UNIVERSITÀ DI PISA

Artificial Intelligence and Data Engineering

Computational Intelligence and Deep Learning

Satellite Images Classification with Convolutional Neural Networks

Members:

Matteo Pasqualetti

Daniele Giaquinta

Contents

1 Introduction

In recent years, satellite imagery has become a valuable source of information for a wide range of applications, including environmental monitoring, urban planning, disaster management, and agriculture. The ever-increasing availability of high-resolution satellite data has led to an unprecedented opportunity to harness the power of artificial intelligence and machine learning for satellite image analysis.

Convolutional Neural Networks (CNNs) have emerged as a powerful tool in the field of computer vision, demonstrating exceptional capabilities in image recognition, classification, and feature extraction. With their ability to automatically learn relevant features from raw pixel data, CNNs have the potential to revolutionize the way we analyze and interpret satellite images.

The primary objective of this project is to explore the application of Convolutional Neural Networks for the classification of satellite images. Satellite image classification involves the task of assigning specific categories or labels to each image based on the presence of distinct features, patterns, or objects. The utilization of CNNs for this purpose holds the promise of achieving high accuracy and robustness, even in the presence of diverse and complex image backgrounds.

In this paper, we delve into the methodology of training and fine-tuning CNN architectures for satellite image classification. We will discuss the preprocessing steps required to prepare the satellite image data for model input, the design choices for CNN architectures tailored to satellite image analysis, and the strategies employed for model optimization and evaluation.

1.1 Dataset

The Dataset used in this paper is based on Sentinel-2 satellite imagery covering 13 spectral bands and consists of 10 LULC classes with a total of 27,000 labeled and geo-referenced images.

The dataset is associated with the publications "Introducing EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification"[3] and "EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification"[4].

2 Use Cases

A CNN model designed to classify satellite images into distinct land uses has numerous practical applications. It can assist urban planners in determining optimal zones for residential, commercial, and industrial areas. In agriculture, the model may be exploited for monitoring crop health and yield prediction. For environmental purposes, it helps track changes in natural landscapes and supports disaster response by identifying affected regions. In real estate, it offers insights into property features and valuation (easily identify abusive buildings). Additionally, the model may be exploited in transportation planning, environmental impact assessments, wildlife conservation, climate change analysis, and infrastructure maintenance.

Classifying small images (64x64 pixels) offers advantages in both resolution and coverage area compared to models that classify larger images.

In terms of resolution, classifying small images is advantageous because end users or applications may not always have access to high-resolution pictures. Achieving high performance with these small images ensures that high performance can also be achieved with larger pictures.

In terms of coverage area, classifying small images enables a single image to encompass a smaller geographic region. This can be advantageous when dealing with large landscapes or dense urban environments. The model can cover a broader range of locations within a dataset, making it suitable for tasks such as urban planning, land use classification, and disaster assessment.

3 Related Works

The papers that served as inspiration for our project are the ones we previously cited [3, 4]. The main disparity in terms of outcomes between our work and theirs lies in our model's performance. While their final outcome attains an accuracy of 98.57%, our ensemble model achieves a score of 95.26%. It's worth noting that even though we don't possess an accuracy advantage, our models attain this score using lightweight JPG files of approximately 4KB each. In contrast, our colleagues employed the complete 107KB images across 13 spectral bands. Our objective is to create an expedient model that operates efficiently with lightweight data.

A more recent study titled "Analysis of deep learning architecture for patch-based land cover classification" has been published [2]. This work seems to employ techniques quite similar to ours, yet it appears to achieve a test accuracy of only 87.8%.

A similar classification problem, utilizing the same dataset, but approached using traditional machine learning techniques instead of deep learning, could also be of interest. This work is titled "Land-Use Land-Cover Prediction from Satellite Images using Machine Learning Techniques" [1].

4 Preprocessing

Since **Pasture** class presented a little bit less elements than other classes, we decided to add 400 more elements after Train/Test split.

	Training	Validation	Test	Total
Before Re-balancing	1440	160	400	2000
After Re-balancing	1800	200	400	2400
River	1800	200	500	2500

Table 1: Pasture Class composition

To increase the number of elements, we decided to take 400 unique random image from the same class and rotate or flip them, by using the code listed at page 7.

Because of Google Colab limitations, we weren't be able to use the 13 spectral bands images nor we wanted to, we therefore used 64x64 pixels images compressed in jpg. We splitted our Dataset in Train Set and Test Set by a ratio of 80%/20%, then Validation Set was extracted from Train Set with a ratio of 90%/10%:

	Training	Validation	Test	Total
AnnualCrop	2160	240	600	3000
Forest	2160	240	600	3000
HerbaceousVegetation	2160	240	600	3000
Highway	1800	200	500	2500
Industrial	1800	200	500	2500
Pasture	**1440**	**160**	**400**	**2000**
PermanentCrop	1800	200	500	2500
Residential	2160	240	600	3000
River	1800	200	500	2500
SeaLake	2160	240	600	3000
Total	*Data 5*	*Data 6*	*Data 6*	*27000*

Table 2: Dataset Composition

4.1 Important functions

The following functions were used in the process just described and during the whole project.

```
# RAN_SEED = 10024062
def set_seed():
    os.environ["PYTHONHASHSEED"]="0"
    np.random.seed(RAN_SEED)
```

```
    rn.seed(RAN_SEED)
    tf.random.set_seed(RAN_SEED)
```

Listing 1: Seeds initialization

```
# VALIDATION_SPLIT = 0.1
# BATCH_SIZE = 32
def load_datasets(BATCH_SIZE):
    set_seed()

    train = tf.keras.preprocessing.image_dataset_from_directory(
        TRAIN_DIR, labels='inferred', label_mode='categorical', class_names=None,
        color_mode='rgb', batch_size=BATCH_SIZE, shuffle=True, seed=RAN_SEED,
        validation_split=VALIDATION_SPLIT, subset='training', follow_links=False,
        image_size=(64,64)
    )

    val = tf.keras.preprocessing.image_dataset_from_directory(
        TRAIN_DIR, labels='inferred', label_mode='categorical', class_names=None,
        color_mode='rgb', batch_size=BATCH_SIZE, shuffle=True, seed=RAN_SEED,
        validation_split=VALIDATION_SPLIT, subset='validation', follow_links=False,
        image_size=(64,64)
    )

    test = tf.keras.preprocessing.image_dataset_from_directory(
        TEST_DIR, labels='inferred', label_mode='categorical',
        class_names=None, color_mode='rgb', batch_size=BATCH_SIZE, shuffle=True,
        seed=RAN_SEED, follow_links=False, image_size=(64,64)
    )

    return train, val, test
```

Listing 2: Splits creation

```
def rotate_or_flip(img):
    flip = [Image.FLIP_LEFT_RIGHT, Image.FLIP_TOP_BOTTOM]
    rotate = [Image.ROTATE_90, Image.ROTATE_180, Image.
   ROTATE_270]

    rotOrFlip = random.randint(0,1)
    if rotOrFlip:
        return img.transpose(flip[random.randint(0,1)])
    else:
        return img.transpose(rotate[random.randint(0,1)])
# -----
pastureDir = os.path.join(TRAIN_DIR, "Pasture")
pastureList = os.listdir(pastureDir)

toAugment = random.sample(pastureList, 400)
for i in toAugment:
    imgPath = os.path.join(pastureDir, i)
    img = Image.open(imgPath)
    mod = rotate_or_flip(img)
    mod.save(pastureDir + "/MOD" + i)
```

Listing 3: Rebalancing

5 CNN from Scratch

In this chapter, we present the outcomes of several architectures designed to address the given problem. We opted for an iterative approach, progressively devising more intricate and detailed attempts to discover the optimal model.
Every model was trained using these configurations:

- optimizer: *ADAM*
- batch size: 32
- learning rate: optimizer's default
- early stop patience: 10
- checkpoint: val_loss

Unless otherwise specified, the ReLU function will be employed in the inner layers, and the Softmax function will be used in the output layer. Additionally, for convolutional layers, a 3x3 kernel size will consistently be applied.

Moreover, following the input layer, a Rescaling layer will always be included

```
def compile_fit(model, train, eps, valid):

    earlyStop = callbacks.EarlyStopping(monitor = 'val_loss',
                                mode = 'min',
                                min_delta = 0,
                                patience = 10,
                                restore_best_weights = True)

    checkName = model.name + ".h5"
    modelPath = os.path.join(MODELS_DIR, checkName)

    checkpoint = callbacks.ModelCheckpoint(filepath =
   modelPath,
                                    monitor = 'val_loss',
                                    mode ='min',
                                    save_best_only = True)

    model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

    return model.fit(
             train,
             epochs=eps,
             validation_data=valid,
             callbacks=[earlyStop, checkpoint])
```

Listing 4: Compilation and fit

5.1 Test 1: Simple model

As the initial model, we developed a basic CNN comprising Conv2D layers to process images with 16, 32, and 64 filters, respectively. Its architecture is illustrated in Figure 1.

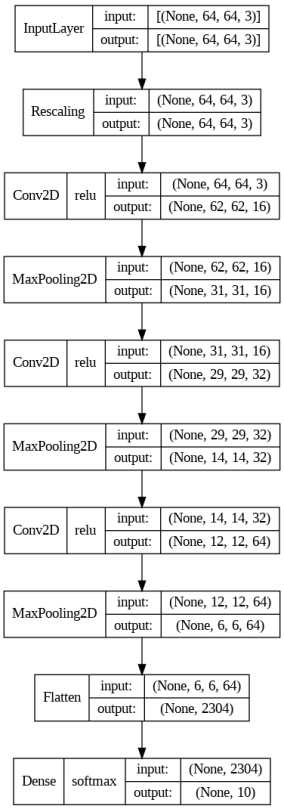
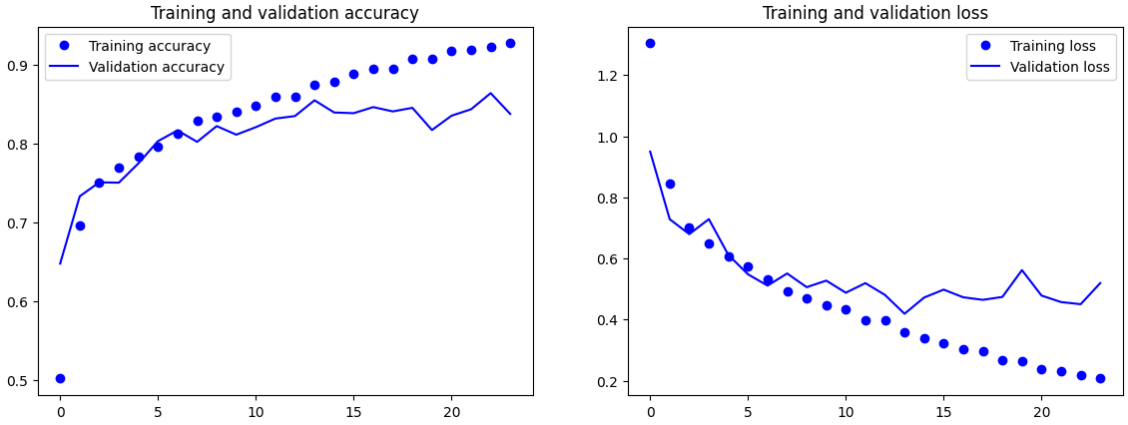


Figure 1: Model 1 Architecture

5.1.1 Training Results

Epochs	Train Loss	Train Accuracy	Validation Loss	Validation Acc.
24	0.3594	87.42%	0.4191	85.50%



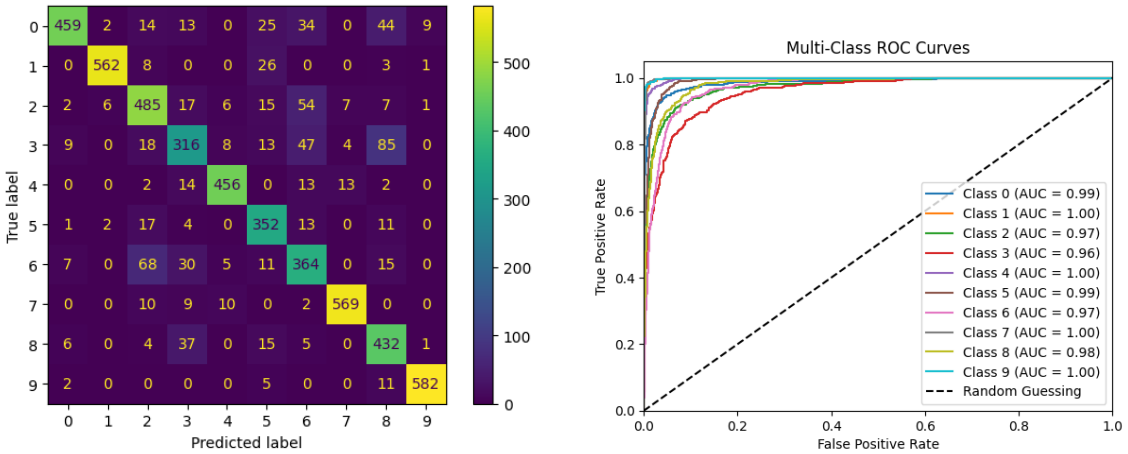
(a) Accuracy (b) Loss

Figure 2: Training results Model 1

As indicated in Table 3 and Figure 2, the results are quite promising considering the model's simplicity. However, there is an issue with overfitting.

This is primarily attributed to the absence of regularization techniques, such as Dropout or L1/L2 Regularization.

5.1.2 Test Results



(a) Confusion Matrix (b) ROC Curve

Figure 3: Test results Model 1

5.2 Test 2: Bigger model and Dropout

For the second test, our aim was to enhance the preceding model by augmenting the count of Conv2D layers to 32, 64, and 128, and incorporating a Dropout layer with a rate of 0.2 after the Flatten layer. The architecture is outlined in Figure 4.

