Università degli Studi di Pisa
Computational Intelligence and Deep Learning

Corso di Laurea Magistrale in Ing. Robotica e dell'Automazione

Project Document

European Mushrooms Classification

February 2023

Team:
Pietro Gori, p.gori3@studenti.unipi.it
Matricola 564872

Vincenzo Degiacomo, v.degiacomo@studenti.unipi.it
Matricola 564713

:
Prof. Beatrice Lazzerini
Phd. Michele Baldassini

Anno Accademico 2022-2023

Contents

1 Introduction

Our goal is the classification of mushroom image in order to recognize their own species. For this project we have considered the principal types of the European zone.

1.1 Related Works

Distinguishing various species of mushrooms can be very complicated to inexperienced eyes. For this reason, several solutions have been explored to solve this task by resorting to ANN. In [1] for example Multi-Layer ANN model was used to train and test the mushroom dataset to predict whether it is edible or poisonous. In particular, with the aim of carrying out environmental monitoring and monitoring effects due to climate change, several works based on deep learning methods have been carried out. In [3] e.g., a CNN from scratch has been implemented to successfully recognize 45 types of mushrooms. Better results have been achieved by [4] and [5] using pre-trained CNN and transfer learning methods.

1.2 Dataset

The dataset has been retrivied from Kaggle [6]. It consists of 6714 images divides into 9 classes:

- Agaricus: 353 files
- Amanita: 750 files
- Boletus: 1073 files
- Cortinarius: 836 files
- Entoloma: 364 files
- Hygrocybe: 316 files
- Lactarius: 1563 files
- Russola: 1148 files
- Suillus: 311 files

It is immediate to note that the dataset is not balanced at all. For this reason we decide to use only 4 classes with a high and similar number of images:

- Amanita: 750 files
- Boletus: 1073 files
- Cortinarius: 836 files
- Russola: 1148 files

Figure 1: Pictures of any dataset class

The train:test ratio selected is 9:1 while the validation set is the 20% of the training set, providing 2740 files for training, 684 files for validation and 382 files for testing.

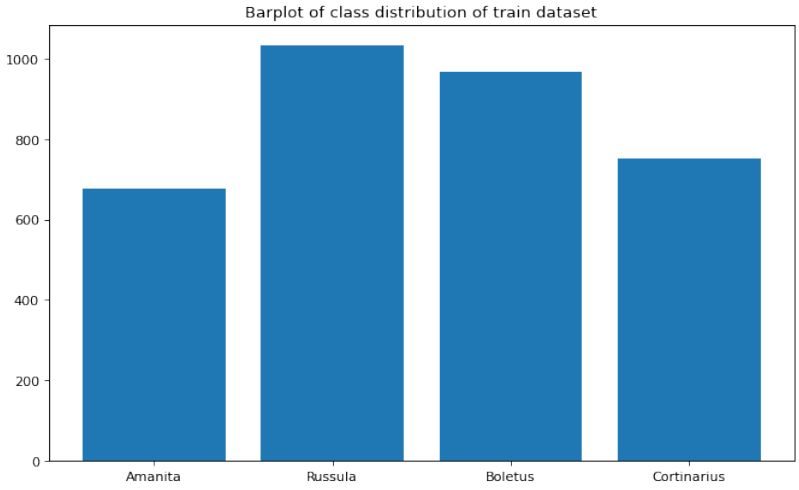


Figure 2: Training dataset distribution.

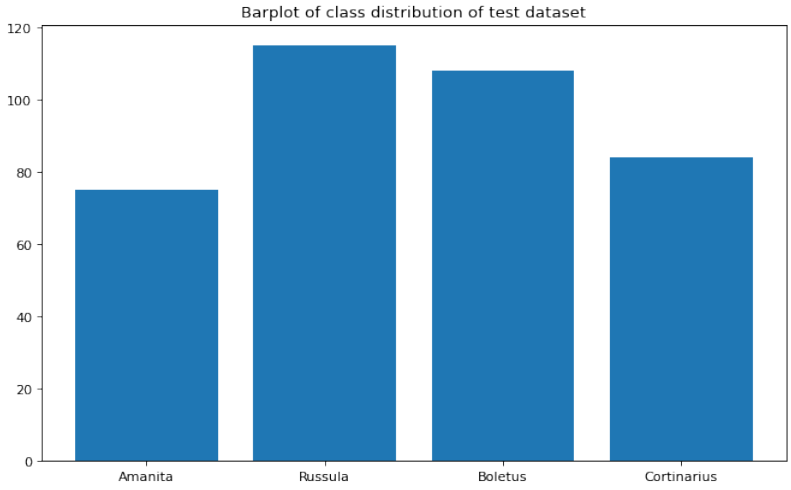


Figure 3: Test dataset distribution.

1.3 Metrics Used

To evaluate the network performance has not been used only the accuracy, in fact, even if we supposed to have the same misclassification cost for all the classes, we also want to verify the performance on minority ones, that was quite consistent in the first version of the dataset. For this reason, we have considered also the F1 on single classes, that corresponds to the harmonic mean of precision and recall, and the F1-macro to have a more compact index of performance. We have also used the ROC AUC (Receiver Operating Characteristic Area Under the Curve) score.
To have an immediate way to verify the goodness of the networks we introduced the confusion matrix; a matrix which relate the true labels with the predicted labels of the images in the test set.

2 Methods and Techniques

2.1 Data Augmentation

Data Augmentation is used to generate more training data from existing training samples through a number of random transformations that produce "believable images", meaning that do not corrupt the class of the image. This is helpful both to fight over-fitting and to help the model to generalize better. As suggested in [3], rotation, flipping, translation, zooming and contrast and brightness variation are best suited when dealing with the kind of images we are dealing with.

```
data_augmentation = tf.keras.Sequential([
layers.RandomFlip('horizontal'),
layers.RandomFlip("vertical"),
layers.RandomRotation(0.2),
layers.RandomContrast(0.2),
layers.RandomZoom(0.3),
layers.RandomBrightness(0.6),
```

```
layers.RandomTranslation(0.2, 0.2)
```

2.2 Reduce Learning Rate on Plateau

Reduce learning rate when a metric has stopped improving. Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.

```
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(
                                monitor='val_accuracy',
                                factor=0.1,
                                mode='max',
                                cooldown=2,
                                patience=2,
                                min_lr=1e-8)
```

2.3 Early Stopping

It stops training when a monitored metric has stopped improving, usually meaning that the model start to over-fit.

```
earlystopping = EarlyStopping(monitor='val_accuracy',
                           patience=10,
                           restore_best_weights=True,
                           verbose=1)
```

2.4 Model Checkpoint

it allows to save the best model obtained before the learner starts to over-fit the training data.

```
checkpoint_model_save = keras.callbacks.ModelCheckpoint(
                                     filepath=save_path,
                                     monitor="val_loss",
                                     verbose=1,
                                     save_best_only=True)
```

3 CNN from Scratch

In this section will be described the Convolutional Neural Network (CNN) that has been made from scratch.
As said in the previous sections, the network has to identify the input images into 4 classes (*Amanita*, *Boletus*, *Cortinarius* and *Russola*).
As every CNN it is made by some so called "conv blocks", which are simply groups of convolutional and pooling layers coupled to create a block. In this network there are 3 conv blocks:

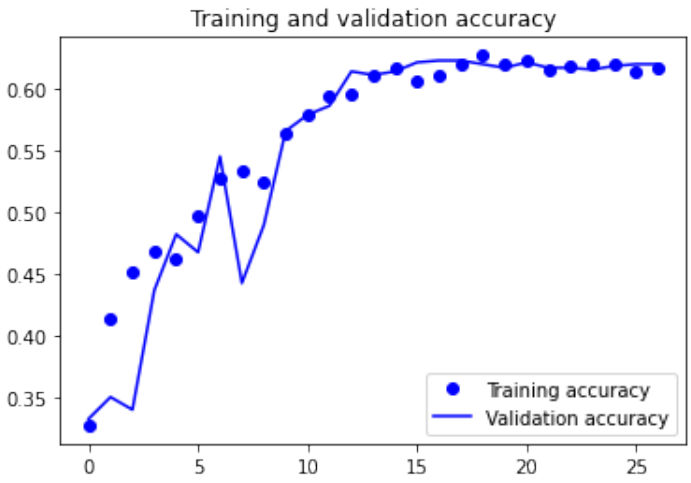
- the first block consists of one *Conv2D* and a *MaxPooling2D* layer with a depth of 32 filters;

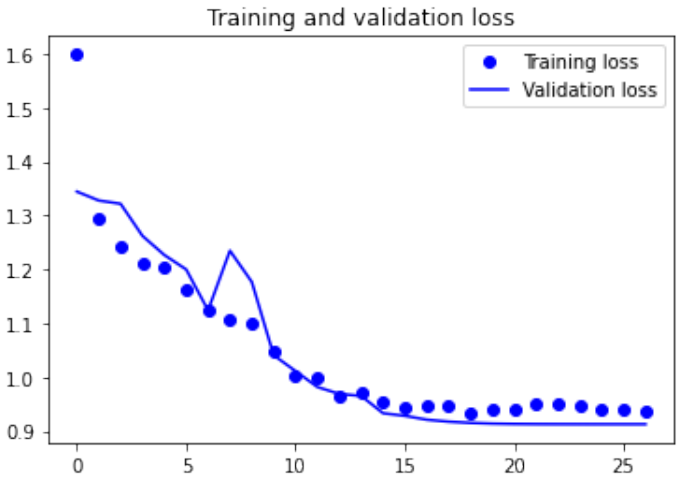
- in the second block the depth is increased to 256 filters and this is done to increase the number of features that the network needs to recognize;
- in the last block the depth is still increased up to 384 filters (this number is choosen because it improves the behaviour of the network) and only here the pooling layer is a *GlobalMaxPooling2D*.

