Fruits and Vegetables

Computational Intelligence and Deep Learning final project
Prof. Beatrice Lazzerini
Alessandro Renda, PhD

Leonardo Turchetti
Lorenzo Tonelli

Msc. in Artificial Intelligence and Data Engineering

January 28, 2022

Contents

1 Introduction

The objective of this project was to use a structured dataset to build, train, test and compare different classifier models using the most common Deep Learning techniques. The dataset used contains images of fruits and vegetables labeled using the specific names. Two classification tasks were tackled:

- **binary classification**: classify between `vegetables` and `fruits`;
- **multiclass classification**: classify using the different vegetables and fruits labels (e.g. `banana`, `apple`, `oranges`, `eggplant`, `bell pepper`);

2 Task 1 - Preprocessing

The dataset used for this work was taken from Kaggle[1]. This dataset contains images of the following food items:

- **fruits**: banana, apple, pear, grapes, orange, kiwi, watermelon, pomegranate, pineapple, mango;
- **vegetables**: cucumber, carrot, capsicum, onion, potato, lemon, tomato, raddish, beetroot, cabbage, lettuce, spinach, soy bean, cauliflower, bell pepper, chilli pepper, turnip, corn, sweetcorn, sweet potato, paprika, jalepeño, ginger, garlic, peas, eggplant.

The original dataset structure consisted of three folders:

- `train` (100 images each);
- `test` (10 images each);
- `validation` (10 images each);

each of the above folders contains subfolders for different fruits and vegetables where in the images for respective food items are present.

[1] `https://www.kaggle.com/kritikseth/fruit-and-vegetable-image-recognition`

```
train
├── apple
│   ├── img_1.jpg
│   ├── img_2.jpg
│   ├── img_3.jpg
│   └── ...
├── banana
│   ├── img_1.jpg
│   ├── img_2.jpg
│   ├── img_3.jpg
│   └── ...
├── beetroot
├── carrot
└── ...
test
├── apple
│   ├── img_1.jpg
│   ├── img_2.jpg
│   ├── img_3.jpg
│   └── ...
├── banana
│   ├── img_1.jpg
│   ├── img_2.jpg
│   ├── img_3.jpg
│   └── ...
├── beetroot
├── carrot
└── ...
validation
├── apple
├── banana
├── beetroot
├── carrot
└── ...
```

The images in the dataset were originally scraped from Bing Image Search.

After starting to do some preliminary tests, building the first scratch classifiers, we began to find very high anomalous values of Testing and Validation accuracy in the results. We have therefore started a very thorough data exploration phase of the dataset. From this operation a series of anomalies and distortions emerged that we went to correct in the preprocessing phase.

2.1 Duplicate Images

The first thing we found is that in the original structure of the dataset there were duplicates between the images present in the training set and those present in the validation and in the test set. So first of all we have unified training, testing and validation by only preserving the labels of each type of fruit and vegetable. The initial dataset was then transformed into the following structure:

```
├── apple
│   ├── img_1.jpg
│   ├── img_2.jpg
│   ├── img_3.jpg
│   └── ...
├── banana
│   ├── img_1.jpg
│   ├── img_2.jpg
│   ├── img_3.jpg
│   └── ...
├── beetroot
│   ├── img_1.jpg
│   ├── img_2.jpg
│   ├── img_3.jpg
│   └── ...
├── carrot
├── garlic
├── ginger
├── kiwi
├── lemon
└── ...
```

Using Python `cv2` module we identified images with the same exact content but different file names; such images were deleted.

2.2 Manual Cleanup

Another problem that came up was that some many of the images found in the directories were not related to the class label but accidentally ended up in the dataset as a result of the web scraping procedure used to collect images from Bing Images; after many attempts of writing a functioning interactive script to visualize and delete such images, the choice was made to perform this task manually since the Jupiter notebook does not provide the level of interaction required in order to progressively visualize all images in the dataset and decide which ones to keep and which ones to delete. The manual cleanup was conducted with the following criteria in mind:

- the `corn` and `sweet corn` classes were merged because with extremely similar images;
- the `capsicum` and `bell pepper` classes were merged because with extremely similar images;
- the `paprika` class was deleted because is not a label that can be associated to fruit or vegetable;
- images containing dishes and meals based on the class label fruit or vegetable were deleted;
- images containing fields of the given fruit or vegetable were deleted;
- images completely unrelated with the class label were deleted;

We noticed the presence of different extensions for the images in the dataset: `png` (212), `jpg` (2325), `jpeg` (48), `JPG` (66); `tf.keras` handles all these different formats without much of a hassle.

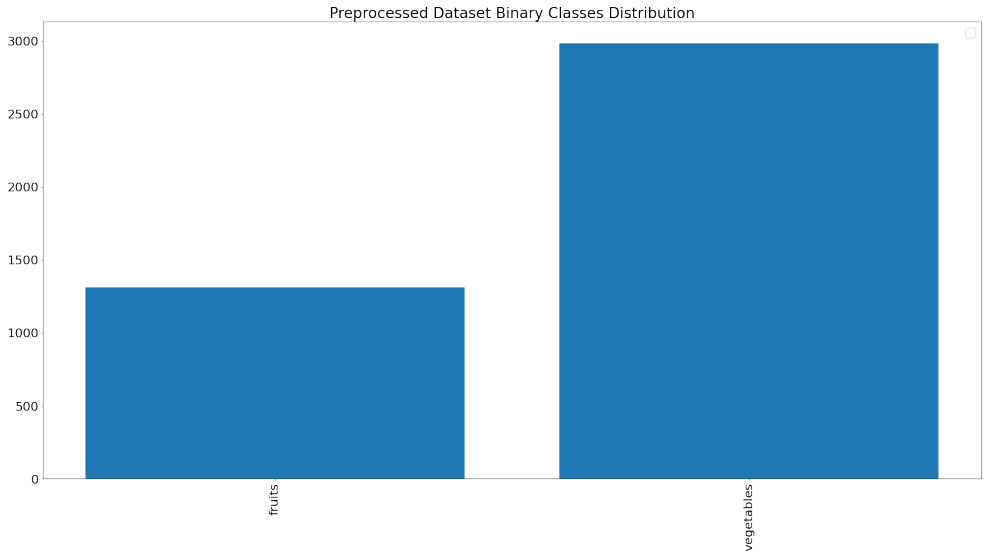
2.3 Binary Classification

Among the objectives we set for our project, we also wanted to develop a model capable of discriminating between `fruits` and `vegetables`, and not only classifying among the subclasses `apple`, `banana`, `beetroot`, `carrot`, `garlic`, `ginger`, `kiwi`, `lemon`, etc.... For ease of use we created a dataset were images are splitted according to the two parent classes with the following structure:

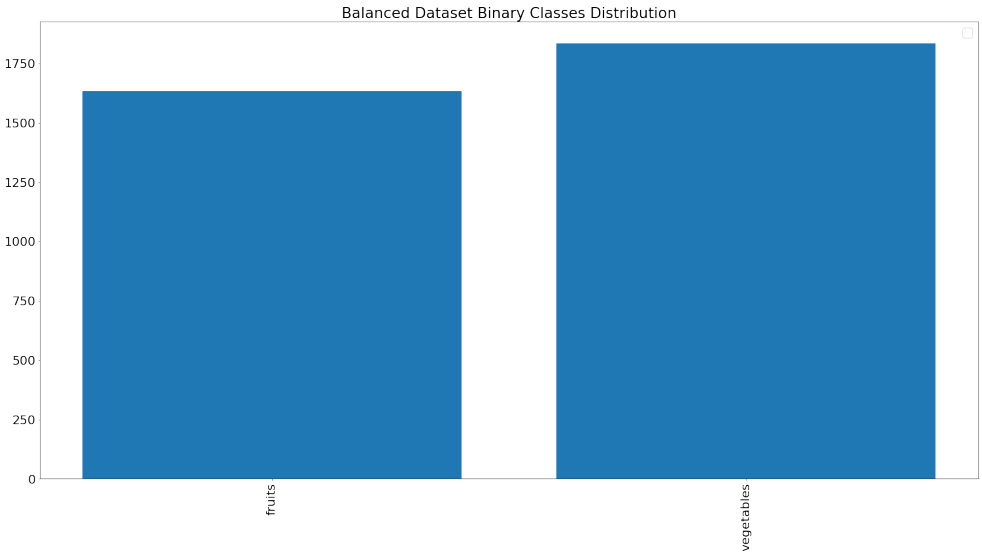
```
├── fruits
│   ├── img_1.jpg
│   ├── img_2.jpg
│   ├── img_3.jpg
│   └── ...
└── vegetables
    ├── img_1.jpg
    ├── img_2.jpg
    ├── img_3.jpg
    └── ...
```

2.4 Imbalanced Dataset

After the creation of the dataset for the binary classification, the classes distribution of this dataset resulted as follows:



So in this case we decided to oversample the fruit class to try to balance the dataset and after this operation the classes distribution of this dataset resulted as follows:



3 Task 2 - Training from scratch

In this section we try to build a CNN from Scratch to solve both the binary and the multi-label classification problems.

3.1 Binary Classification

In order to build the CNN model, an incremental approach was adopted. This incremental approach consisted of developing several models in a trial and error fashion in order to find the best one[2]. The images of the dataset were resized to $180x180$. The number of epochs used during the training stage is increased as the complexity of the structure of the network increases. This incremental approach allowed us to detect with precision the moment in which the network start displaying overfitting behavior. For each of the proposed models, we provide the plot of the accuracy and of the loss function computed on both the training and test sets.

3.1.1 Model 1

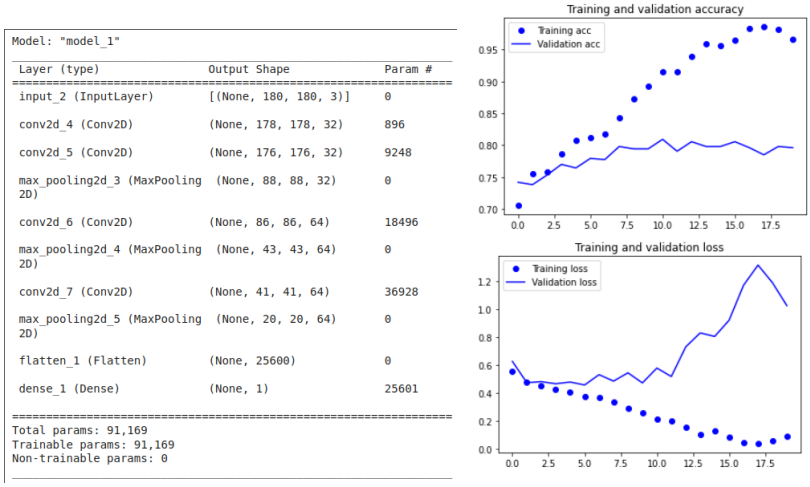


Figure 1: Task 2 - Model 1 Summary and Accuracy Loss curves

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 1	6	0.779	0.720	0.458	0.561

Table 1: Metrics Model 1

[2]For additional information please see: `Task 2.1_Binary - Training from scratch.ipynb`.

Validation loss keeps increasing after epoch 6 which means that the network is basically learning by heart. Although the very minimal structure of this initial network, an average accuracy of 75% was achieved.

3.1.2 Model 2

As an initial attempt of increasing the overall accuracy of the network, a convolutional layer with a filter to 128 was added.

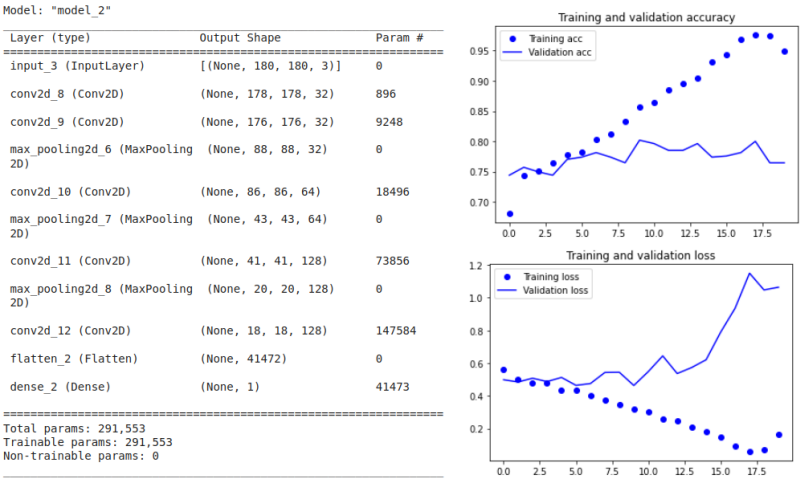


Figure 2: Task 2 - Model 2 Summary and Accuracy Loss curves

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 2	10	0.802	0.734	0.463	0.589

Table 2: Metrics Model 0

Validation loss keeps increasing after epoch 10, at this point overfitting is starting. An average accuracy of 75% shows that no evident improvement was obtained compared to the previous model.

3.1.3 Model 3

Not willing to give up on our previous attempt, an additional convolutional layer was added and the filter size was doubled in order to obtain the model of the third attempt.

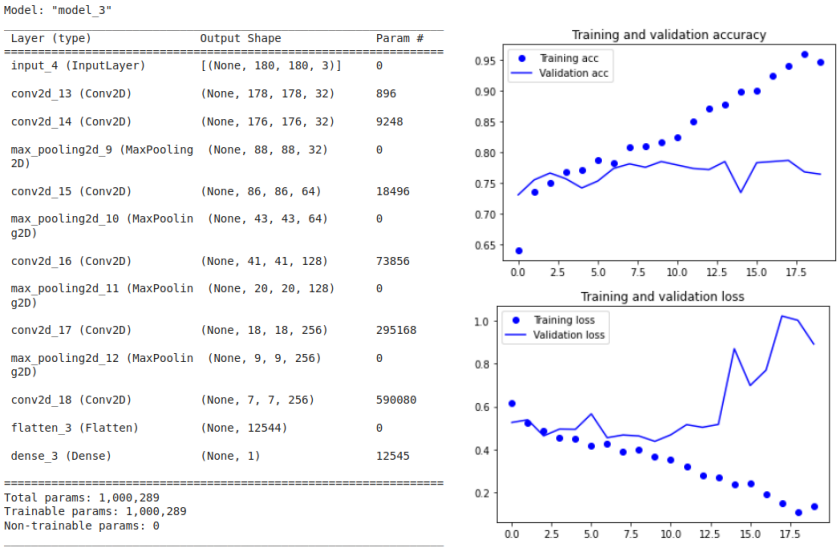


Figure 3: Task 2 - Model 3 Summary and Accuracy Loss curves

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 3	10	0.785	0.766	0.437	0.476

Table 3: Metrics Model 0

However, also in this case no tangible improvements were observed.

3.1.4 Model 4

In this case, a fully connected NN-layer was added in the output network.

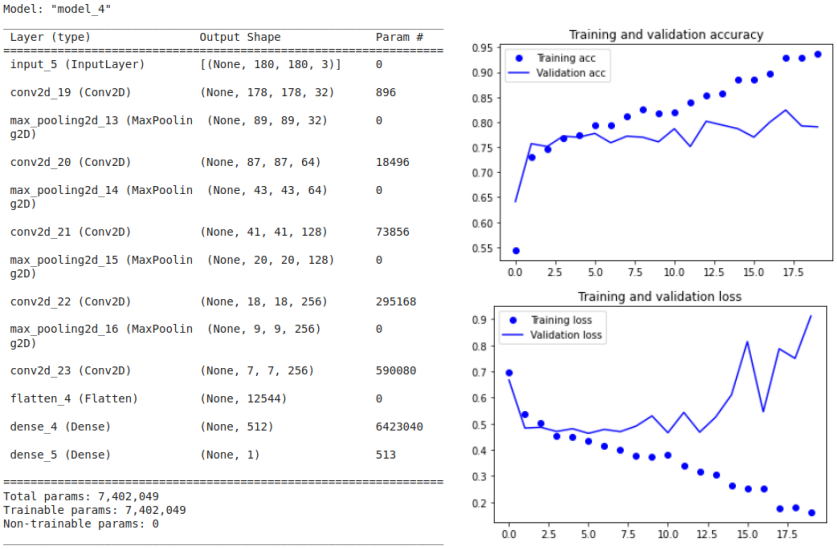


Figure 4: Task 2 - Model 4 Summary and Accuracy Loss curves

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 4	6	0.778	0.786	0.463	0.451

Table 4: Metrics Model 4

In this case, accuracy does not increase above 76% and overfitting clearly starts affecting the network after epoch 4. This is easily understandable given the high number of network parameters $(7,402,049)$.

3.1.5 Model 5

Data augmentation is another way we can reduce overfitting on models. This is the approach adopted to obtain the next incremental model. Data augmentation was obtained by means of Keras preprocessing layers (`RandomFlip`, `RandomRotation`, `RandomTranslation`, `RandomZoom`). The augmentation code is the following:

```
[ ]  data_augmentation = keras.Sequential(
       [
         layers.RandomFlip("horizontal"),
         layers.RandomRotation(0.2),
         layers.RandomTranslation(height_factor=0.1, width_factor=0.1, fill_mode="constant", fill_value=255),
         layers.RandomZoom(0.1, fill_mode="nearest")
       ]
     )
```

Figure 5: Data Augmentation code

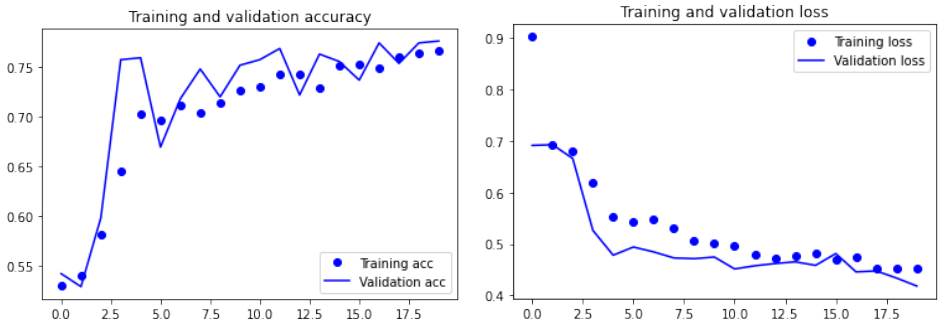


Figure 6: Task 2 - Model 5 Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 5	20	0.783	0.783	0.418	0.412

Table 5: Metrics Model 0

As a result, the most notable improvement lies in the fact that overfitting affect the network in later training epoch if compared to previous models.

3.1.6 Model 6

A single model can be used to simulate having a large number of different network architectures by randomly dropping out nodes during training. This is called dropout and offers a very computationally cheap and remarkably effective regularization method to reduce overfitting and improve generalization error in deep neural networks of all kinds. Keras `layers.Dropout` was used to this end.

Dropout rate 0.5

With a dropout rate of 0.5 overfitting does not affect the network anymore and the overall average accuracy after 40 epochs approaches 81%.

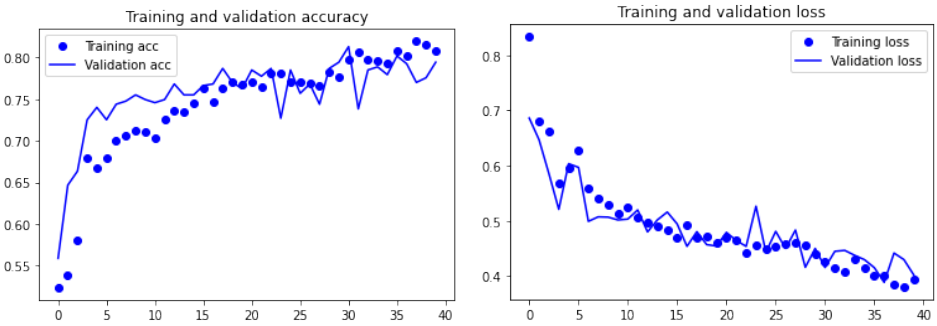


Figure 7: Task 2 - Model 6 (Dropout rate 0.5) Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 6a	37	0.793	0.809	0.389	0.369

Table 6: Metrics Model 6a

Dropout rate 0.2 With the dropout rate set to 0.2, the average accuracy increases even more and approaches 86% after 40 epochs of training.

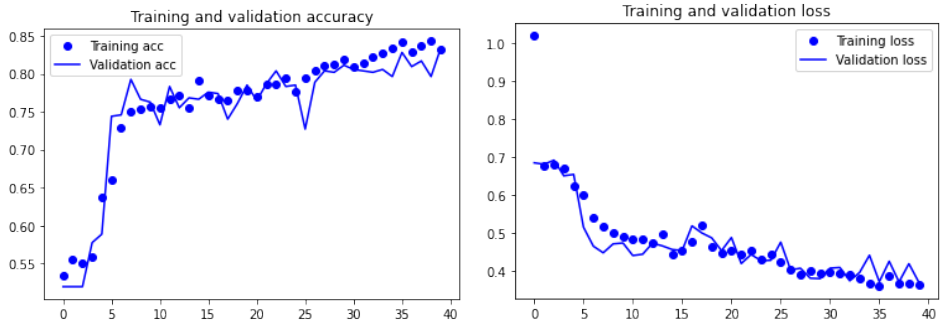


Figure 8: Task 2 - Model 6 (Dropout rate 0.2) Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 6b	40	0.834	0.814	0.372	0.388

Table 7: Metrics Model 6b

3.1.7 Model 7

Another way to prevent overfitting is to use Regularization. A combination of L1 and L2 Regularization was used. They are the most common types of regularization. The hyperparameters chosen are: for L1 $1^{-}5$ and $1^{-}5$ for L2.

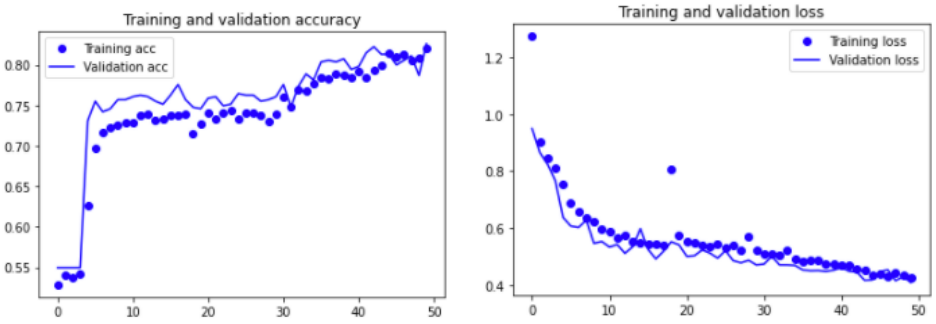


Figure 9: Task 2 - Model 7 (L2 - L1 regularization) Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 7	50	0.826	0.797	0.401	0.482

Table 8: Metrics Model 7

We don't observe any significant improvement by applying Regularization technique.

3.1.8 Final Observations

The increase in the complexity of the architecture has increased overall testing accuracy by about 10% if we take the model 1 base as a reference and the best model Model6b.
The confusion matrix of the worst and best models obtained are reported.