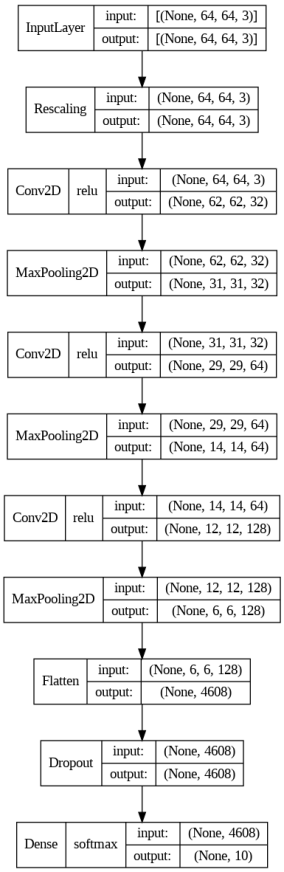
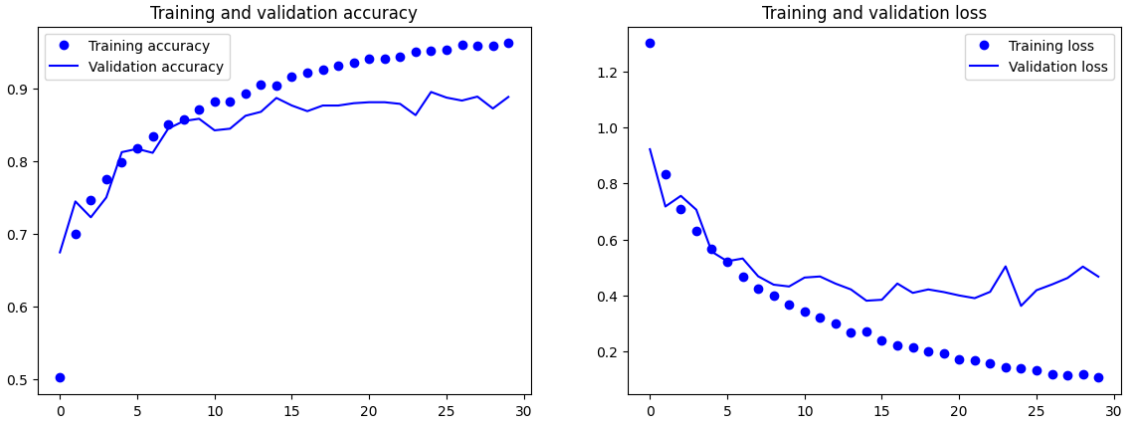


Figure 4: Model 2 Architecture

5.2.1 Training Results

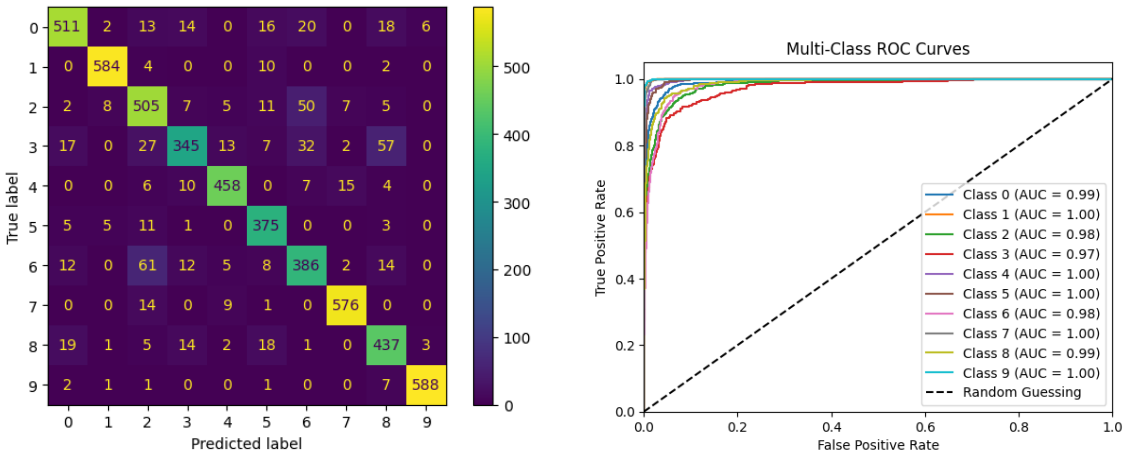
Epochs	Train Loss	Train Accuracy	Validation Loss	Validation Acc.
30	0.1392	95.15%	0.3621	89.50%



(a) Accuracy (b) Loss

Figure 5: Training results Model 2

The achieved results show a slight improvement compared to the previous model; however, addressing overfitting remains a concern that requires attention.



(a) Confusion Matrix (b) ROC Curve

Figure 6: Test results Model 2

Considering the outcomes on these two datasets, it becomes apparent that the majority of issues stem from misclassifications of classes 2, 3, and 6. To delve deeper:

- Class 2 **HerbaceousVegetation** classified as 6 **PermanentCrop**
- Class 6 **PermanentCrop** classified as Class 2 **HerbaceousVegetation**
- Class 3 **Highway** classified as Class 8 **River**

This observation is comprehensible, given the similarity between classes 2 and 6. However, the same does not hold for classes 3 and 8: Highways are more prone to being misinterpreted as Rivers, rather than the other way around.

Based on these findings, it can be inferred that the network is not yet proficient in distinguishing between these classes.

5.2.2 Test Results

Class	Precision	Recall	F1-Score	Support
0	0.9187	0.8100	0.8609	600
1	0.9605	0.9733	0.9669	600
2	0.7942	0.8617	**0.8265**	600
3	0.8330	0.7280	**0.7769**	500
4	0.9408	0.9220	0.9313	500
5	0.8747	0.9075	0.8908	400
6	0.7884	0.7900	**0.7892**	500
7	0.9493	0.9667	0.9579	600
8	0.7898	0.9020	0.8422	500
9	0.9830	0.9617	0.9722	600
Accuracy			0.8848	5400
Macro Avg	0.8797	0.8796	0.8781	5400
Weighted Avg	0.8840	0.8824	0.8818	5400

5.3 Test 3: Dense layer, L2, Dropout

In an effort to mitigate the overfitting issues observed, we opted to escalate the utilization of Dropout between the layers. Additionally, we introduced a Dense layer with 64 neurons, incorporating L2-Regularization and ReLU activation, just before the output layer.

Despite employing the EarlyStop technique, we chose to double the number of epochs in case the model could benefit from more extended training.

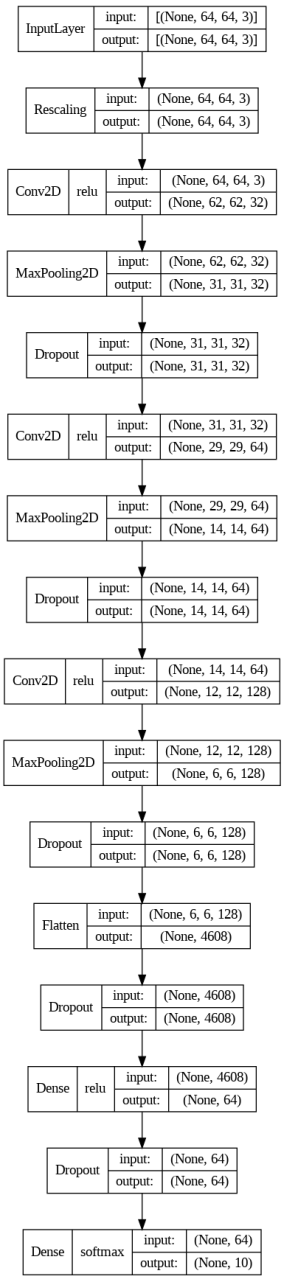
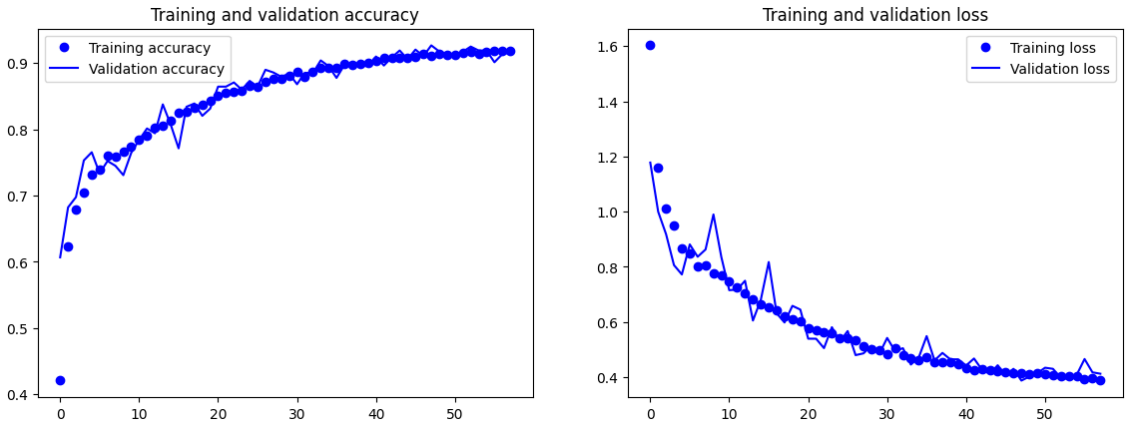


Figure 7: Model 3 Architecture

5.3.1 Training Results

Epochs	Train Loss	Train Accuracy	Validation Loss	Validation Acc.
58	0.4150	91.09%	0.3875	92.68%



(a) Accuracy (b) Loss

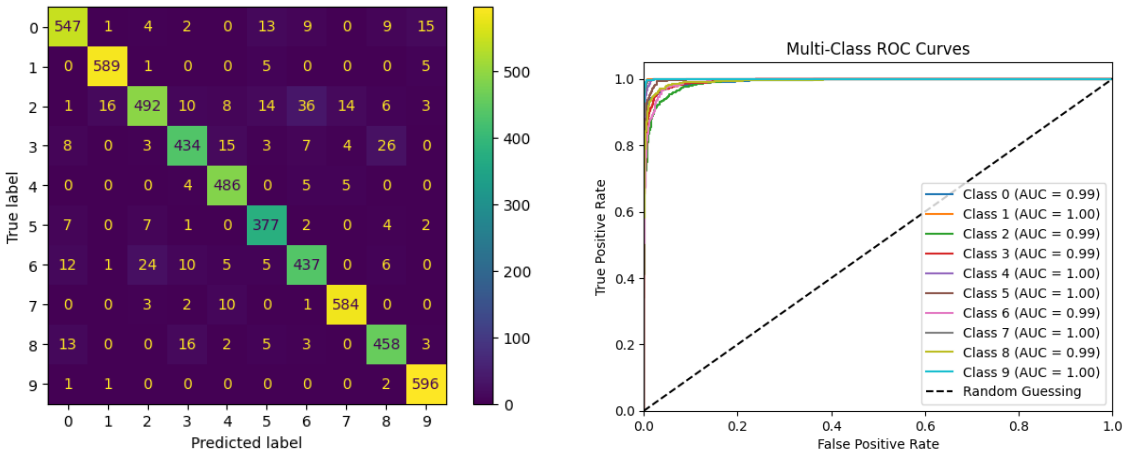
Figure 8: Training results Model 3

As evident from the training results, we successfully managed to eliminate overfitting.

The decision to amplify dropout rates and incorporate a normalization layer has proven highly effective.

Now, our focus is on refining the model even further, with the objective of increasing Validation Accuracy while simultaneously ensuring minimal Loss. This iterative process aims to strike an optimal balance between training and generalization, thus leading to a more robust and proficient model.

5.3.2 Test Results



(a) Confusion Matrix (b) ROC Curve

Figure 9: Test results Model 3

As depicted in the Confusion Matrix presented in Figure 9.a, we continue to encounter issues related to misclassifications on the same classes as observed during Test 2. This pattern suggests that the model is still struggling to effectively discern the distinctions between these particular classes.

Class	Precision	Recall	F1-Score	Support
0	0.9287	0.9117	0.9201	600
1	0.9688	0.9817	0.9752	600
2	0.9213	0.8200	**0.8677**	600
3	0.9061	0.8680	**0.8866**	500
4	0.9240	0.9720	0.9474	500
5	0.8934	0.9425	0.9173	400
6	0.8740	0.8740	**0.8740**	500
7	0.9621	0.9733	0.9677	600
8	0.8963	0.9160	0.9060	500
9	0.9551	0.9933	0.9739	600
Accuracy			0.9259	5400
Macro Avg	0.9230	0.9253	0.9236	5400
Weighted Avg	0.9258	0.9259	0.9253	5400

Table 3: Classification Report

5.4 Test 4: Deeper Network and additional Dense layer

In this particular test, we are embarking on a more extensive enhancement of the model by introducing an additional Dense layer and an extra Conv2D layer. The composition of the model is visually depicted in Figure 10. This augmented architecture aims to capitalize on the capabilities of deeper layers for improved feature extraction and classification performance.