Other noteworthy things are that between the first and the second block there is a *BatchNormalization* layer that normalizes its inputs in different ways during training and during inference.
Any *Conv2D* layer exploit the *ReLu* as activation function because it has many advantages, for example it doesn't require expensive computation and it allows to avoid vanishing gradient problem.
In addiction in the last two convolutional layers padding is used to exploit even corners in order to recognize features and improve the behaviour of the CNN.
After the conv blocks there are some fully connected layers used for classification. The *Flatten* layer simply flattens its inputs which then pass through a *Dropout* layer, used to prevent overfitting, and at the end there is a *Dense* layer. That is the one which has the task to classify the input images (for this is called 'classifier') and it uses the *softmax* as activation function.
The first layer is used to do data augmentation to increase fictitiously the dataset. The number of conv blocks and the number of filters for each block have been chosen following a trial-and-error rule. Particularly at the beginning there were only few layers and then other blocks were added as long as the accuracy increased, until at a certain point it reaches a plateau. At that point the number of filters in the convolution layers has been changed trying to improve the training and validation accuracy.
This network is a lot less complex than the pretrained ones, but it gives acceptable results as can be seen in the next figures. The desired goal was to create a CNN from scratch that wasn't too complex (so trainable with an acceptable amount of resources) but at the same time can give a good result in terms of predictions, as said in [3].
In the compile phase has been used a *Sparse Categorical Crossentropy* as loss function, because there are more then two classes to identify. The optimizer used is *Adam* because even if is slower then *RMSprop*, it can explore a greater part of the solution space; the only one metric used is *accuracy*, that has to be maximized to have better performances.
As said in the previous sections, during training Early Stopping and Reduce Learning Rate On Plateau were used as methods to fight against overfitting and then Model Checkpoint is used to save the trained model, saving the version of the model that achieved the best performance at the end of an epoch.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 96, 96, 3)]	0
rescaling (Rescaling)	(None, 96, 96, 3)	0
conv2d (Conv2D)	(None, 85, 85, 32)	13856

```
max_pooling2d (MaxPooling2D  (None, 84, 84, 32)       0
)

batch_normalization (BatchN  (None, 84, 84, 32)       128
ormalization)

conv2d_1 (Conv2D)            (None, 84, 84, 256)      73984

max_pooling2d_1 (MaxPooling  (None, 42, 42, 256)      0
2D)

conv2d_2 (Conv2D)            (None, 42, 42, 384)      885120

global_max_pooling2d (Globa  (None, 384)              0
lMaxPooling2D)

flatten (Flatten)            (None, 384)              0

dropout (Dropout)            (None, 384)              0

classifier (Dense)           (None, 4)                1540

=================================================================
Total params: 974,628
Trainable params: 974,564
Non-trainable params: 64
_________________________________________________________________
```





The performance are explained from the confusion matrix and the classification report.

	Report precision	**recall**	**f1-score**	**support**
Amanita	0.68	0.28	0.40	75
Boletus	0.68	0.82	0.75	108
Cortinarius	0.53	0.37	0.44	84
Russula	0.55	0.78	0.65	115
accuracy			0.60	382
macro avg	0.61	0.56	0.56	382
weighted avg	0.61	0.60	0.58	382

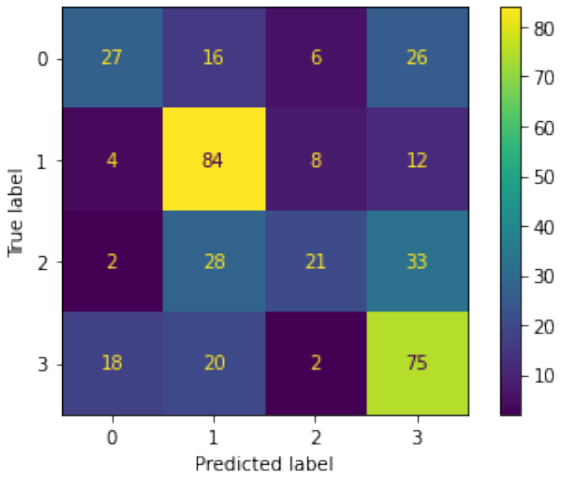


Figure 4: Confusion Matrix

4 Pre-Trained Models

4.1 VGG16

The original VGG16 architecture is shown in the Figure below. The network has 14.714.688 parameters and expects an input of fixed dimension of 224x224x3 and generates an output of 1000 probabilities (since it's been previously trained on the ImageNet database).

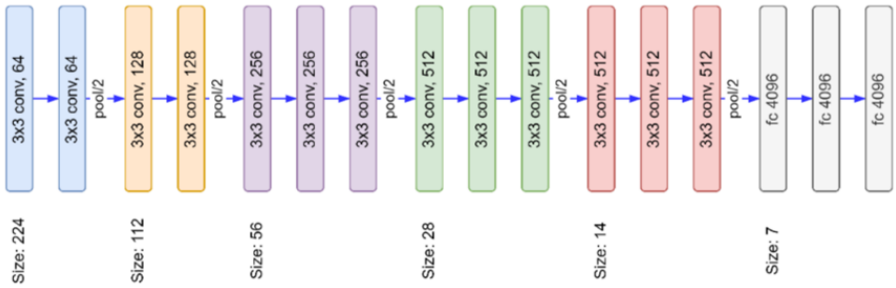


Figure 5: VGG16 structure.

Due to the VGG16 version used includes also a *MaxPooling2D*, it has been chosen to introduce only a *Dropout(0.5)* layer, to avoid overfitting, and a *Dense(class_ number, activation='softmax')* layer.

4.1.1 Features Extraction

The first transfer learning approach used is *Feature Extraction*, implemented blocking all the parameters of the pretrained networks and training only the added top layers.

```
Model: "VGG16_FE"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 299, 299, 3)]     0

 sequential (Sequential)     (None, 299, 299, 3)       0

 tf.__operators__.getitem_1   (None, 299, 299, 3)      0
 (SlicingOpLambda)

 tf.nn.bias_add_1 (TFOpLambd  (None, 299, 299, 3)      0
 a)

 vgg16 (Functional)          (None, 512)               14714688

 dropout_1 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 4)                 2052
```

```
=================================================================
Total params: 14,716,740
Trainable params: 2,052
Non-trainable params: 14,714,688
_________________________________________________________________
```

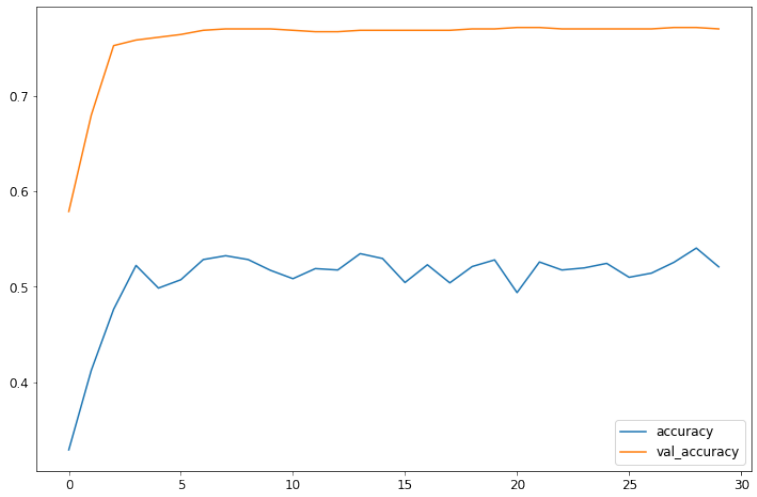


Figure 6: Training and validation accuracy.

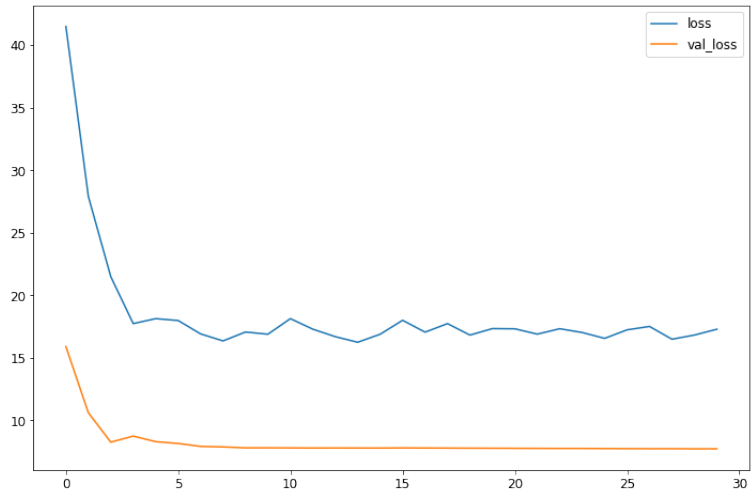


Figure 7: Training and validation loss.