Model	Validation Loss	Testing Accuracy
model1	0.458	0.720
model2	0.463	0.734
model3	0.437	0.767
model4	0.776	0.787
model5	0.451	0.783
model6a	0.388	0.809
model6b	0.371	0.814
model7	0.407	0.797

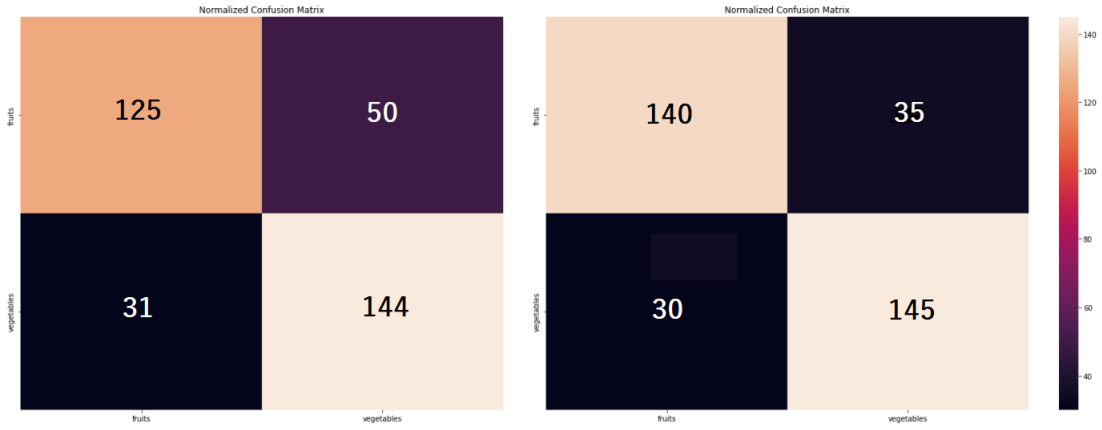


Figure 10: Confusion Matrix Base Model and Model6b

3.2 Multi-Label Classification

In this section we try to get a CNN network capable of classifying objects belonging to more than 2 classes. Specifically we have 33 classes that correspond to sub-classes of the two classes used for binary classification. The approach used to obtain the best network is modular and heuristic. We will start to evaluate the performance of 4 different CNN networks each with a different architecture. The best two, evaluated considering a mix between val-loss, testing-accuracy and f1-score, will be modified by adding dense layers. Augmentation will be introduced, two drop out versions and finally L1 and L2 regularization.

The maximum batch number will initially be 30. When the augmentation is introduced the maximum batch number will be increased to 60, because we expect a slower network overfitting due to the use of augmentation, dropout and regularization.

3.2.1 Design convolutional network

The architecture of the starting model, **Model 0**, is shown in the figure. From that model four additional architectures will be built.

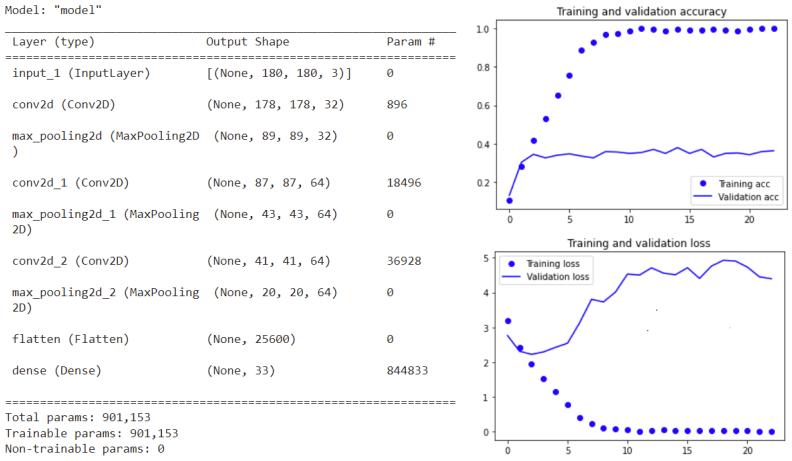


Figure 11: Task 2 - Model 0 Summary and Accuracy Loss curves

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 0	3	0.3119	0.295	2.2253	2.409

Table 9: Metrics Model 0

Model 1 is based on model 0 with an additional convolutional layer and max-pooling layer.

Layer (type)	Output Shape	Param #
input_6 (InputLayer)	[(None, 180, 180, 3)]	0
conv2d_22 (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d_22 (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_23 (Conv2D)	(None, 87, 87, 32)	9248
max_pooling2d_23 (MaxPooling2D)	(None, 43, 43, 32)	0
conv2d_24 (Conv2D)	(None, 41, 41, 64)	18496
max_pooling2d_24 (MaxPooling2D)	(None, 20, 20, 64)	0
conv2d_25 (Conv2D)	(None, 18, 18, 64)	36928
max_pooling2d_25 (MaxPooling2D)	(None, 9, 9, 64)	0
flatten_5 (Flatten)	(None, 5184)	0
dense_9 (Dense)	(None, 33)	171105

Total params: 236,673
Trainable params: 236,673

Figure 12: Task 2 - Model 1 Summary

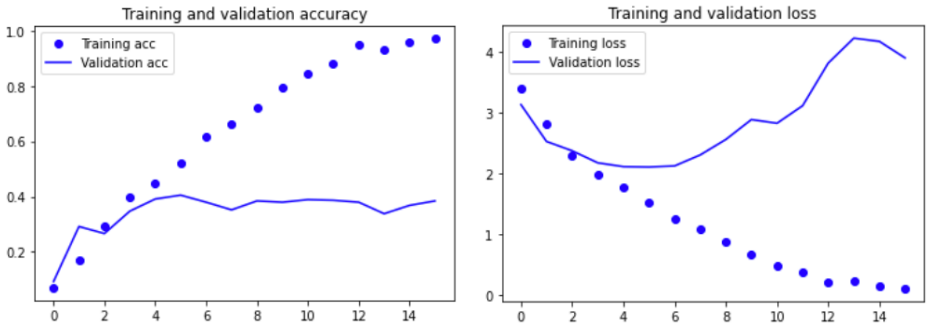


Figure 13: Task 2 - Model 1 Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 1	6	0.405	0.346	2.104	2.417

Table 10: Metrics Model 1

The network **Model 2** is based on model architecture 1 but changing the size of the filters of the last convolutional layer from 64 to 128.

```
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         [(None, 180, 180, 3)]     0

conv2d_25 (Conv2D)           (None, 178, 178, 32)      896

max_pooling2d_22 (MaxPoolin  (None, 89, 89, 32)        0
g2D)

conv2d_26 (Conv2D)           (None, 87, 87, 32)        9248

max_pooling2d_23 (MaxPoolin  (None, 43, 43, 32)        0
g2D)

conv2d_27 (Conv2D)           (None, 41, 41, 64)        18496

max_pooling2d_24 (MaxPoolin  (None, 20, 20, 64)        0
g2D)

conv2d_28 (Conv2D)           (None, 18, 18, 64)        36928

max_pooling2d_25 (MaxPoolin  (None, 9, 9, 64)          0
g2D)

conv2d_29 (Conv2D)           (None, 7, 7, 128)         73856

max_pooling2d_26 (MaxPoolin  (None, 3, 3, 128)         0
g2D)

flatten_6 (Flatten)          (None, 1152)              0

dense_6 (Dense)              (None, 33)                38049

=================================================================
Total params: 177,473
Trainable params: 177,473
```

Figure 14: Task 2 - Model 2 Summary

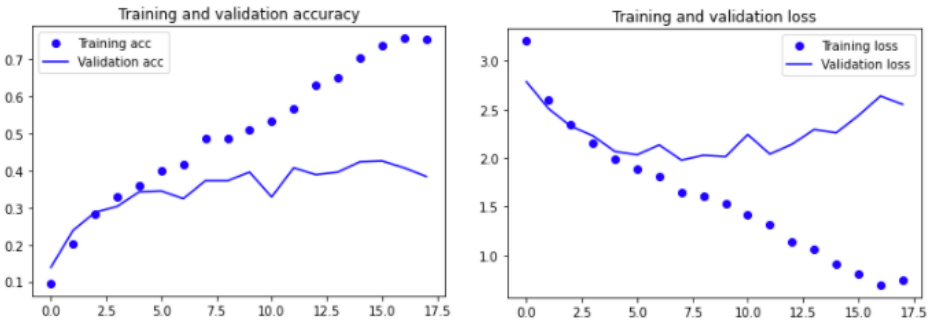


Figure 15: Task 2 - Model 2 Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 2	8	0.373	0.376	1.978	1.961

Table 11: Metrics Model 2

A 64 convolutional layer and the relative max_pooling have been removed. The **Model 3** model is shown in the figure.

```
Layer (type)                 Output Shape              Param #
=================================================================
input_11 (InputLayer)        [(None, 180, 180, 3)]     0

conv2d_42 (Conv2D)           (None, 178, 178, 32)      896

max_pooling2d_39 (MaxPoolin  (None, 89, 89, 32)        0
g2D)

conv2d_43 (Conv2D)           (None, 87, 87, 32)        9248

max_pooling2d_40 (MaxPoolin  (None, 43, 43, 32)        0
g2D)

conv2d_44 (Conv2D)           (None, 41, 41, 64)        18496

max_pooling2d_41 (MaxPoolin  (None, 20, 20, 64)        0
g2D)

conv2d_45 (Conv2D)           (None, 18, 18, 128)       73856

max_pooling2d_42 (MaxPoolin  (None, 9, 9, 128)         0
g2D)

flatten_10 (Flatten)         (None, 10368)             0

dense_13 (Dense)             (None, 33)                342177

=================================================================
Total params: 444,673
Trainable params: 444,673
```

Figure 16: Task 2 - Model 3 Summary

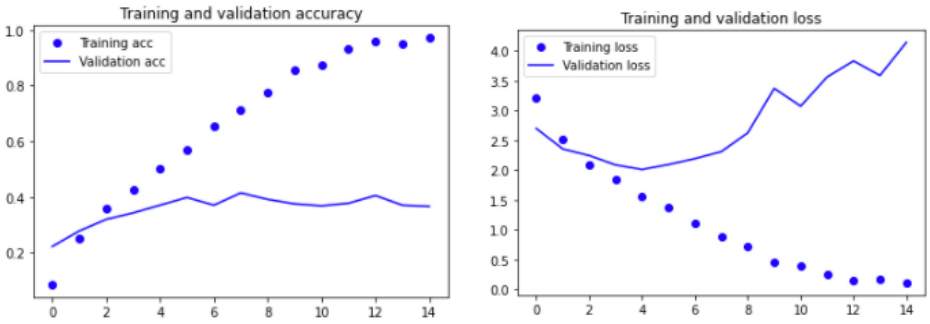


Figure 17: Task 2 - Model 3 Accuracy and Loss

Model	**Epoch Stop**	**Validation Accuracy**	**Testing Accuracy**	**Validation Loss**	**Testing Loss**
Model 3	5	0.3704	0.323	2.011	2.202

Table 12: Metrics Model 3

Model 4 was built starting from model 3. A convolutional layer with 256 filters and max-pooling was added.

```
Layer (type)                 Output Shape              Param #
=================================================================
input_13 (InputLayer)        [(None, 180, 180, 3)]     0

conv2d_51 (Conv2D)           (None, 178, 178, 32)      896

max_pooling2d_48 (MaxPoolin  (None, 89, 89, 32)        0
g2D)

conv2d_52 (Conv2D)           (None, 87, 87, 32)        9248

max_pooling2d_49 (MaxPoolin  (None, 43, 43, 32)        0
g2D)

conv2d_53 (Conv2D)           (None, 41, 41, 64)        18496

max_pooling2d_50 (MaxPoolin  (None, 20, 20, 64)        0
g2D)

conv2d_54 (Conv2D)           (None, 18, 18, 128)       73856

max_pooling2d_51 (MaxPoolin  (None, 9, 9, 128)         0
g2D)

conv2d_55 (Conv2D)           (None, 7, 7, 256)         295168

max_pooling2d_52 (MaxPoolin  (None, 3, 3, 256)         0
g2D)

flatten_12 (Flatten)         (None, 2304)              0

dense_15 (Dense)             (None, 33)                76065

=================================================================
Total params: 473,729
Trainable params: 473,729
```

Figure 18: Task 2 - Model 4 Summary

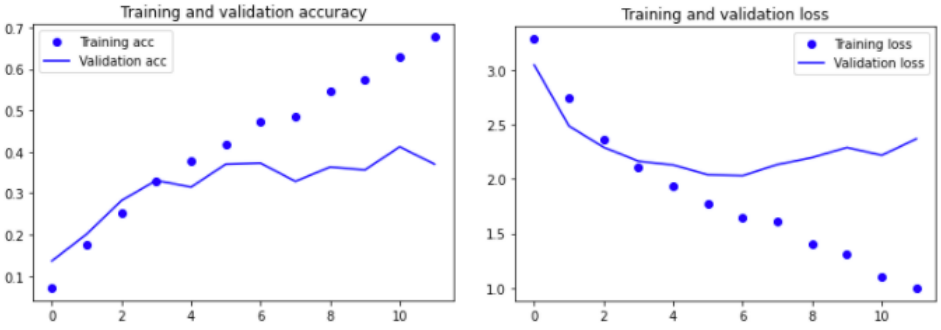


Figure 19: Task 2 - Model 4 Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 4	7	0.373	0.315	2.013	2.201

Table 13: Metrics Model 4

The table shows the results obtained so far. The architectures we take into account are **Model 1** e **Model 2**.

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 0	3	0.3119	0.294	2.423	2.311
Model 1	6	0.4051	0.346	2.104	2.417
Model 2	8	0.373	0.376	1.978	1.961
Model 3	5	0.3704	0.323	2.011	2.202
Model 4	7	0.373	0.315	2.013	2.201

Table 14: Metrics models

3.2.2 Implementing network classifier

At each of the two chosen networks, we get two new networks. The first by adding a 128-neuron hidden layer to the full-connected network and the second by adding a 512-neuron layer.

Model 1_1 is based on Model 1 but by adding to the full-connected network a hidden layer of 128 neurons.

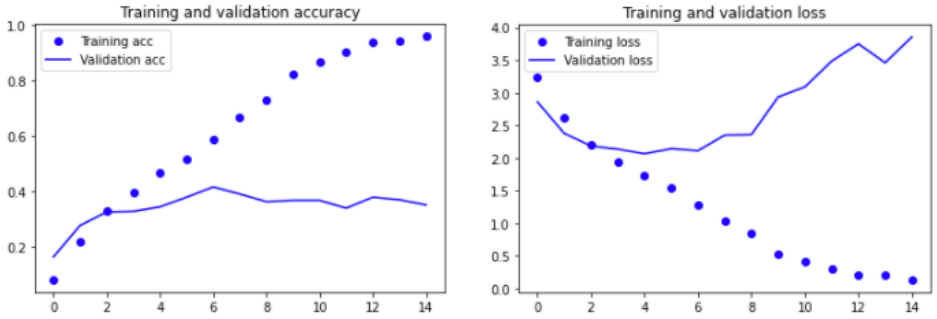


Figure 20: Task 2 - Model 1_1 Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 1_1	5	0.345	0.351	2.066	2.032

Table 15: Metrics Model 1_1

Model 1_2 is based on Model 1 but by adding to the full-connected network a hidden layer of 512 neurons.

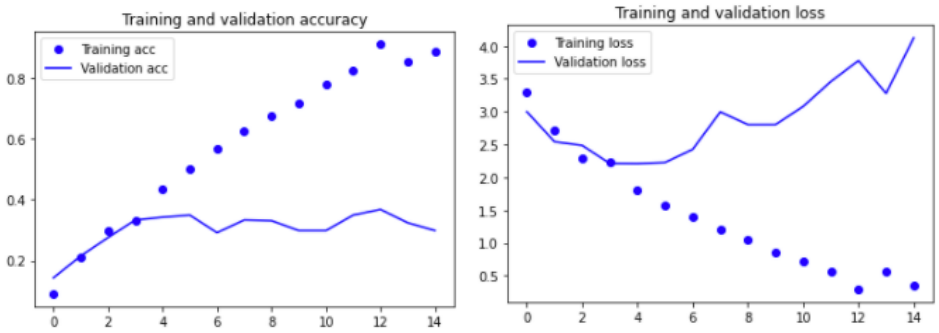


Figure 21: Task 2 - Model 1_2 Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 1_2	5	0.342	0.315	2.208	2.307

Table 16: Metrics Model 1_2

Model 2_1 We add to the full-connected network of the Model 2 a hidden layer of 128 neurons.

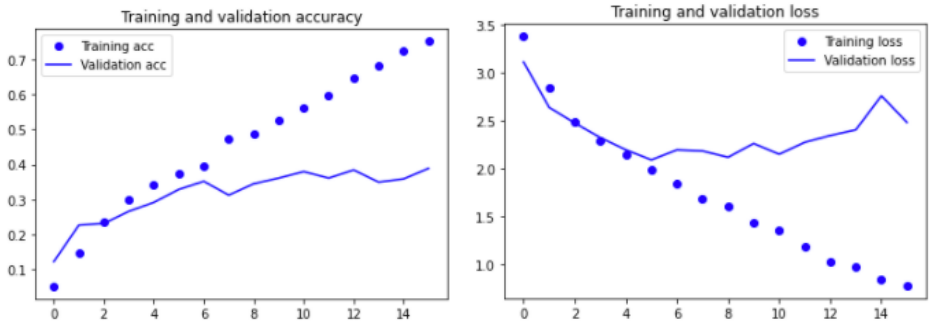


Figure 22: Task 2 - Model 2_1 Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 2_1	6	0.329	0.352	2.091	2.278

Table 17: Metrics Model 2_1

Model 2_2 We add to the full-connected network of the Model 2 a hidden layer of 512 neurons.

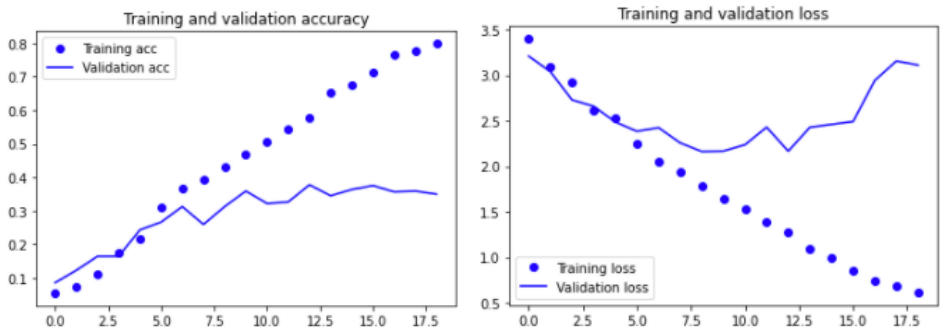


Figure 23: Task 2 - Model 2_2 Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 2_2	9	0.313	0.328	2.162	2.197

Table 18: Metrics Model 2_2

The introduction of hidden layers in the full-connected network has slightly improved the testing_accuracy and val_loss of **Model 1**.

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 1	6	0.4051	0.346	2.104	2.417
Model 2	8	0.373	0.376	1.978	1.961
Model 1_1	5	0.345	0.351	2.066	2.032
Model 1_2	5	0.342	0.315	2.208	2.307
Model 2_1	6	0.329	0.352	2.091	2.278
Model 2_2	9	0.313	0.328	2.162	2.197

Table 19: Metrics Models

3.2.3 Augmentation technique

In this section we apply the augmentation technique to increase the predictive capacity of a network and reduce overfitting. The models chosen to apply this technique are model 1, model 1_2, model 2 and model 2_1. Because we expect slower overfitting, the network training process has been increased as the number of max batches to 60. The augmentation code used is the usual one used for task 2 - binary class from scratch.

Model 1a

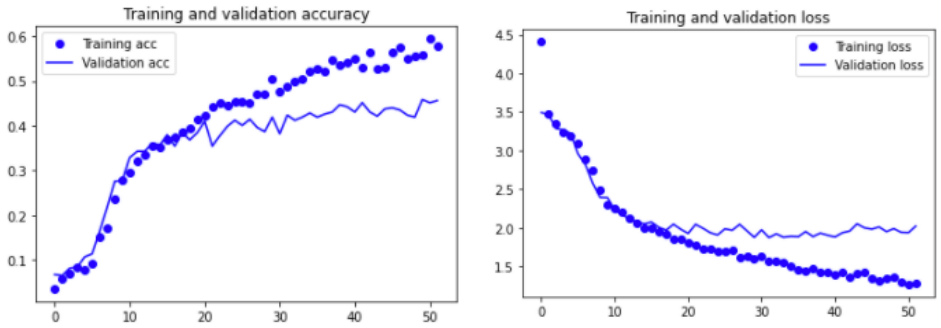


Figure 24: Task 2 - Model 1a Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 1a	32	0.424	0.400	1.870	1.980

Table 20: Metrics Model 1a

Model 1_2a

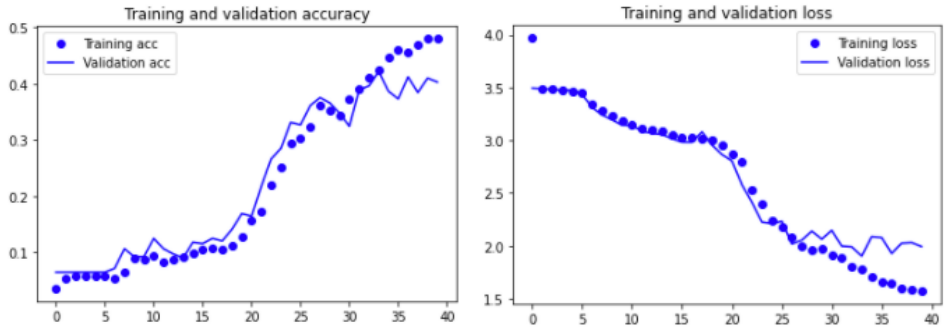


Figure 25: Task 2 - Model 1_2a Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 1_2a	34	0.421	0.406	1.903	1.966

Table 21: Metrics 1_2a

Model 2a

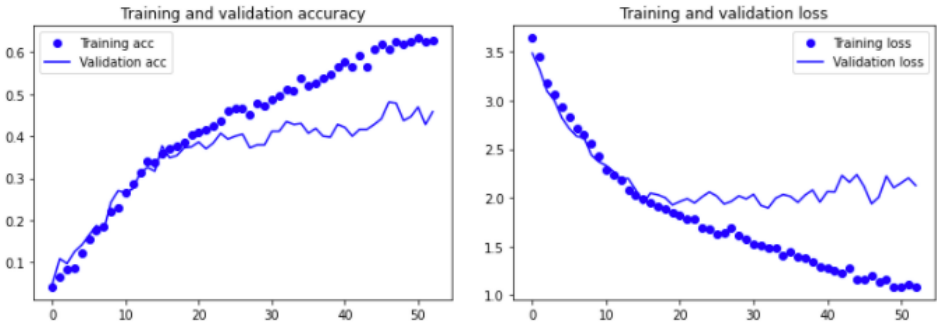


Figure 26: Task 2 - Model 2a Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 2a	33	0.435	0.436	1.895	1.888

Table 22: Metrics Model 2a

Model 2_1a

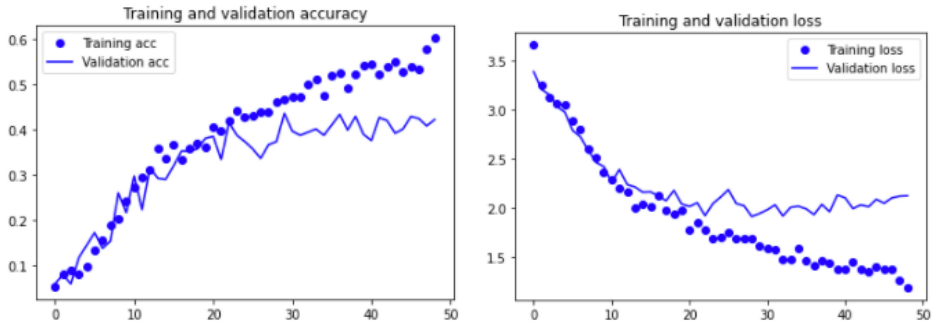


Figure 27: Task 2 - Model 2_1a Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 2_1a	29	0.373	0.381	1.911	1.853

Table 23: Metrics Model 2_1a

By the introduction of the augmentation technique, we can see how much the average epoch stop has increased. Additionally, the metrics increase for all models.

