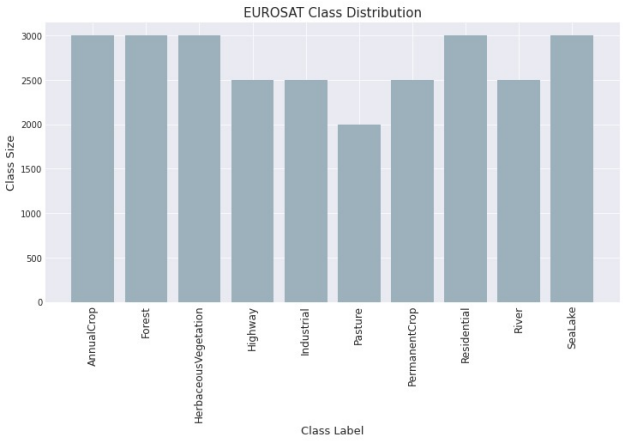


Figure 17: EuroSAT class distribution

potential class confusions. For instance, a *River* image could be misidentified as a *Highway*. Similarly, an image showing a highway junction with nearby buildings might be confused for an *Industrial* area. To overcome these challenges, it's crucial to develop a classifier that can adeptly discern these subtle differences.

For the data preparation phase, we adhered to the recommended train, validation, and test splits of 70-15-15 percent, with data suffling, as suggested by the authors. Additionally, we implemented image normalization using the mean and standard deviations from the ImageNet (Deng et al., 2009) dataset. This practice is commonly employed in image-based datasets, including the current example, as it leverages the robustness of these standardized values, ensuring consistent and effective preprocessing of the input images.

On the other hand, we did a data augmentation strategy, where we employed a set of transformations to augment the training dataset, aiming to enhance model robustness and improve generalization performance. The applied transformations include random horizontal and vertical flips, random rotations up to 60 degrees, and random resized cropping with varying scales and aspect ratios. See Figure 18 for actual examples. Additionally, we initially experimented with color jittering, involving adjustments to brightness, contrast, saturation, and hue; nonetheless, empirical results indicated that taking out this transformation yielded superior outcomes. Furthermore, we evaluated the impact of concatenating the original training dataset with the augmented one, finding that training solely on the augmented images produced more favorable results compared to the concatenated approach. This was a possibility, given that sometimes training with only the augmented dataset provides a better generalization capability for the test set, although you miss on the original data. Overall, these final decisions were made based on empirical performance assessments of later work, guiding our augmentation strategy towards configurations that demonstrated enhanced model learning and classification capabilities.

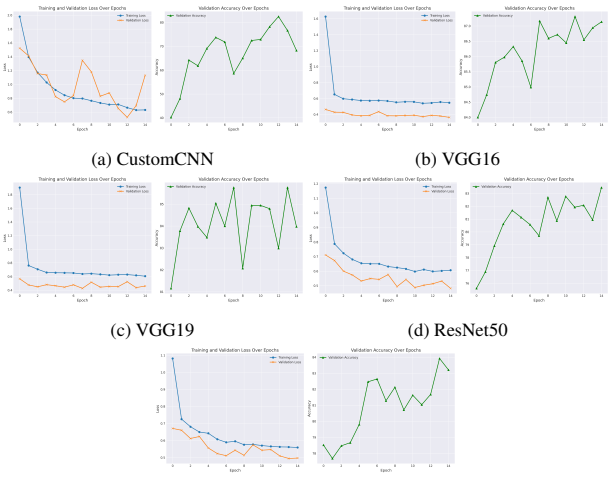
Figure 18: EuroSAT augmented dataset examples

5.4. Modeling

In our exploration of the CIFAR-10 dataset, we trained five distinct models to compare their performance in a static setup. These models include the CustomCNN from Task 1 (Table 1), pre-trained VGG16 and VGG19 (Simonyan and Zisserman, 2014) models with modified classifiers, and pre-trained ResNet50 and ResNet152 models with altered fully connected layers.