It is immediately noticeable how validation accuracy exceeds train accuracy as early as the first epoch. This strange phenomenon will be investigated later in section 4.1.3

The performance are explained using the confusion matrix and the classification report.

Report	precision	recall	f1-score	support
Amanita	0.64	0.48	0.55	75
Boletus	0.68	0.90	0.78	108
Cortinarius	0.70	0.54	0.61	84
Russula	0.64	0.67	0.66	115
accuracy			0.67	382
macro avg	0.67	0.65	0.65	382
weighted avg	0.67	0.67	0.66	382

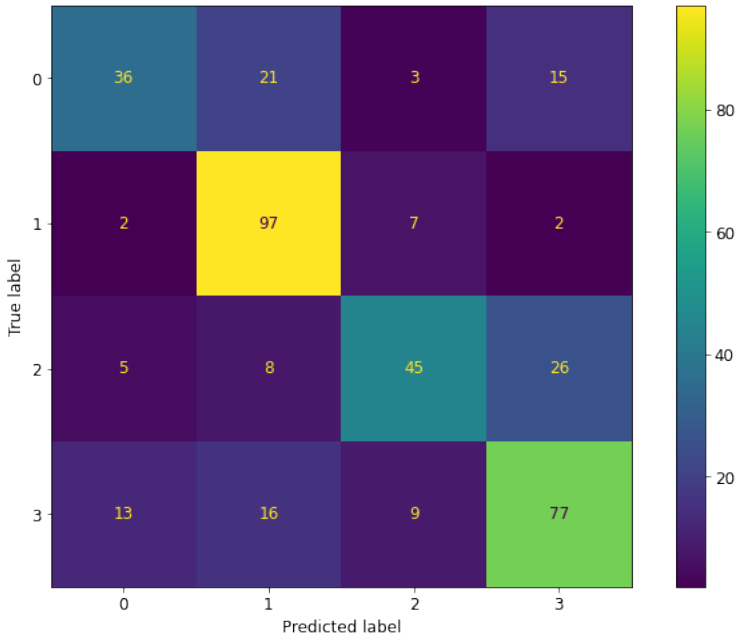


Figure 8: Confusion Matrix.

4.1.2 Fine Tuning

Starting from the model obtained with the *Feature Extraction*, the *block5_ conv3* (Conv2D) layer of the VGG16 has been unfreezed and then trained together with top layer.

```
Model: "VGG16_FT"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 299, 299, 3)]     0

 sequential (Sequential)     (None, 299, 299, 3)       0

 tf.__operators__.getitem_1   (None, 299, 299, 3)      0
 (SlicingOpLambda)
```

```
tf.nn.bias_add_1 (TFOpLambd  (None, 299, 299, 3)      0
a)

vgg16 (Functional)           (None, 512)              14714688

dropout_1 (Dropout)          (None, 512)              0

dense_1 (Dense)              (None, 4)                2052

=================================================================
Total params: 14,716,740
Trainable params: 2,361,860
Non-trainable params: 12,354,880
_________________________________________________________________
```

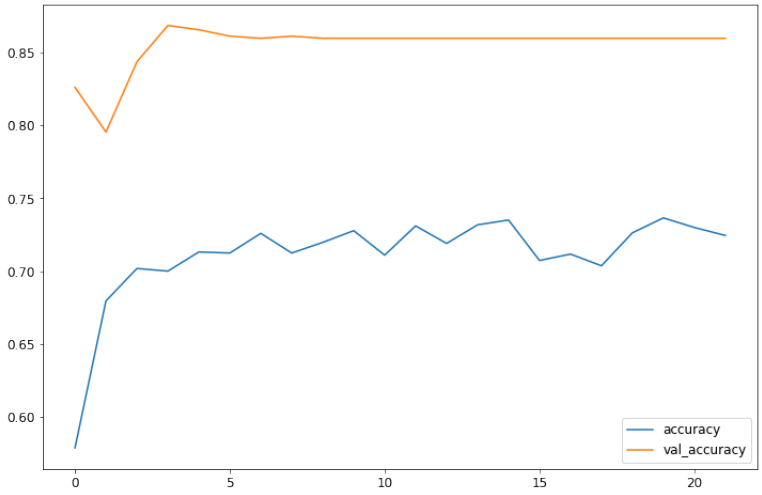


Figure 9: Training and validation accuracy.

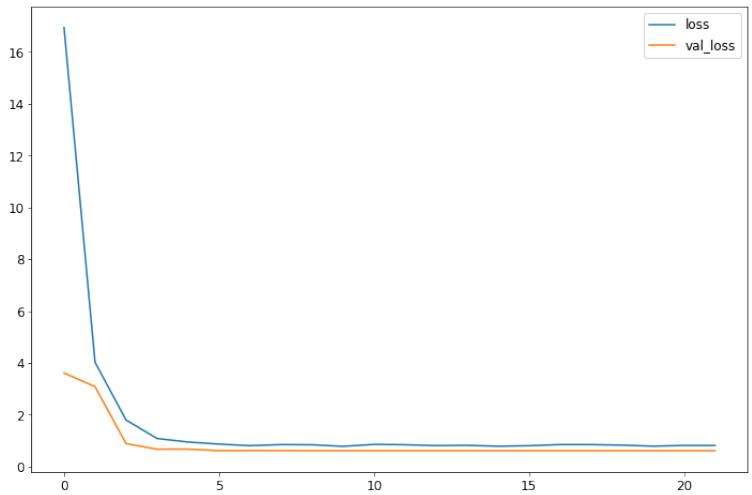


Figure 10: Training and validation loss.

The performance are explained from the confusion matrix and the classification report.

Report	**precision**	**recall**	**f1-score**	**support**
Amanita	0.63	0.84	0.72	75
Boletus	0.88	0.91	0.89	108
Cortinarius	0.82	0.61	0.70	84
Russula	0.80	0.76	0.78	115
accuracy			0.78	382
macro avg	0.78	0.78	0.77	382
weighted avg	0.79	0.78	0.78	382

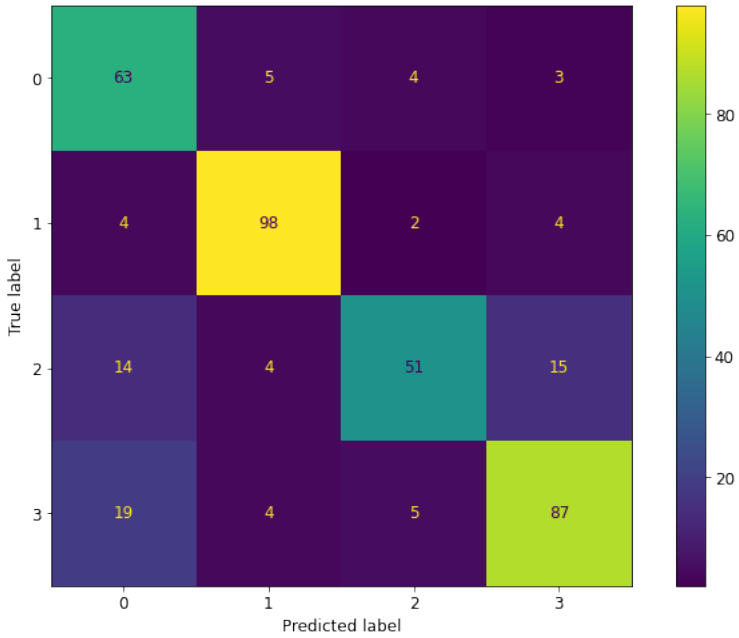


Figure 11: Confusion Matrix.

4.1.3 Validation Accuracy Investigation

Trying to understand why validation accuracy (loss) was higher (lower) than train validation (loss), the first idea was to attribute the phenomenon to the high value assigned to the *Dropout* layer. The latter in fact disables some features during training that are used during validation instead. But by reducing or eliminating it completely, the above behavior was only partially reduced. At that point, we began to speculate that the *Data Augmentation* layer was responsible. This layer in fact acts only on the train dataset, making the classification task much more complex than the one performed on the validation dataset, which will therefore have a higher accuracy and a lower loss value.