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 1a	32	0.424	0.400	1.870	1.980
Model 1_2a	34	0.421	0.406	1.903	1.966
Model 2a	33	0.435	0.436	1.895	1.888
Model 2_1a	29	0.373	0.381	1.911	1.853

Table 24: Metrics augmented models

3.2.4 Dropout technique

Taking model 2a and model 1_2a as a reference, let's try to further reduce overfitting using the droupout technique. Parameters 0.2 and 0.5 have been used.

Model 1_2aD2

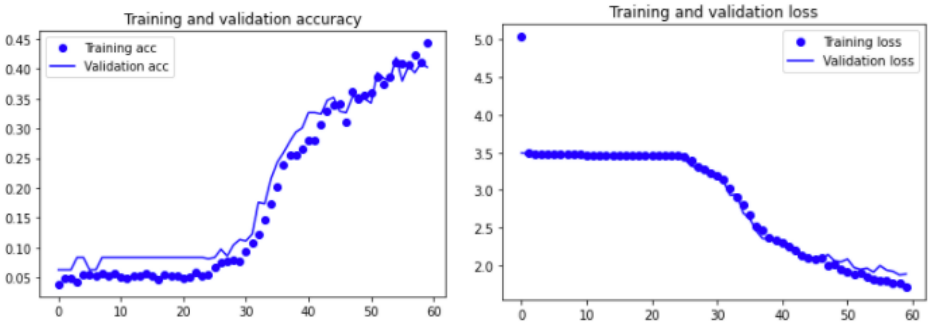


Figure 28: Task 2 - Model 1_2aD2 Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 1_2aD2	59	0.410	0.352	1.874	2.081

Table 25: Metrics Model 1_2aD2

Model 1_2aD5

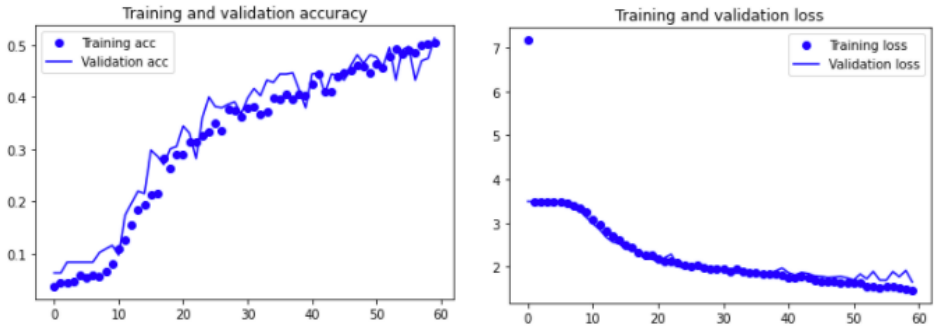


Figure 29: Task 2 - Model 1_2aD5 Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 1_2aD5	60	0.5139	0.473	1.649	1.747

Table 26: Metrics 1_2aD5

Model 2aD5

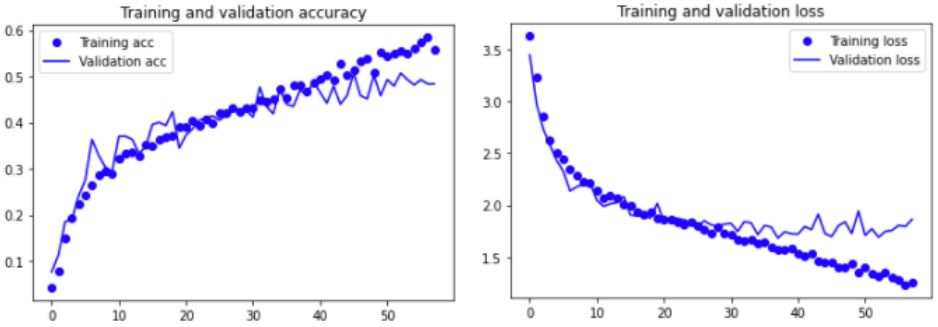


Figure 30: Task 2 - Model 2aD5 Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 2aD5	38	0.470	0.479	1.682	1.701

Table 27: Metrics 2aD5

Model 2aD2

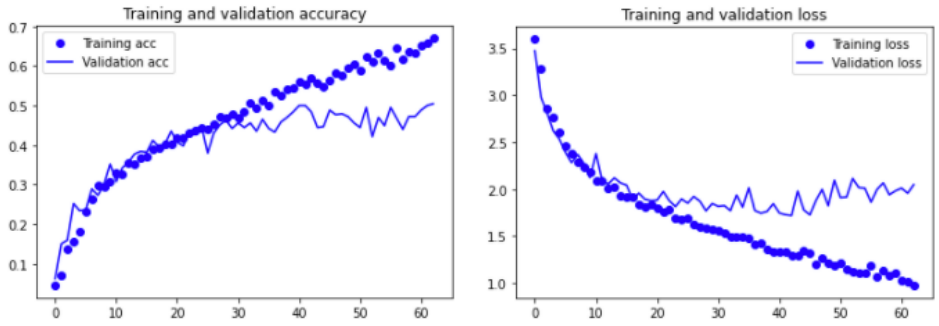


Figure 31: Task 2 - Model 2aD2 Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 2aD2	43	0.484	0.497	1.718	1.680

Table 28: Metrics 2aD2

The use of the dropout has increased the average value of the epoch stop, therefore increased the average training time of the models. However, it gave beneficial results to the 2aD2,2aD5 and 1_2aD5 models which increased the testing accuracy and reduced the validation loss.

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 1_2aD2	59	0.410	0.352	1.874	2.081
Model 1_2aD5	60	0.5139	0.473	1.649	1.789
Model 2aD2	43	0.484	0.497	1.718	1.680
Model 2aD5	38	0.470	0.479	1.682	1.701

Table 29: Metrics Models using Dropout

3.2.5 Regularization

We implement L1 and L2 regularization to further reduce overfitting.

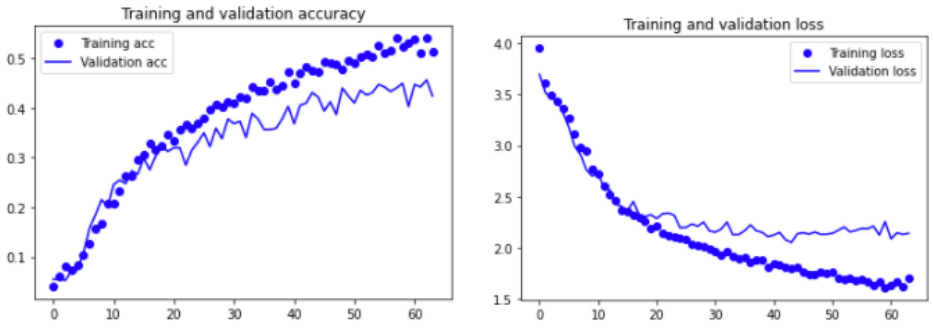


Figure 32: Task 2 - Model 1_2aD5L1L2 Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 1_2aD5L1L2	44	0.430	0.430	2.053	2.052

Table 30: Metrics Model 1_2aD5L1L2

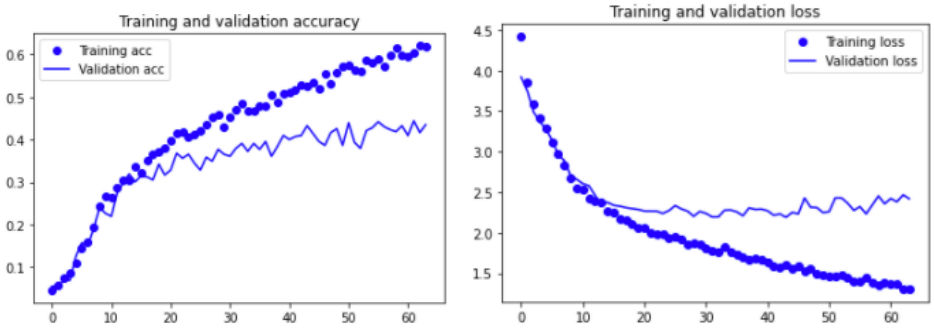


Figure 33: Task 2 - Model 2aD2L1L2 Accuracy and Loss

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 2aD2L1L2	44	0.4329	0.406	2.190	2.282

Table 31: Metrics Model

The use of L1 and L2 regularization did not lead to significant improvements. Even the ability to predict has deteriorated.

3.2.6 Conclusion and general summary

Below is a table showing the starting model and the 3 best models obtained.

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Model 0	3	0.3119	0.294	2.423	2.409
Model 1_2aD5	60	0.5139	0.473	1.649	1.789
Model 2aD2	43	0.484	0.497	1.718	1.680

Table 32: Metrics best models

Below are two confusion matrixes, the first relating to model 0 and the second to model 2aD5. The first confusion matrix shows that the network cannot detect any of the 32 classes. The second shows that the model is able to identify some classes but is not yet precise in identifying features useful for distinguishing classes of similar objects. For example potato and sweetpotato.

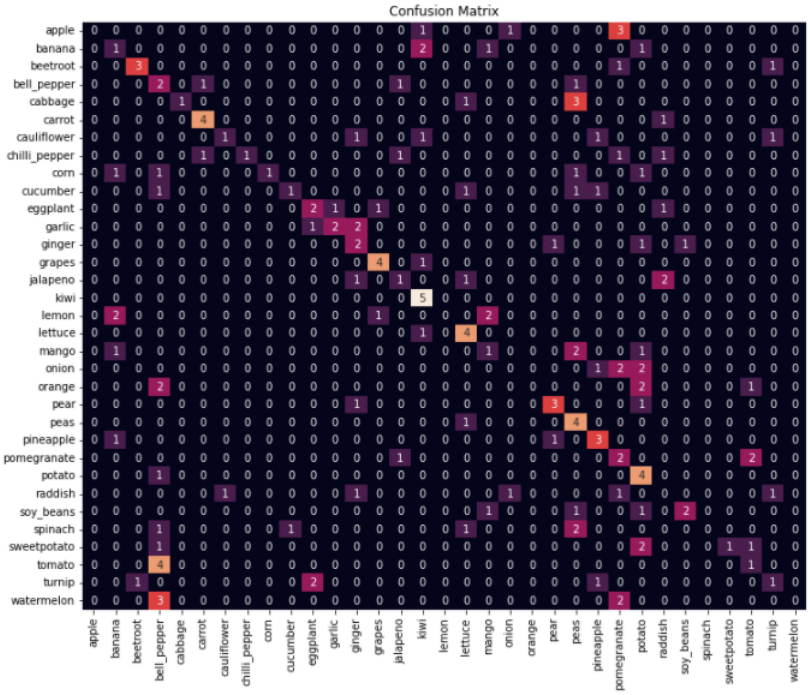


Figure 34: Task 2 - Model 0 Confusion Matrix

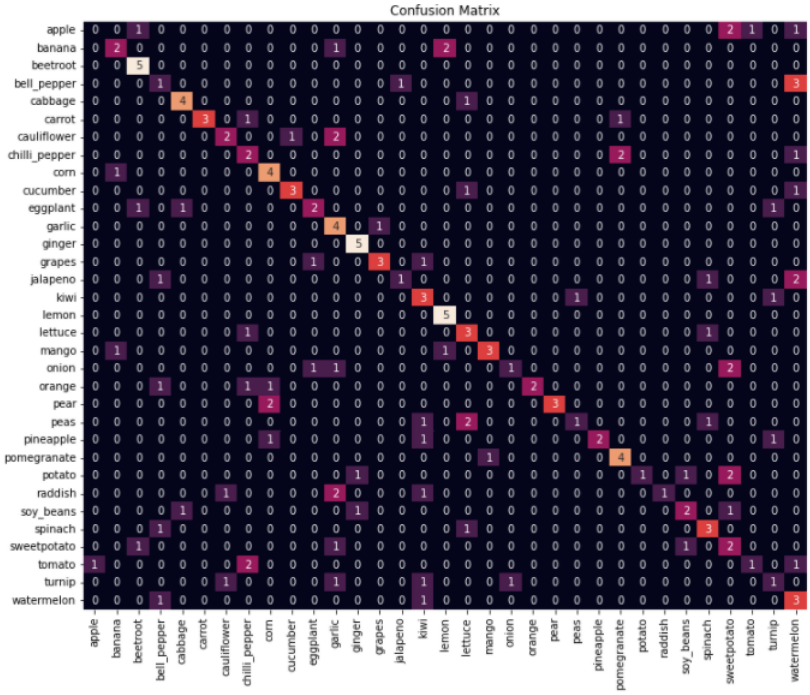


Figure 35: Task 2 - Model 2aD2 Confusion Matrix

4 Task 3 - Pre-trained

This section describe the use of 3 different pre-trained networks to solve both the binary and the multi-label classification problem. In particular we try to use VGG16, InceptionV3 and ResNet50.

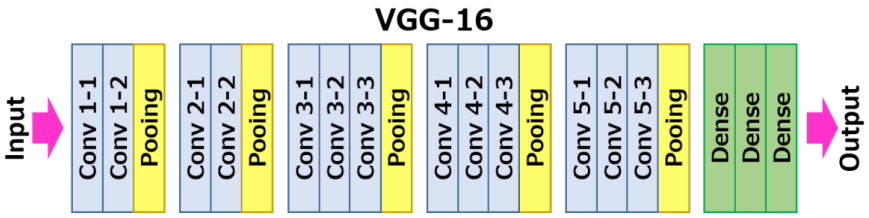


Figure 36: VGG16 Architecture

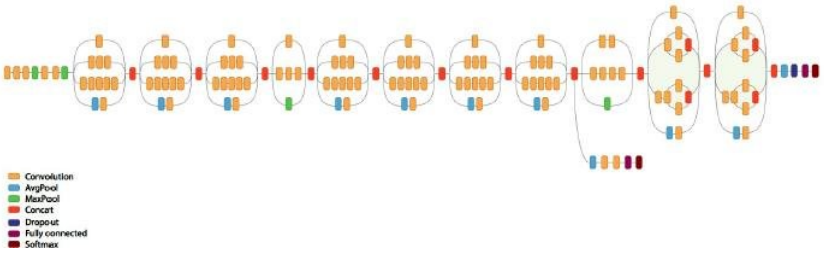


Figure 37: InceptionV3 Architecture

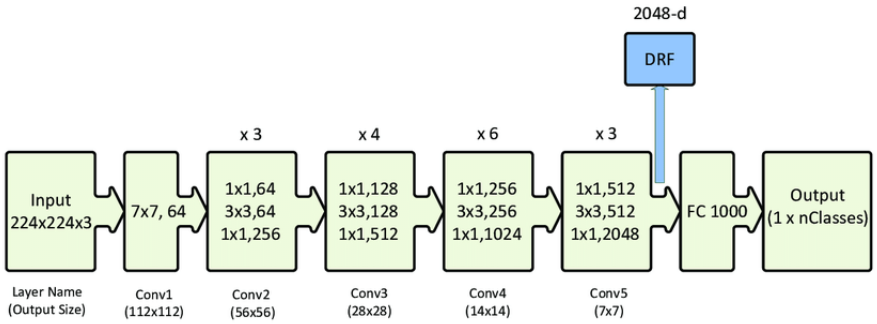


Figure 38: ResNet50 Architecture

4.1 Binary Classification

For each pre-trained network we report the result from 5 different models both using the Feature Extraction approach and the Fine Tuning approach. All of these models are trained using the optimizer Rmsprop with the default "learning rate" (0.001), the loss function "binary_crossentropy" , 50 epochs, an Early Stopping Condition with "patience" = 10, "batch size" = 32 and a resize of the input images of 224x224. The dataset was divided as follows: the Test Set was obtained using the "validation_split" attribute (with value 0.1) of the "image_dataset_from_directory" function, and the Validation Set using the "validation_split" attribute (with value 0.1) of the "fit" function. In all of the models we start to use the Feature Extraction approach and then to try to increase the performance we use the Fine Tuning approach.

4.1.1 Data Augmentation

In order to try to improve the Overfitting problem during the training of the different models at a certain point we introduce the concept of Augmentation for generate more training data from existing training samples, by "augmenting" the samples via a number of random transformations that yield believable-looking images. The code of the layer to insert the augmentation inside the models to train is shown below.

```
[ ]  data_augmentation = keras.Sequential(
       [
         layers.RandomFlip("horizontal"),
         layers.RandomRotation(0.2),
         layers.RandomTranslation(height_factor=0.1, width_factor=0.1, fill_mode="constant", fill_value=255),
         layers.RandomZoom(0.1, fill_mode="nearest")
       ]
     )
```

Figure 39: Data Augmentation code

4.1.2 VGG16

Base Model In this first simple model we combined the pre-trained network with a fully connected net with a Flatten layer and a Dense layer of 256.

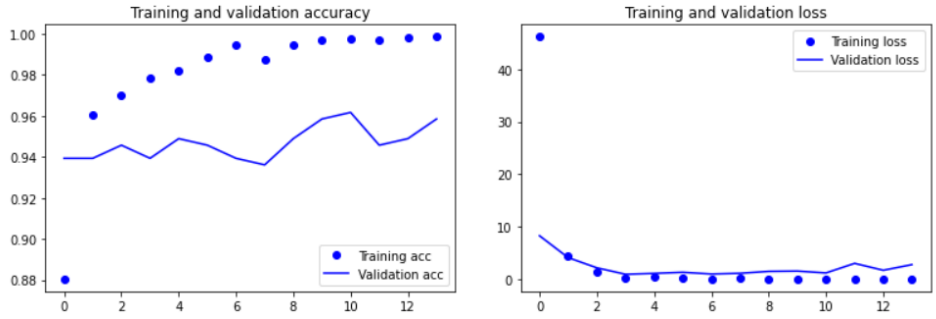


Figure 40: Base Model Feature Extraction

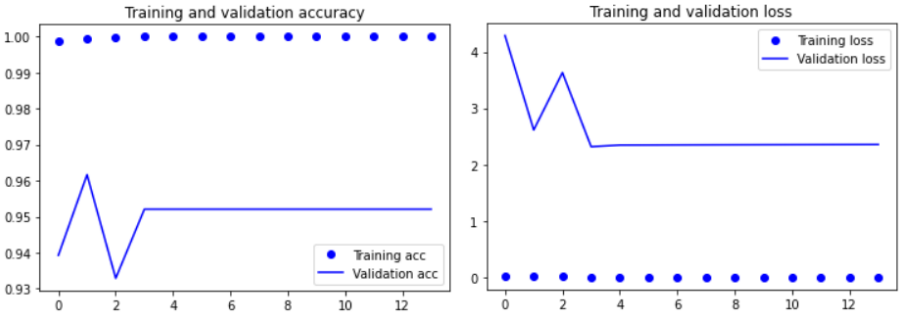


Figure 41: Base Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Base-Fe	14	0.939	0.925	0.942	7.511
Base-Ft	14	0.952	0.930	2.318	5.990

Table 33: Metrics Base Models

As we can see from the graphs and the table the network perform very well with an accuracy of 92.5% and 93%, but it goes in Overfitting very soon and it have a Testing Loss very high.

Augmented Model To try to combat the Overfitting problem we add to the previous model a layer that augmented the data using the structure that we describe on Section 4.1.1.

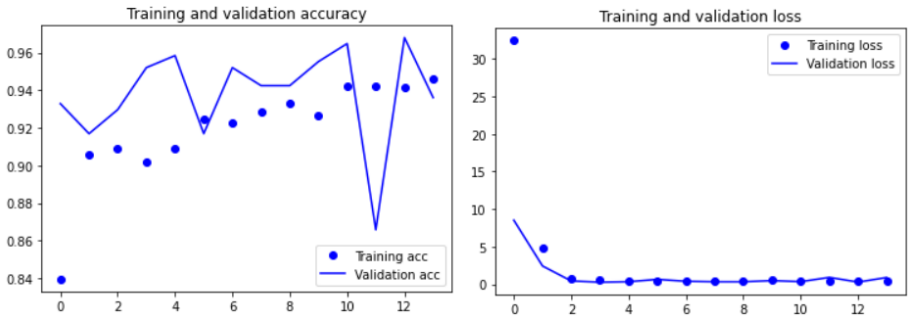


Figure 42: Augmented Model Feature Extraction

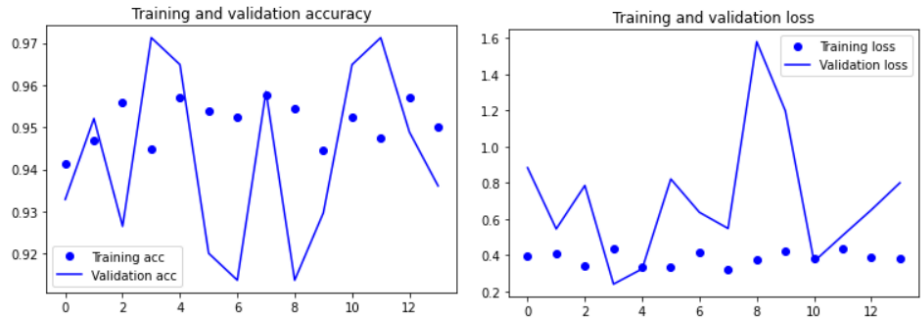


Figure 43: Augmented Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Augmen-Fe	14	0.952	0.922	0.267	0.804
Augmen-Ft	14	0.971	0.925	0.239	1.430

Table 34: Metrics Augmented Models

As we can see from the graphs and the table the network has very similar accuracy than the previous models, but with a value of Testing Loss much better than before. The problem of Overfitting remain unchanged.

Dropout Models Another way to reduce the Overfitting problem is to use the Dropout, so we add to the previous model a layer to insert the Dropout. We try to use two different values(0.2 and 0.5).