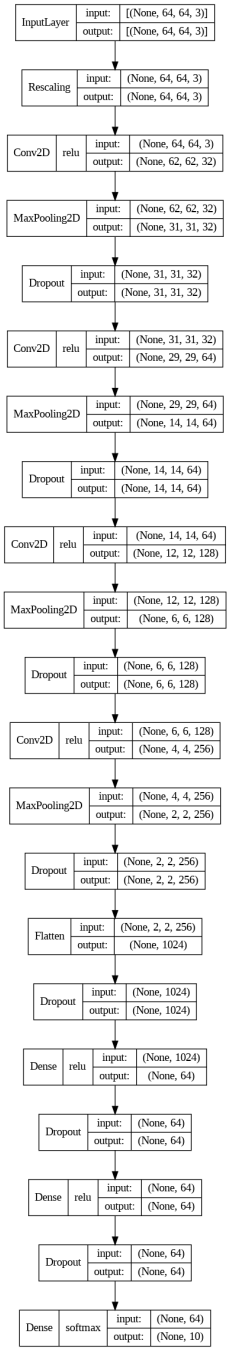
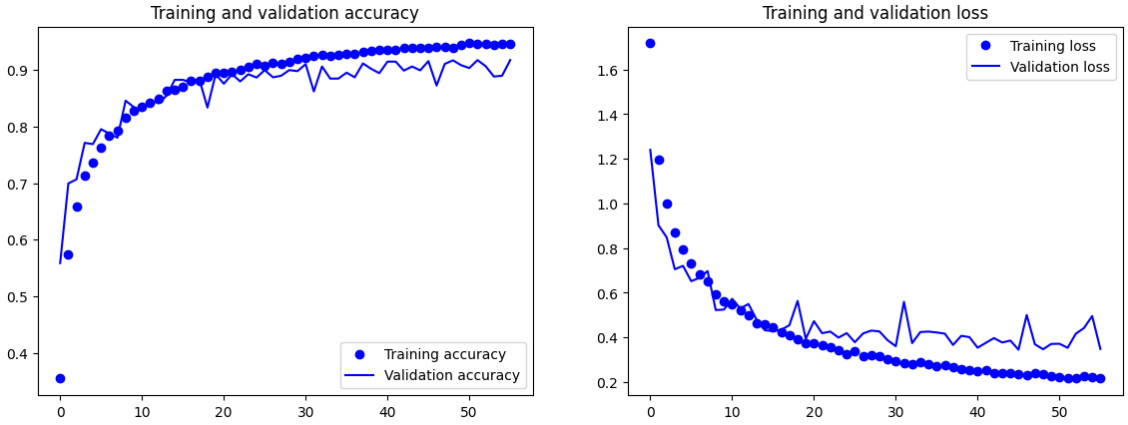


Figure 10: Model 4 Architecture

5.4.1 Training Results

Epochs	Train Loss	Train Accuracy	Validation Loss	Validation Acc.
56	0.2357	93.97%	0.3447	91.59%

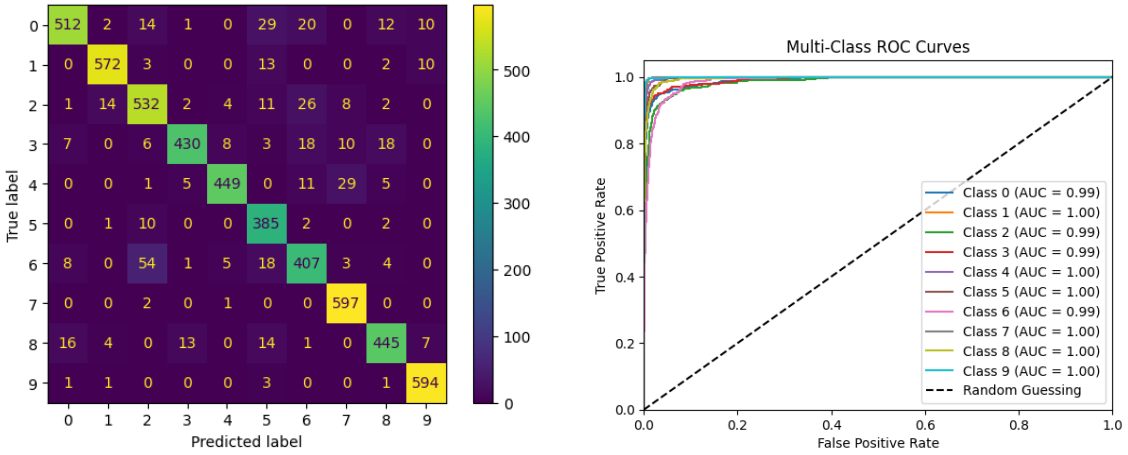


(a) Accuracy (b) Loss

Figure 11: Training results Model 4

Despite achieving a notably high accuracy, the introduction of new layers has once again resulted in a slight resurgence of overfitting tendencies. This trade-off between improved performance and potential overfitting highlights the delicate balance that needs to be struck when extending the model's complexity.

5.4.2 Test Results



(a) Confusion Matrix (b) ROC Curve

Figure 12: Test results Model 4

5.5 Test 5: Doubled Conv2D and increased Dropout

As illustrated in Figure 13, we initiated this iteration from Model 3 and proceeded to double the configuration of each Conv2D layer. Additionally, we slightly elevated the Dropout rate at each Dense layer. This strategic adjustment aims to counteract potential overfitting challenges that can emerge due to the augmentation of these layers. This approach endeavors to strike a balance between model complexity and the risk of overfitting, thereby promoting both enhanced performance and robust generalization.

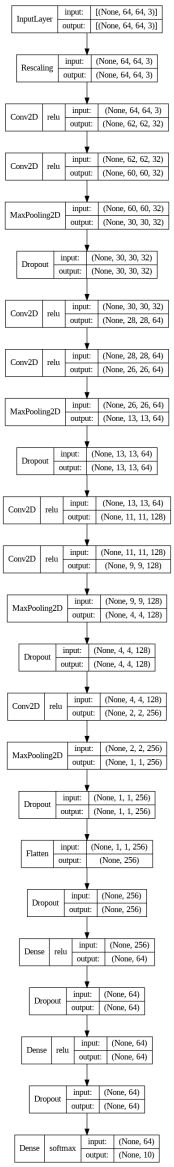
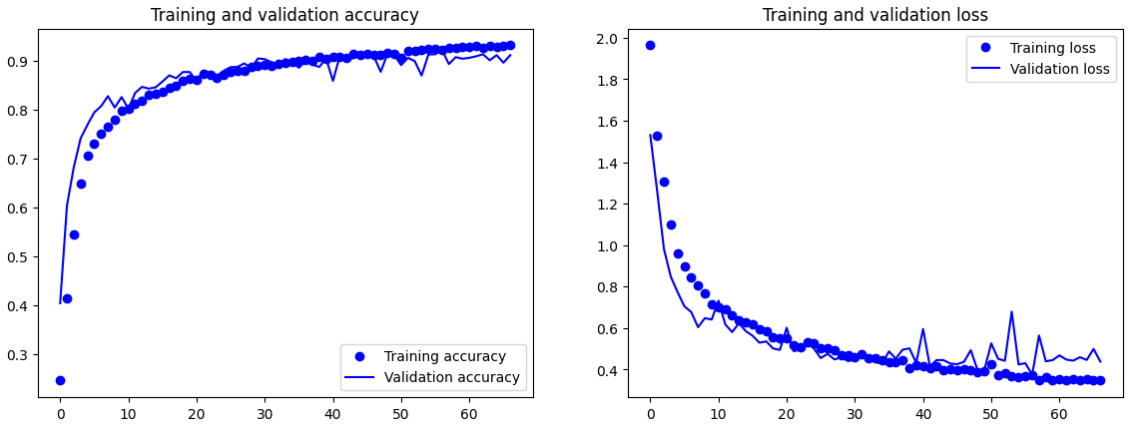


Figure 13: Model 5 Architecture

5.5.1 Training Results

Epochs	Train Loss	Train Accuracy	Validation Loss	Validation Acc.
67	0.3723	92.34%	0.3749	92.27%

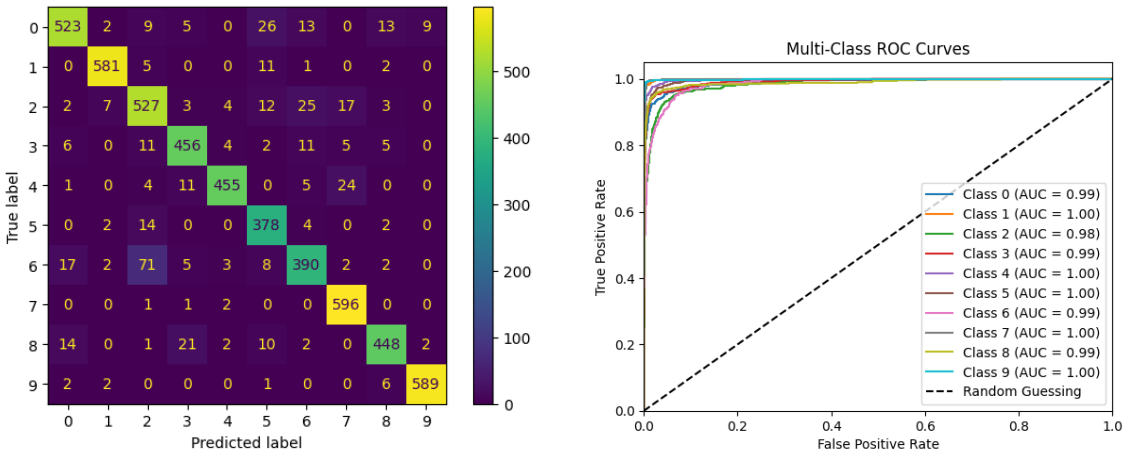


(a) Accuracy (b) Loss

Figure 14: Training results Model 5

The outcomes closely resemble those of Model 3, with one notable distinction: the Validation Scores exhibit a higher degree of variability and noise compared to previous iterations. This increased variability could be attributed to the adjustments made in layer configurations and dropout rates, leading to a more intricate interplay between training and validation performance.

5.5.2 Test Results



(a) Confusion Matrix (b) ROC Curve

Figure 15: Test results Model 5

As indicated by the Confusion Matrix, a gradual shift in challenges is discernible, primarily centered around classes 2 and 6. This suggests that the model is beginning to address and concentrate its difficulties on these specific classes, while other aspects of classification are becoming more stable.

5.6 Test 6: Model 5 no EarlyStop

For the final test, we opted to select Model 5 and train it over 100 epochs to evaluate the results comprehensively. While retaining the Checkpoint mechanism to obtain the best model, we excluded the EarlyStop strategy.

To address the mild overfitting observed in Model 5, we slightly increased the Dropout rates within the Dense layers. Moreover, we introduced L2 regularization in the second Dense layer. This strategic refinement aims to fine-tune the model's balance between training performance and generalization capabilities.

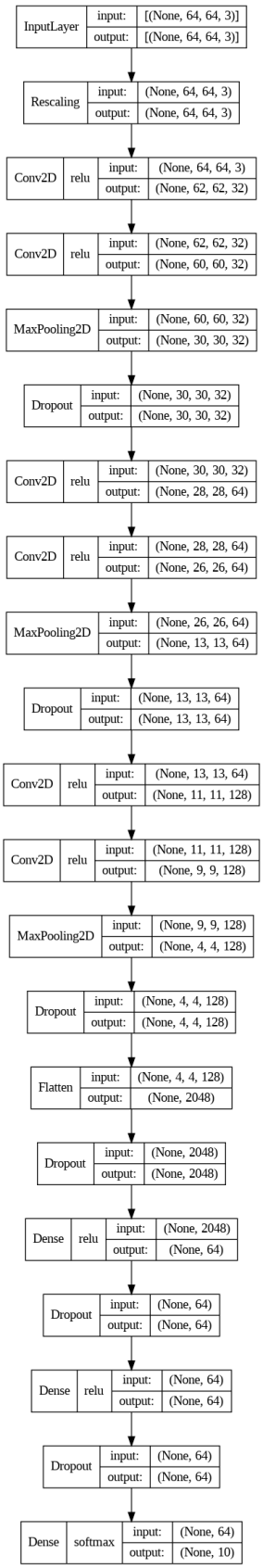
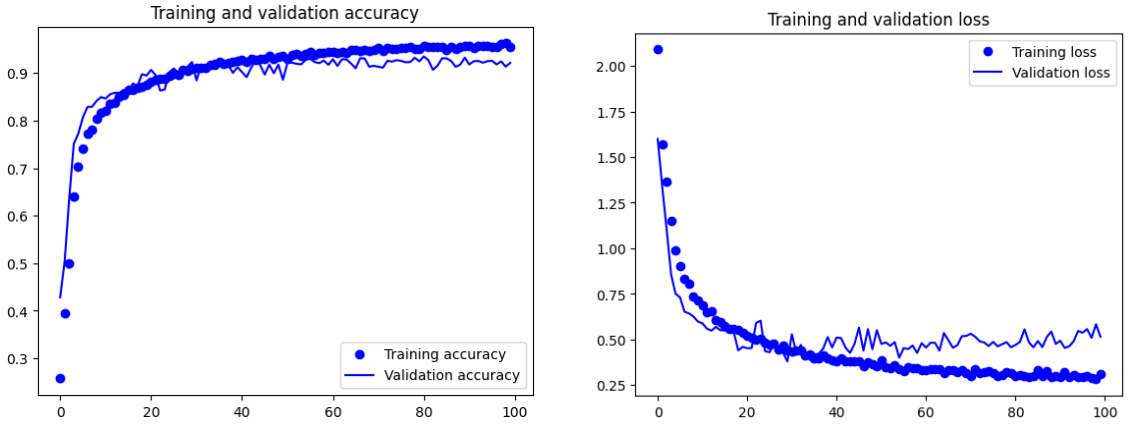


Figure 16: Model 6 Architecture

5.6.1 Training Results

Epochs	Train Loss	Train Accuracy	Validation Loss	Validation Acc.
30	0.4412	90.83%	0.3796	92.32%

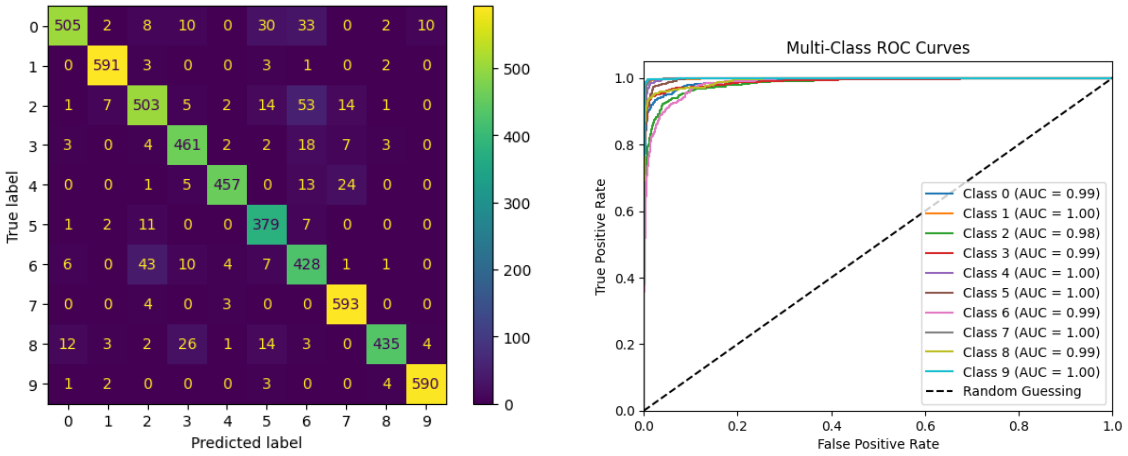


(a) Accuracy (b) Loss

Figure 17: Training results Model 6

Even though this model's performance is comparable to that of Model 3, it exhibits a pattern of gradually increasing overfitting tendencies after approximately epoch 60. This observation highlights the delicate challenge of balancing performance gains with the potential risks of overfitting, especially in longer training scenarios.

5.6.2 Test Results



(a) Confusion Matrix (b) ROC Curve

Figure 18: Test results Model 6

As show again by Confusion Matrix in Figure 18.a, we didn't achieve to get rid of Classes 2 and 6 missclassification, but we achieved to reduce a lot Class 3 and 8.

5.7 Recap and considerations

In this chapter, we explored various models to address the classification challenge.

Through our analysis, we observed that the incorporation of techniques such as Dropout and Normalization played a crucial role in mitigating overfitting concerns within our context.