At this point we decide to train a second model removing both *Droput* and *Data Augmentation* layers, finding out even better performance than the first one after fine tuning. The following is the result obtained:

```
Model: "VGG16_FT_no_DataAugumentation"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_8 (InputLayer)        [(None, 299, 299, 3)]     0

 tf.__operators__.getitem_3   (None, 299, 299, 3)      0
 (SlicingOpLambda)

 tf.nn.bias_add_3 (TFOpLambd  (None, 299, 299, 3)      0
 a)
```

```
 vgg16 (Functional)          (None, 512)               14714688

 dense_3 (Dense)             (None, 4)                 2052

=================================================================
Total params: 14,716,740
Trainable params: 2,361,860
Non-trainable params: 12,354,880
_________________________________________________________________
```

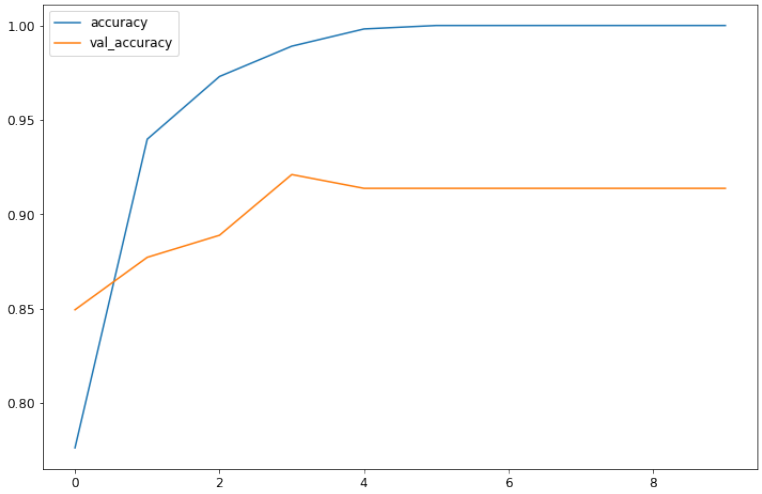


Figure 12: Training and validation accuracy.

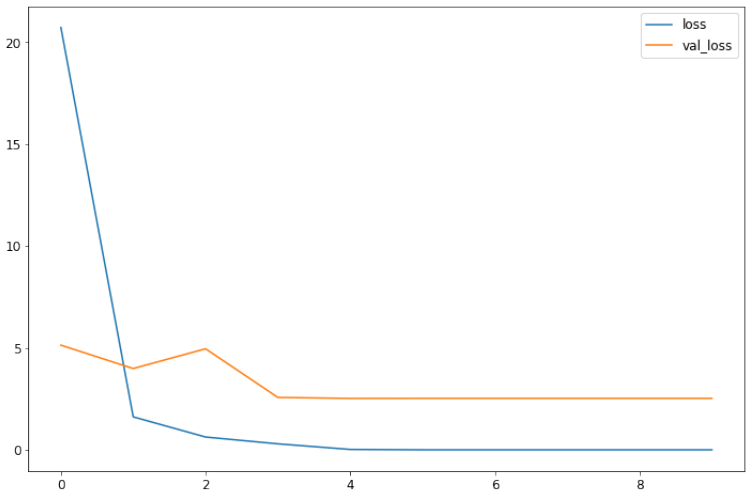


Figure 13: Training and validation loss.

The phenomenon is not present anymore. The performance are explained from the confusion matrix and the classification report.

Report	precision	recall	f1-score	support
Amanita	0.78	0.87	0.82	75
Boletus	0.92	0.96	0.94	108
Cortinarius	0.87	0.79	0.82	84
Russula	0.85	0.82	0.84	115
accuracy			0.86	382
macro avg	0.86	0.86	0.86	382
weighted avg	0.86	0.86	0.86	382

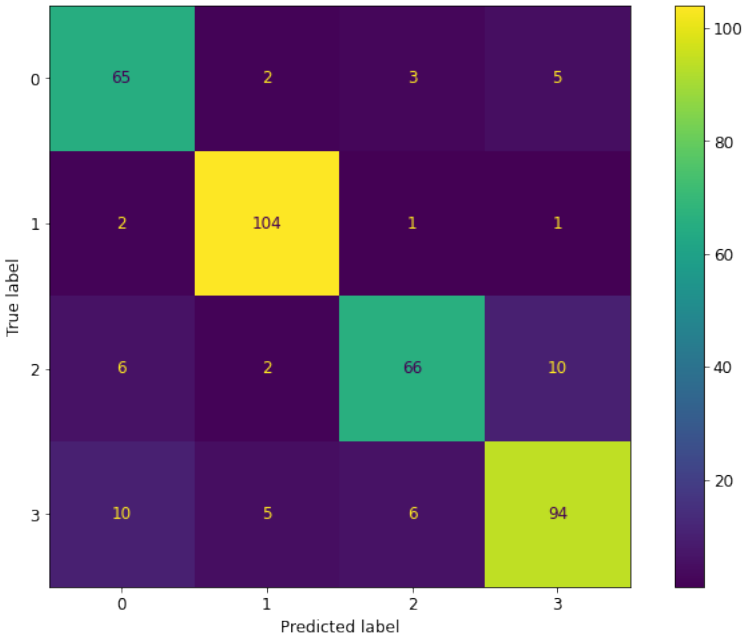
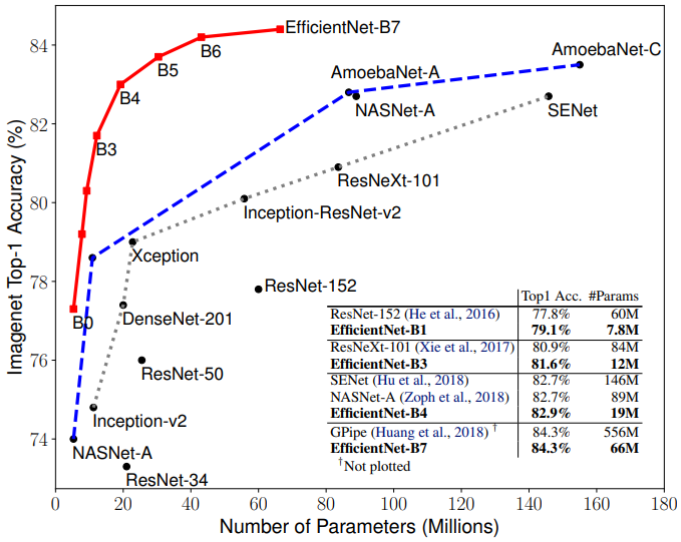


Figure 14: Confusion Matrix.

The contra of this model are that without *Data Augmentation* and *Dropout* it loses in robustness.

4.2 EfficientNetB7

Searching in the literature, EfficientNet results to be a state-of-art CNN. As shown in [2], in its simplest version EfficientNetB0, it is able to outperform in accuracy ResNet50, using less than half number of parameters.



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

Figure 15: Pretrained CNN comparison.

In particular, we selected EfficientNetB7 which is able to achieve a 98.8% on Flowers dataset [7] using transfer learning approach. EfficientNetB7 can be obtained starting from the base EfficentNetB0 architecture and applying *Compound Scaling Method* as explained in [2].

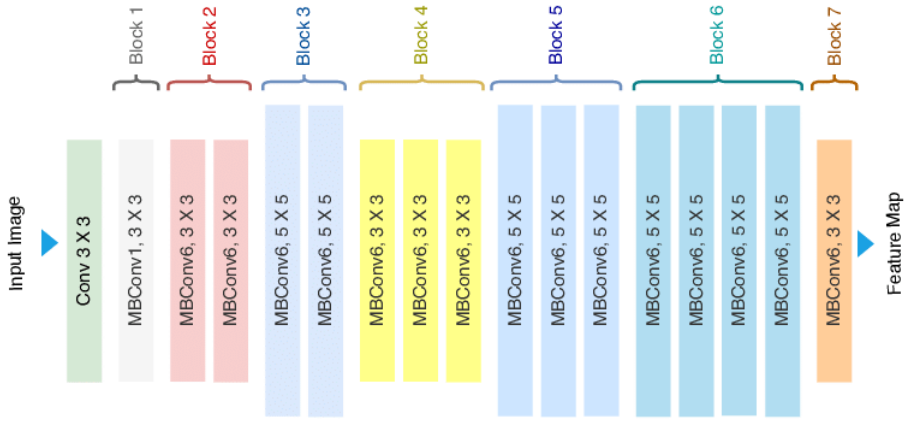


Figure 16: EfficientNetB0 structure.

Since the pretrained CNN already includes a *MaxAveragePolling* layer before the top layers, has been decided to introduce only a *Dropout(0.28)* layer and a

Dense(class_number, activation='softmax') layer. As shown below, even with only these two layers, the performances of the network are highly satisfactory.

4.2.1 Features Extraction

The first transfer learning approach used is *Feature Extraction*, implemented blocking all the parameters of the pretrained networks and training only the added top layers.

```
Model: "EFFNET_FE"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential (Sequential)      (None, 299, 299, 3)       0

efficientnetb7 (Functional)  (None, 2560)              64097687

dropout (Dropout)            (None, 2560)              0

dense (Dense)                (None, 4)                 10244

=================================================================
Total params: 64,107,931
Trainable params: 10,244
Non-trainable params: 64,097,687
_________________________________________________________________
```

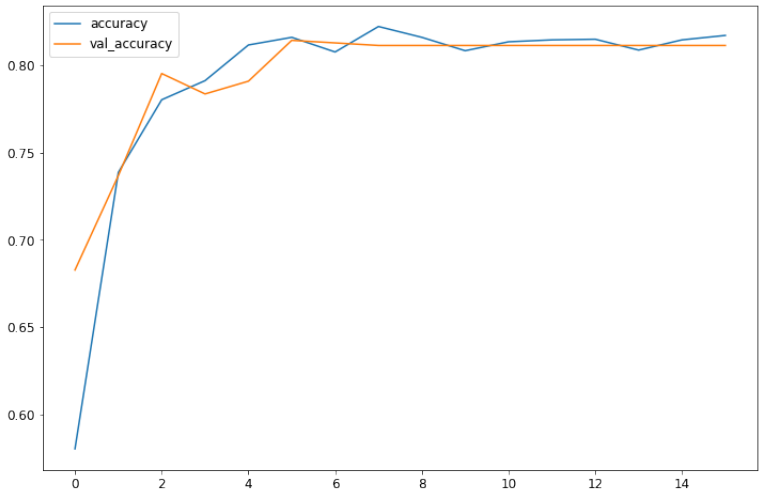


Figure 17: Training and validation accuracy.