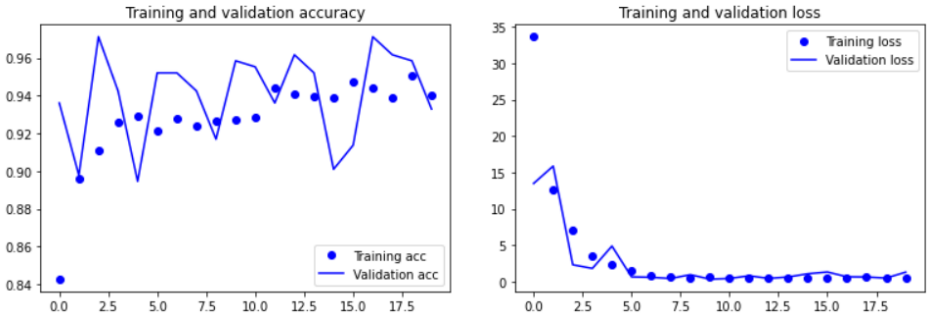


Figure 44: Dropout 0.2 Model Feature Extraction

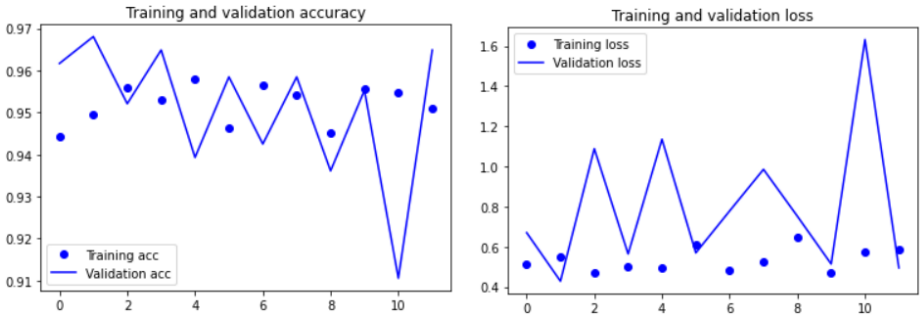


Figure 45: Dropout 0.2 Model Fine Tuning

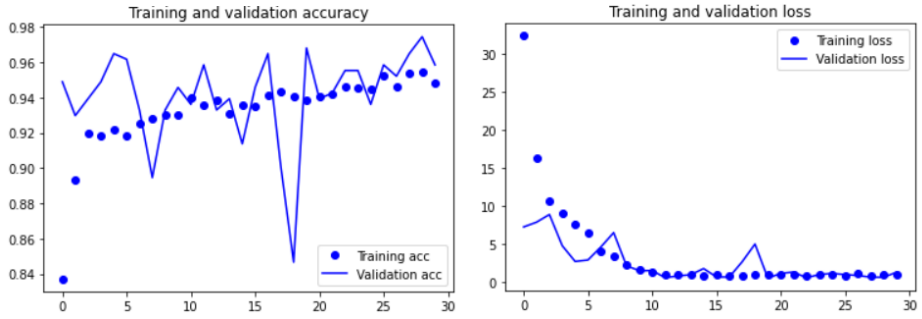


Figure 46: Dropout 0.5 Model Feature Extraction

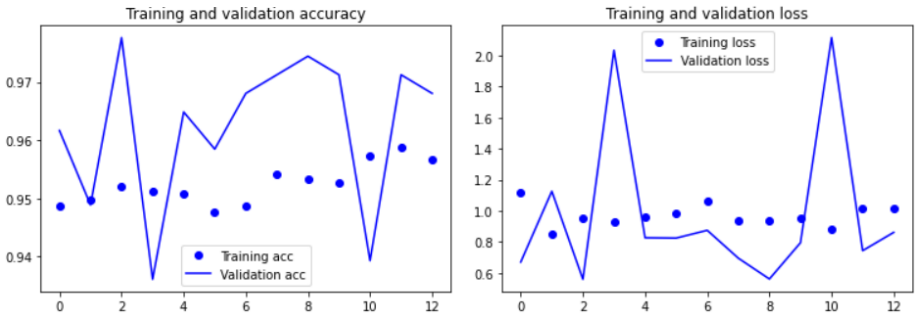


Figure 47: Dropout 0.5 Model Fine Tuning

Model	**Epoch Stop**	**Validation Accuracy**	**Testing Accuracy**	**Validation Loss**	**Testing Loss**
Drop.0.2-Fe	20	0.959	0.922	0.315	1.484
Drop.0.2-Ft	12	0.968	0.954	0.429	1.40
Drop.0.5-Fe	30	0.968	0.942	0.477	2.611
Drop.0.5-Ft	13	0.978	0.948	0.559	2.182

Table 35: Metrics Dropout Models

As we can see from the graphs and the table now the network start to increase the Testing accuracy respect to the previous models. The Testing Loss, in particular for the Drop.0.2 model, is unchanged. The Overfitting begins to reduce in particular for Feature Extraction's models.

Regularized Model The last technique that we do to try to reduce the Overfitting problem is add the Regularization to the previous model with Dropout 0.2. So on the hidden layer of the fully connected net we forcing the weights to only take small values. We use the Regularization L1_L2 with corresponding values L1 = 1e-5 and L2 = 1e-4.

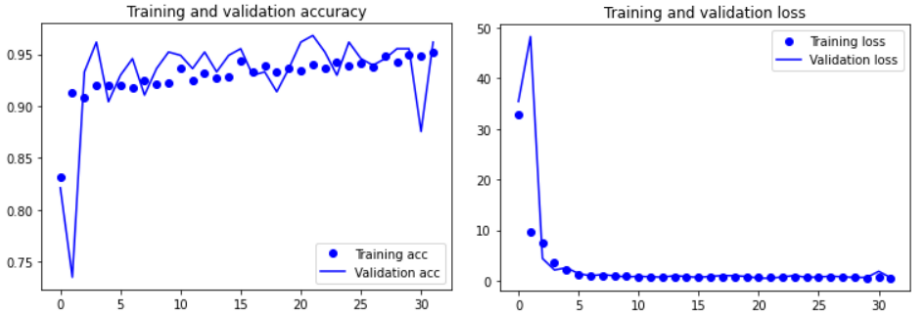


Figure 48: Regularized Model Feature Extraction

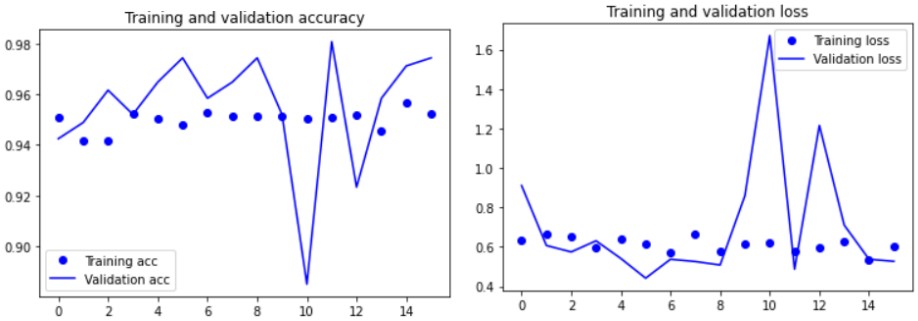


Figure 49: Regularized Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Regul-Fe	32	0.968	0.948	0.496	0.905
Regul-Ft	16	0.974	0.945	0.441	1.139

Table 36: Metrics Regularized Models

As we can see from the graphs and the table now has very similar accuracy than the previous models, but with a value of Testing Loss low than before. The Overfitting has another little improvement both for Feature Extraction's and Fine Tuning's models.

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Base-Fe	14	0.939	0.925	0.942	7.511
Base-Ft	14	0.952	0.93	2.318	5.99
Augmen-Fe	14	0.952	0.922	0.267	0.804
Augmen-Ft	14	0.971	0.925	0.239	1.430
Drop.0.2-Fe	20	0.959	0.922	0.315	1.484
Drop.0.2-Ft	12	0.968	0.954	0.429	1.40
Drop.0.5-Fe	30	0.968	0.942	0.477	2.611
Drop.0.5-Ft	13	0.978	0.948	0.559	2.182
Regul-Fe	32	0.968	0.948	0.496	0.905
Regul-Ft	16	0.974	0.945	0.441	1.139

Table 37: Recap Metrics VGG16's Models

This table report 5 different metrics that we collect after the training of all the models that we describe above. Being the final objective that of finding the best compromise between testing accuracy increase and testing loss decrease, we can point out that the optimal solution is in this case given by the model obtained using data augmentation and a drop ratio equal to 0.2 accordingly to the *Fine Tuning* approach. We report also the two Confusion Matrix of the first model and the best model for this pretrained network in order to see the improvement between the two models.

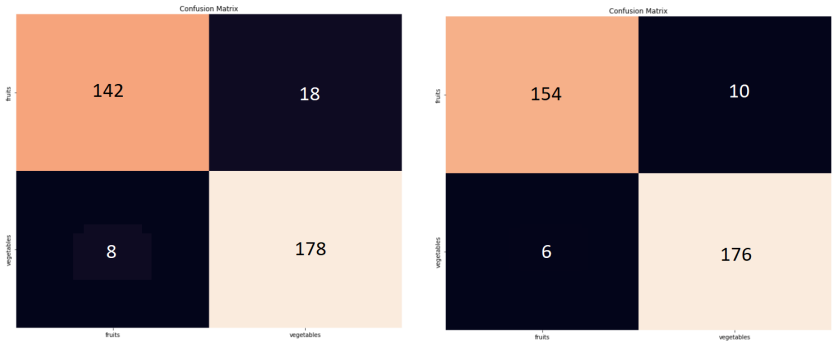


Figure 50: Confusion Matrix Base Model-Fe and Drop.0.2-Ft

4.1.3 InceptionV3

Base Model In this first simple model we combined the pre-trained network with a fully connected net with a Flatten layer and a Dense layer of 256.

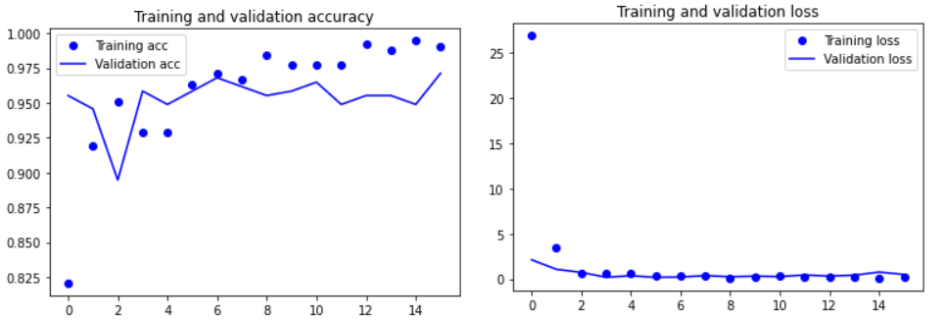


Figure 51: Base Model Feature Extraction

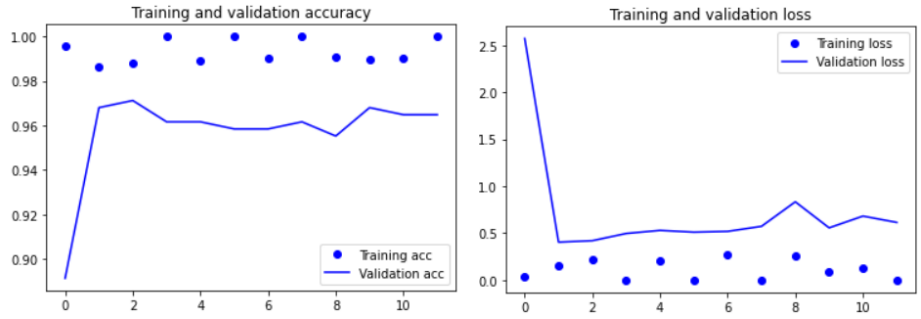


Figure 52: Base Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Base-Fe	16	0.959	0.945	0.176	0.639
Base-Ft	12	0.968	0.948	0.405	0.747

Table 38: Metrics Base Models

As we can see from the graphs and the table the network perform very well with an accuracy about 94.5% and 94.8% and a good Testing Loss respect to the initial models of VGG16, but also in this case it goes in Overfitting very soon.

Augmented Model To try to combat the Overfitting problem we add to the previous model a layer that augmented the data using the structure that we describe on Section 4.1.1.

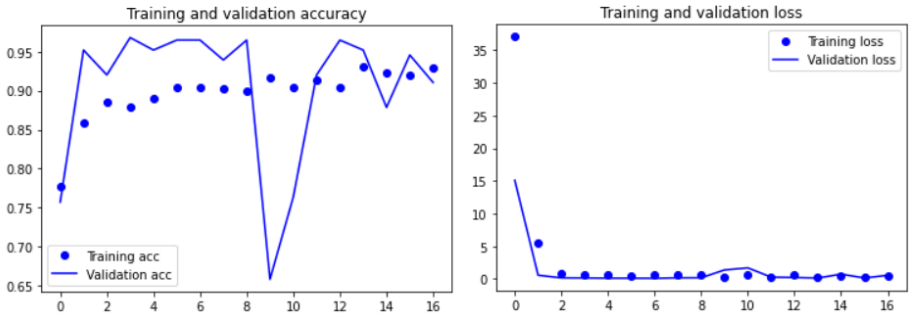


Figure 53: Augmented Model Feature Extraction

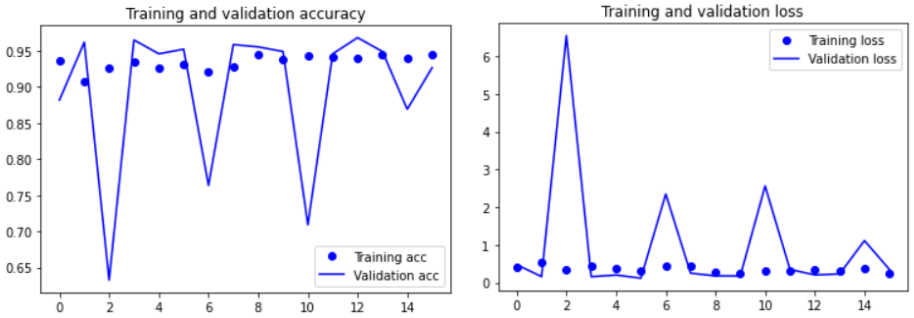


Figure 54: Augmented Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Augment-Fe	17	0.965	0.925	0.109	0.553
Augment-Ft	16	0.952	0.936	0.121	0.203

Table 39: Metrics Augmented Models

As we can see from the graphs and the table the network has a decrease of about 1%/2% of Testing accuracy but with a decrease of Testing Loss respect to previous models. The Overfitting problem is still practically unchanged.

Dropout Models Another way to reduce the Overfitting problem is to use the Dropout, so we add to the previous model a layer to insert the Dropout. We try to use two different values(0.2 and 0.5).

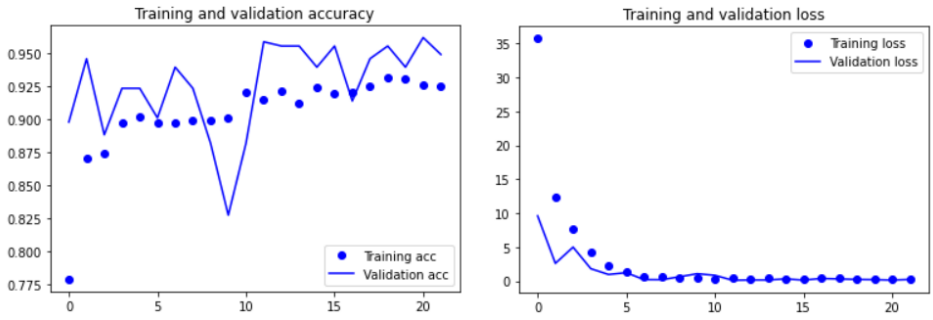


Figure 55: Dropout 0.2 Model Feature Extraction

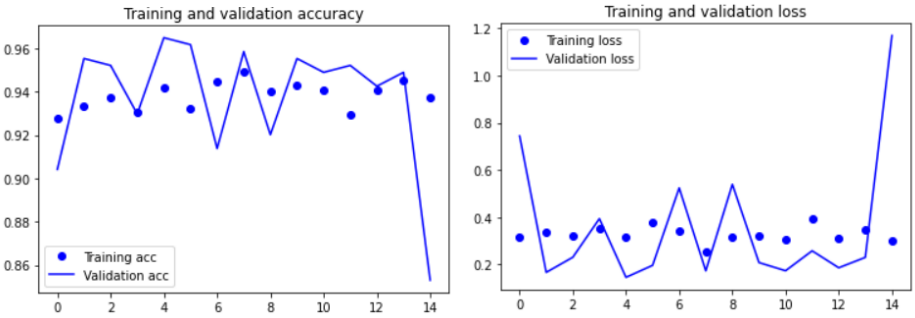


Figure 56: Dropout 0.2 Model Fine Tuning

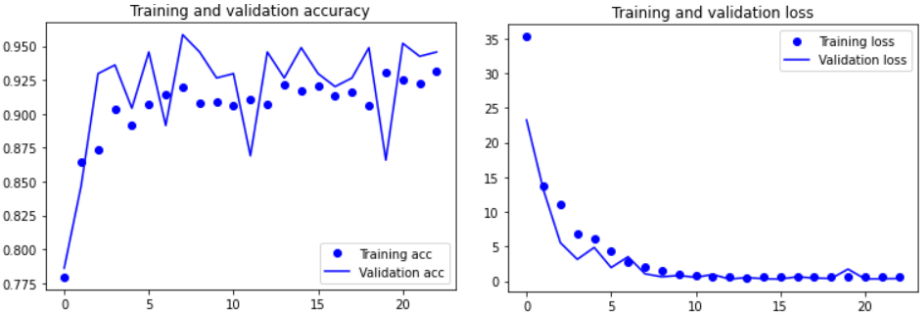


Figure 57: Dropout 0.5 Model Feature Extraction

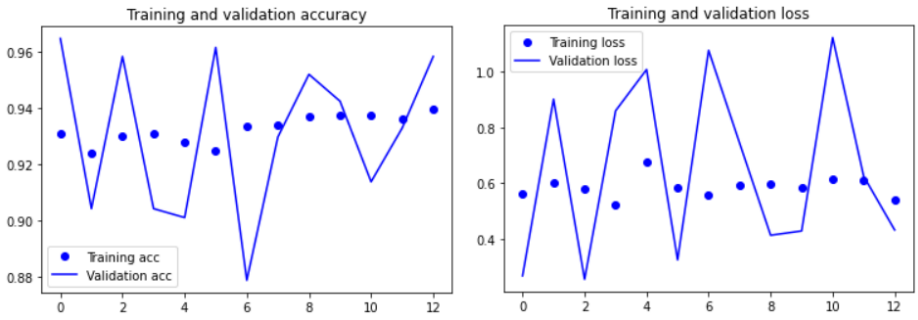


Figure 58: Dropout 0.5 Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Drop. 0.2-Fe	22	0.959	0.936	0.125	0.278
Drop. 0.2-Ft	15	0.965	0.867	0.146	1.086
Drop. 0.5-Fe	23	0.948	0.948	0.286	0.406
Drop. 0.5-Ft	13	0.959	0.942	0.256	0.562

Table 40: Metrics Dropout Models

As we can see from the graphs and the table these the network Drop.0.2 with the Fine Tuning approach has a strong decrease of about 7% of Testing accuracy with a increase of Testing Loss respect to previous models. Instead, the other models have similar values. The Overfitting begins to reduce in particular for Feature Extraction's models.

Regularized Model The last technique that we do to try to reduce the Overfitting problem is add The Regularization to the previous model with Dropout 0.2. So on the hidden layer of the fully connected net we forcing the weights to only take small values. We use the Regularization L1_L2 with corresponding values L1 = 1e-5 and L2 = 1e-4.

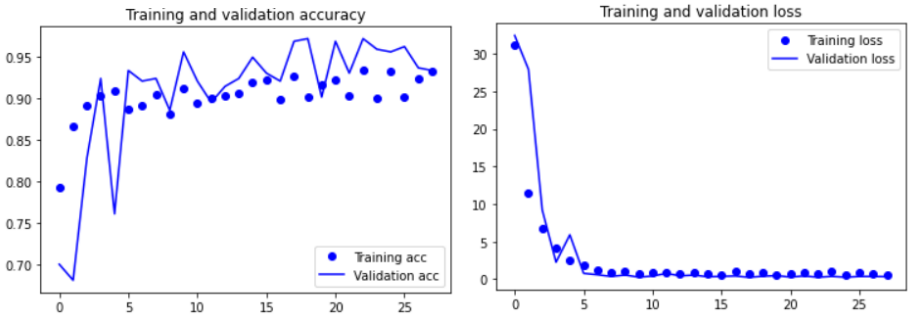


Figure 59: Regularized Model Feature Extraction

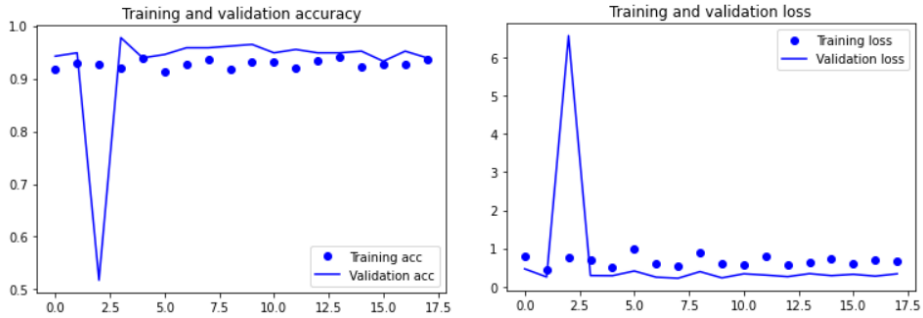


Figure 60: Regularized Model Fine Tuning

Model	**Epoch Stop**	**Validation Accuracy**	**Testing Accuracy**	**Validation Loss**	**Testing Loss**
Regular-Fe	28	0.968	0.945	0.247	0.346
Regular-Ft	18	0.959	0.957	0.218	0.340

Table 41: Metrics Regularized Models

As we can see from the graphs and the table these the network with the Fine Tuning approach has an increase about 1% of Testing accuracy with a decrease of Testing Loss respect to previous models. The Overfitting has another improvement in particular for Feature Extraction's model.

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Base-Fe	16	0.959	0.945	0.176	0.639
Base-Ft	12	0.968	0.948	0.405	0.747
Augment-Fe	17	0.965	0.925	0.109	0.553
Augment-Ft	16	0.952	0.936	0.121	0.203
Drop. 0.2-Fe	22	0.959	0.936	0.125	0.278
Drop. 0.2-Ft	15	0.965	0.867	0.146	1.086
Drop. 0.5-Fe	23	0.948	0.948	0.286	0.406
Drop. 0.5-Ft	13	0.959	0.942	0.256	0.562
Regular-Fe	28	0.968	0.945	0.247	0.346
Regular-Ft	18	0.959	0.957	0.218	0.340

Table 42: Recap Metrics InceptionV3's Models

This table report 5 different metrics that we collect after the training of all the models that we describe above. Being the final objective that of finding the best compromise between testing accuracy increase and testing loss decrease, we can point out that the optimal solution is in this case given by the model obtained using data augmentation, a drop ratio equal to 0.2 and regularization L1_L2 accordingly to the *Fine Tuning* approach. We report also the two Confusion Matrix of the first model and the best model for this pretrained network in order to see the improvement between the two models.