The architectures of the classifiers for the VGG and ResNet models were adjusted to fit our dataset's requirements. For instance, VGG models' classifiers were replaced with a sequence of linear layers, ReLU activations, and dropout layers, reducing the feature dimension step by step until reaching the number of classes. Similarly, the fully connected part of the ResNet models was substituted with a tailored architecture comprising linear layers interspersed with ReLU activations and dropout layers, as well. Details on the architectures of each specific model, can be found in the folder *open_project* of the delivered files (in the *architecture_{datetime}.txt* of each of the 5 models).

Data augmentation techniques as mentioned in Section 5.3 were applied to the training dataset during the comparison experiments. Each model was then trained with a batch size of 512 for 15 epochs using the Adam optimizer, with an initial learning rate of 0.001 and a learning rate factor for reduction set at 0.5, alongside a weight decay of 0.001 for regularization and the cross-entropy loss. Training logs, losses, and accuracy figures were saved for each model to monitor progress and performance. Moreover, the training process was meticulously executed, utilizing an early stopping mechanism with a patience of 5 epochs to prevent overfitting and a model checkpoint strategy to save the best performing model based on validation accuracy. The *ReduceLROnPlateau* scheduler was used to adjust the learning rate based on validation performance, with the aim of fine-tuning the models' convergence. The results of the com-

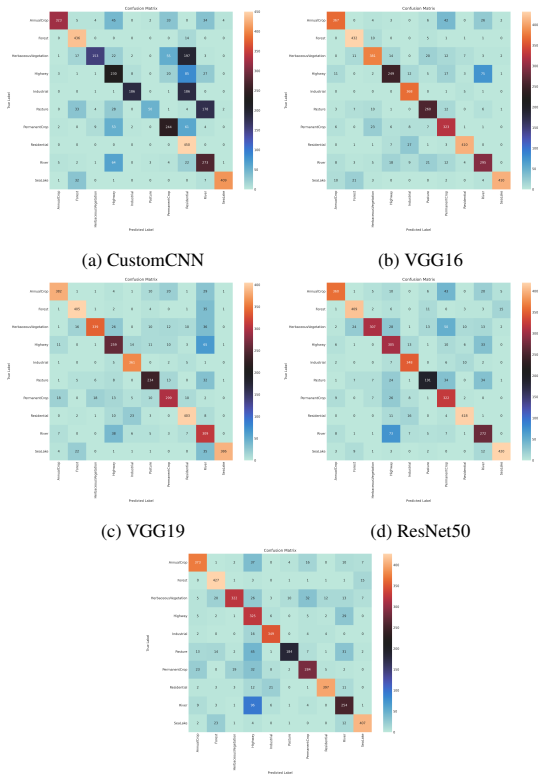


(a) CustomCNN (b) VGG16

(c) VGG19 (d) ResNet50

(e) ResNet152

Figure 19: Comparison of training and validation loss functions



(a) CustomCNN (b) VGG16

(c) VGG19 (d) ResNet50

(e) ResNet152

Figure 20: Comparison of test confusion matrices

parison between the 5 models are presented below.

Table 3 presents the number of total, frozen and trainable parameters in each case, together with the training time for each model and the final best validation accuracy. Furthermore Figures 19 and 20, present the training and validation losses of those 5 models together with their respective confusion matrices on the test set. Thus, by comparing the results of Table3 and Figures 19 and 20 it was decided that VGG16, was providing the best results, while it was only taking approx. 1 minute and 30 seconds more to train compared to the fastest trained model (ResNet50). At the same time, from the training-validation losses of all models, it is shown that their generalization is present and that, training for longer epochs could improve their overall performance, in all cases. Finally, performance-wise the *CustomCNN* architecture was achieving the worst results both in terms of training and validation, as well as for the test set. Additionally, results on the f1-score, recall and precision of predictions for all models, for each respective class are included in the delivered files, however, those results have exactly the same order as the accuracy measure, and for that reason it was decided to not be presented in this work. To conclude, the VGG16 model was selected for proceeding further, with specialized fine-tuning and attempts to achieve the greatest performance possible (Subsection 5.5).