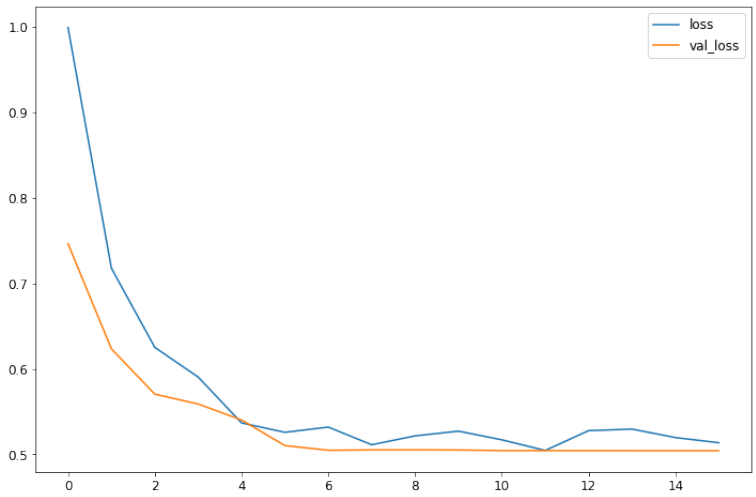


Figure 18: Training and validation loss.

The performance are explained from the confusion matrix and the classification report.

Report	**precision**	**recall**	**f1-score**	**support**
Amanita	0.89	0.87	0.88	75
Boletus	0.77	1.00	0.87	108
Cortinarius	0.91	0.46	0.61	84
Russula	0.76	0.83	0.80	115
accuracy			0.81	382
macro avg	0.83	0.79	0.79	382
weighted avg	0.82	0.81	0.79	382

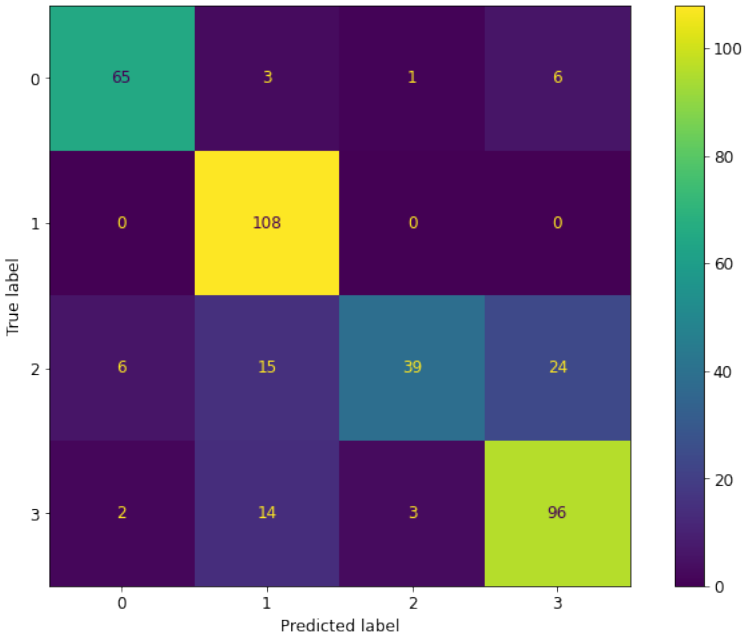


Figure 19: Confusion Matrix.

4.2.2 Fine Tuning

Starting from the model obtained with the *Feature Extraction*, the *top_ conv* (Conv2D) layer of the EfficientNetB7 has been unfreezed and then trained together with top layer.

```
Model: "EFFNET_FT"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, None, None, None)  0

 efficientnetb7 (Functional)  (None, 2560)             64097687

 dropout (Dropout)           (None, 2560)              0

 dense (Dense)               (None, 4)                 10244

=================================================================
Total params: 64,107,931
Trainable params: 1,653,764
Non-trainable params: 62,454,167
_________________________________________________________________
```

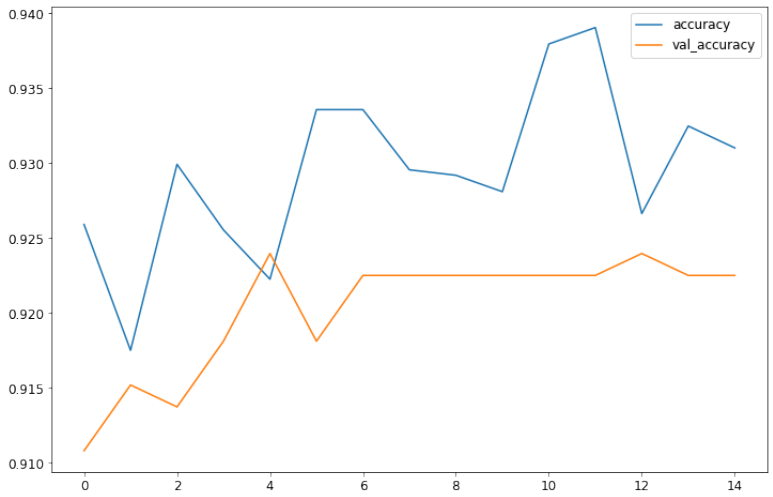


Figure 20: Training and validation accuracy.

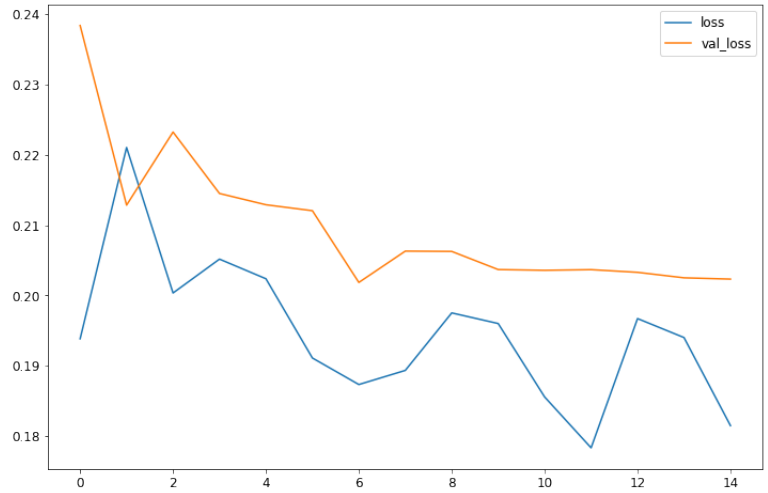


Figure 21: Training and validation loss.

The performance are explained from the confusion matrix and the classification report.

	Report precision	**recall**	**f1-score**	**support**
Amanita	0.95	0.95	0.95	75
Boletus	0.96	1.00	0.98	108
Cortinarius	0.99	0.88	0.93	84
Russula	0.94	0.97	0.96	115
accuracy			0.96	382
macro avg	0.96	0.95	0.95	382
weighted avg	0.96	0.96	0.96	382

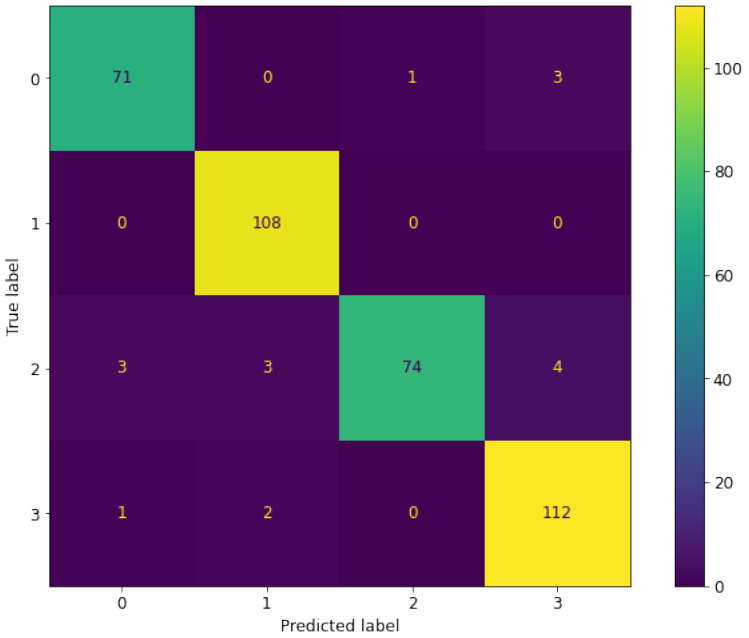


Figure 22: Confusion Matrix

5 Error Analysis

In this section a general study on the total amount of miss-classified images by all the models is carrying out with the simultaneous analysis of the whole set of correct classified images by all the models; it has allowed to understand the general behavior of each network. This kind of study is called 'Error Analysis'.