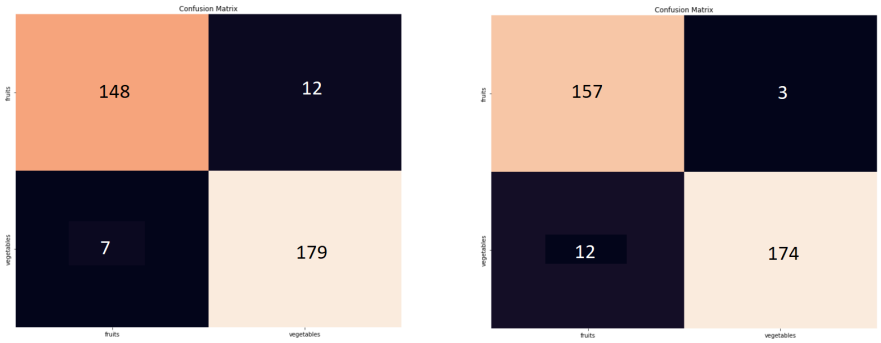


Figure 61: Confusion Matrix Base Model-Fe and Regular-Ft

4.1.4 ResNet50

Base Model In this first simple model we combined the pre-trained network with a fully connected net with a Flatten layer and a Dense layer of 256.

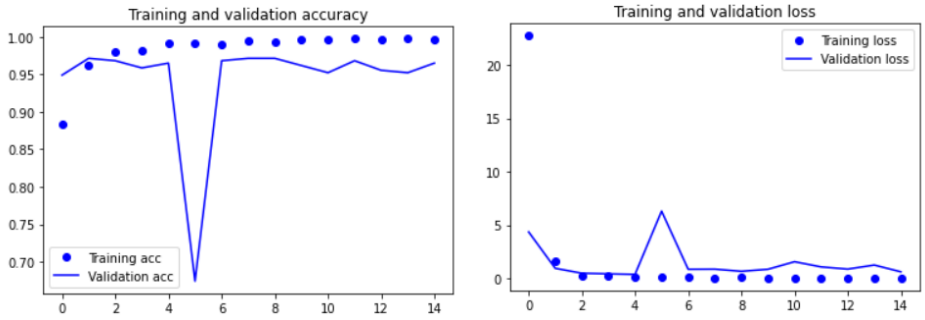


Figure 62: Base Model Feature Extraction

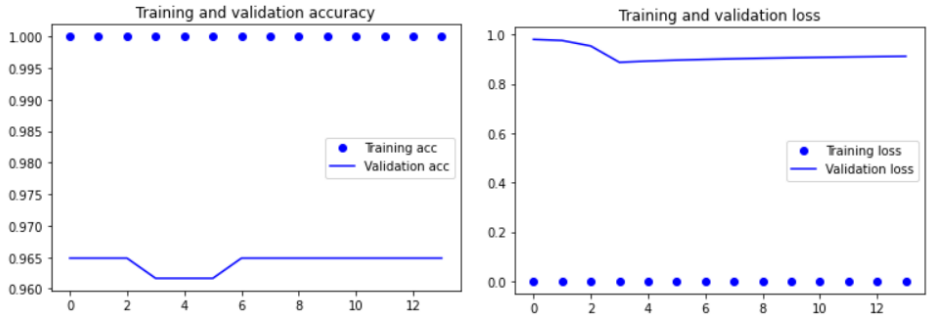


Figure 63: Base Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Base-Fe	15	0.965	0.962	0.383	1.589
Base-Ft	14	0.971	0.968	0.888	1.481

Table 43: Metrics Base Models

As we can see from the graphs and the table the network perform very well with an accuracy about 96.2% and 96.8% and a good Testing Loss respect to the initial models of VGG16, but also in this case it goes in Overfitting very soon.

Augmented Model To try to combat the Overfitting problem we add to the previous model a layer that augmented the data using the structure that we describe on Section 4.1.1.

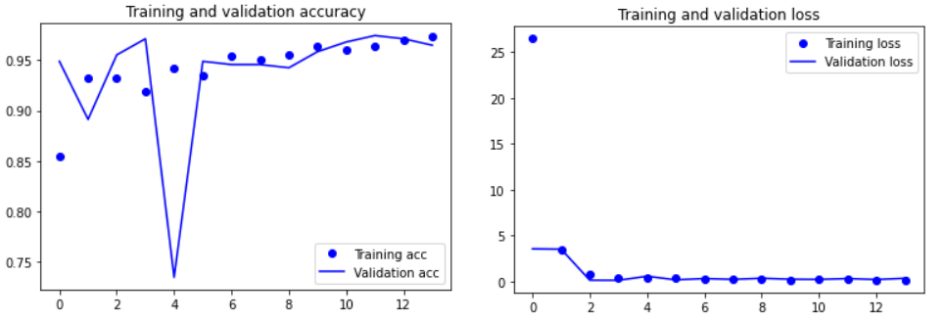


Figure 64: Augmented Model Feature Extraction

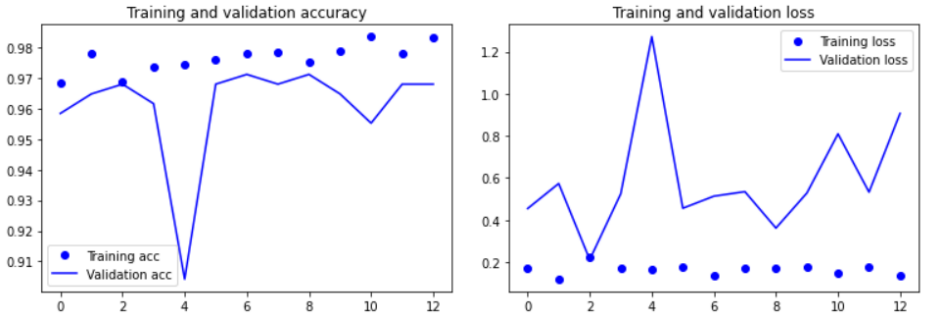


Figure 65: Augmented Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Augment-Fe	14	0.971	0.948	0.132	0.628
Augment-Ft	13	0.968	0.965	0.213	0.898

Table 44: Metrics Augmented Models

As we can see from the graphs and the table the network with the Feature Extraction approach has a decrease of about 1%/2% of Testing accuracy respect to previous models, while the Fine Tuning's model has similar value. About the Testing Loss there are a good decrease in both the approaches. The Overfitting problem is still practically unchanged.

Dropout Models Another way to reduce the Overfitting problem is to use the Dropout, so we add to the previous model a layer to insert the Dropout. We try to use two different values(0.2 and 0.5).

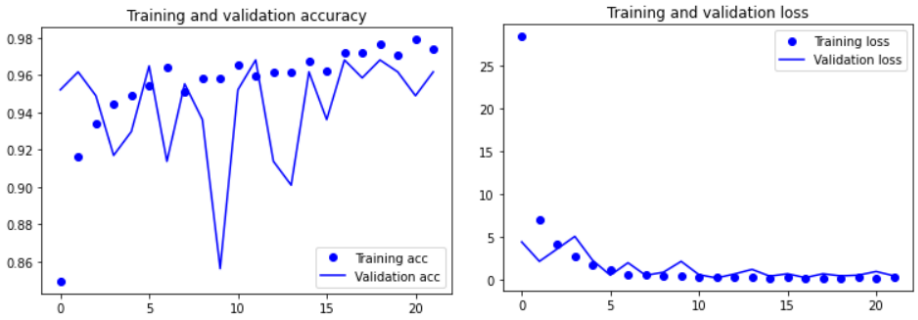


Figure 66: Dropout 0.2 Model Feature Extraction

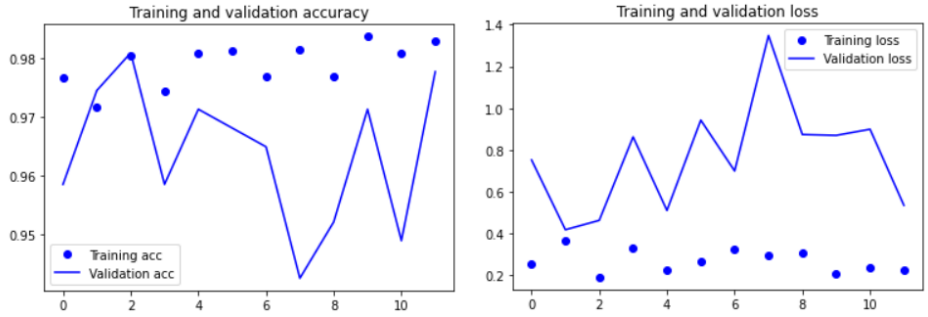


Figure 67: Dropout 0.2 Model Fine Tuning

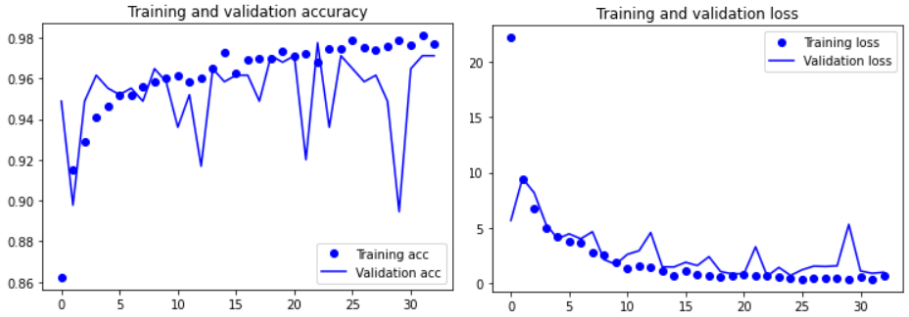


Figure 68: Dropout 0.5 Model Feature Extraction

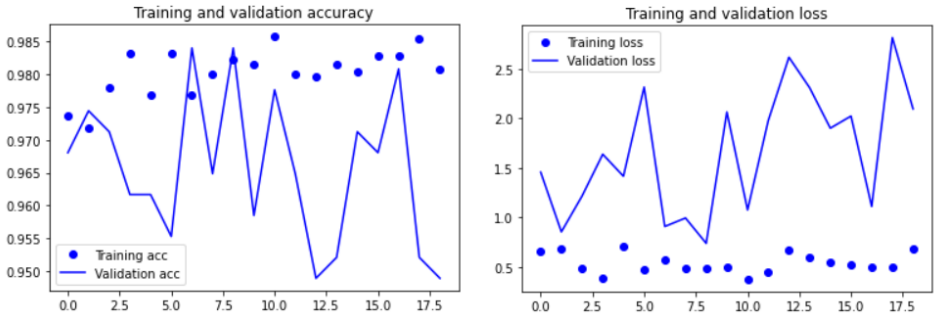


Figure 69: Dropout 0.5 Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Drop. 0.2-Fe	22	0.968	0.945	0.252	0.706
Drop. 0.2-Ft	12	0.974	0.962	0.418	0.811
Drop. 0.5-Fe	33	0.978	0.951	0.676	2.139
Drop. 0.5-Ft	19	0.984	0.945	0.739	3.292

Table 45: Metrics Dropout Models

As we can see from the graphs and the table these the networks Drop.0.2 has similar values of Testing accuracy and Testing Loss respect to corresponding previous models. Instead, the models with Dropout 0.5 have a strong deterioration of Testing Loss. The Overfitting begins to reduce in particular for Feature Extraction's models.

Regularized Model The last technique that we do to try to reduce the Overfitting problem is add The Regularization to the previous model with Dropout 0.2. So on the hidden layer of the fully connected net we forcing the weights to only take small values. We use the Regularization L1_L2 with corresponding values L1 = 1e-5 and L2 = 1e-4.

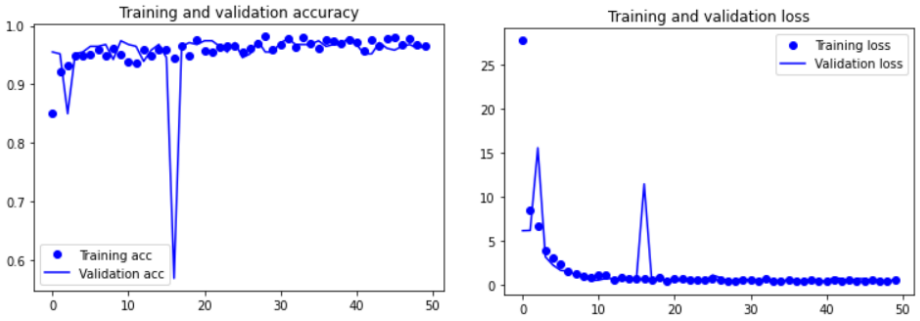


Figure 70: Regularized Model Feature Extraction

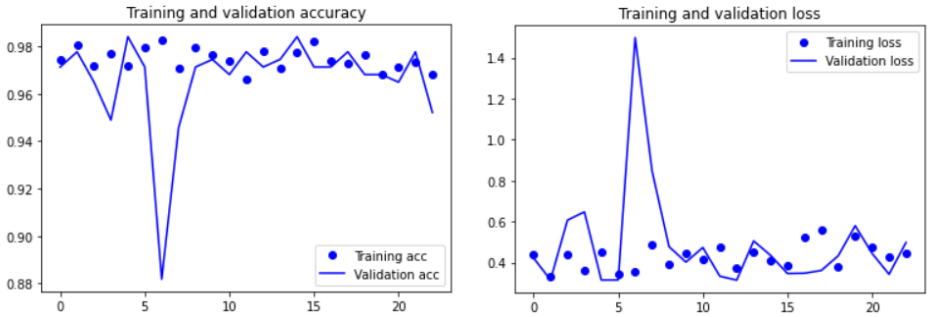


Figure 71: Regularized Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Regular-Fe	50	0.971	0.954	0.299	0.836
Regular-Ft	23	0.971	0.936	0.314	0.857

Table 46: Metrics Regularized Models

As we can see from the graphs and the table these the two networks have a decrease about 1%/2% of Testing accuracy and a similar value of Testing Loss respect to Dropout 0.2 previous models. The Overfitting problem has a strong improvement in particular for Feature Extraction's model.

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Base-Fe	15	0.965	0.962	0.383	1.589
Base-Ft	14	0.971	0.968	0.888	1.481
Augment-Fe	14	0.971	0.948	0.132	0.628
Augment-Ft	13	0.968	0.965	0.213	0.898
Drop. 0.2-Fe	22	0.968	0.945	0.252	0.706
Drop. 0.2-Ft	12	0.974	0.962	0.418	0.811
Drop. 0.5-Fe	33	0.978	0.951	0.676	2.139
Drop. 0.5-Ft	19	0.984	0.945	0.739	3.292
Regular-Fe	50	0.971	0.954	0.299	0.836
Regular-Ft	23	0.971	0.936	0.314	0.857

Table 47: Recap Metrics ResNet50's Models

This table report 5 different metrics that we collect after the training of all the models that we describe above. Being the final objective that of finding the best compromise between testing accuracy increase and testing loss decrease, we can point out that the optimal solution is in this case given by the model obtained using data augmentation and a drop ratio equal to 0.2 accordingly to the *Fine Tuning* approach. We report also the two Confusion Matrix of the first model and the best model for this pretrained network in order to see the improvement between the two models.

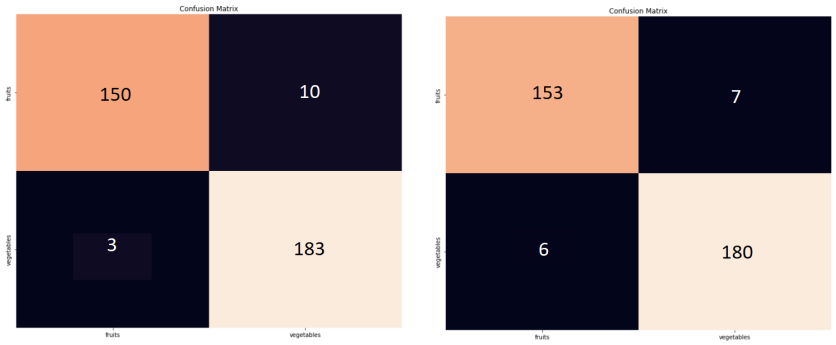


Figure 72: Confusion Matrix Base Model-Fe and Drop.0.2-Ft

4.1.5 Roc Curve

In this section we use the Roc Curve method to try to choose the best model among the 3 different pretrianed networks. As we can see from the following figure we have a tie between the best model of Inception and that of ResNet.

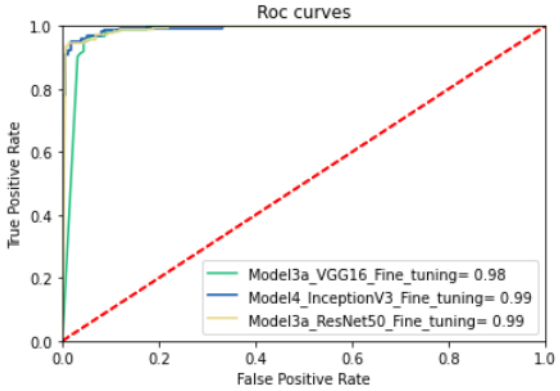


Figure 73: Roc Curve best Model for each Pretrained

4.2 Multi-Label Classification

For each pre-trained network we report the result from 5 different models both using the Feature Extraction approach and the Fine Tuning approach. All of these models are trained using the optimizer Rmsprop with the default "learning rate" (0.001), the loss function "categorical_crossentropy" 50 epochs, an Early Stopping Condition with "patience" = 10, "batch size" = 32 and a resize of the input images of 224x224. The dataset was divided as follows: the Test Set was obtained using the "validation_split" attribute (with value 0.1) of the "image_dataset_from_directory" function, and the Validation Set using the "validation_split" attribute (with value 0.1) of the "fit" function. In all of the models we start to use the Feature Extraction approach and then to try to increase the performance we use the Fine Tuning approach.

4.2.1 Data Augmentation

In order to try to improve the Overfitting problem during the training of the different models at a certain point we introduce the concept of Augmentation for generate more training data from existing training samples, by "augmenting" the samples via a number of random transformations that yield believable-looking images. The code of the layer to insert the augmentation inside the models to train is shown below.

```
[ ]  data_augmentation = keras.Sequential(
       [
         layers.RandomFlip("horizontal"),
         layers.RandomRotation(0.2),
         layers.RandomTranslation(height_factor=0.1, width_factor=0.1, fill_mode="constant", fill_value=255),
         layers.RandomZoom(0.1, fill_mode="nearest")
       ]
     )
```

Figure 74: Data Augmentation code

4.2.2 VGG16

Base Model In this first simple model we combined the pre-trained network with a fully connected net with a Flatten layer and a Dense layer of 256.

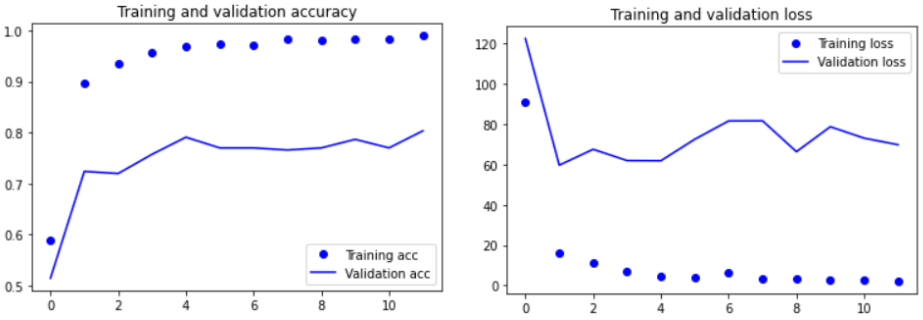


Figure 75: Base Model Feature Extraction

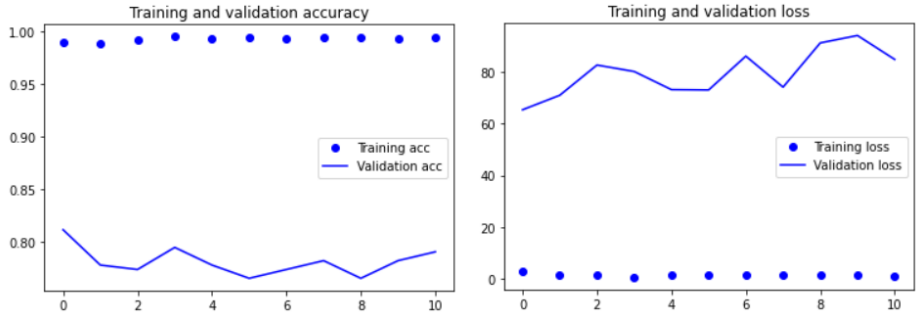


Figure 76: Base Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Base-Fe	12	0.794	0.762	59.649	80.713
Base-Ft	11	0.812	0.747	65.354	101.101

Table 48: Metrics Base Models

As we can see from the graphs and the table the network perform with an accuracy of 76.2% and 74.2%, but it has a very high Testing Loss and it goes in Overfitting very soon.

Augmented Model To try to combat the Overfitting problem we add to the previous model a layer that augmented the data using the structure that we describe on Section 4.2.1.

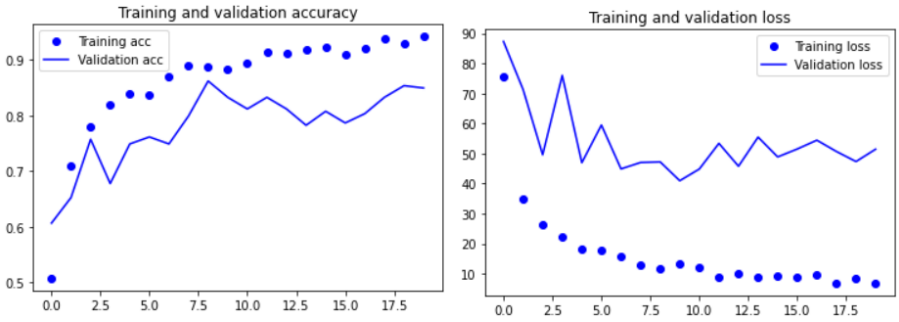


Figure 77: Augmented Model Feature Extraction

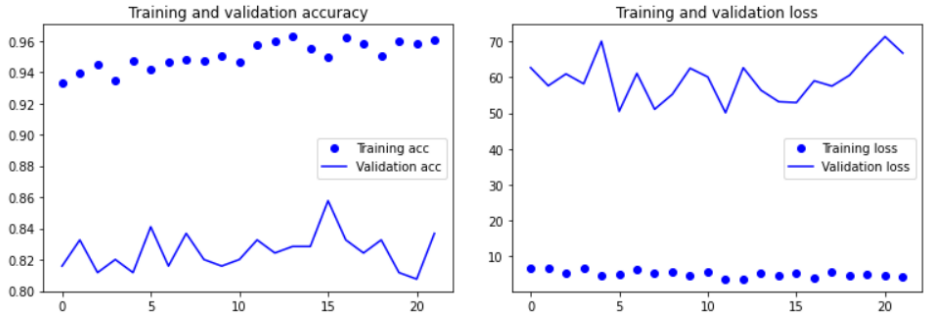


Figure 78: Augmented Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Augment-Fe	20	0.833	0.789	40.958	66.875
Augment-Ft	22	0.833	0.804	50.088	84.723

Table 49: Metrics Augmented Models

As we can see from the graphs and the table the network with both the approaches has a strong increase of Testing accuracy and a strong decrease of Testing Loss respect to corresponding previous model. The Overfitting problem has an improvement.