5.5. Final Model

Continuing on, since the pre-trained VGG16 model with the modified classifier—comprising a sequence of linear layers, ReLU activations, and dropout layers—yielded the best results in the previous comparison, it was selected for the final fine-tuning and optimization tailored to the specific solution. The architecture of

Firstly, the hyperparameters of the training process were modified in order to achieve better results. Training epochs were modified to be equal to 100, and weight decay for regularization was set at 0.0001 to prevent overfitting for the larger number of epochs to be trained, while the remaining hyperparameters remained the same.

Then 4 separate tries were conducted. The first one includes only the changes in the hyperparameters mentioned in the previous paragraph. During the second attempt, the balance of the dataset and the inclusion of both the original and the augmented images took place. In the third try, only the augmented dataset was used while the removal of the color jittering effect in the data augmentation process was tested. And finally in the forth try, the set-up was exactly the same as the third approach, although just the patience threshold of early stopping mechanism was changed to 15.

The differences between those tries were very similar, with the only significant exception being the removal of the color jittering effect on the augmented dataset, and patience being equal to 15 (the 4th try -*VGG16_final3* in the repository's folders-, is considered the final model and the architecture of the system is also included in the respective file, inside the *VGG16_final3* folder). It is important to mention here that the training of the model was stopped by the early stopping mechanism at epoch 37 after 40 minutes and 21 seconds of training. The results of this approach are presented in the next subsection.

5.6. Postmodeling Analysis

The training and validation losses, together with the validation accuracy of the final model are presented in Figure 21. It is very interesting that with such configuration the model achieves a decent result of 91% accuracy on validation test,

Table 3: Model comparison

Model	Total Parameters	Trainable Params.	Non-Trainable Params.	Training Time	Validation Accuracy
CustomCNN	1,337,898	1,337,898	0	38m 19s	82.22%
VGG16	68,205,386	53,490,698	14,714,688	20m 1s	87.29%
VGG19	73,515,082	53,490,698	20,024,384	20m 16s	85.77%
ResNet50	26,136,138	2,628,106	23,508,032	18m 24s	83.46%
ResNet152	60,771,914	2,628,106	58,143,808	19m 22s	83.95%

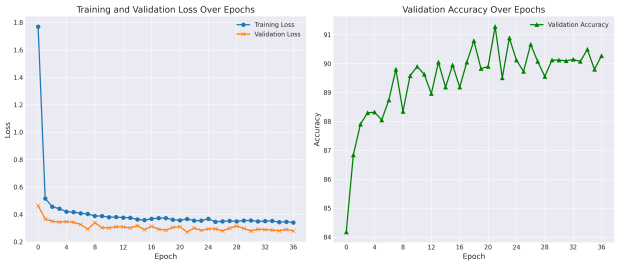


Figure 21: Final model: Loss and accuracy results

and 90% on the test set respectively. At the same time, it is observed that both the training and validations losses are continuously decreasing in parallel, showcasing that overfitting is absent. The confusion matrix of the final model for the test set is presented in Figure 22. From the confusion matrix, the worst classification case is present for the *Highway* class which is being confused with the *River* class, and vice versa. Consequently, the comments mentioned in Subsection 5.3, about the potential class confusions, were true. Usually, indeed a *River* image could be misidentified as a *Highway* and similarly, an image showing an *AnnualCrop* is ussually confused with *PermanentCrop*, or *HerbaceousVegetation* is confused with *PermanentCrop* which in both cases are very similar land-fields. Although, *Highway* and *River* are the classes with the worse results, and at the same time they have 2500 example images, compared to 3000 of the remaining classes (except *Pasture* that has 200) when balancing the dataset with resampling was tried, the results were the same, indicating the existance of bias in the predictability power of the model for the specific classes. More details on the class-specific metrics of accuracy, recall and f2-scores can be found in the delivered files.

5.7. Interpretability of the Final Model

To continue with, in this section, the interpretability of the model is taking place. It includes details on the activations of the model, the top 10 image examples of randomly selected units of the dense layers of the network, and a visualization of the dimensionality reduction of the activations, following techniques previously presented in Section 3. The inclusion of the filter visualization of the final model was considered reduntant, since similar results were presented in Section 3, and since the VGG16 architecture was used, the number of convolutional layers of the system was equal to 13, however, the images are included in the delivered files.