To do this, the images that were miss-classified by all the four best classifiers have been plotted and compared to correctly classified images.

The classifiers considered are:

- Model 1: CNN from scratch 3
- Model 2: VGG16 without Data Augmentation 4.1.3
- Model 3: VGG16 4.1.2
- Model 4: EfficientNetB7 4.2.2


Figure 23: Correct classified (Left) vs incorrect classified image (Right)


Figure 24: Correct classified (Left) vs incorrect classified image (Right)


Figure 25: Correct classified (Left) vs incorrect classified image (Right)

Figure 26: Correct classified (Left) vs incorrect classified image (Right)

5.1 Summary

```
Images that are misclassified by all classifiers:  8
--------
Images that are misclassified only by FromScratch:  79
Images that are correctly classified only by FromScratch:  2
--------
Images that are misclassified only by VGG16_no_DA:  3
Images that are correctly classified only by VGG16_no_DA:  3
--------
Images that are misclassified only by VGG16:  34
Images that are correctly classified only by VGG16:  0
--------
Images that are misclassified only by EfficientNet:  3
Images that are correctly classified only by EfficientNet:  7
```

6 Model Comparison

The following tables are a summary comparison of the four classifiers mentioned in the previous section, based on the metrics mentioned in the section 1.3.

Model	**Accuracy**	**Loss**	**ROC AUC**	**F1-Macro**
CNN from Scratch	0.60	0.93	0.73	0.50
VGG16 no DA	0.93	2.61	0.94	0.86
VGG16	0.78	7.54	0.82	0.77
EfficentNetB7	0.96	0.14	0.97	0.95

Table 1: Models comparison.

Model	**F1-Amanita**	**F1-Boletus**
CNN from Scratch	0.43	0.66
VGG16 no DA	0.82	0.94
VGG16	0.72	0.89
EfficentNetB7	0.95	0.98
Model	**F1-Cortinarius**	**F1-Russula**
CNN from Scratch	0.35	0.57
VGG16 no DA	0.82	0.84
VGG16	0.70	0.78
EfficentNetB7	0.93	0.96

Table 2: F1 score for each class.

As might be expected from looking at the Confusion Matrix in the figure 22, the best performance is achieved by EfficientNetB7 for each index. This is mainly due to the high starting accuracy of the aforementioned pretrained network.

7 Explainability

In this section will be analyzed how the models are able to predict the class of mushrooms images received in input through the analysis of the intermediate activations, the filters pattern of a certain layer and the class activation heatmaps in an image. It's notable the fact that for this analysis it has been used only the best pretrained VGG16 network.

7.1 Intermediate activations

We have built a multi-output model which, given an image as input, allows you to print on the screen all the activations maps obtained from each convolutional and Max-pooling layer of the network.
Following are shown some activations of the network created from scratch, that has many convnet-max pooling blocks, each one producing a number of activation maps that depends on the number of kernels used and varies from 32 to 384.

Figure 27: The 32 activation maps produced by the first layer of the from scratch model

First layer. Looking at Figure 27, we can see how the level of abstraction with which the images are considered is very low, in fact initially the network deals with identifying some generic characteristics, such as corner or edge, and for this reason almost all the information present in the image is preserved.
In the notebooks it is also possible to view the activation maps extracted from the other models and it can be seen that for the first layers the information extracted is almost the same. This highlights why in transfer learning through fine-tuning usually the first layers are left blocked, in fact it is possible to reuse already pre-trained features without reducing performances.
In the next figure it is possible to see how the abstraction level increases with deep layers and try to recognize more and more particular features. Therefore, since the output is negative, the ReLU, which is used as activation function, clears the final output of the feature map.

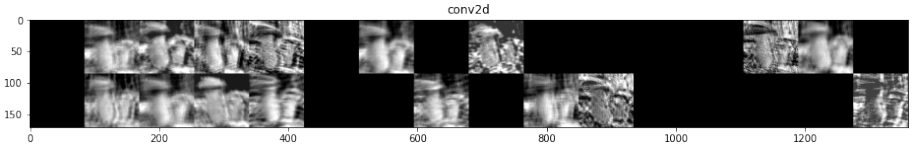


Figure 28: The activation maps produced by the first layer of the from scratch model

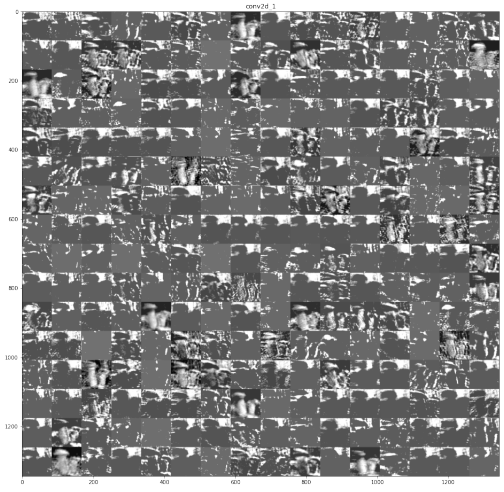


Figure 29: The activation maps produced by the second layer of the from scratch model

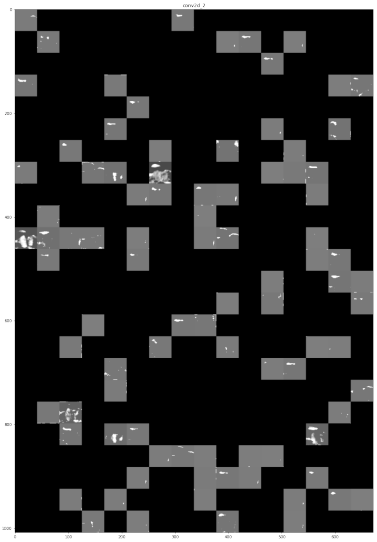


Figure 30: The activation maps produced by the third layer of the from scratch model

7.2 Visualizing convnet filters

In this section we are going to display the visual pattern that each filter is meant to respond to, so which input image maximizes the response of a specific filter. In this case, gradient ascent is used in the input space (the pixels of the image), in order to maximize the response of a specific filter.
To do this, a feature extractor was create; this is used to define a function that returns a scalar value quantifying how much a given input image activates a given filter in the layer, so given the image, the layer and the index of the filter, it's possible to compute the activation and the mean value of the filter activations (fitness), usually excluding borders.
Basically this section is used to show, with images, what a certain layer is looking for when it's processing an input image.

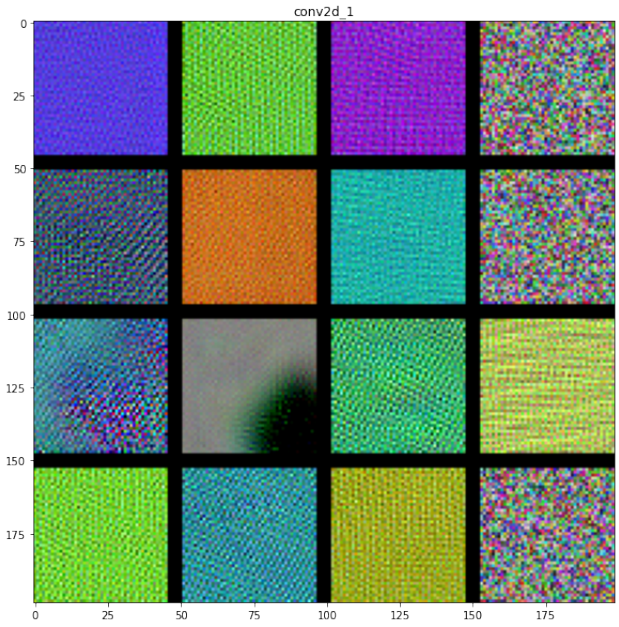


Figure 31: Preferred pattern of every single filter in the second convolutional layer of the from scratch model

In Figure 31 every square represent, in an image, the filters that are applied by the second convolutional layer to the input images during processing for the from scratch model.
The following figures represent the same thing but for the second convolutional layer of the pretrained VGG16 and for the third convolutional layer of the pretrained EfficientNetB7 respectively.