Dropout Models Another way to reduce the Overfitting problem is to use the Dropout, so we add to the previous model a layer to insert the Dropout. We try to use two different values(0.2 and 0.5).

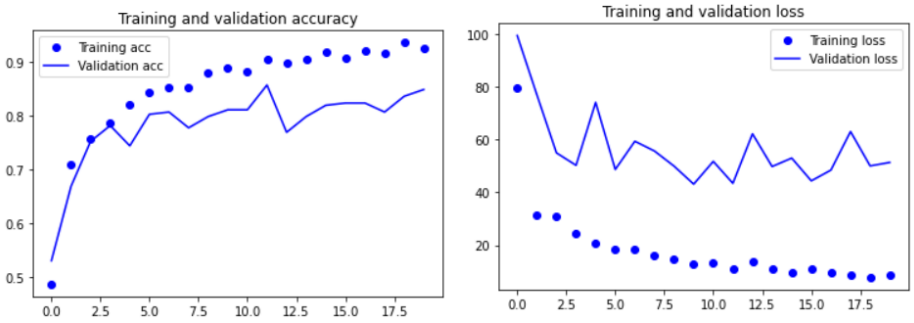


Figure 79: Dropout 0.2 Model Feature Extraction

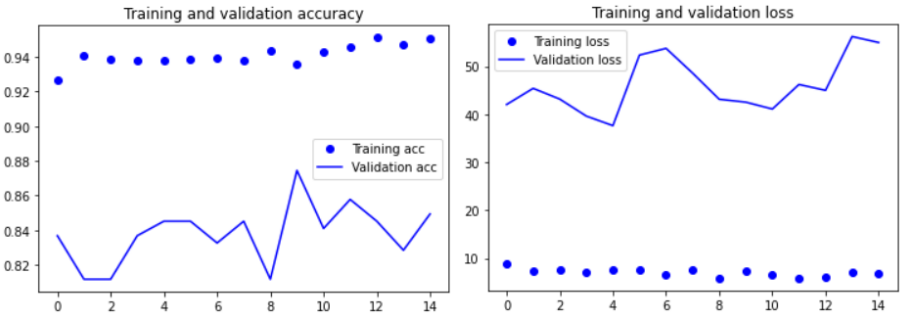


Figure 80: Dropout 0.2 Model Fine Tuning

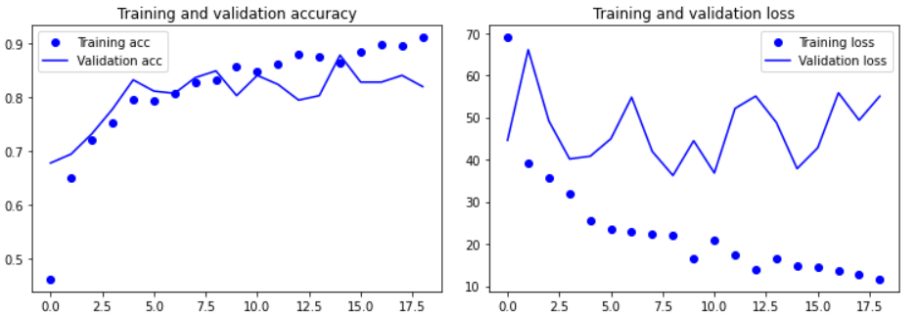


Figure 81: Dropout 0.5 Model Feature Extraction

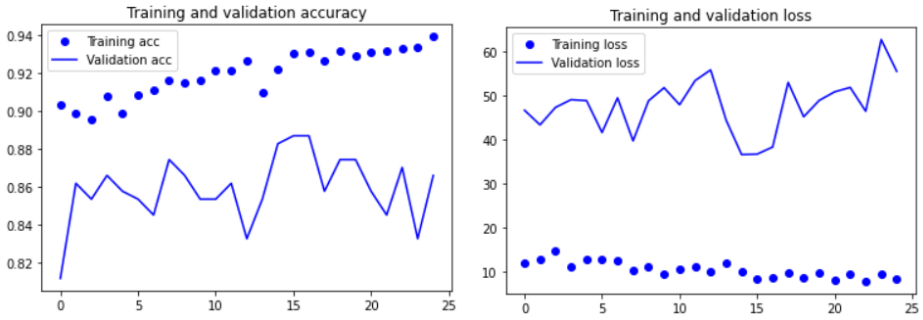


Figure 82: Dropout 0.5 Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Drop. 0.2-Fe	20	0.822	0.815	43.063	70.371
Drop. 0.2-Ft	15	0.845	0.823	37.638	66.083
Drop. 0.5-Fe	19	0.849	0.789	36.230	54.635
Drop. 0.5-Ft	25	0.883	0.819	36.655	63.756

Table 50: Metrics Dropout Models

As we can see from the graphs and the table these the networks with Dropout equal to 0.2 has increase of Testing accuracy and similar values of Testing Loss respect to corresponding previous models. Instead, for the models with Dropout 0.5 only the one with Fine Tuning approach have an improvement in both Testing Accuracy and Loss. The Overfitting has different behaviour among the 4 models.

Regularized Model The last technique that we do to try to reduce the Overfitting problem is add The Regularization to the previous model with Dropout 0.2. So on the hidden layer of the fully connected net we forcing the weights to only take small values. We use the Regularization L1_L2 with corresponding values L1 = 1e-5 and L2 = 1e-4.

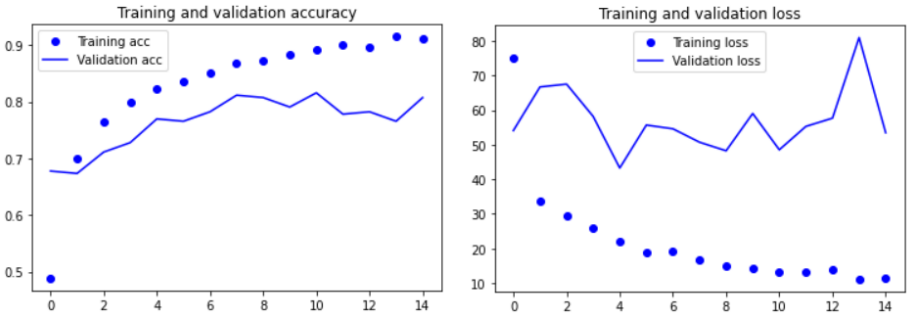


Figure 83: Regularized Model Feature Extraction

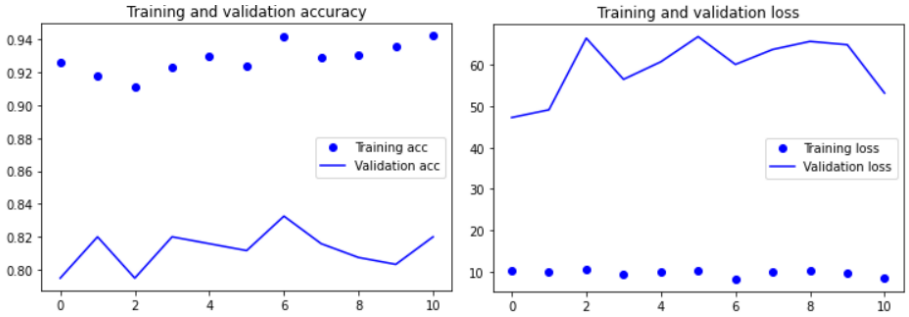


Figure 84: Regularized Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Regular-Fe	15	0.828	0.800	43.237	57.574
Regular-Ft	11	0.845	0.815	47.250	53.604

Table 51: Metrics Regularized Models

As we can see from the graphs and the table these the two networks have a little decrease about 1% of Testing accuracy and an improvement of Testing Loss respect to corresponding Dropout 0.2 previous models. The Overfitting problem has a strong deterioration in both the approaches.

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Base-Fe	12	0.794	0.762	59.649	80.713
Base-Ft	11	0.812	0.747	65.354	101.101
Augment-Fe	20	0.833	0.789	40.958	66.875
Augment-Ft	22	0.833	0.804	50.088	84.723
Drop. 0.2-Fe	20	0.822	0.815	43.063	70.371
Drop. 0.2-Ft	15	0.845	0.823	37.638	66.083
Drop. 0.5-Fe	19	0.849	0.789	36.230	54.635
Drop. 0.5-Ft	25	0.883	0.819	36.655	63.756
Regular-Fe	15	0.828	0.800	43.237	57.574
Regular-Ft	11	0.845	0.815	47.250	53.604

Table 52: Recap Metrics VGG16's Models

This table report 5 different metrics that we collect after the training of all the models that we describe above. Here, unlike the binary case, a first evaluation among the models can be finding the best compromise between testing accuracy increase and testing loss decrease. So we can point out that the 3 optimal solutions are in this case given by the model obtained using data augmentation and a drop ratio equal to 0.2 accordingly to the *Fine Tuning* approach, the model obtained using data augmentation and a drop ratio equal to 0.5 accordingly to the *Fine Tuning* approach and the model obtained using data augmentation, a drop ratio equal to 0.2 and regularization L1_L2 accordingly to the *Fine Tuning* approach.

4.2.3 InceptionV3

Base Model In this first simple model we combined the pre-trained network with a fully connected net with a Flatten layer and a Dense layer of 256.

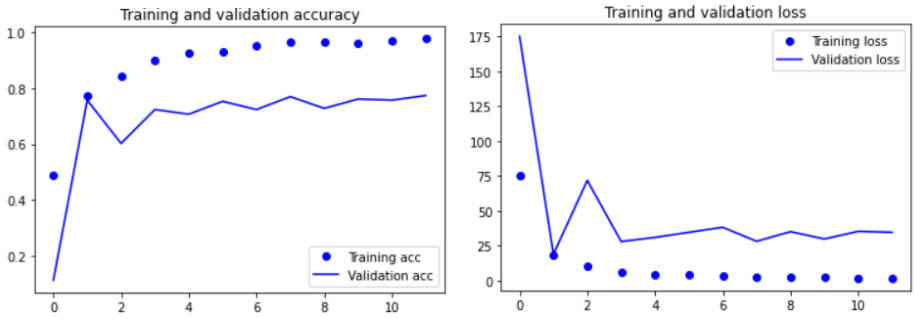


Figure 85: Base Model Feature Extraction

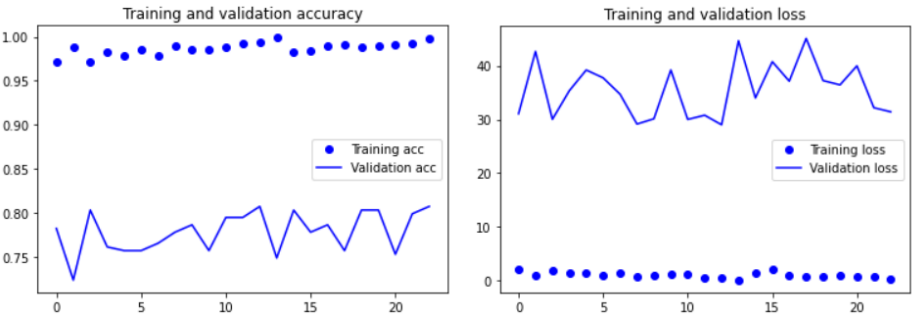


Figure 86: Base Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Base-Fe	12	0.787	0.766	18.871	30.058
Base-Ft	23	0.808	0.759	29.008	35.196

Table 53: Metrics Base Models

As we can see from the graphs and the table the networks perform with an accuracy of 76.6% and 75.9% and have a value of Testing Loss very low respect to the VGG16's initial models. The network using Feature Extraction approach goes in Overfitting very soon.

Augmented Model To try to combat the Overfitting problem we add to the previous model a layer that augmented the data using the structure that we describe on Section 4.2.1.

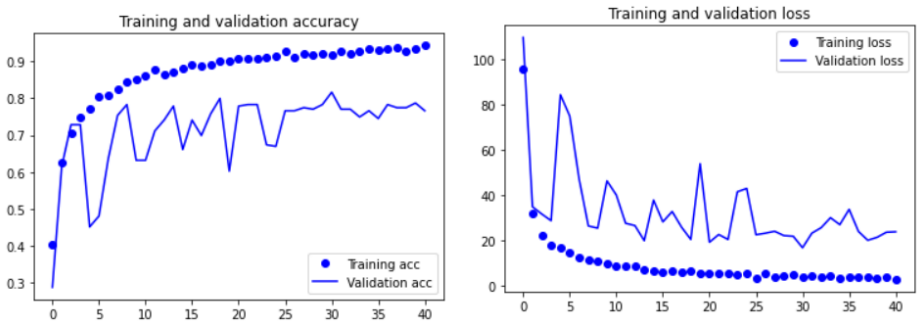


Figure 87: Augmented Model Feature Extraction

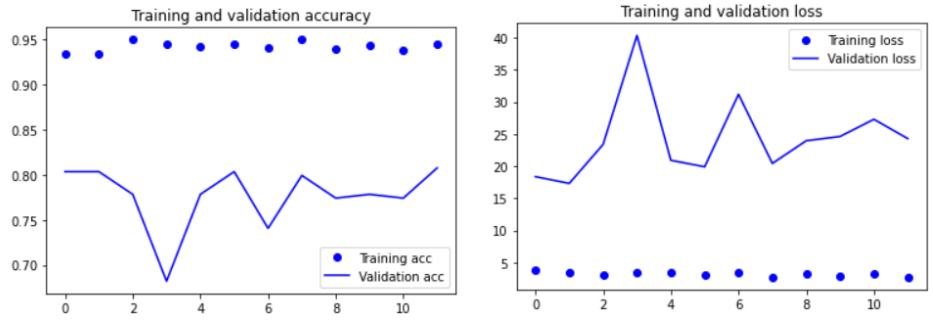


Figure 88: Augmented Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Augment-Fe	41	0.816	0.740	16.780	28.746
Augment-Ft	12	0.803	0.785	17.317	23.396

Table 54: Metrics Augmented Models

As we can see from the graphs and the table the network with both the approaches a strong decrease of Testing Loss respect to corresponding previous model, while for Testing Accuracy they have different behaviour. The Overfitting problem has a strong improvement only for the Feature Extraction's model.

Dropout Models Another way to reduce the Overfitting problem is to use the Dropout, so we add to the previous model a layer to insert the Dropout. We try to use two different values(0.2 and 0.5).

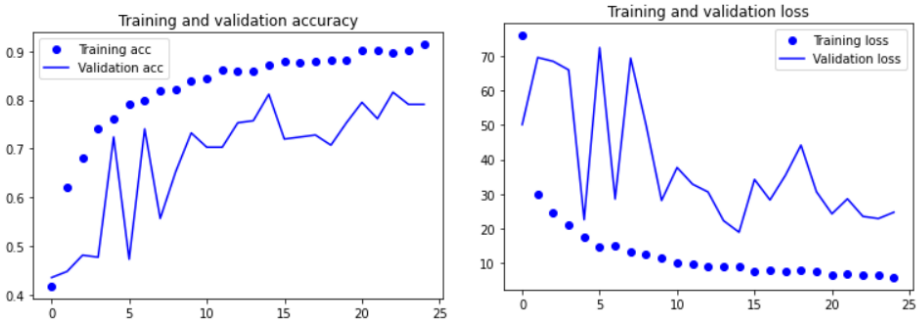


Figure 89: Dropout 0.2 Model Feature Extraction

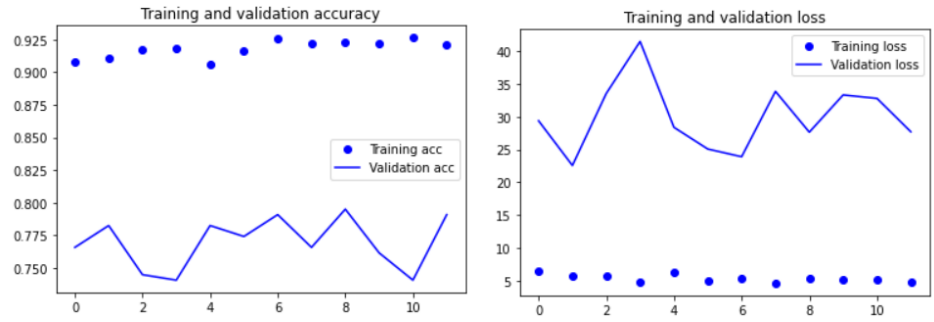


Figure 90: Dropout 0.2 Model Fine Tuning

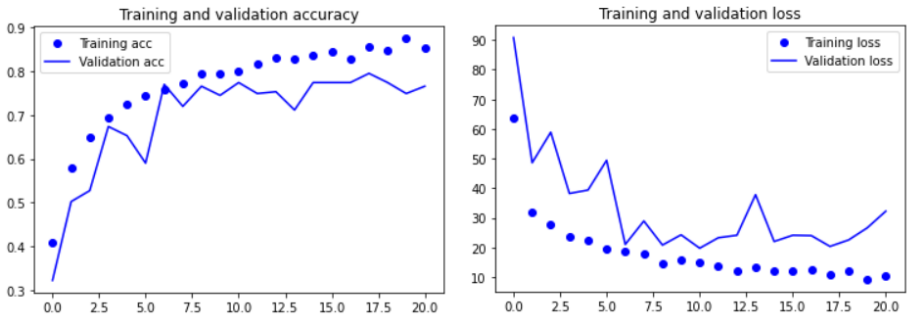


Figure 91: Dropout 0.5 Model Feature Extraction

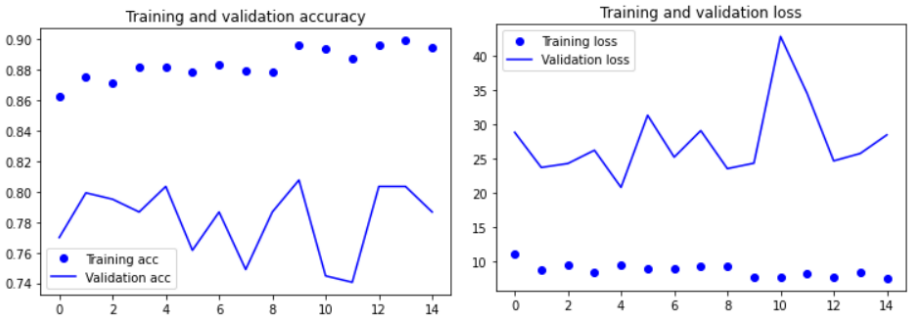


Figure 92: Dropout 0.5 Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Drop. 0.2-Fe	25	0.812	0.781	18.959	22.409
Drop. 0.2-Ft	12	0.792	0.781	22.563	24.487
Drop. 0.5-Fe	21	0.774	0.766	19.758	30.215
Drop. 0.5-Ft	15	0.803	0.781	20.815	30.582

Table 55: Metrics Dropout Models

As we can see from the graphs and the table these the networks with Dropout equal to 0.2 has similar value of Testing accuracy and Loss respect to previous Fine Tuning's model. Instead, for the models with Dropout 0.5 the one with Feature Extraction approach has a deterioration in both the metrics and the one Fine Tuning approach only in Testing Loss. The Overfitting has different behaviour among the 4 models.

Regularized Model The last technique that we do to try to reduce the Overfitting problem is add The Regularization to the previous model with Dropout 0.2. So on the hidden layer of the fully connected net we forcing the weights to only take small values. We use the Regularization L1_L2 with corresponding values L1 = 1e-5 and L2 = 1e-4.

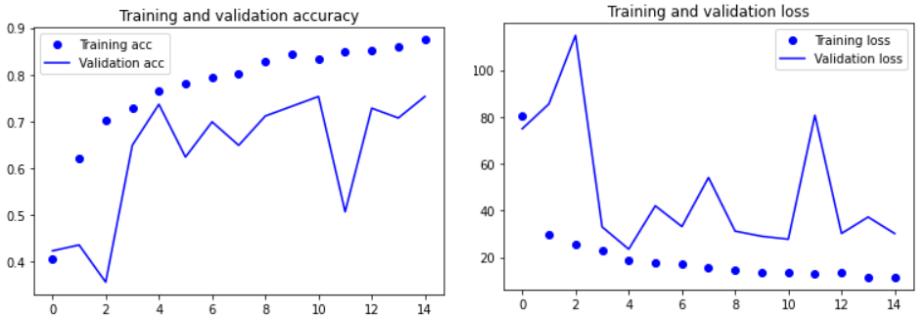


Figure 93: Regularized Model Feature Extraction

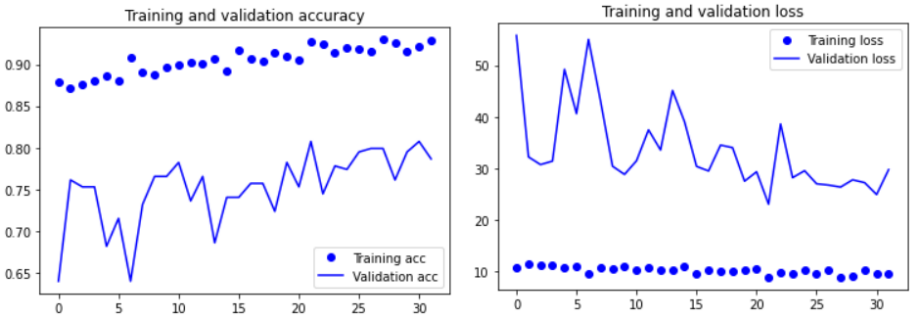


Figure 94: Regularized Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Regular-Fe	15	0.796	0.766	23.503	25.750
Regular-Ft	32	0.808	0.785	23.038	28.699

Table 56: Metrics Regularized Models

As we can see from the graphs and the table these the two networks have a similar values of Testing accuracy and an improvement of Testing Loss respect to corresponding Dropout 0.2 previous models. The Overfitting problem has a improvement in particular for the Fine Tuning's model.

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Base-Fe	12	0.787	0.766	18.871	30.058
Base-Ft	23	0.808	0.759	29.008	35.196
Augment-Fe	41	0.816	0.740	16.780	28.746
Augment-Ft	12	0.803	0.785	17.317	23.396
Drop. 0.2-Fe	25	0.812	0.781	18.959	22.409
Drop. 0.2-Ft	12	0.792	0.781	22.563	24.487
Drop. 0.5-Fe	21	0.774	0.766	19.758	30.215
Drop. 0.5-Ft	15	0.803	0.781	20.815	30.582
Regular-Fe	15	0.796	0.766	23.503	25.750
Regular-Ft	32	0.808	0.785	23.038	28.699

Table 57: Recap Metrics InceptionV3's Models

This table report 5 different metrics that we collect after the training of all the models that we describe above. Here, unlike the binary case, a first evaluation among the models can be finding the best compromise between testing accuracy increase and testing loss decrease. So we can point out that the 3 optimal solutions are in this case given by the model obtained using data augmentation accordingly to the *Fine Tuning* approach, the model obtained using data augmentation and a drop ratio equal to 0.2 accordingly to the *Feature Extraction* approach and the model obtained using data augmentation, a drop ratio equal to 0.2 and regularization L1_L2 accordingly to the *Fine Tuning* approach.