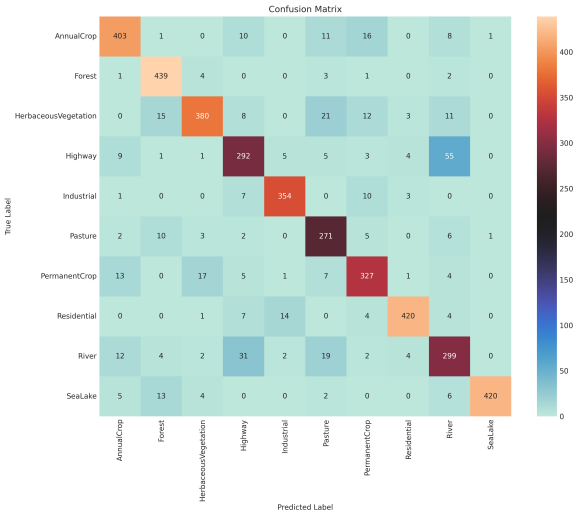
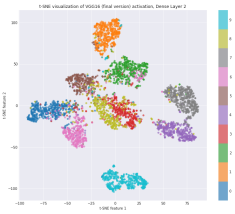
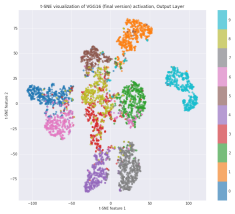


Figure 22: Final model: Confusion matrix on test set

(a) Dense Layer 1

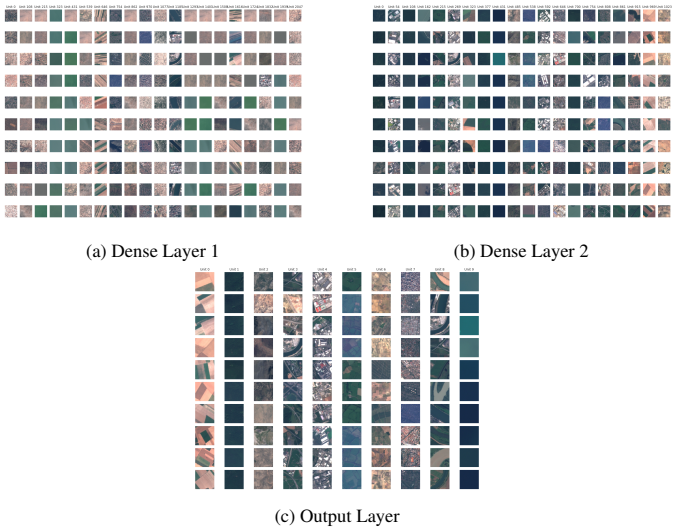


(b) Dense Layer 2



(c) Output Layer

Figure 23: t-SNE visualizations of the final model activations across different layers.



(a) Dense Layer 1

(b) Dense Layer 2

(c) Output Layer

Figure 24: Top 10 image examples for randomly selected units in each layer of the final model.

The t-SNE visualizations (Figures 23a, 23b, and 23c) demonstrate the clustering of the activation patterns within the dense layers of our final model. Distinct clusters indicate that the model is effectively discriminating between different features or classes, even from the first connected layer of the architecture. The progression from the first to the output layer shows a trend towards increased separation, suggesting that the network is refining its representations for clearer class discrimination. At the final layer, it can only be seen that the confusions between the classes is usually happening between classes 2, 3 and 8 (which respectively refer to *HerbaceousVegetation*, *Highway* and *River*).

Similarly, the top 10 image examples for randomly selected units (Figures 24a, 24b, and 24c) provide insights into the features that activate specific neurons. For instance, in the first dense layer we can already see some units that focus the activation on specific classes, such as Unit 646 for *AnnualCrop* or Unit 323 for *SeaLake*. Nonetheless, we can also observe that classification wouldn't be effective yet, as in this case we have also some *SeaLake* images on the top for other units that have much more entropy, which means that they still encompass various classes. As expected, for dense layer 2 the variability of images per unit is lower, and finally for the output layer we can see all 10 neurons clearly aimed for classification of each of the EuroSAT classes. As some examples, we have Unit 0 clearly directed to *AnnualCrop* class activation, Unit 4 for *Industrial* and Unit 8 for *River*.