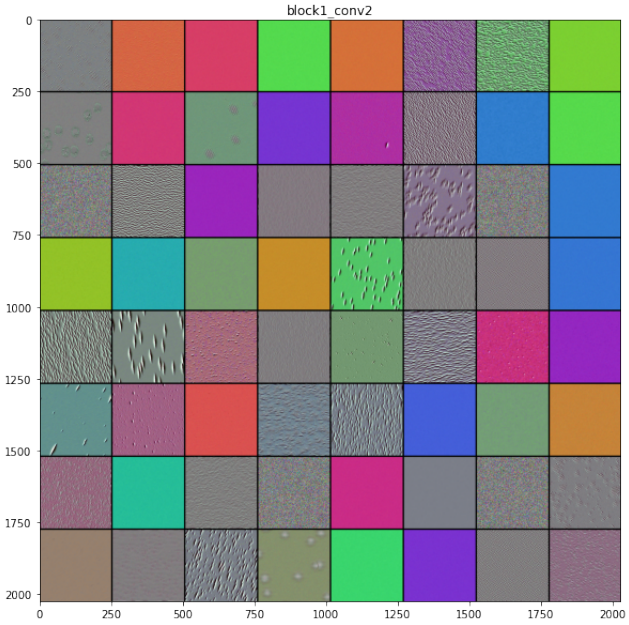


Figure 32: Preferred pattern of every single filter in the second convolutional layer of the VGG16 pretrained model

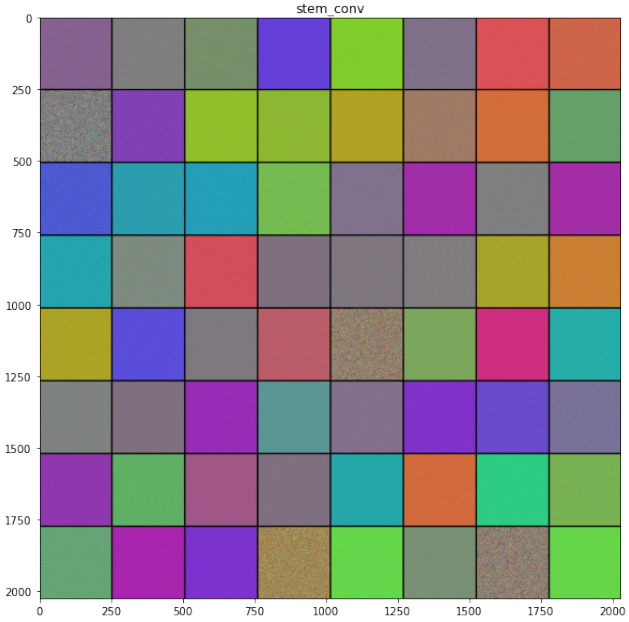


Figure 33: Preferred pattern of every single filter in the first convolutional layer of the EfficientNetB7 pretrained model

From the figures it's clear that the pattern recognized by the filters are very simple (i.e. vertical or horizontal lines) and this is due to the fact that the layers chosen are at the beginning of the networks, so if deeper layers are kept it may

be possible that the pattern result more complex.

7.3 Visualizing heatmaps of class activation

This section is particularly important to understand how our models take decision and if they take that focusing on the same part of the image.
To obtain it has been built a model that can produce an heatmap of class activation over input image, that is a grid that visually show which location is more considered to detect the right class.
This heatmap is given by the channel-wise mean of the feature maps produced by the last convolutional layer, that weighs them depending on the importance of each channel with regard to the top predicted class.
Following are shown heatmaps analysis obtained with our models and using four images, respectively of the *Amanita*, the *Boletus*, the *Cortinarius* and the *Russula* class, that were correctly classified by all the models.

7.3.1 Amanita

As can be seen from the following images, recognition of this species is mostly done through identification of the mushroom's volva. Notably, for the from scratch network, the other relevant parts of the fungus do not produce any kind of activation.

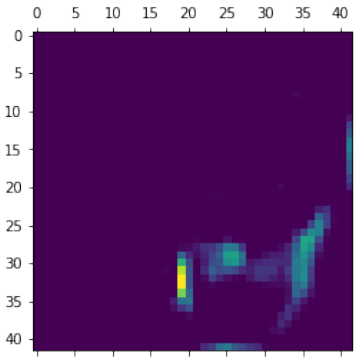


Figure 34: From scratch

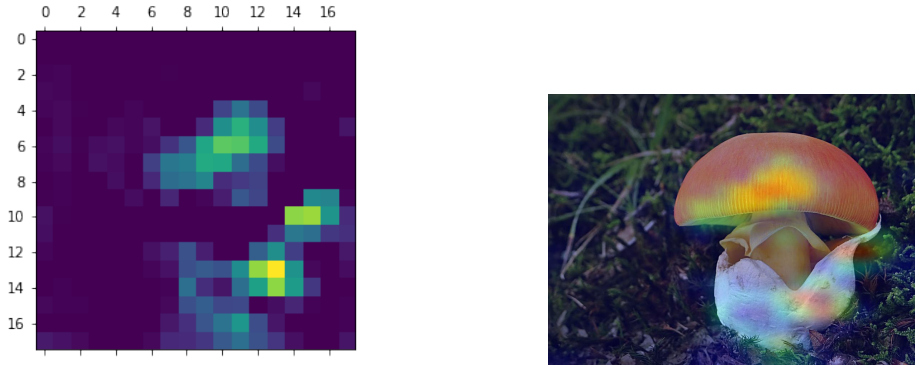


Figure 35: VGG16

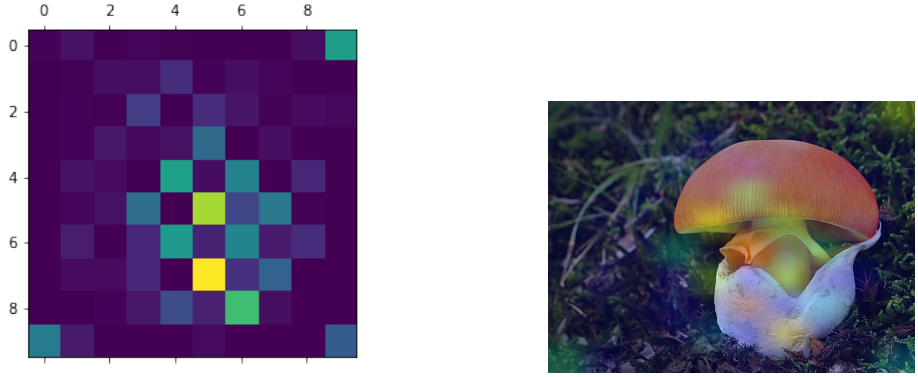


Figure 36: EfficientNetB7

7.3.2 Boletus

Here it's clear that all models are able to recognize the mushroom, but they put relevance in different parts of it. The from scratch model recognizes the hat of the two bigger mushrooms, partially ignoring the smallest one. The VGG16 network is able to identify almost all the two biggest mushrooms, but the best overall results have been achieved with the EfficientNetB7 network, because it can recognize a whole region where there are mushrooms.

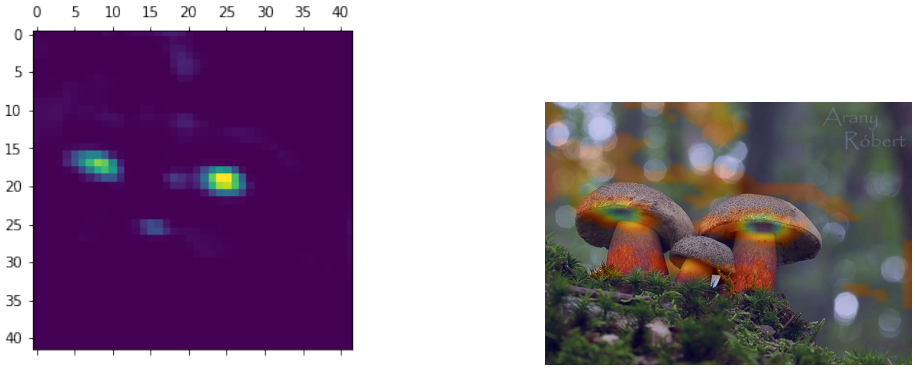


Figure 37: From scratch

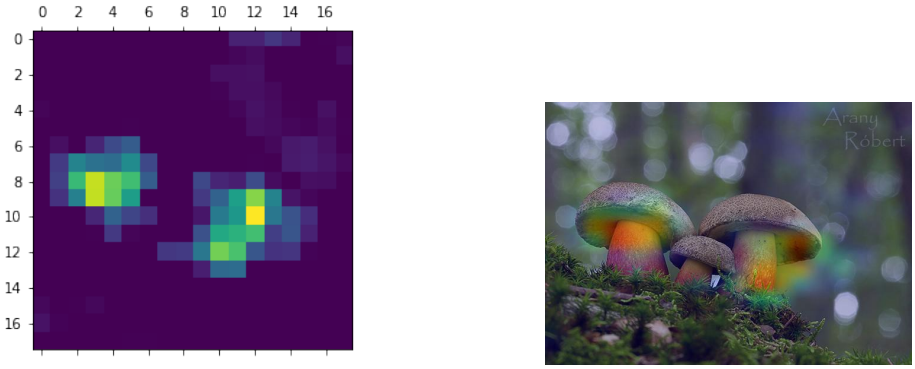


Figure 38: VGG16

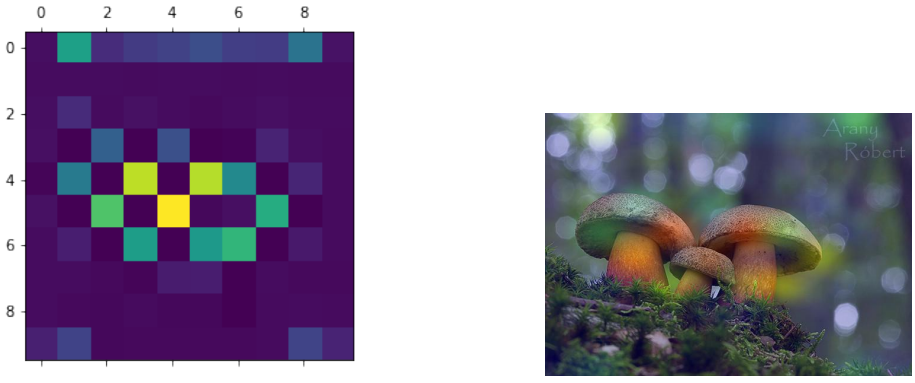


Figure 39: EfficientNetB7

7.3.3 Cortinarius

Similarly to the Boletus class, the from scratch network can recognize the hat of the left mushroom, but the other models identify in a better way the other mushroom. Probably this is due to the small number of layer of the from scratch model, so it can not see too specific features. In addiction for this class, the VGG16 network behaves slightly better then the EfficientNetB7.