4.2.4 ResNet50

Base Model In this first simple model we combined the pre-trained network with a fully connected net with a Flatten layer and a Dense layer of 256.

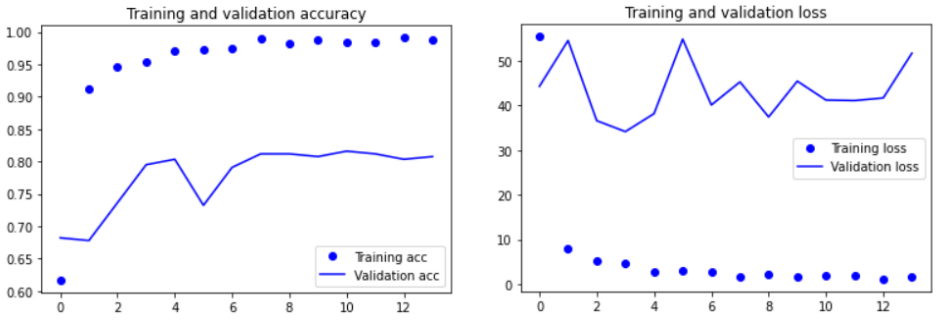


Figure 95: Base Model Feature Extraction

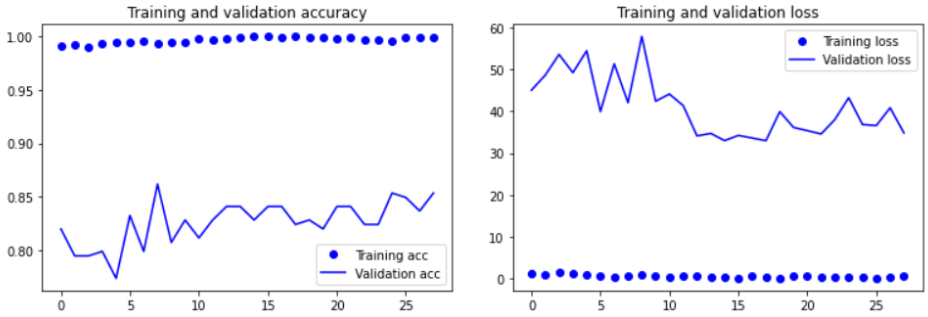


Figure 96: Base Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Base-Fe	14	0.795	0.740	34.164	60.801
Base-Ft	28	0.848	0.834	32.981	35.625

Table 58: Metrics Base Models

As we can see from the graphs and the table the networks perform with an accuracy of 74% and 83.4% and have a value of Testing Loss very low respect to the VGG16's initial models but higher than the InceptionV3's initial models. The network using Feature Extraction approach goes in Overfitting very soon, while the one using Fine Tuning approach goes in Overfitting later.

Augmented Model To try to combat the Overfitting problem we add to the previous model a layer that augmented the data using the structure that we describe on Section 4.2.1.

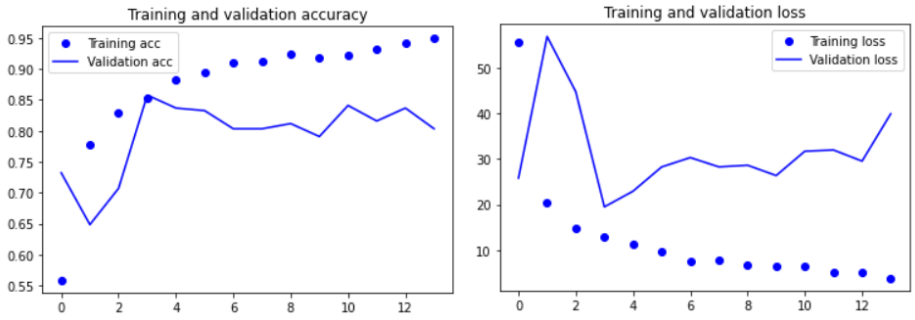


Figure 97: Augmented Model Feature Extraction

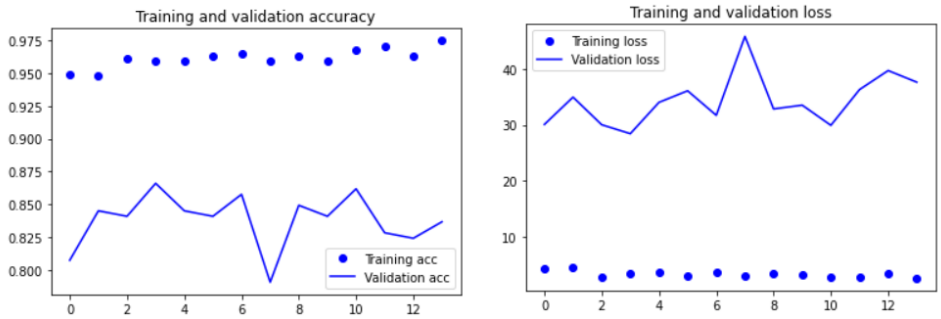


Figure 98: Augmented Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Augment-Fe	14	0.858	0.796	19.483	40.728
Augment-Ft	14	0.866	0.796	28.421	48.308

Table 59: Metrics Augmented Models

As we can see from the graphs and the table the network with both the approaches a better value of Testing accuracy and Loss respect to Feature Extraction's previous model, but a worse values respect to the Fine Tuning's previuos model. The Overfitting problem has a similar behaviour to the Feature Extraction's previous model.

Dropout Models Another way to reduce the Overfitting problem is to use the Dropout, so we add to the previous model a layer to insert the Dropout. We try to use two different values(0.2 and 0.5).

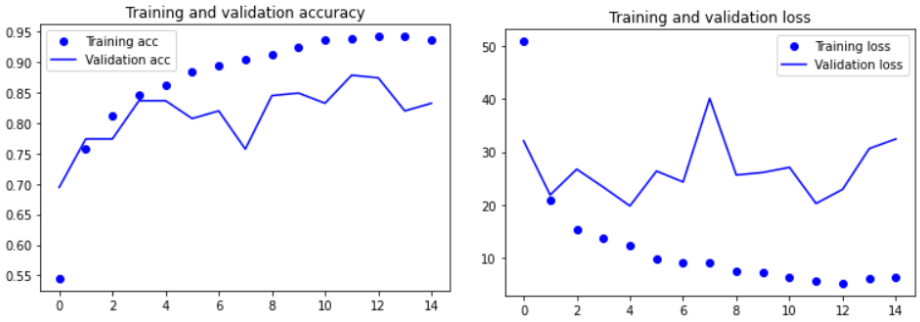


Figure 99: Dropout 0.2 Model Feature Extraction

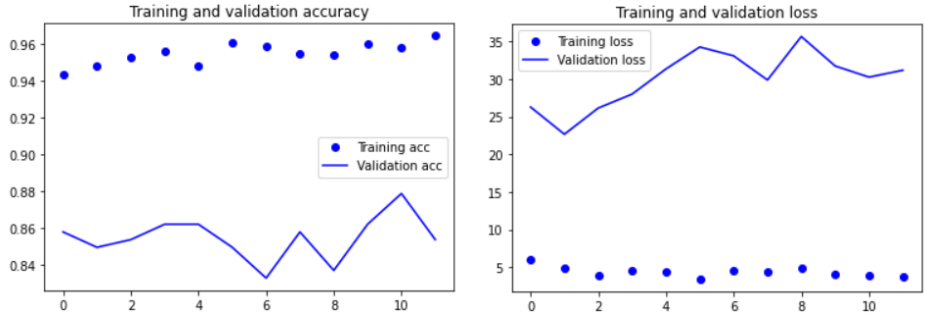


Figure 100: Dropout 0.2 Model Fine Tuning

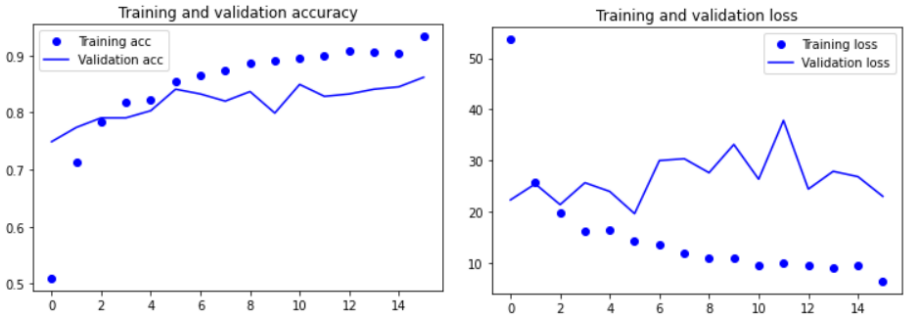


Figure 101: Dropout 0.5 Model Feature Extraction

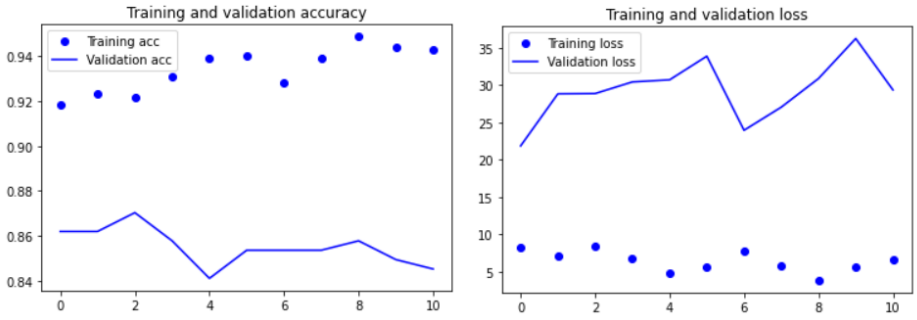


Figure 102: Dropout 0.5 Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Drop. 0.2-Fe	15	0.837	0.785	19.796	53.705
Drop. 0.2-Ft	11	0.849	0.830	22.669	38.097
Drop. 0.5-Fe	16	0.841	0.785	19.604	41.584
Drop. 0.5-Ft	11	0.862	0.819	21.864	36.060

Table 60: Metrics Dropout Models

As we can see from the graphs and the table these the networks with Fine Tuning approach has an improvement of Testing accuracy and Loss respect to previous model. Instead, the models with Feature Extraction method have similar values in both the metrics. The Overfitting has different behaviour among the 4 models.

Regularized Model The last technique that we do to try to reduce the Overfitting problem is add The Regularization to the previous model with Dropout 0.2. So on the hidden layer of the fully connected net we forcing the weights to only take small values. We use the Regularization L1_L2 with corresponding values L1 = 1e-5 and L2 = 1e-4.

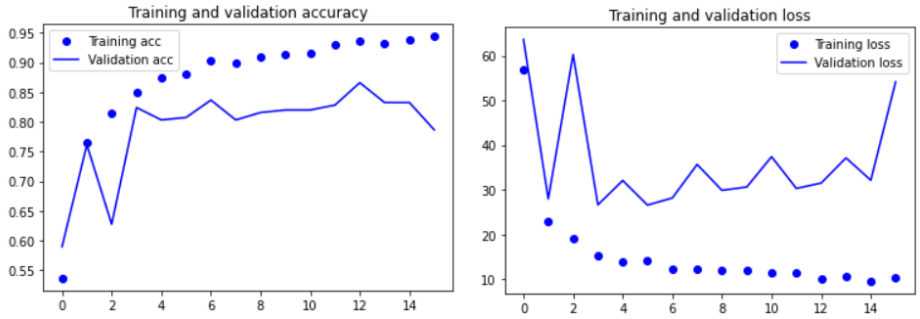


Figure 103: Regularized Model Feature Extraction

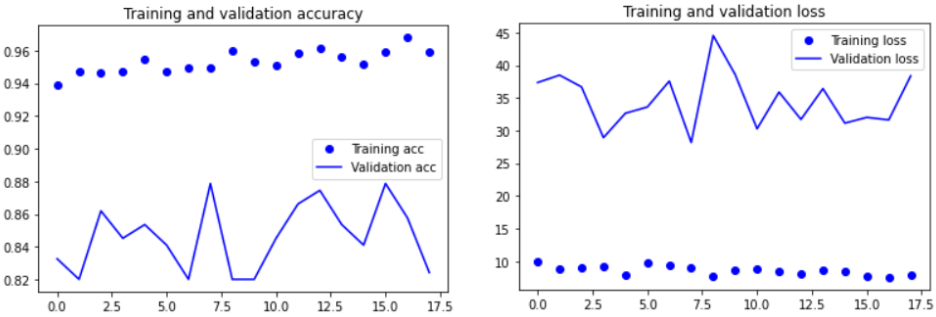


Figure 104: Regularized Model Fine Tuning

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Regular-Fe	16	0.808	0.759	26.585	58.987
Regular-Ft	18	0.879	0.808	28.176	38.492

Table 61: Metrics Regularized Models

As we can see from the graphs and the table these the model with Feature Extractoin approach has a deteriotation in both Testing accuracy and Loss respect to the previous models, while the model with Fine Tuning method has similar value in both the metrics. The Overfitting problem has a little improvement in particular for the Fine Tuning's model.

Model	Epoch Stop	Validation Accuracy	Testing Accuracy	Validation Loss	Testing Loss
Base-Fe	14	0.795	0.740	34.164	60.801
Base-Ft	28	0.848	0.834	32.981	35.625
Augment-Fe	14	0.858	0.796	19.483	40.728
Augment-Ft	14	0.866	0.796	28.421	48.308
Drop. 0.2-Fe	15	0.837	0.785	19.796	53.705
Drop. 0.2-Ft	11	0.849	0.830	22.669	38.097
Drop. 0.5-Fe	16	0.841	0.785	19.604	41.584
Drop. 0.5-Ft	11	0.862	0.819	21.864	36.060
Regular-Fe	16	0.808	0.759	26.585	58.987
Regular-Ft	18	0.879	0.808	28.176	38.492

Table 62: Recap Metrics ResNet50's Models

This table report 5 different metrics that we collect after the training of all the models that we describe above. Here, unlike the binary case, a first evaluation among the models can be finding the best compromise between testing accuracy increase and testing loss decrease. So we can point out that the 3 optimal solutions are in this case given by the base model obtained accordingly to the *Fine Tuning* approach, the model obtained using data augmentation and a drop ratio equal to 0.2 accordingly to the *Fine Tuning* approach and the model obtained using data augmentation, a drop ratio equal to 0.5 accordingly to the *Fine Tuning* approach.

4.2.5 Evaluation Metrics

In this section we do a second evaluation of 4 important metrics between the 3 best models for each pretrained network that we have identified in the previous paragraphs.

VGG16

Model	Precision	Recall	F1-Score	Support
Drop. 0.2-Ft	0.809	0.789	0.781	265
Drop. 0.5-Ft	0.830	0.811	0.809	265
Regular-Ft	0.843	0.815	0.815	265

This table report the 4 metrics that describe the 3 models. Being the final objective that of finding the best compromise between precision increase and recall increase we can point out that the best optimal solution is in this case given by the model obtained using data augmentation, a dropout ratio equal to 0.2 and a regularization L1_L2 accordingly to the *Fine Tuning* approach. We report also the two Confusion Matrix of the first model and the best model for this pretrained network.

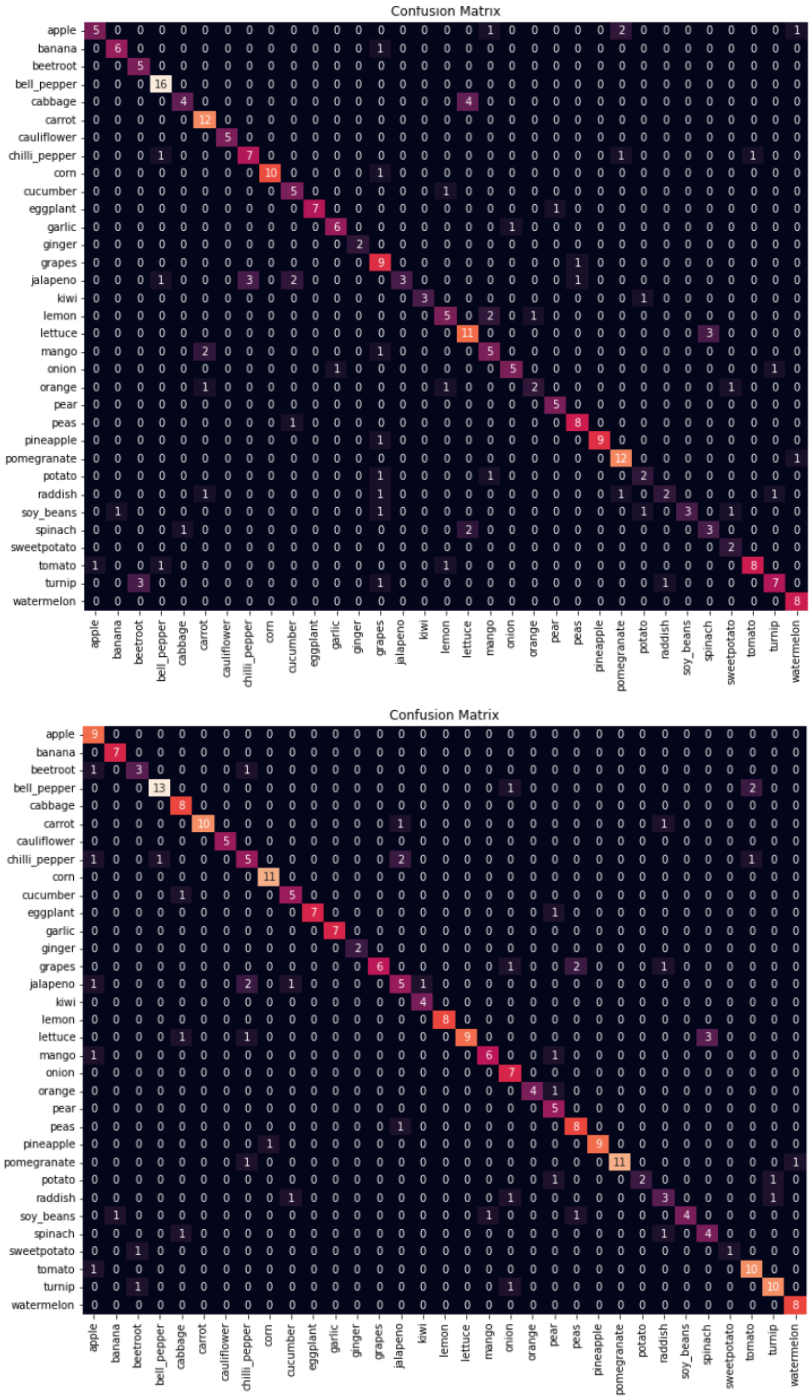
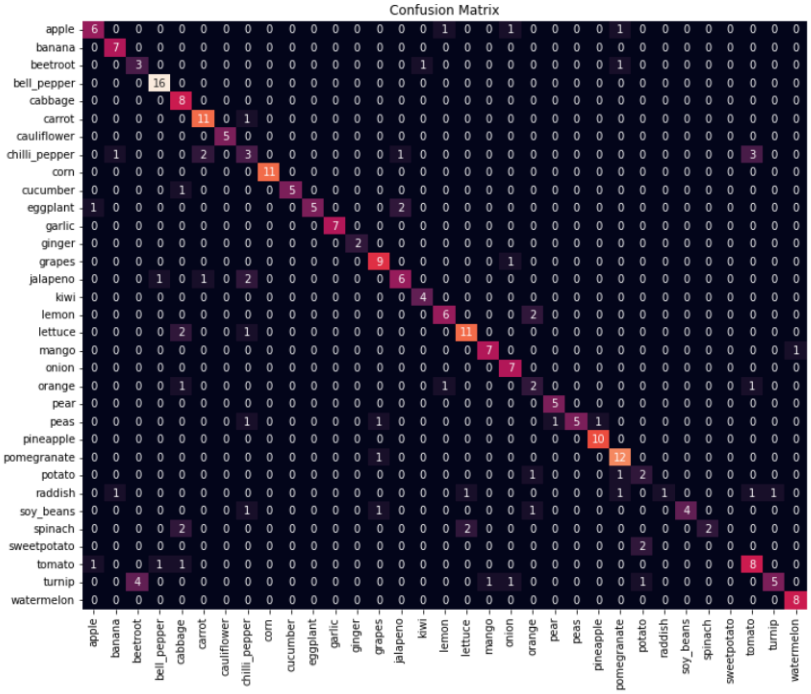


Figure 105: Confusion Matrix Base Model-Fe and Regular-Ft

InceptionV3

Model	Precision	Recall	F1-Score	Support
Augment-Ft	0.821	0.808	0.799	265
Drop. 0.2-Fe	0.764	0.774	0.754	265
Regular-Ft	0.812	0.781	0.779	265

This table report the 4 metrics that describe the 3 models. Being the final objective that of finding the best compromise between precision increase and recall increase we can point out that the best optimal solution is in this case given by the model obtained using data augmentation accordingly to the *Fine Tuning* approach. We report also the two Confusion Matrix of the first model and the best model for this pretrained network.



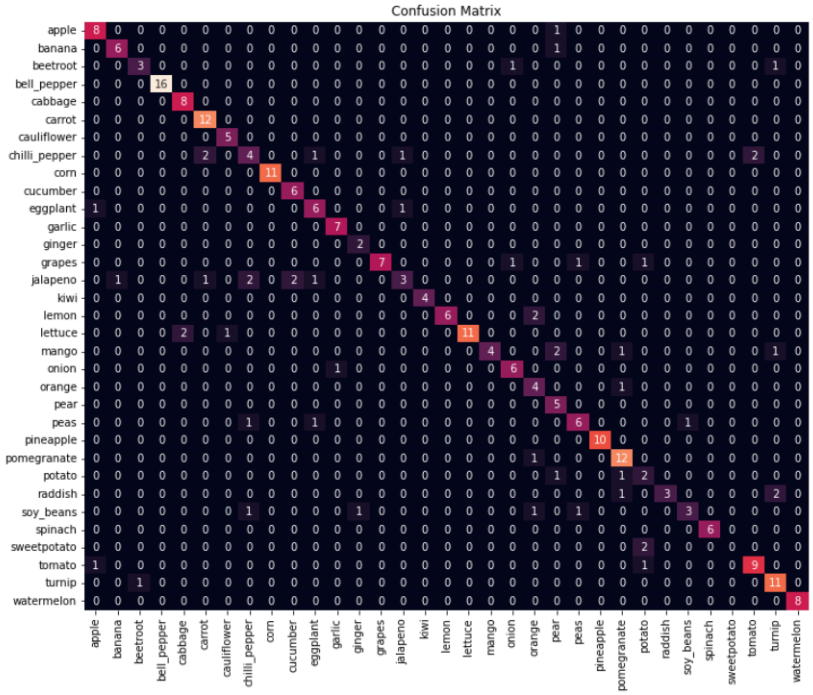


Figure 106: Confusion Matrix Base Model-Fe and Augmented-Ft

ResNet50

Model	Precision	Recall	F1-Score	Support
Base-Ft	0.852	0.819	0.812	265
Drop. 0.2-Ft	0.815	0.796	0.792	265
Drop. 0.5-Ft	0.826	0.781	0.778	265

This table report the 4 metrics that describe the 3 models. Being the final objective that of finding the best compromise between precision increase and recall increase we can point out that the best optimal solution is in this case given by the base model obtained accordingly to the *Fine Tuning* approach. We report also the two Confusion Matrix of the first model and the best model for this pretrained network.