Although the topic of Augmentation was not extensively covered in any of the conducted tests, we refrained from delving into it to maintain the chapter's concise length. Notably, we found that Augmentation did not yield performance improvements across any of the test cases, highlighting its limited impact in our specific scenario.

Model	**Train Loss**	**Train Accuracy**	**Validation Loss**	**Validation Acc.**
Model 1	0.3594	87.42%	0.4191	85.50%
Model 2	0.1392	95.15%	0.3621	89.50%
Model 3	**0.4150**	**91.09%**	**0.3875**	**92.68%**
Model 4	0.2357	93.97%	0.3447	91.59%
Model 5	0.3723	92.34%	0.3749	92.27%
Model 6	0.4412	90.83%	0.3796	92.32%

Table 4: Final results

After careful evaluation, we have chosen Model 3 as the most suitable candidate for our best model. This decision is grounded in its ability to strike an optimal balance between accuracy, loss, and simplicity. This model's performance, characterized by a harmonious interplay of these key factors, aligns well with our objective of achieving effective classification while maintaining a manageable and interpretable architecture.

6 Pre-Trained Models

We chose VGG16 and ResNet50 from Keras as our pre-trained models. VGG16 is a deep neural network architecture known for its simplicity and effectiveness in image classification. It consists of 16 layers, mostly using 3x3 convolutional filters and 2x2 max-pooling layers. VGG16 captures features at different scales, making it useful for tasks like recognizing objects in images. Its pre-trained weights enable quick adaptation to new tasks, making it a popular choice for transfer learning.

Using VGG16 for satellite image land use classification is beneficial due to its hierarchical feature extraction, pre-trained weights, transfer learning potential, ability to handle large input sizes, and suitability for diverse landscapes. It offers good generalization and customization while aiding visual interpretation of features.

ResNet50 is a powerful deep neural network architecture renowned for its innovation in tackling the vanishing gradient problem through residual connections. With 50 layers, it's designed to capture intricate features and patterns, making it highly effective for image classification tasks. ResNet50's pre-trained weights allow seamless adaptation to new tasks, rendering it a popular choice for transfer learning.

Employing ResNet50 for satellite image land use classification offers distinct advantages. Its advanced residual connections enable it to efficiently capture complex relationships within the data, allowing for the recognition of nuanced land use patterns. The pre-trained weights empower quick model customization, particularly valuable when working with limited labeled data. ResNet50 accommodates larger input sizes, a crucial trait for processing satellite images. Its adaptability to diverse landscapes, coupled with its interpretability, makes it a suitable candidate for enhancing visual insights.

6.1 VGG16 Simple Model

After importing the VGG16 convolutional base the first simple model was created as follows:

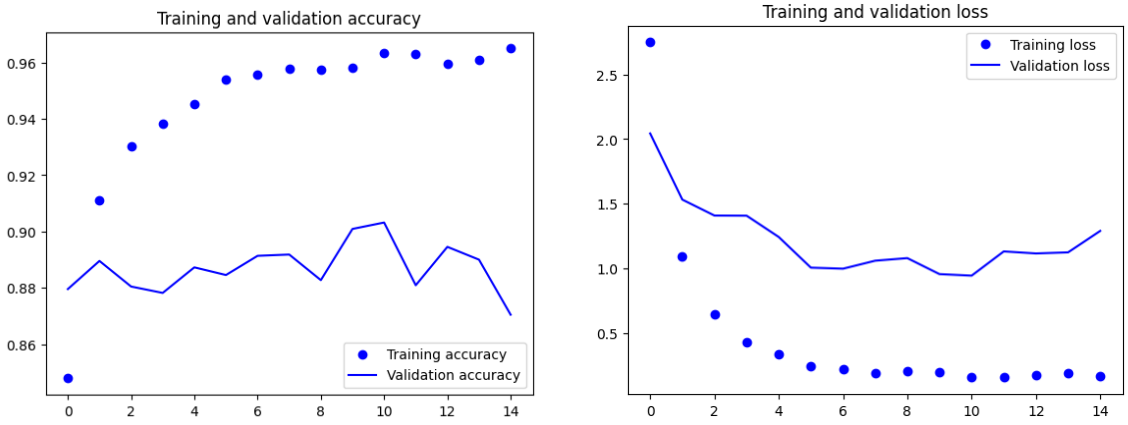
```
inputs = keras.Input(shape=(64, 64, 3))
x = inputs
x = keras.applications.vgg16.preprocess_input(x)
x = conv_base(x)
x = layers.Flatten()(x)
x = layers.Dense(256)(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="categorical_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])
```

Listing 5: VGG16 base

The data are preprocessed using the library function and are then sent into the CNN. The data is then flattened and goes through a 256-neurons dense layer. The output is of course the probability distribution for the classes, the chosen optimizer function is Adam and the loss function is categorical cross entropy.

6.1.1 Training Reuslts

Epochs	Train Loss	Train Accuracy	Validation Loss	Validation Acc.
15	0.1567	96.34%	0.9434	90.32%



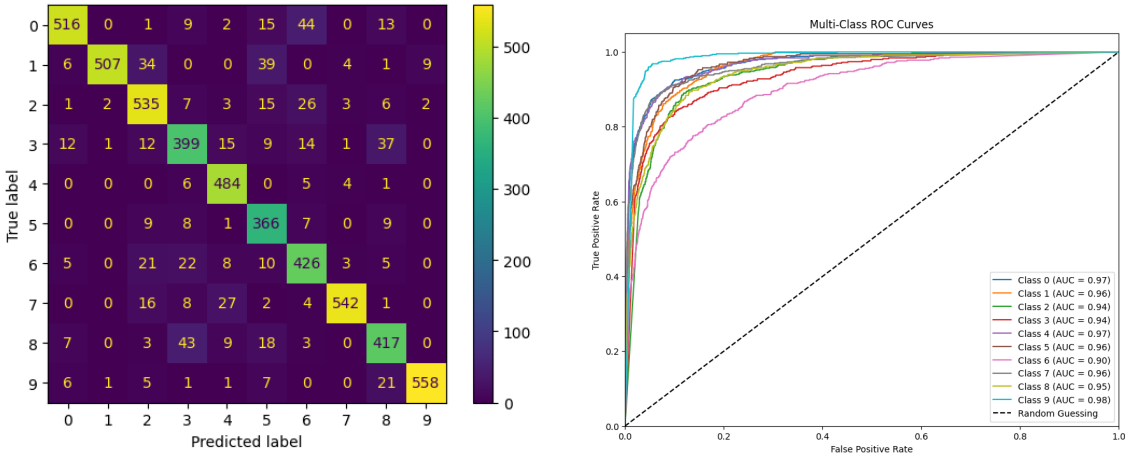
(a) Accuracy (b) Loss

Figure 19: Training results VGG16 simple model

As can easily be observed we encounter a scenario of very heavy overfitting. We need to address this issue.

6.1.2 Test Results

Class	Precision	Recall	F1-Score	Support
0	0.9268	0.8650	0.8948	600
1	0.9924	0.8667	0.9253	600
2	0.8851	0.8600	0.8724	600
3	0.7905	0.8300	0.8098	500
4	0.9280	0.9540	0.9408	500
5	0.7581	0.9400	0.8393	400
6	0.7993	0.9000	0.8467	500
7	0.9874	0.9167	0.9507	600
8	0.8858	0.8220	0.8527	500
9	0.9528	0.9750	0.9638	600
Accuracy: 0.8924, Macro Avg: 0.8906, Weighted Avg: 0.8985, Total: 5400				



(a) Confusion Matrix (b) ROC Curve

Figure 20: Test results VGG16 simple model

There does not seem to be a solid pattern in the way the model makes mistakes.

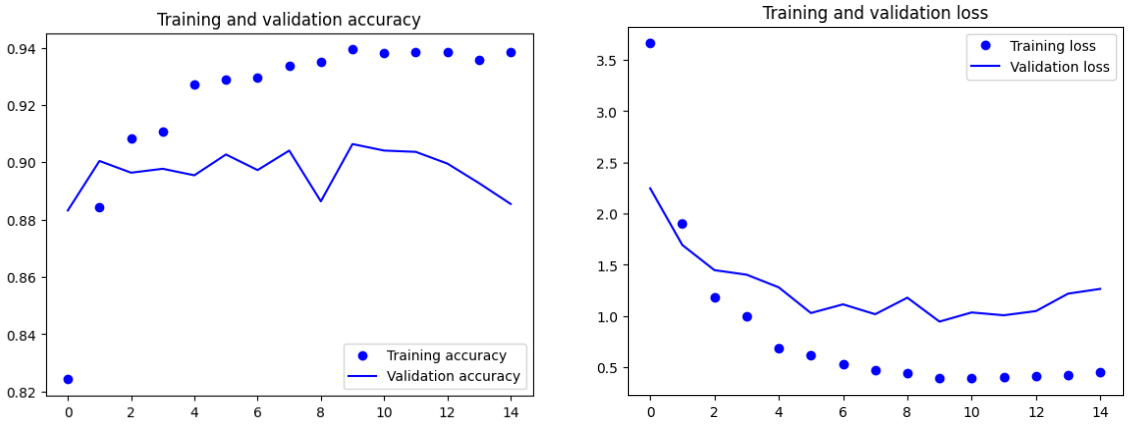
6.2 VGG16 Model with Dropout and Data Augmentation

A **dropout layer is added with the value set to 0.5** as well as a **data augmentation layer**.

It may not be possible to increase the performance of the model but we still aim to eliminate the overfitting

6.2.1 Training Reuslts

Epochs	Train Loss	Train Accuracy	Validation Loss	Validation Acc.
15	0.3895	93.93%	0.9457	90.64%



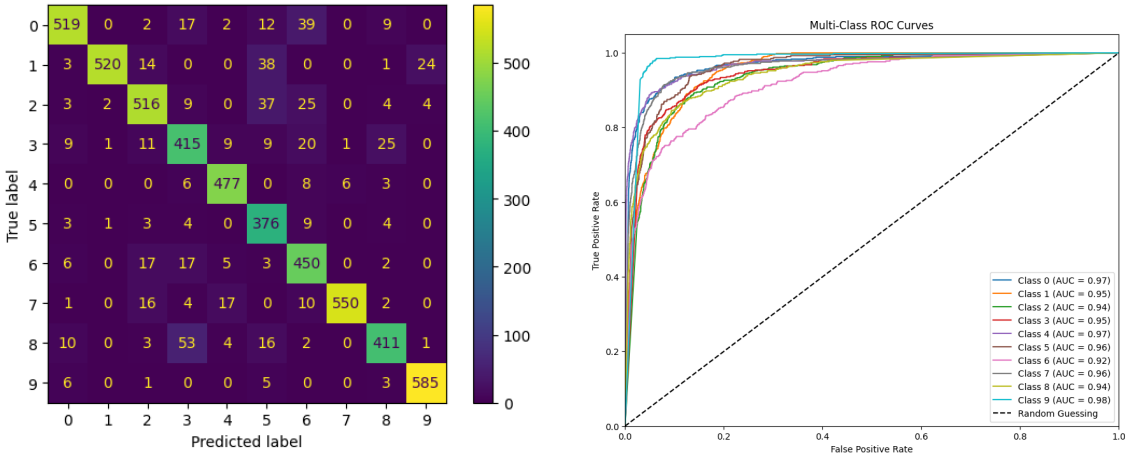
(a) Accuracy (b) Loss

Figure 21: Training results VGG16 model with dropout and data augmentation

The issue of overfitting is not yet fully solved, and the accuracy has also lowered a bit.

6.2.2 Test Results

Class	Precision	Recall	F1-Score	Support
0	0.9331	0.8600	0.8951	600
1	0.9922	0.8450	0.9127	600
2	0.8412	0.8917	0.8657	600
3	0.7932	0.7980	0.7956	500
4	0.8800	0.9680	0.9219	500
5	0.7609	0.9150	0.8309	400
6	0.8053	0.8520	0.8280	500
7	0.9731	0.9033	0.9369	600
8	0.8160	0.8340	0.8249	500
9	0.9807	0.9300	0.9547	600
Accuracy: 0.8796, Macro Avg: 0.8776, Weighted Avg: 0.8859, Total: 5400				



(a) Confusion Matrix (b) ROC Curve

Figure 22: Test results VGG16 model with dropout and data augmentation

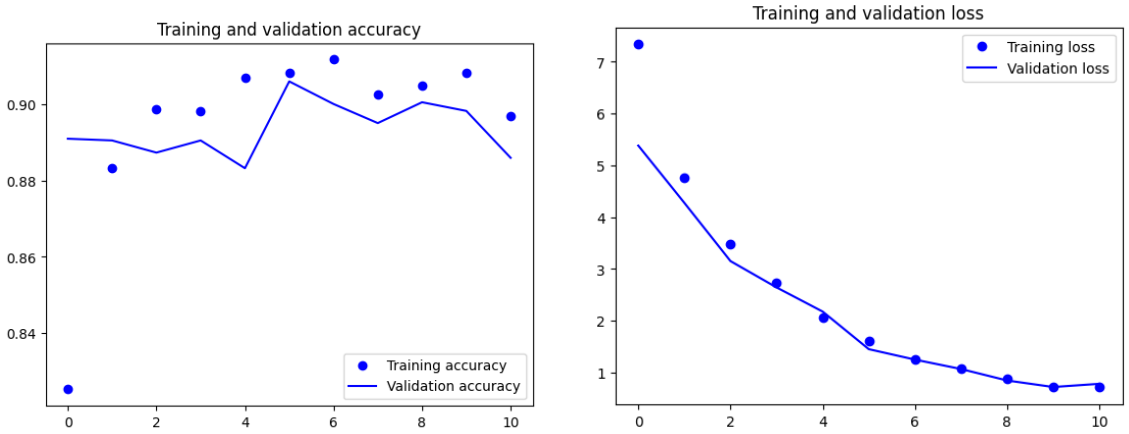
6.3 VGG16 Model with L2 Regularization

A **L2 regularization parameter set to 0.01** is added into the dense layer of 256 neurons.

This will eliminate the problem of overfitting once and for all.