5.8. Scientific and personal conclusions

On a personal level for both of us, navigating through this set of models and trying to find the optimal configuration underscored the iterative nature of deep learning experiments and the importance of thoughtful architecture design in achieving superior classification results. This experience not only widened our understanding of neural network dynamics but also cultivated an appreciation for the art and science of model selection and customization in the pursuit of optimal performance. There are a lot of possibilities when designing a model and decision making moments, so we've learnt that is key to document and go step-by-step in the process in order to compare the isolated effects of every transformation we make in order to find better performance.

5.9. Possible extensions and known limitations

Firstly, the current final configuration provides promising results, but there is room for exploring further complexities within the custom layers. Extending the custom layers to a more intricate level and systematically assessing the impact on model performance could offer more insights into the optimal depth and structure for improved classification accuracy, we could try until which complexity the custom layers can arrive and aim for getting those extra correct classifications. Additionally, while our exploration encompassed various deep convolutional robust methods, an intriguing trend for future investigation involves experimenting with transformer-based structures for image classification. This initiative could be particularly valuable with an expanded dataset or leveraging the multi-spectral information available in the EuroSAT dataset. Determining the point at which transformers become advantageous and understanding their efficacy in the context of EuroSAT's characteristics could lead to enhanced classification outcomes, so that could be a possible extension of this research. Finally, a known limitation of the current model pertains to the classification challenges associated with the River class, which reduces the overall results a bit. One potential extension involves the implementation of an ensemble architecture, which means taking into account more than one model in order to finally give a more robust prediction. By introducing an additional model or mechanism specifically designed to address the complexities of the *River* class, we could aim to improve overall classification accuracy. This ensemble approach could include a dedicated model for *River* class predictions, complementing the existing architecture and contributing to a more nuanced and strong classification system.

6. Conclusion

In the course of this project, we deepened into the multifaceted exploration into various aspects of deep learning, encompassing tasks such as investigating the intricacies of deep and specifically convolutional modeling, assessing the impact of regularization, delving into transfer learning techniques, and interpreting the behaviors of these sophisticated models. The comprehensive nature of this step-by-step project afforded us a robust foundation for understanding the nuances of deep learning methodologies. Despite the challenges posed by time, given the substantial workload, we view this project at the end as highly valuable for our learning goals. Through our efforts, we not only encountered and overcame obstacles but also gained profound insights and knowledge. Therefore, the experience has undeniably contributed to our growth and improvements in the field of deep learning.

References

Comas, P., 2023. The ParisComas dataset. https://www.dropbox.com/scl/fo/2ksvzeqwahh3nuoodohc5/h?rlkey=w5gv115837ucyij5qzg6fbzuj&dl=0.

Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L., 2009. Imagenet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, IEEE. pp. 248–255.

ESA, E.S.A., 2023. The Copernicus Programme. https://sentiwiki.copernicus.eu/web/copernicus-programme.

Günen, M.A., 2022. Performance comparison of deep learning and machine learning methods in determining wetland water areas using eurosat dataset. Environmental Science and Pollution Research 29, 21092–21106. doi:https://link.springer.com/article/10.1007/s11356-021-17177-z.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778.

Helber, P., Bischke, B., Dengel, A., Borth, D., 2019. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 12, 2217–2226.

Krizhevsky, A., 2009. CIFAR-10 and CIFAR-100 datasets. https://www.cs.toronto.edu/~kriz/cifar.html.

Philbin, J., Zisserman, A., 2017. The Paris dataset. https://www.robots.ox.ac.uk/~vgg/data/parisbuildings/.

Simonyan, K., Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 .

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.

UPC, I.P.G., 2015. Terrassa 900 dataset. https://imatge.upc.edu/web/resources/terrassa-buildings-4126.