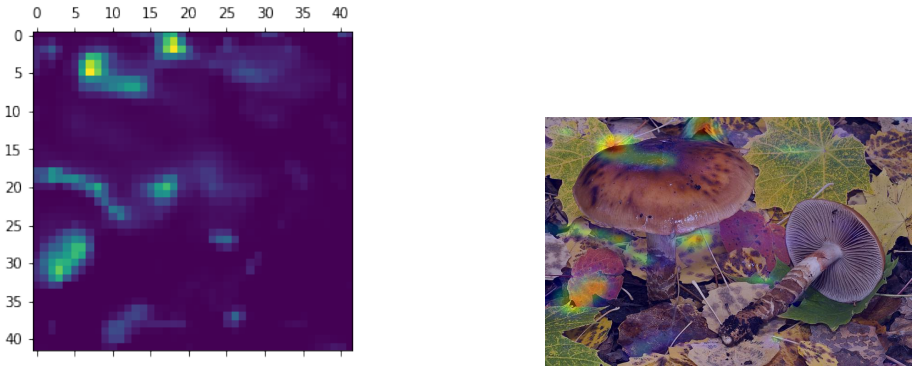


Figure 40: From scratch

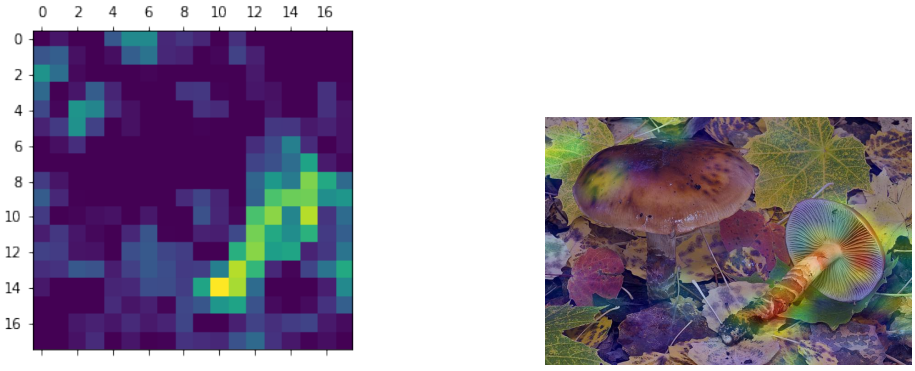


Figure 41: VGG16

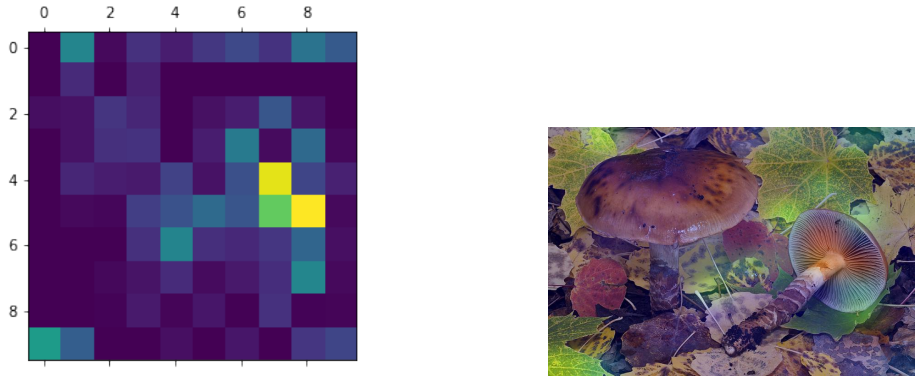


Figure 42: EfficientNetB7

7.3.4 Russula

In the following image, taken from the test set, is evident the most interesting region to be detected.

Figure 43: From scratch

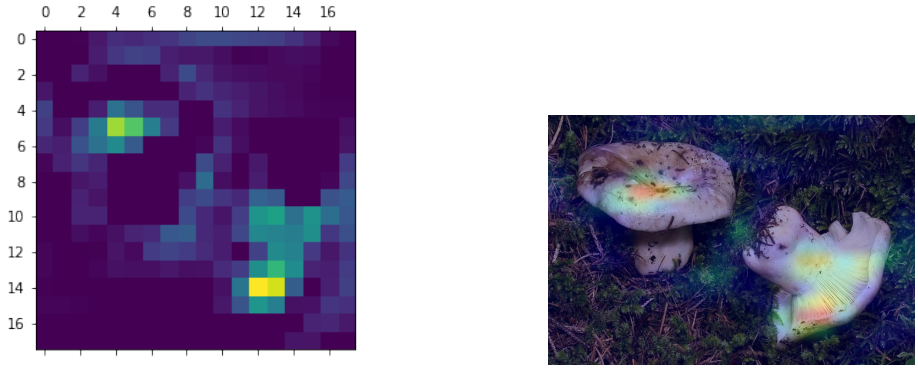


Figure 44: VGG16

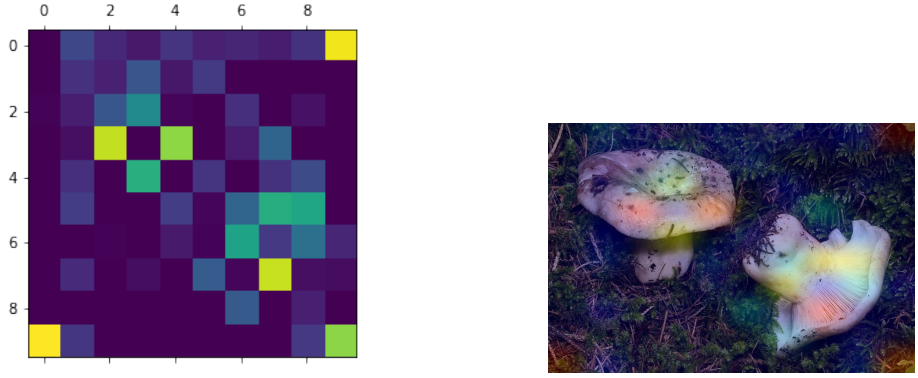


Figure 45: EfficientNetB7

For the *Boletus* and the *Russula* classes the behaviours of the three models are more or less the same, in fact the from scratch identifies the hat of one mushroom, while the other models can recognize all the mushrooms in the figures. As in the first case, EfficientNetB7 has a better behaviour than VGG16, even if in Figure 45 and 36 there are some outliers.

8 Conclusion

This project provides a series of methods to recognize and classify some mushrooms species.
Because the task is relatively simple, models developed with transfer learning methods manage to achieve very high levels of accuracy. The situation changes for the from scratch model, which manages to achieve a maximum accuracy of 0.60% on the test dataset. Certainly, a deeper network would have performed better, but required more resources and computational cost. At the end it's possible to say that all methods are quite good to accomplish this task.

It was also very interesting to perform the Explainability Analysis in order to better understand what was happening within each layer of the various models and why one performs better than the other.

References

[1] Eyad Alkronz et al. "Prediction of Whether Mushroom is Edible or Poisonous Using Back-propagation Neural Network." In: *International Journal of Corpus Linguistics* 3 (Mar. 2019), pp. 1–8.

[2] Mingxing Tan and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *CoRR* abs/1905.11946 (2019). arXiv: 1905.11946. URL: http://arxiv.org/abs/1905.11946.

[3] Jitdumrong Preechasuk et al. "Image Analysis of Mushroom Types Classification by Convolution Neural Networks". In: *Proceedings of the 2019 2nd Artificial Intelligence and Cloud Computing Conference*. AICCC 2019. Kobe, Japan: Association for Computing Machinery, 2020, pp. 82–88. ISBN: 9781450372633. DOI: 10.1145/3375959.3375982. URL: https://doi.org/10.1145/3375959.3375982.

[4] Nusrat Zahan et al. "A Deep Learning-Based Approach for Edible, Inedible and Poisonous Mushroom Classification". In: *2021 International Conference on Information and Communication Technology for Sustainable Development (ICICT4SD)* (2021), pp. 440–444.

[5] Mark Jayson Y. Sutayco and Meo Vincent C. Caya. "Identification of Medicinal Mushrooms using Computer Vision and Convolutional Neural Network". In: *2022 6th International Conference on Electrical, Telecommunication and Computer Engineering (ELTICOM)*. 2022, pp. 167–171. DOI: 10.1109/ELTICOM57747.2022.10038007.

[6] URL: https://www.kaggle.com/datasets/maysee/mushrooms-classification-common-genuss-images.

[7] URL: https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/1ECTVN.