6.3.1 Training Reuslts

Epochs	Train Loss	Train Accuracy	Validation Loss	Validation Acc.
15	0.7125	90.80%	0.7204	89.82%



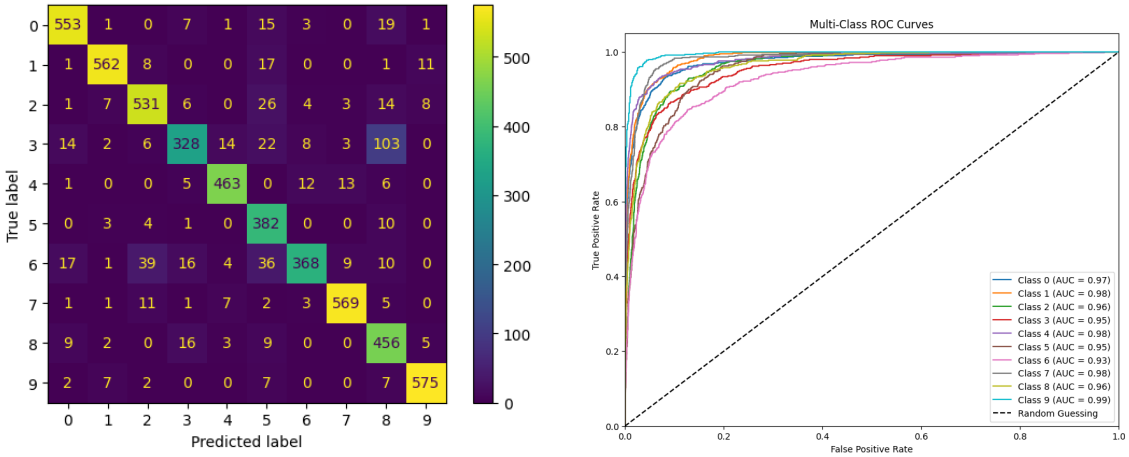
(a) Accuracy (b) Loss

Figure 23: Training results VGG16 model resilient to overfitting

The issue of overfitting is fully solved and the accuracy has also increased a bit.

6.3.2 Test Results

Class	Precision	Recall	F1-Score	Support
0	0.9232	0.9217	0.9224	600
1	0.9590	0.9367	0.9477	600
2	0.8835	0.8850	0.8843	600
3	0.8632	0.6560	0.7455	500
4	0.9411	0.9260	0.9335	500
5	0.7403	0.9550	0.8341	400
6	0.9246	0.7360	0.8196	500
7	0.9531	0.9483	0.9507	600
8	0.7227	0.9120	0.8064	500
9	0.9583	0.9583	0.9583	600
Accuracy: 0.8865, Macro Avg: 0.8869, Weighted Avg: 0.8941, Total: 5400				



(a) Confusion Matrix (b) ROC Curve

Figure 24: Test results VGG16 model resilient to overfitting

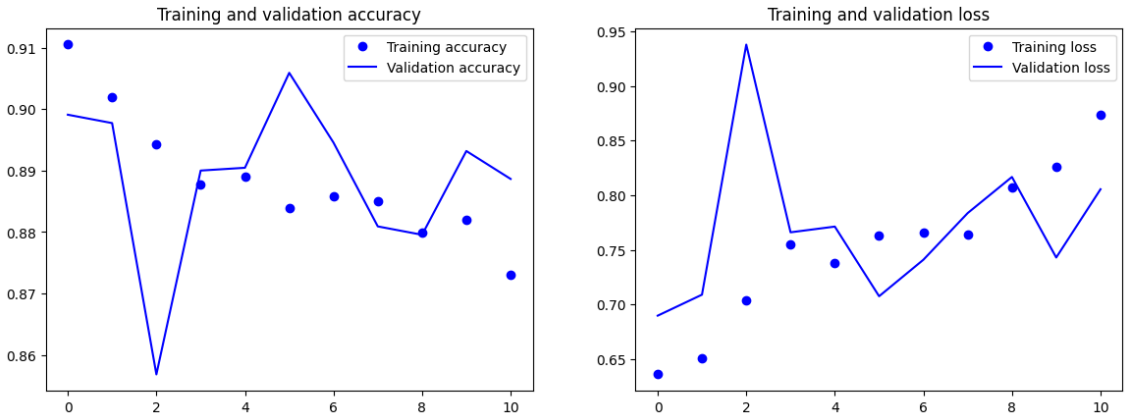
6.4 VGG16 Model with Fine Tuning

Lastly, we decided to try and **fine tune the last convolutional layer**. While this might not be the best idea (fine-tuning assumes that a model has been trained on similar data and with similar classes), we decided to try this approach anyway because we lack precise knowledge of the VGG16 model and because of the limited size of our dataset; we cannot definitively state that this will lead to worse results.

As reference model **we used the one with L2 regularization**, which looks to be the more stable one.

6.4.1 Training Reuslts

Epochs	Train Loss	Train Accuracy	Validation Loss	Validation Acc.
15	0.7632	88.39%	0.7075	90.59%

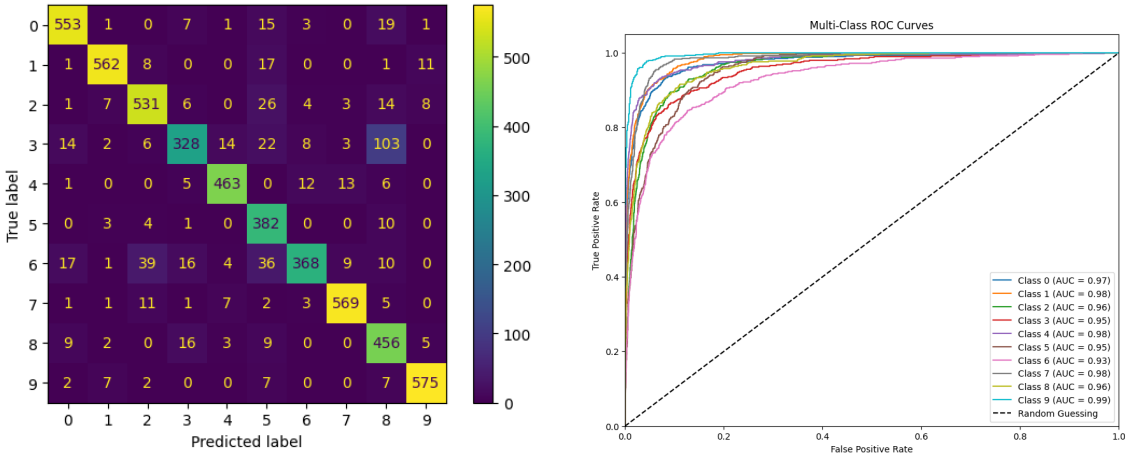


(a) Accuracy (b) Loss

Figure 25: Training results VGG16 model fine tuned

6.4.2 Test Results

Class	Precision	Recall	F1-Score	Support
0	0.8208	0.9467	0.8793	600
1	0.9721	0.9283	0.9497	600
2	0.8832	0.8950	0.8891	600
3	0.8889	0.6240	0.7333	500
4	0.9503	0.9180	0.9339	500
5	0.7875	0.9450	0.8591	400
6	0.9258	0.7240	0.8126	500
7	0.9171	0.9767	0.9459	600
8	0.7794	0.8620	0.8186	500
9	0.9365	0.9833	0.9593	600
Accuracy: 0.8852, Macro Avg: 0.8862, Weighted Avg: 0.8898, Total: 5400				



(a) Confusion Matrix (b) ROC Curve

Figure 26: Test results VGG16 model resilient to overfitting

The results in the training process are quite unstable and the model should be discarded.

6.4.3 Chosen Model

Out of the 4 models, the best pre-trained based on VGG16 is the third one with dropout, data augmentation and L2 regularization. That is the model to choose for future experiments.

Model	Train Loss	Train Accuracy	Validation Loss	Validation Accuracy
Model 1	0.1567	96.34%	0.9434	90.32%
Model 2	0.3895	93.93%	0.9457	90.64%
Model 3	**0.7125**	**90.80%**	**0.7204**	**89.82%**
Model 4	0.7632	88.39%	0.7075	90.59%

Table 5: Final results for the specified models

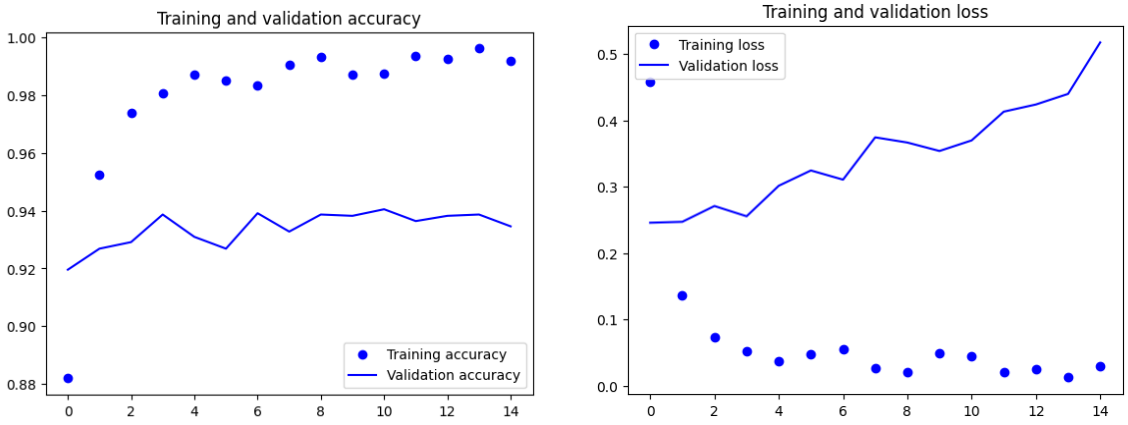
Remember that even if the validation accuracy is the lowest of the bunch, this isn't true when we look at the test accuracy. This model appears to be the most general one, hence the choice.

6.5 ResNet50 Simple Model

The ResNet50 pre-trained model was created following the same steps used for the VGG16 model; after importing the ResNet50 convolutional base the data are preprocessed using the library function and are then sent into the CNN. The data are then flattened and goes through a 256-neurons dense layer. The output is of course the probability distribution for the classes, the chosen optimizer function is Adam and the loss function is categorical cross entropy.

6.5.1 Training Reuslts

Epochs	Train Loss	Train Accuracy	Validation Loss	Validation Acc.
15	0.4584	88.19%	0.2458	91.95%



(a) Accuracy (b) Loss

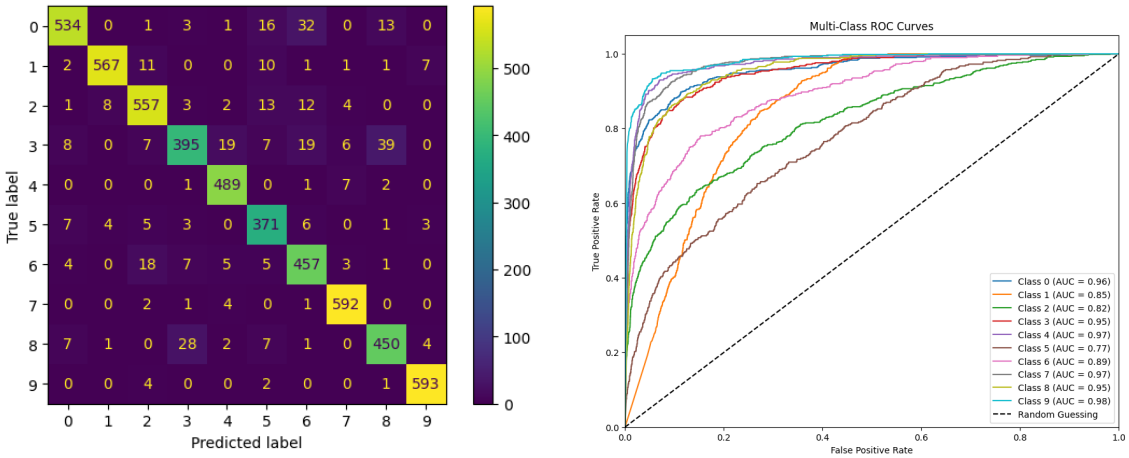
Figure 27: Training results R50 simple model

Once again the simple model is affected by overfitting that we need to address; it is to notice though that the difference between the training values and validation values are not so big when put into scale with the values on the y-axis.

6.5.2 Test Results

Class	Precision	Recall	F1-Score	Support
0	0.9485	0.8900	0.9183	600
1	0.9776	0.9450	0.9610	600
2	0.9207	0.9283	0.9245	600
3	0.8957	0.7900	0.8395	500
4	0.9368	0.9780	0.9569	500
5	0.8608	0.9275	0.8929	400
6	0.8623	0.9140	0.8874	500
7	0.9657	0.9867	0.9761	600
8	0.8858	0.9000	0.8929	500
9	0.9769	0.9883	0.9826	600
Accuracy: 0.9269, Macro Avg: 0.9231, Weighted Avg: 0.9275, Total: 5400				

Table 6: Classification report



(a) Confusion Matrix (b) ROC Curve

Figure 28: Test results R50 simple model

There does not seem to be a solid pattern in the way the model makes mistakes.

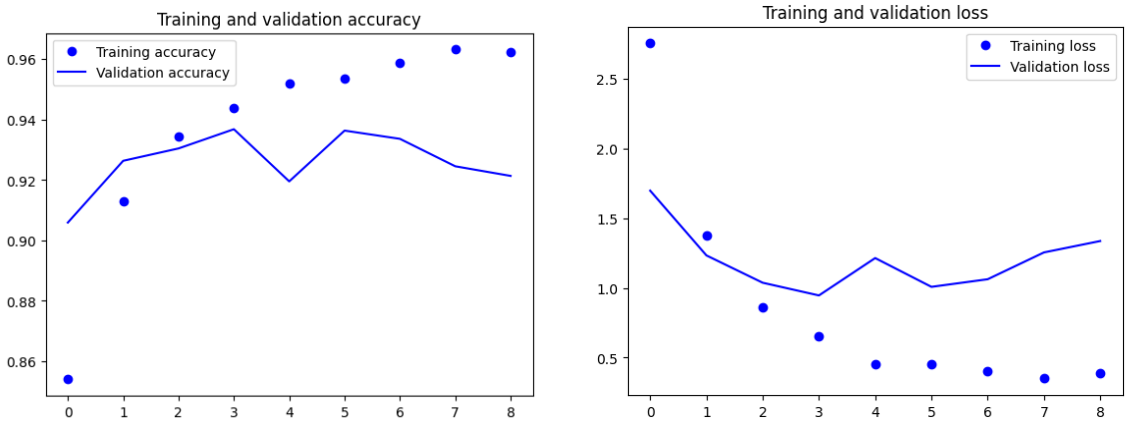
6.6 ResNet50 Model with Dropout and Data Augmentation

A **dropout layer is added with the value set to 0.5** as well as a **data augmentation layer**.