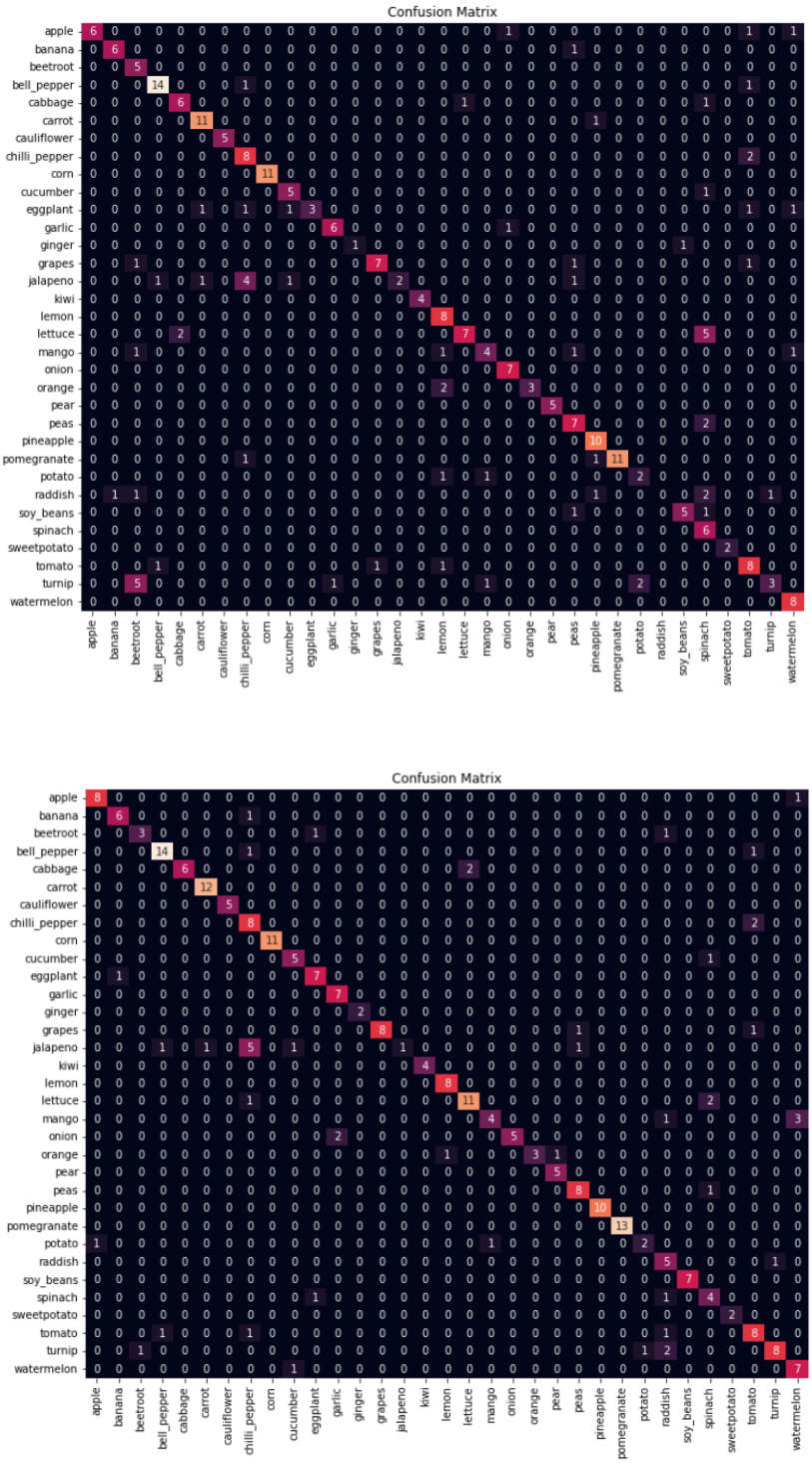


Figure 107: Confusion Matrix Base Model-Fe and Base Model-Ft

5 Task 4 - Ensemble

In this task our aim is to get a better prediction than the best models obtained for binary classification and multi-label classification. The technique used is named ensemble and it consists to cluster a set of classifiers to produce a composite classifier that outperforms the base classifiers. For each context (Binary Scratch, Multi-Label Scratch, Binary Pretrained, Multi-Label Pretrained), we choose the best three classifiers that will be used to perform the Majority Voting and the Average Voting.

5.1 Binary Scratch

The ensemble is composed by the models: Model 5, Model 6b and Model 7 that were obtained during task 2.2 and will be used to perform Majority Voting and Average Voting techniques. The confusion matrix for each method are shown below.

5.1.1 Majority

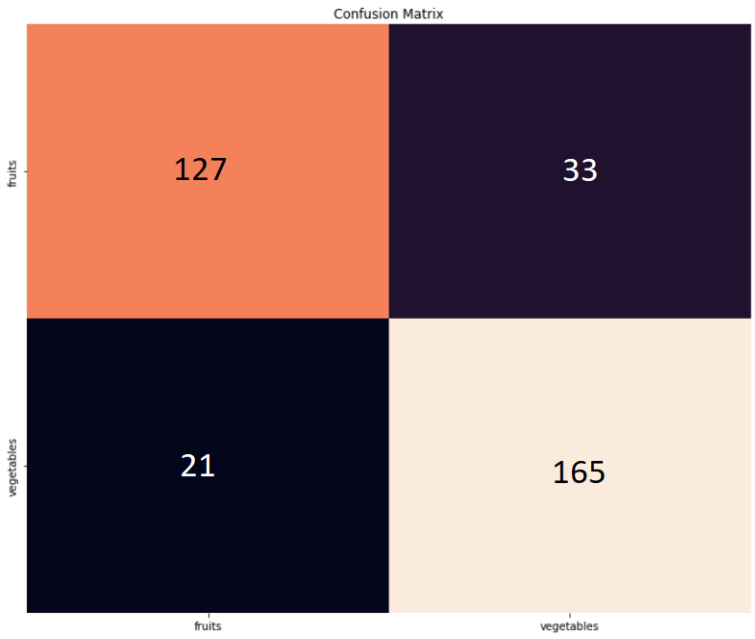


Figure 108: Confusion Matrix Ensemble Model Majority method

accuracy			0.84	346
macro avg	0.85	0.84	0.84	346
weighted avg	0.84	0.84	0.84	346

Figure 109: Metrics Ensemble Model Majority method

5.1.2 Average

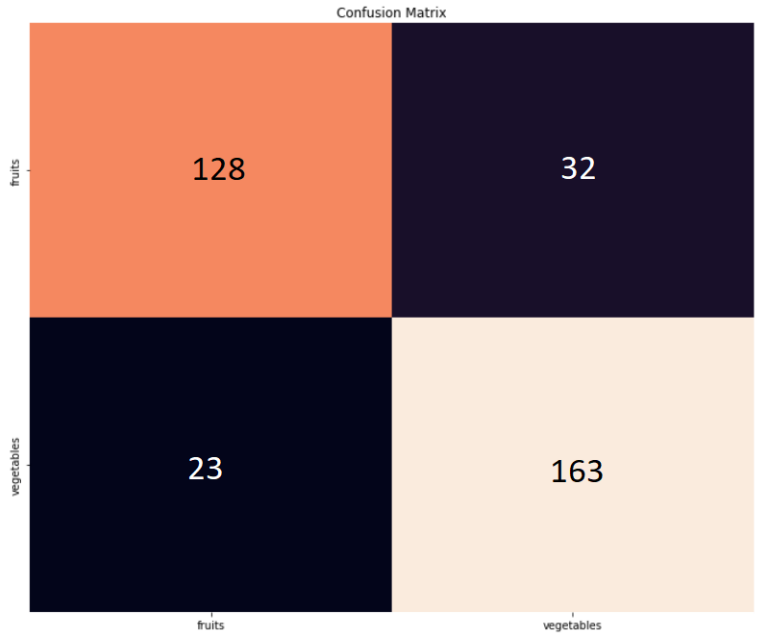


Figure 110: Confusion Matrix Ensemble Model Average method

accuracy			0.84	346
macro avg	0.84	0.84	0.84	346
weighted avg	0.84	0.84	0.84	346

Figure 111: Metrics Ensemble Model Average method

From the direct comparison of the two confusion matrix obtained, it is possible to observe a better accuracy for the ensemble associated with Majority Voting. Furthermore, this ensemble has increased the accuracy of about 5% compared to the best model of the Binary-Scratch section.

5.2 Multi-Label Scratch

The ensemble is composed by the models: Model 2aD5, Model 1_2aD5 and Model 2aD2 that were obtained during task 2.2 and they are used to perform Majority Voting and Average Voting techniques.

5.2.1 Majority

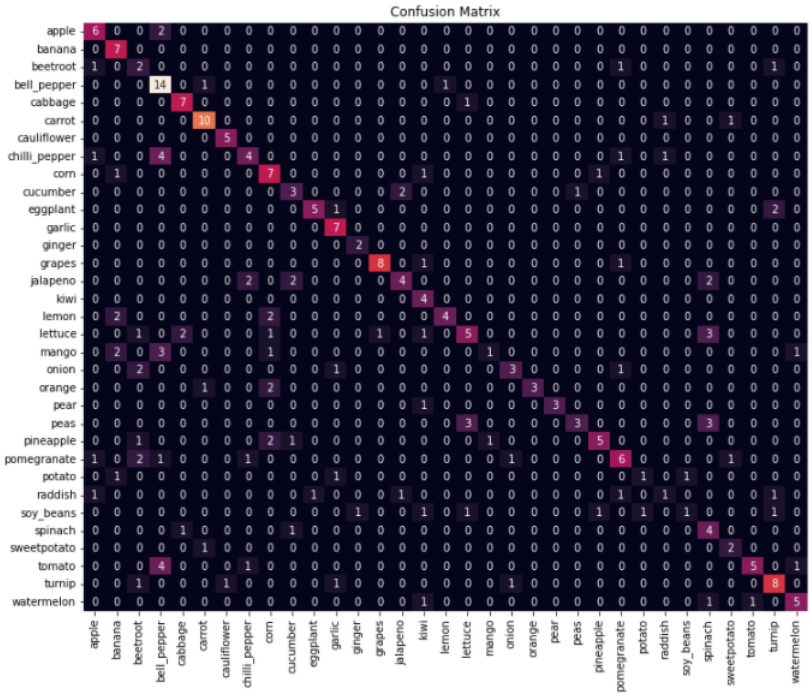


Figure 112: Confusion Matrix Ensemble Model Majority method

accuracy			0.58	265
macro avg	0.61	0.60	0.57	265
weighted avg	0.62	0.58	0.57	265

Figure 113: Metrics Ensemble Model Majority method

5.2.2 Average

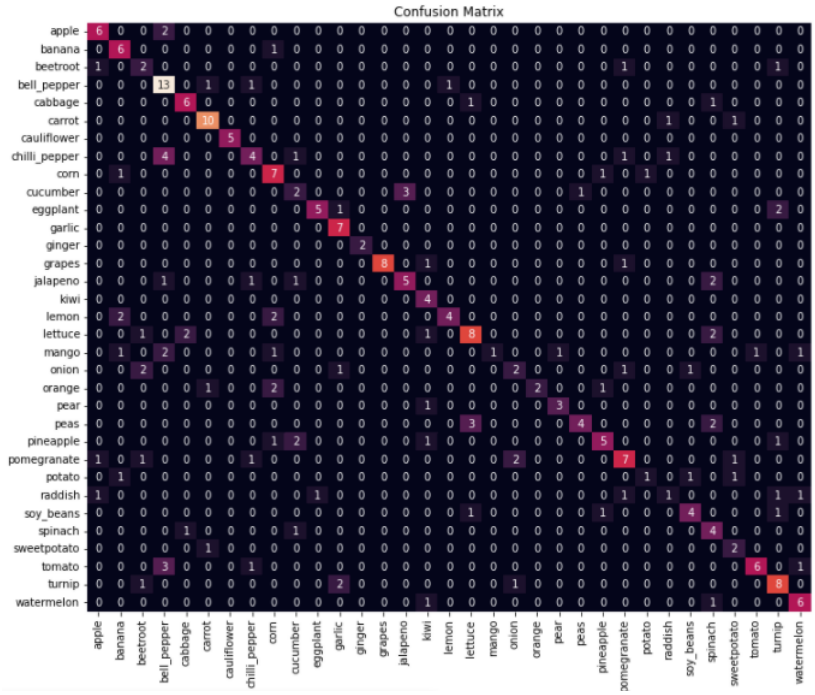


Figure 114: Confusion Matrix Ensemble Model Average method

accuracy			0.60	265
macro avg	0.64	0.61	0.59	265
weighted avg	0.65	0.60	0.59	265

Figure 115: Metrics Ensemble Model Average method

The Average Voting technique shows a better situation than the Majority Average one. The accuracy reached is better than 10% of the Model 2aD2's accuracy.

5.3 Binary Pretrained

The ensemble is composed by the best model of VGG16, ResNet50 and InceptionV3 networks. For the VGG16 network is selected 'Drop.0.2-FT', for InceptionV3 'Regular-FT' and for ResNet50 the model 'Drop0.2-FT'.

5.3.1 Majority

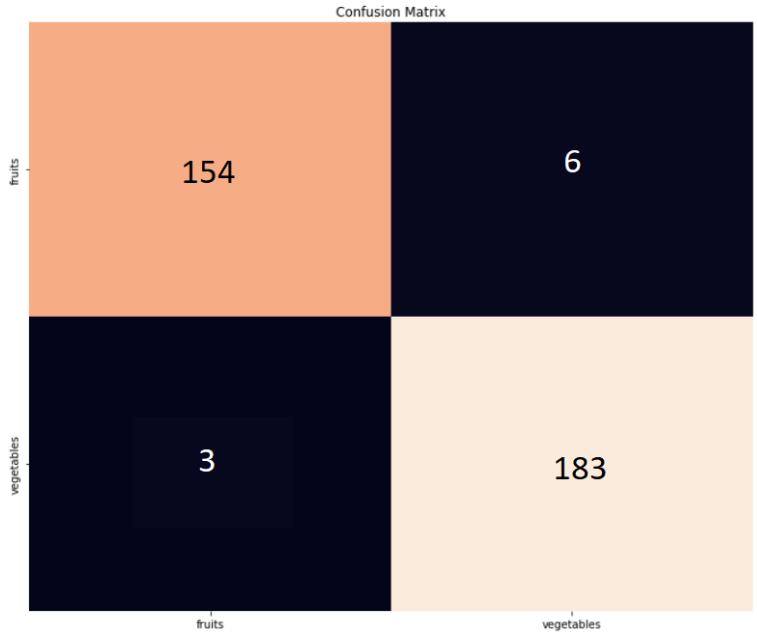


Figure 116: Confusion Matrix Ensemble Model Majority method

accuracy			0.97	346
macro avg	0.97	0.97	0.97	346
weighted avg	0.97	0.97	0.97	346

Figure 117: Metrics Ensemble Model Majority method

5.3.2 Average

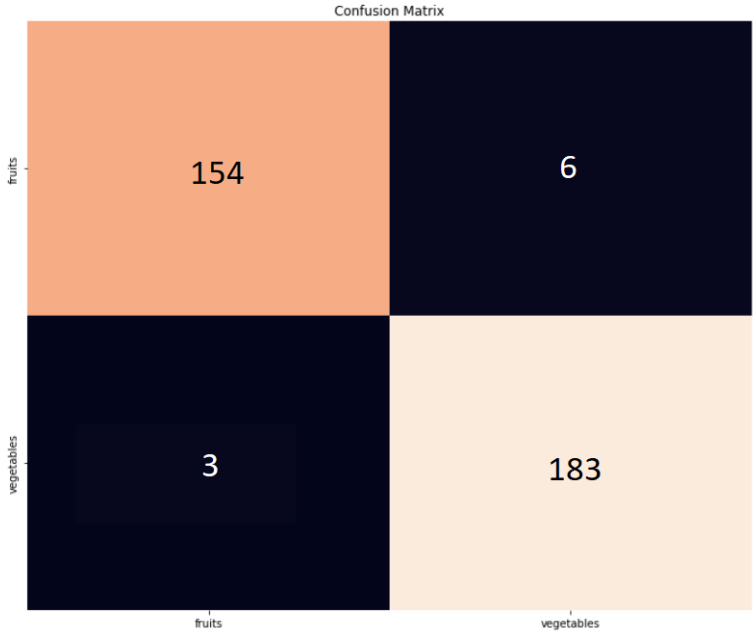


Figure 118: Confusion Matrix Ensemble Model Average method

accuracy			0.97	346
macro avg	0.97	0.97	0.97	346
weighted avg	0.97	0.97	0.97	346

Figure 119: Metrics Ensemble Model Average method

From the direct comparison of the two confusion matrix obtained, it is not possible to choose the best technique. Furthermore, both ensembles have increased the accuracy testing of about 2% compared to the best model of the Binary Pretrained section.

5.4 Multi-Label Pretrained

The ensemble is composed by the best model of VGG16, ResNet50 and InceptionV2 networks. For the VGG16 network is selected 'Regular-FT', for InceptionV3 'Augmented-FT' and for ResNet50 the model 'Base-FT'.

5.4.1 Majority

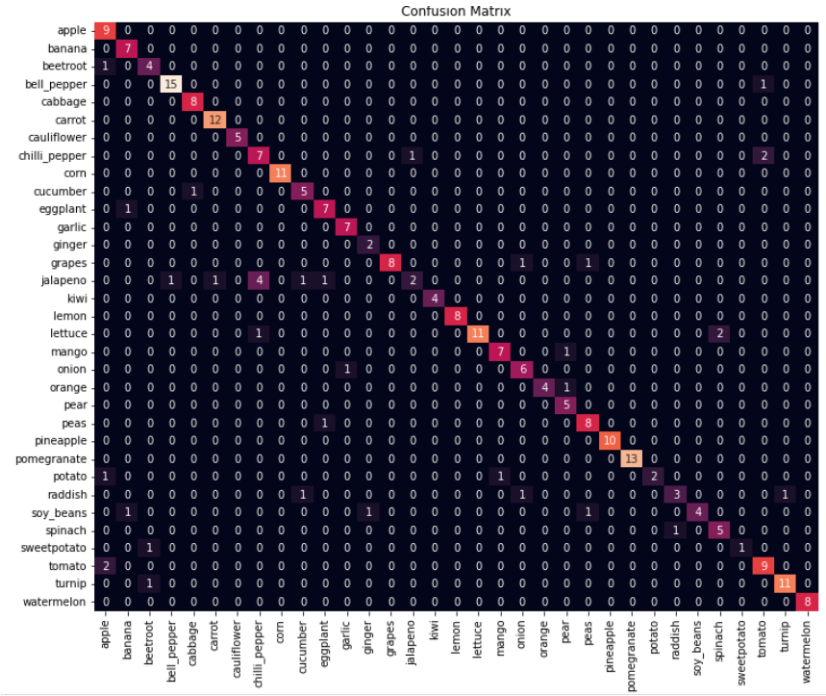


Figure 120: Confusion Matrix Ensemble Model Majority method

accuracy			0.86	265
macro avg	0.86	0.85	0.84	265
weighted avg	0.87	0.86	0.85	265

Figure 121: Metrics Ensemble Model Majority method

5.4.2 Average

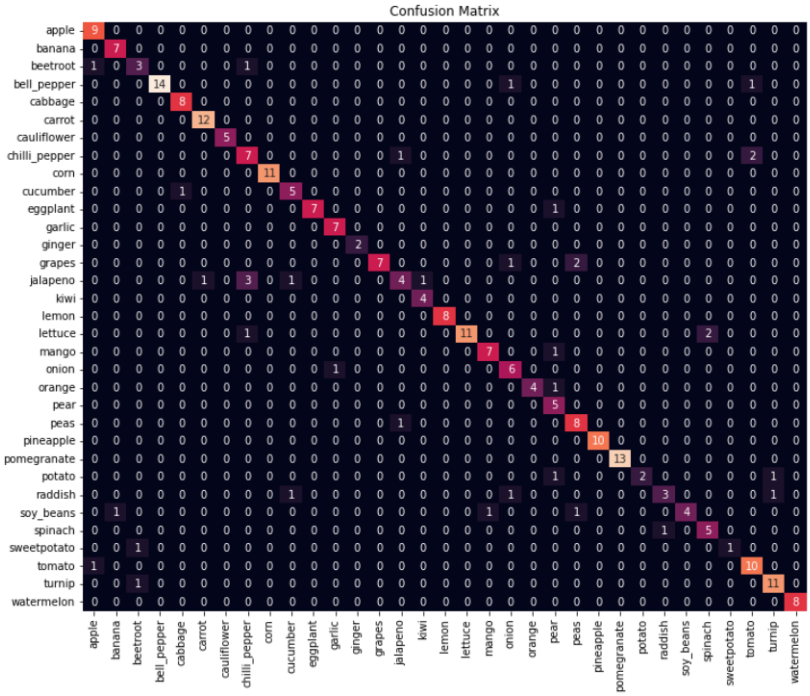


Figure 122: Confusion Matrix Ensemble Model Average method

accuracy			0.86	265
macro avg	0.87	0.85	0.84	265
weighted avg	0.88	0.86	0.86	265

Figure 123: Metrics Ensemble Model Average method

The average method performs slightly better than the Majority method. Furthermore it increases the F1-score of 5%-6% respect to the best model of the Multi-Label Pretrained section.

6 Conclusion

The goal we set ourselves was achieved for the binary classification of fruit and vegetables, while in the multi-class classification it was partially achieved. In particular, a good accuracy threshold of 84% was reached in the scratch binary classification, while 97% for pretrained networks. The high level of accuracy can be justified by the fact that the fruit and vegetable category contain very different features.
The scenario was different for the 33-class classification. The goals scored by scratch did not achieve good results, not even with the ensemble technique. A good result was obtained through the pretrained networks which by using the ensemble reached an accuracy of about 86%.
The substantial gap between multiclass scratch and pretrained can be justified by the fact that within the ImageNet dataset, through which the weights we used to load all the pretrained networks were obtained, there are classes in common with our dataset.
However, even in the case of pretrained networks, it should be noted that some classes have a lower prediction precision, for example in the confusion matrix we can see the problem of distinguishing the class chilli pepper and jalapeno, which have correspondingly precision 58% and 67%.
In order to further increase the accuracy, for the multiclass classification, it is suggested to increase the number of images for the highly correlated classes (chilli pepper and jalapeno, for example) without unbalancing the dataset too much.