It may not be possible to increase the performance of the model but we still aim to eliminate the overfitting.

6.6.1 Training Reuslts

Epochs	Train Loss	Train Accuracy	Validation Loss	Validation Acc.
15	0.6530	94.40%	0.9454	93.68%



(a) Accuracy (b) Loss

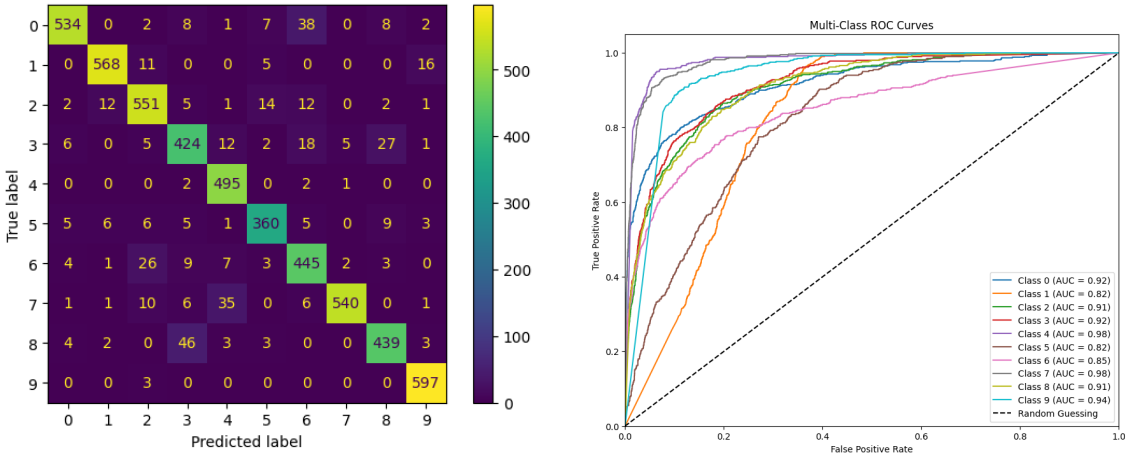
Figure 29: Training results R50 model with dropout and data augmentation

The issue of overfitting is not yet fully solved, and the accuracy has also lowered a bit.

6.6.2 Test Results

Class	Precision	Recall	F1-Score	Support
0	0.9604	0.8900	0.9239	600
1	0.9627	0.9467	0.9546	600
2	0.8974	0.9183	0.9077	600
3	0.8396	0.8480	0.8438	500
4	0.8919	0.9900	0.9384	500
5	0.9137	0.9000	0.9068	400
6	0.8460	0.8900	0.8674	500
7	0.9854	0.9000	0.9408	600
8	0.8996	0.8780	0.8887	500
9	0.9567	0.9950	0.9755	600
Accuracy: 0.9172, Macro Avg: 0.9153, Weighted Avg: 0.9188, Total: 5400				

Table 7: Classification report



(a) Confusion Matrix (b) ROC Curve

Figure 30: Test results R50 model with dropout and data augmentation

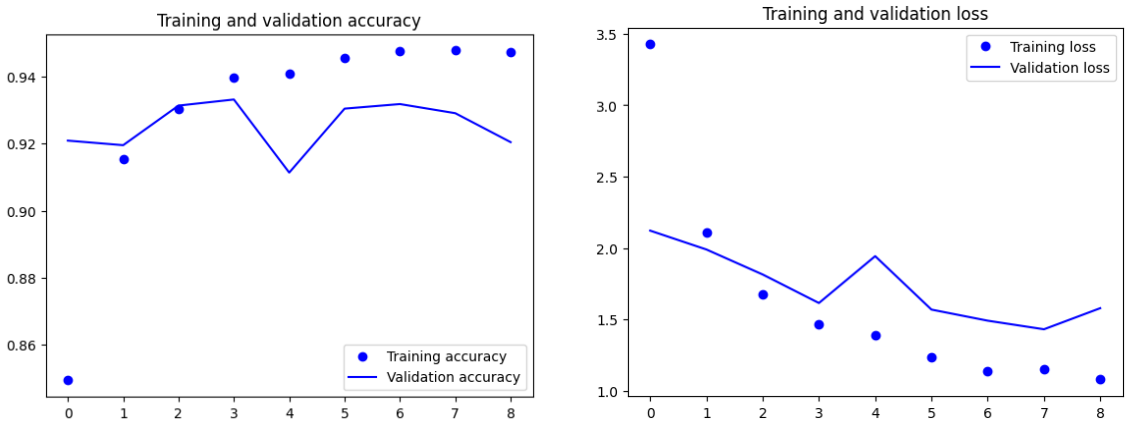
6.7 ResNet50 Model with L2 Regularization

A **L2 regularization parameter set to 0.001** is added into the dense layer of 256 neurons.

With regularization set to 0.01 like in the VGG16 model we lose performance, we will use a lower value here (0.001).

6.7.1 Training Reuslts

Epochs	Train Loss	Train Accuracy	Validation Loss	Validation Acc.
15	1.1547	94.78%	1.4320	92.91%



(a) Accuracy (b) Loss

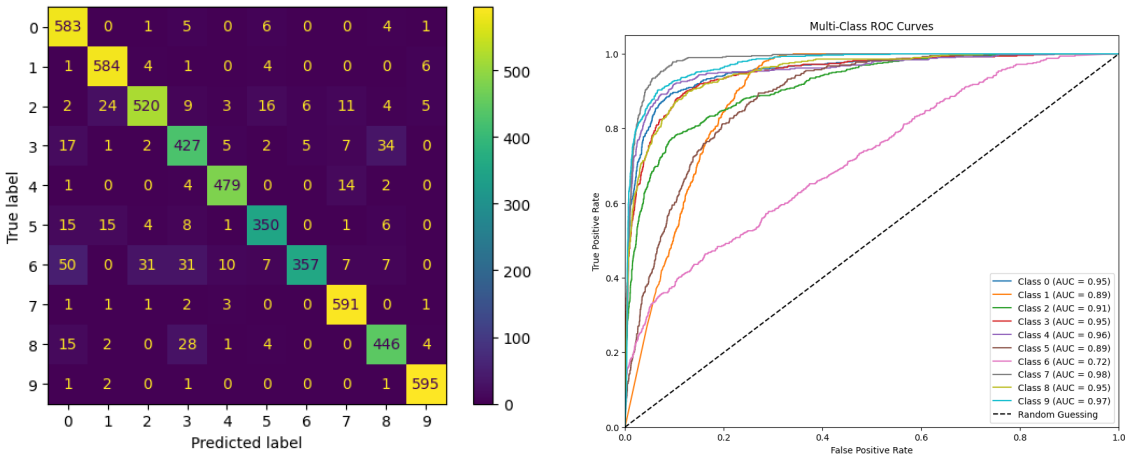
Figure 31: Training results R50 model resilient to overfitting

Even though the trend of overfitting seems to be more under control, the simple model still had lower values for validation and training loss and a higher value for validation the test accuracy.

6.7.2 Test Results

Class	Precision	Recall	F1-Score	Support
0	0.8499	0.9717	0.9067	600
1	0.9285	0.9733	0.9504	600
2	0.9236	0.8667	0.8942	600
3	0.8275	0.8540	0.8406	500
4	0.9542	0.9580	0.9561	500
5	0.8997	0.8750	0.8872	400
6	0.9701	0.7140	0.8226	500
7	0.9366	0.9850	0.9602	600
8	0.8849	0.8920	0.8884	500
9	0.9722	0.9917	0.9818	600
Accuracy: 0.9133, Macro Avg: 0.9147, Weighted Avg: 0.9157, Total: 5400				

Table 8: Classification report



(a) Confusion Matrix (b) ROC Curve

Figure 32: Test results R50 model resilient to overfitting

6.8 ResNet50 Model with Fine Tuning

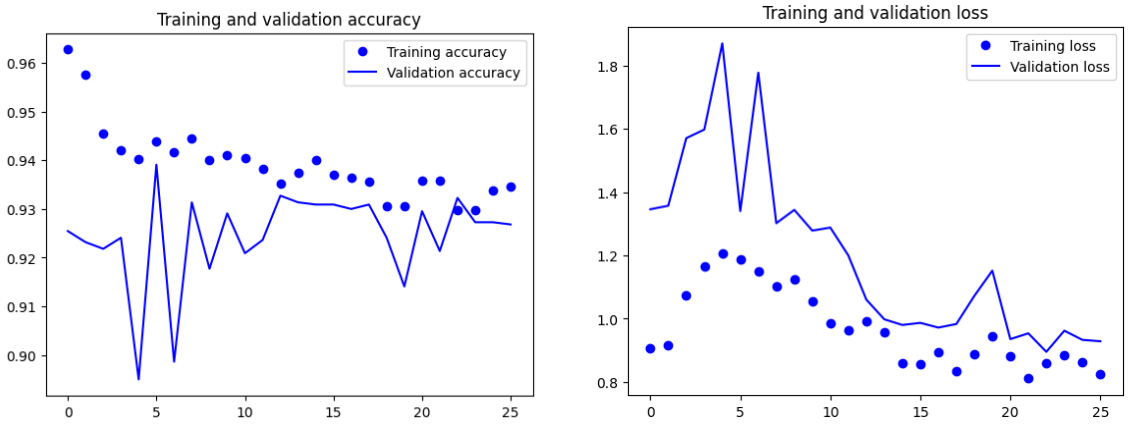
Again, we decided to try and **fine tune the last convolutional layer**. While this might not be the best idea (fine-tuning assumes that a model has been trained on similar data and with similar classes), we decided to try this approach anyway

because we lack precise knowledge of the R50 model and because of the limited size of our dataset; we cannot definitively state that this will lead to worse results.

As reference model **we used the one with L2 regularization**.

6.8.1 Training Reuslts

Epochs	Train Loss	Train Accuracy	Validation Loss	Validation Acc.
30	0.8606	92.98%	0.8954	93.23%



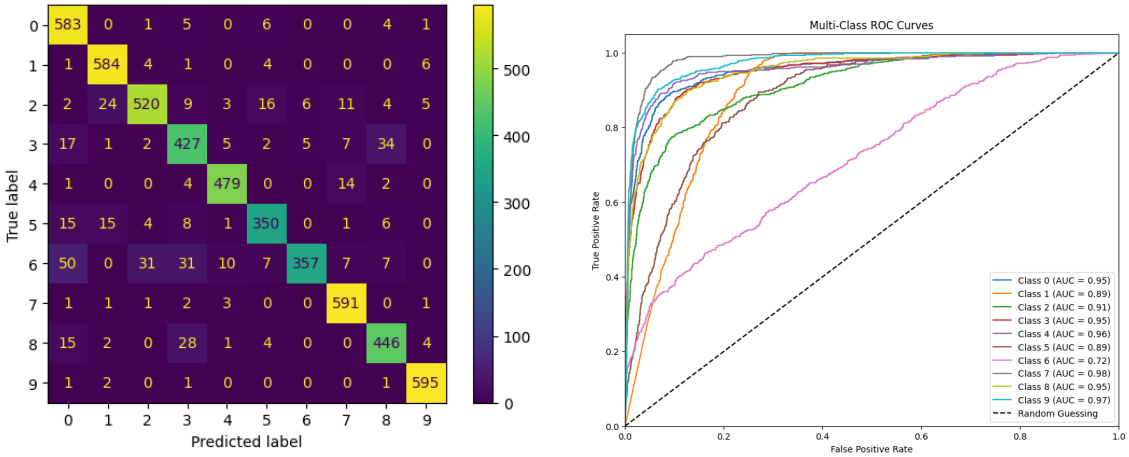
(a) Accuracy (b) Loss

Figure 33: Training results R50 model fine tuned

6.8.2 Test Results

Class	Precision	Recall	F1-Score	Support
0	0.9811	0.7767	0.8670	600
1	0.9705	0.9317	0.9507	600
2	0.9092	0.9017	0.9054	600
3	0.7683	0.8820	0.8212	500
4	0.9791	0.9380	0.9581	500
5	0.8493	0.9300	0.8878	400
6	0.8102	0.8540	0.8315	500
7	0.9715	0.9650	0.9682	600
8	0.8388	0.9260	0.8802	500
9	0.9898	0.9700	0.9798	600
Accuracy: 0.9072, Macro Avg: 0.9068, Weighted Avg: 0.9132, Total: 5400				

Table 9: Classification report



(a) Confusion Matrix (b) ROC Curve

Figure 34: Test results R50 model resilient to overfitting

The results in the training process are again quite unstable and the performance are a bit lower than the model with L2 regularization.

6.8.3 Chosen Model

Out of the 4 models, the best pre-trained based on R50 is the first one without any dropout, data augmentation and L2 regularization. That is the model to choose for feature experiments.

Model	Train Loss	Train Accuracy	Validation Loss	Validation Accuracy
Model 1	**0.4584**	**88.19%**	**0.2458**	**91.95%**
Model 2	0.6530	94.40%	0.9454	93.68%
Model 3	1.1547	94.78%	1.4320	92.91%
Model 4	0.8606	92.98%	0.8954	93.23%

Table 10: Final results for the specified models

Once again even if the validatin accuracy is the lowest of the bunch, that is not true when we look at the test accuracy which is the best of the four models. The loss is also the lowest and that is the main reason tho choose the simple model over the others; it seems that measures to avoid overfitting only lowered the performance of the CNN.

The simple model will be used for the final phase of our project.

7 Visualization

In this chapter we are going to analyze what our models learned.

7.1 Activations

In this section, we will delve into an analysis of the initial convolutional layers of each model, aiming to decipher the features they have learned at their respective depths.

We have deliberately excluded deeper layers from our examination. This choice is rooted in the understanding that as we traverse deeper into the network, the acquired representations tend to become progressively more abstract and less interpretable. The complexity of these later layers often hinders the clear identification of specific features, making it challenging to extract meaningful insights from their representations.

In this chapter, the following image was used as a sample.

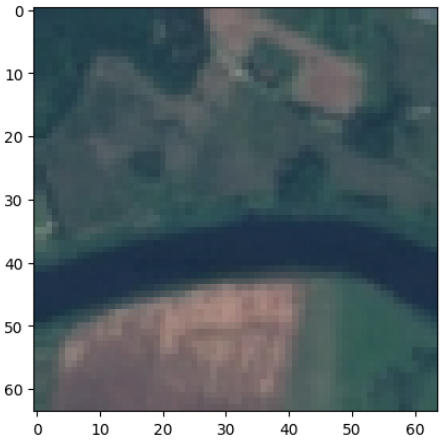


Figure 35: River

7.1.1 Model3

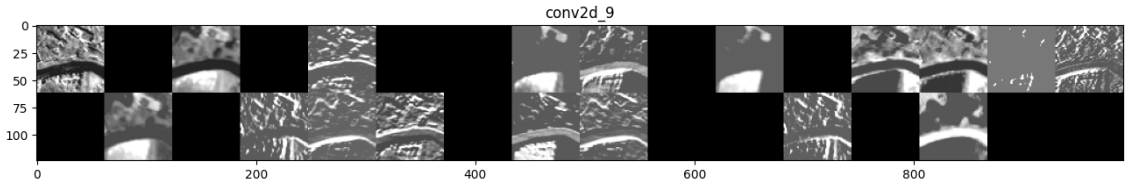


Figure 36: First Conv layer

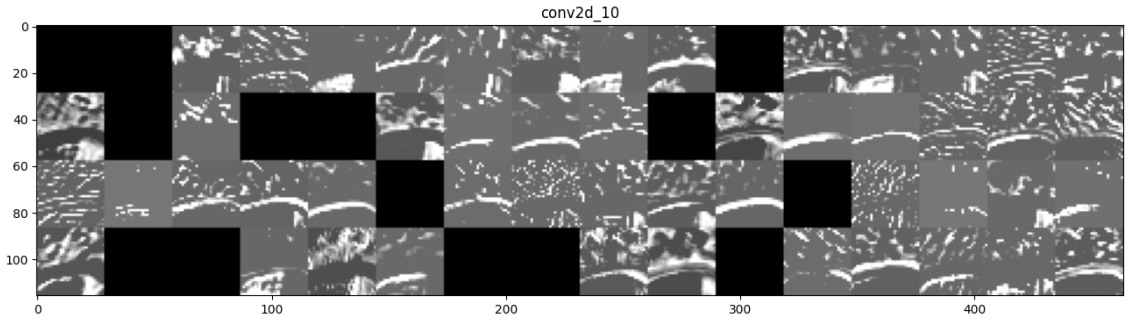


Figure 37: Second Conv layer

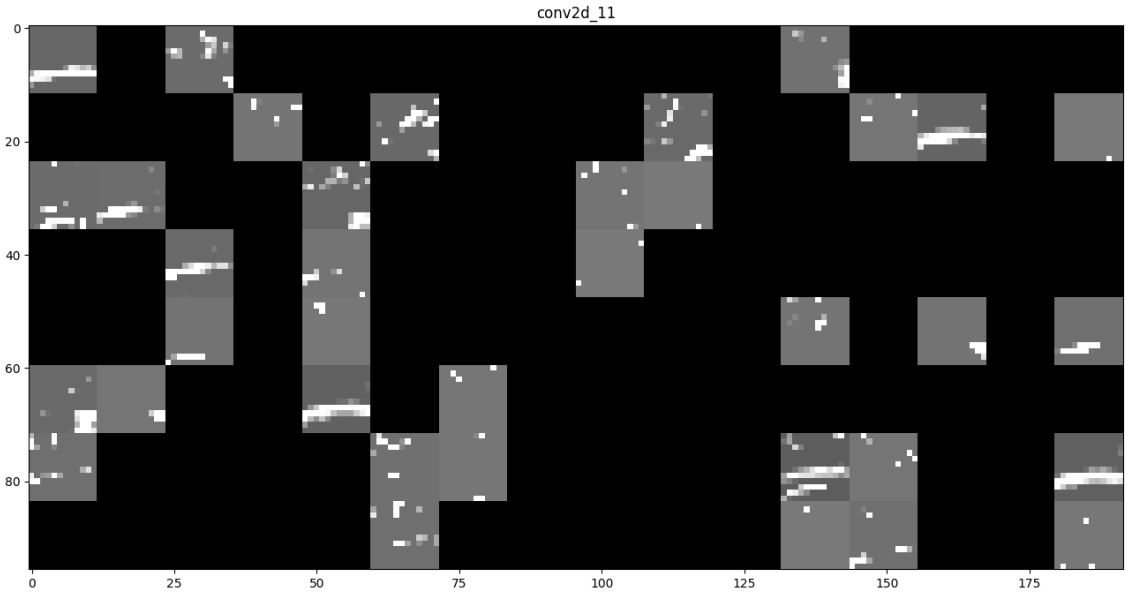


Figure 38: Third Conv layer

7.1.2 VGG16

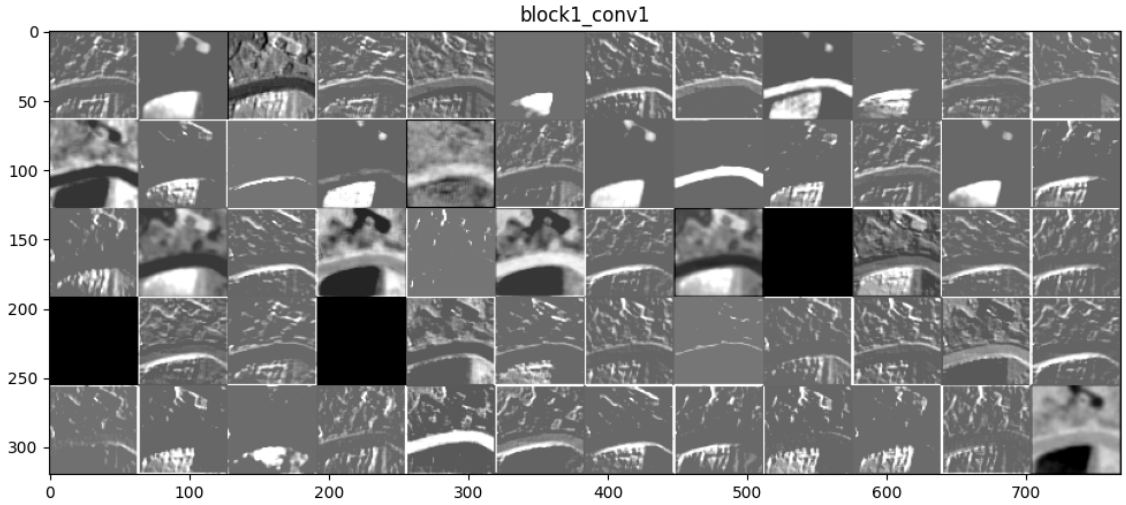


Figure 39: First Conv layer

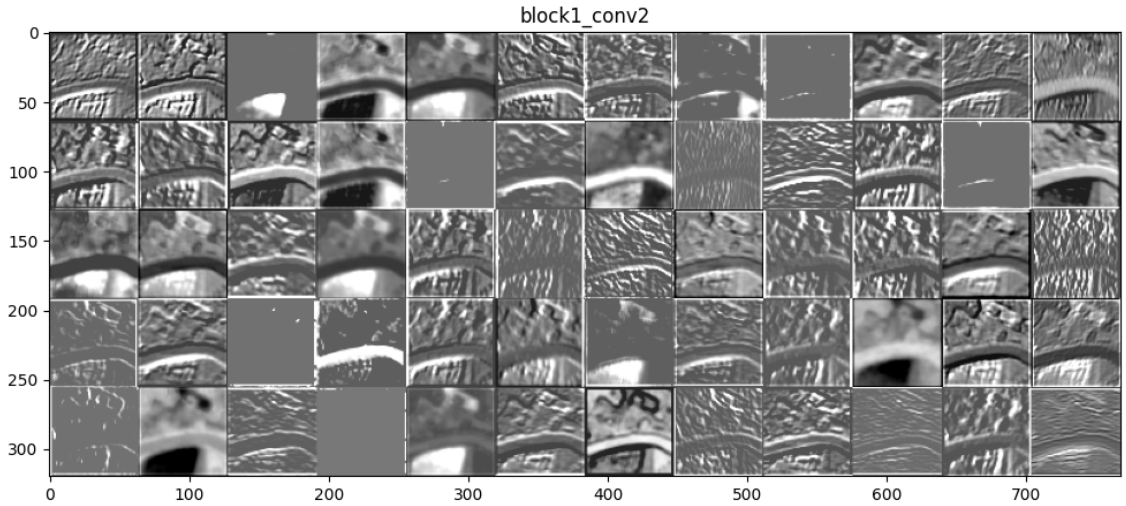


Figure 40: Second Conv layer

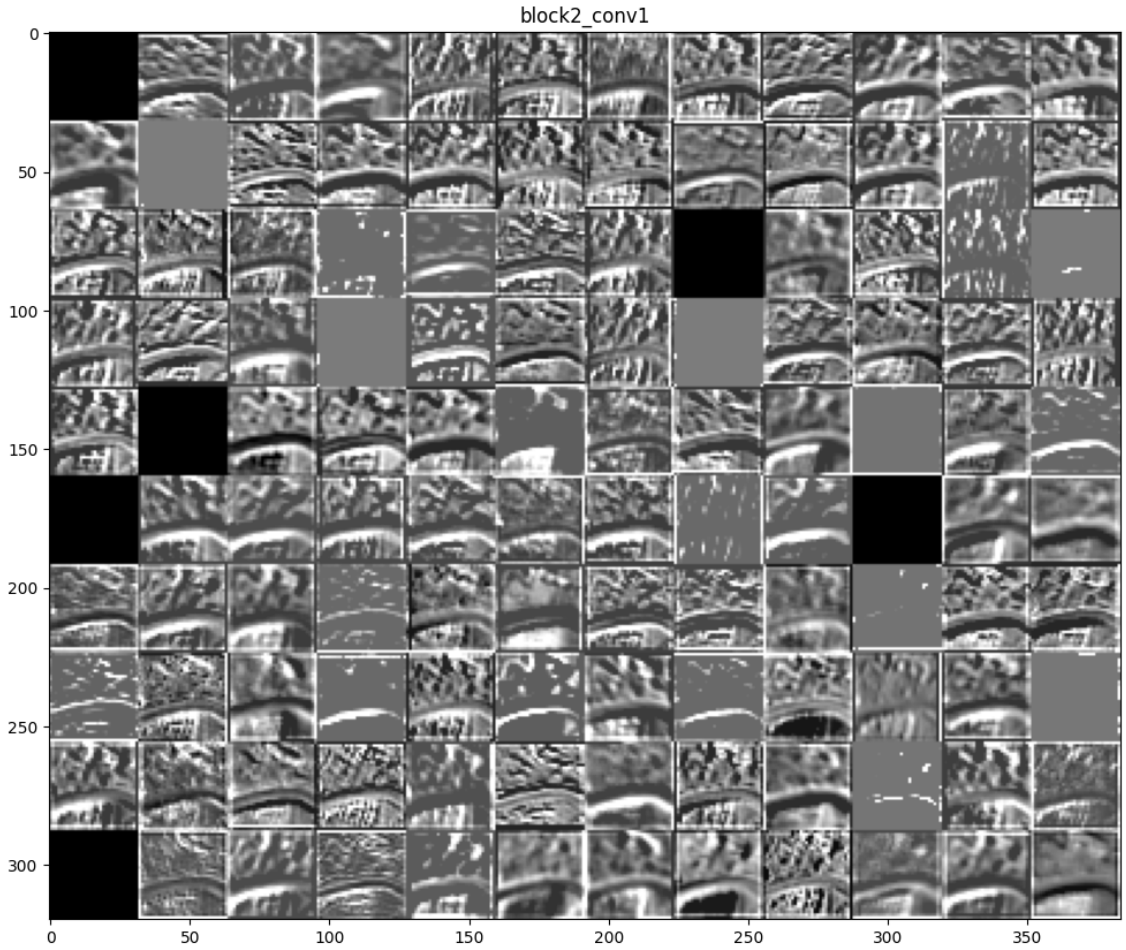


Figure 41: Third Conv layer

7.1.3 ResNet50

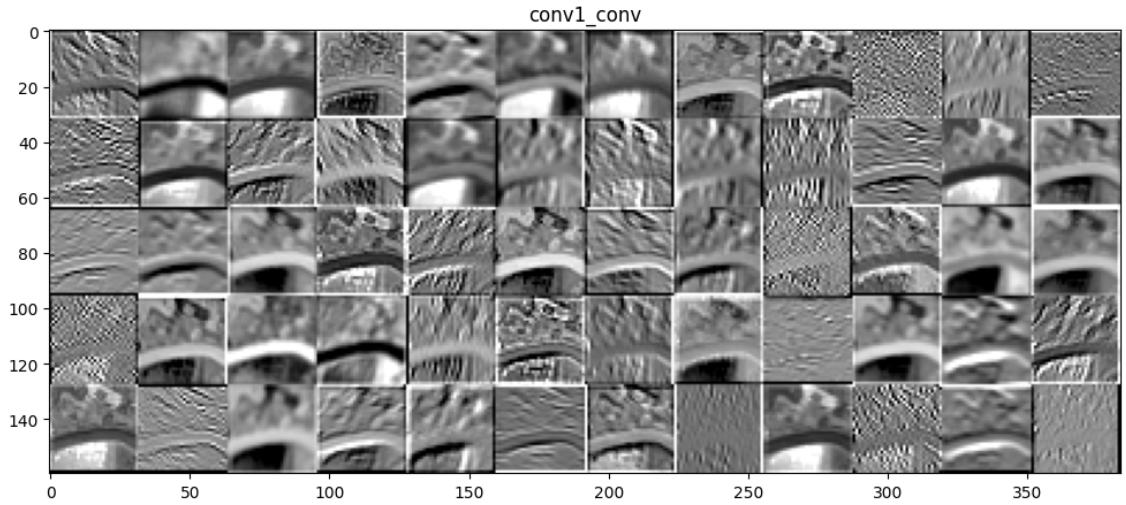


Figure 42: First Conv layer

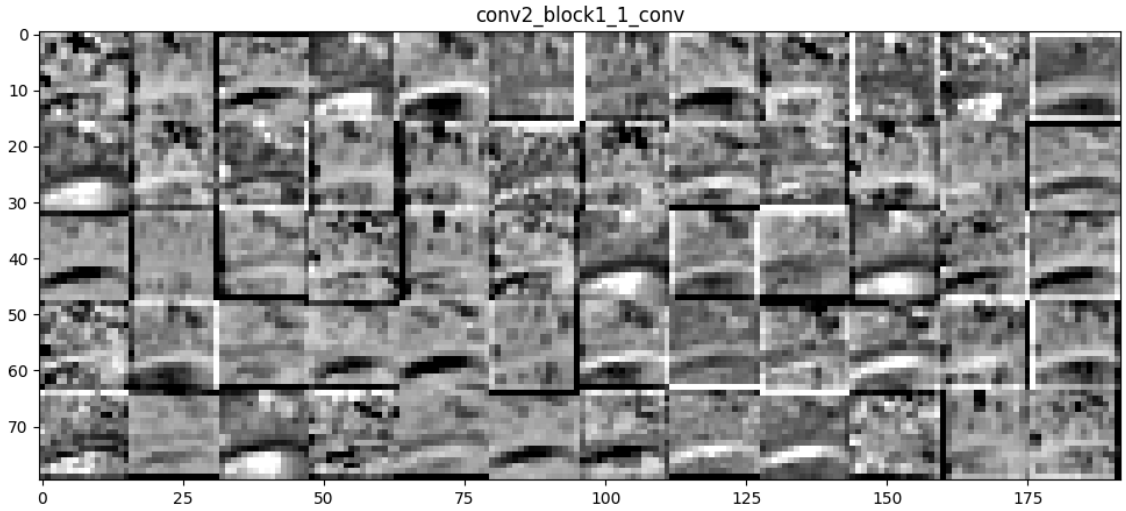


Figure 43: Second Conv layer

7.2 Heatmaps

The heatmap serves as a valuable tool for comprehending the regions within the input image that the network prioritized during its prediction process. Through the heatmap, it is possible to unveil the specific areas of the image that significantly influenced the CNN's classification decision.

It's worth noting that both VGG16 and ResNet50 produced heatmaps consisting of 2x2 pixel images. This was a direct result of our original images being 64x64 pixels, and the CNN's processing led to a reduction in spatial dimensions.

In response to this limitation, we have chosen to utilize intermediate convolutional layers to compute larger activation maps, rather than relying solely on the final layer. This strategic shift enables us to capture a more detailed representation of the image's influential regions, thus enhancing the interpretability and granularity of the heatmaps.

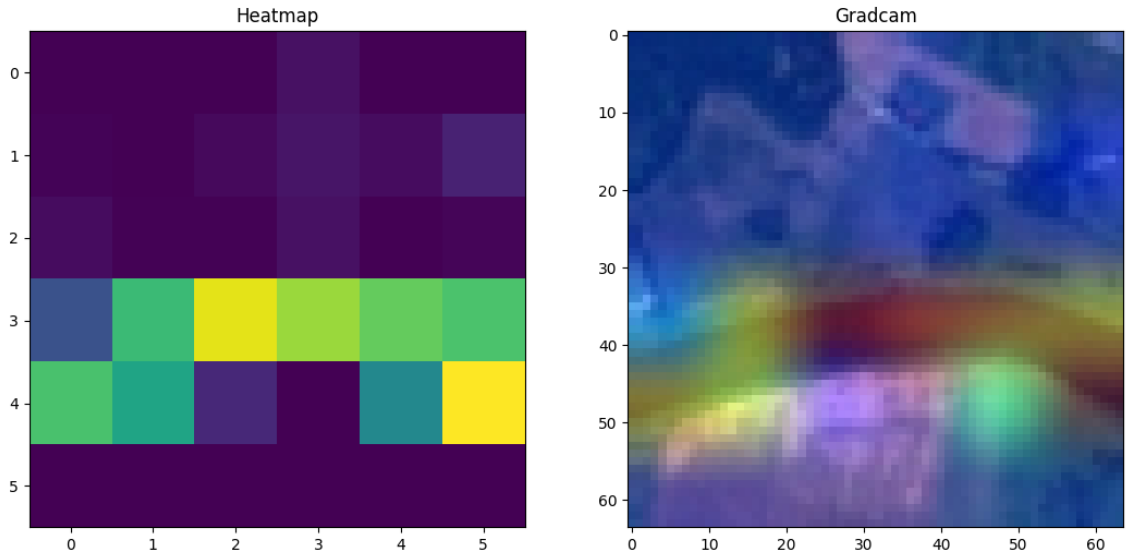


Figure 44: Model3 Heatmap

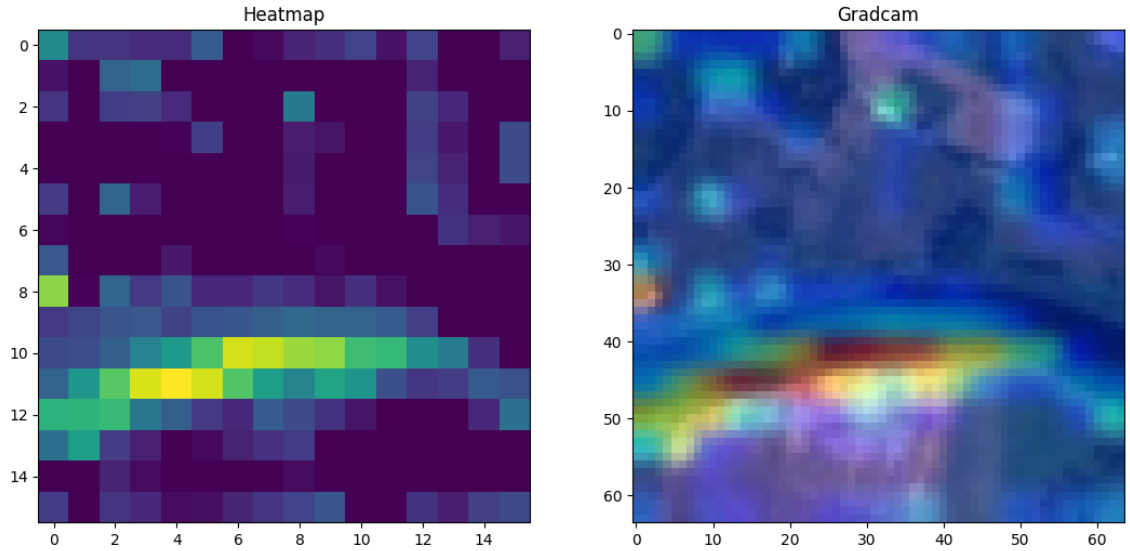


Figure 45: VGG16 Heatmap

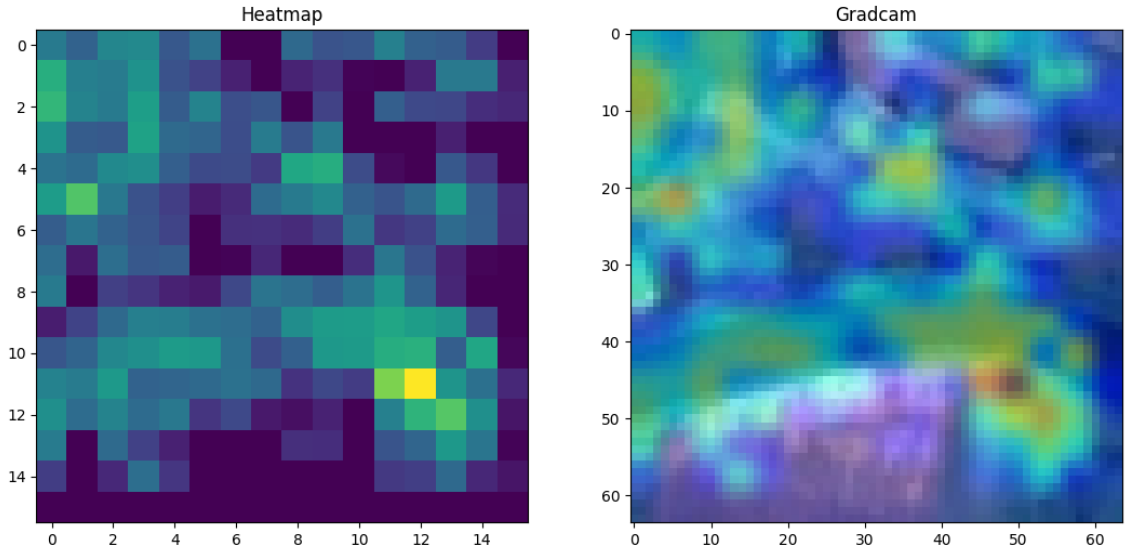


Figure 46: ResNet50 Heatmap

Across all images, it is evident from both the heatmaps and activation maps that the convolutional network effectively discerns the distinct characteristics of rivers, particularly their gentle undulating forms. Notably, ResNet50 exhibits less pronounced activations in comparison.

Interestingly, ResNet50 showcases a more diffused pattern of activations across the entire image, as opposed to deep and concentrated markings on these distinctive shapes. This difference in activation patterns between the two models offers insight into their respective interpretive abilities and the ways they capture image features.

8 Ensemble

In the end, we selected one model from our own custom-built CNNs, another from the various iterations of pre-trained VGG16 models, and yet another from the pre-trained ResNet50 models. With the goal of constructing the most optimal classifier, we opted to experiment with an ensemble approach that could potentially enhance performance even further.

8.1 Ensemble Function

The ensemble technique we opted for relies on a majority vote methodology, while according higher priority to our model due to its superior performance across various scenarios.

This entails that in cases where two or more models concur on a prediction, that particular prediction is adopted. However, in instances where the models' opinions diverge, we default to the solution provided by our model. The code snippet below illustrates the segment of code used to generate these ensemble predictions:

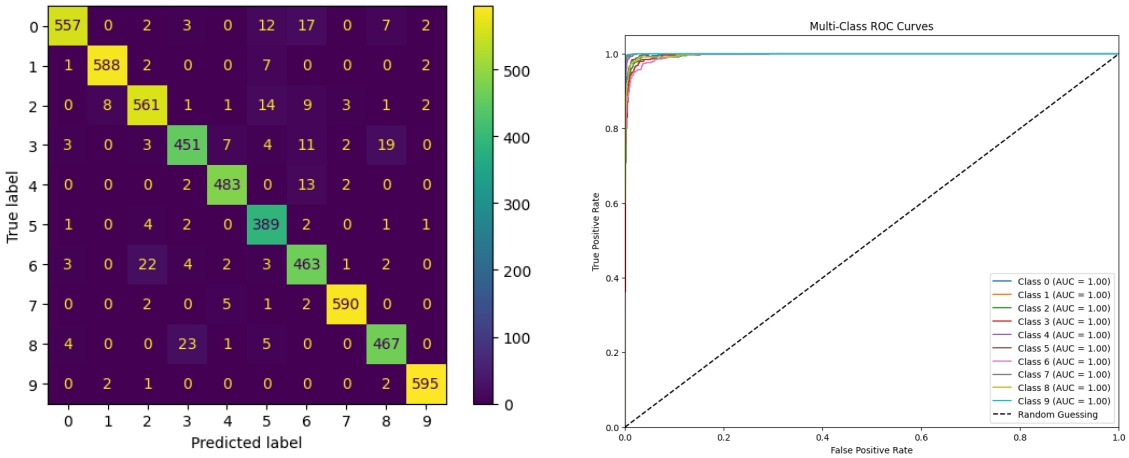
```
# compute the predictions for all the models
our_test_scores = our_model.predict(data)
vgg_test_scores = vgg_model.predict(data)
resnet_test_scores = resnet_model.predict(data)
# Create an array containing the sum of the scores of the three models
final_scores = np.zeros_like(our_test_scores)
final_scores = our_test_scores + vgg_test_scores + resnet_test_scores

# There are three cases:
# CASE 1: a cell contains a value equal to 3 and nine equal to 0; in this case we must transform 3->1
# CASE 2: a cell contains a value equal to 2, one equal to 1 and eight equal to 0; 2->1, 1->0, 0->0
# CASE 3: a cell contains three values equal to 1 and seven equal to 0; we adopt the prediction offered by our model
for i in range(final_scores.shape[0]):
    for j in range(final_scores.shape[1]):
        value = final_scores[i, j]

        if value == 3:
            final_scores[i, j] = 1
        elif value == 2:
            final_scores[i, j] = 1
            final_scores[i, (j + 1) % 10:] = 0
        elif np.sum(final_scores[i] == 1) == 3:
            final_scores[i] = our_test_scores[i]
```

Listing 6: Ensemble function

8.1.1 Test Results

Class	Precision	Recall	F1-Score	Support
0	0.9789	0.9283	0.9530	600
1	0.9833	0.9800	0.9816	600
2	0.9397	0.9350	0.9373	600
3	0.9280	0.9020	0.9148	500
4	0.9679	0.9660	0.9670	500
5	0.8943	0.9725	0.9317	400
6	0.8956	0.9260	0.9105	500
7	0.9866	0.9833	0.9850	600
8	0.9359	0.9340	0.9349	500
9	0.9884	0.9917	0.9900	600
Accuracy: 0.9526, Macro Avg: 0.9498, Weighted Avg: 0.9532, Total: 5400				

Table 11: Classification report



(a) Confusion Matrix (b) ROC Curve

Figure 47: Test results ensemble model

The ensemble model demonstrates higher performance in terms of accuracy, F1 scores, and nearly maximized AUC values. This outcome is expected, of course, given that a majority vote approach works exceptionally well with three models. Notably, Class 6 remains the most challenging class.

9 Conclusions

Throughout this project, we developed several CNN models from scratch as well as utilizing various pre-trained models based on the VGG16 and ResNet50 architectures to classify satellite images based on their land usage.

We successfully crafted a CNN model from scratch that exhibited exceptional performance, surpassing even the pre-trained models. It's noteworthy that all models achieved high performance across the board.

Furthermore, we delved into understanding what the models learned in specific convolutional layers. We also explored heatmaps, revealing that the models effectively identified distinct areas within images to inform their decisions. For instance, rooftops were highlighted in images featuring buildings, indicating that the models were leveraging these elements for predictions.

Lastly, we constructed an ensemble model comprising our best model and the top pre-trained models. This ensemble yielded even better results in terms of accuracy and robustness. We hold confidence in the performance of the latter and envision its potential usefulness in real-life applications.

References

[1] Tapan Kumar Das, Dillip Kumar Barik, and K V G Raj Kumar. Land-use land-cover prediction from satellite images using machine learning techniques. In *2022 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COM-IT-CON)*, volume 1, pages 338–343, 2022.

[2] Hisyam Fahmi and Wina Permana Sari. Analysis of deep learning architecture for patch-based land cover classification. In *2022 6th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, pages 1–5, 2022.

[3] Patrick Helber, Benjamin Bischke, and Andreas Dengel. Introducing eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *Remote Sensing*, 10(10):1637, 2018.

[4] Patrick Helber, Benjamin Bischke, and Andreas Dengel. 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(7):2217–2226, 2019.