HOMEWORK 1

IMAGE CLASSIFICATION

TEAM:	I_TRE_NEURONI
AUTHORS:	RICCARDO CAMPI, MATTEO BIANCHI, LEONARDO GALEAZZI
VERSION:	v1.0
PROFESSORS:	M. MATTEUCCI, G. BORACCHI, F. LATTARI, E. LOMURNO
DATE:	28/11/2021

SUMMARY

1. INTRODUCTION

This is the **first Homework** of the Artificial Neural Networks and Deep Learning course.

In this homework the groups are required to **classify images of leaves**, which are divided into categories according to the species of the plant to which they belong. Being a classification problem, given an image, the goal is to predict the correct class label.

Figure 1: an example of leaf images

2. DATASET

The dataset provided by the competition's promoters is a **folder containing 17 728 files**, grouped into several categories. In particular, there are **14 different types of leaves** with whom is possible to classify the images (Tomato, Orange, Soybean, Grape, Corn, Apple, Peach, Pepper, Potato, Strawberry, Cherry, Squash, Blueberry, Raspberry).

2.1 CLASS-IMBALANCE PROBLEM

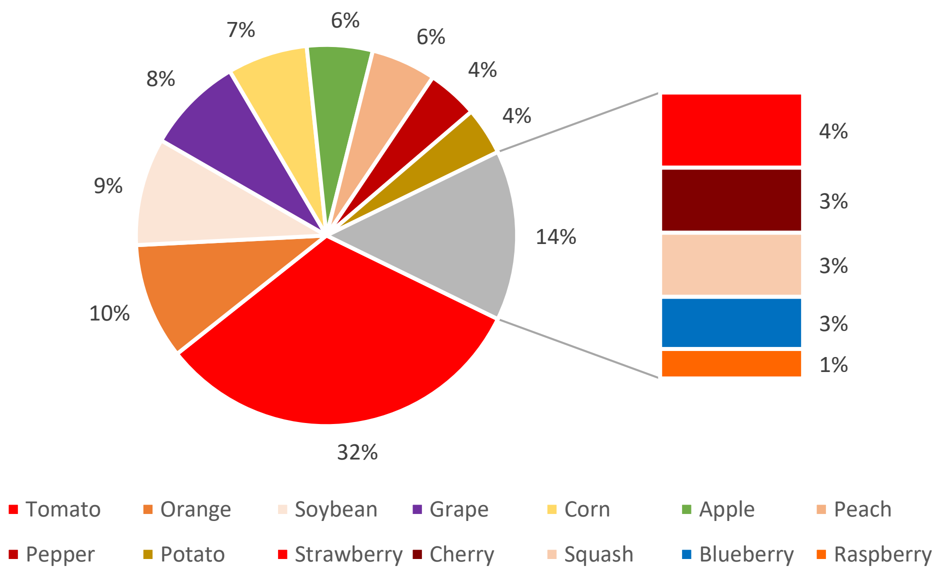


Table 1: class-imbalance problem

As shown in *Table 1: class-imbalance problem*, some classes contain many more images than the others. In particular, the sum of Tomato, Orange and Soybean represents more than half of the entire distribution.

This problem is known as **class-imbalance**. Due to this, the fitted model **tends to be biased** towards the majority class data, which leads to lower accuracy during the testing phase.

2.1.1 UNDER-SAMPLING

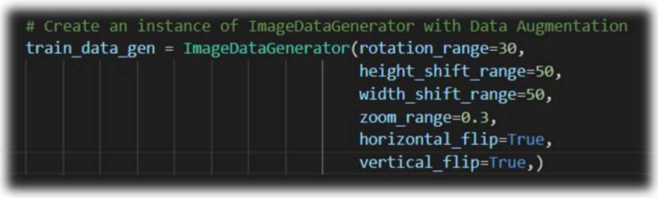
One of the most used techniques to **bring the required balance** in the data is called **under-sampling**. In particular, for this homework under-sampling was used to partially solve the problem by removing some files in larger classes.

2.2 IMAGE DATA AUGMENTATION

Image data augmentation is a technique that can be used to **artificially expand the size of a training dataset** by creating modified versions of images in the dataset.

Training models on more data can result in more skilful models, and the augmentation techniques can create **variations of the images** that can improve the ability of the fit models to **generalize** what they have learned to new images.

For this homework were used the 4 image data augmentation types. We decided to keep the basic augmentation settings given us in the lectures.



```
# Create an instance of ImageDataGenerator with Data Augmentation
train_data_gen = ImageDataGenerator(rotation_range=30,
                                    height_shift_range=50,
                                    width_shift_range=50,
                                    zoom_range=0.3,
                                    horizontal_flip=True,
                                    vertical_flip=True,)
```

Figure 2: ImageDataGenerator object with augmentation techniques

The code showed in *Figure 2: ImageDataGenerator object with augmentation techniques* was **used for all of the trained models**.

2.2 OBSERVATIONS

Rotation and shift will leave areas of the frame with no pixel data, and we chose to fill them with black pixels. For this reason, we left the default fill mode as the original images have a black background and the border (other fill modes would have given us augmented images too different from the originals or too unrealistic).

3 TRAINING

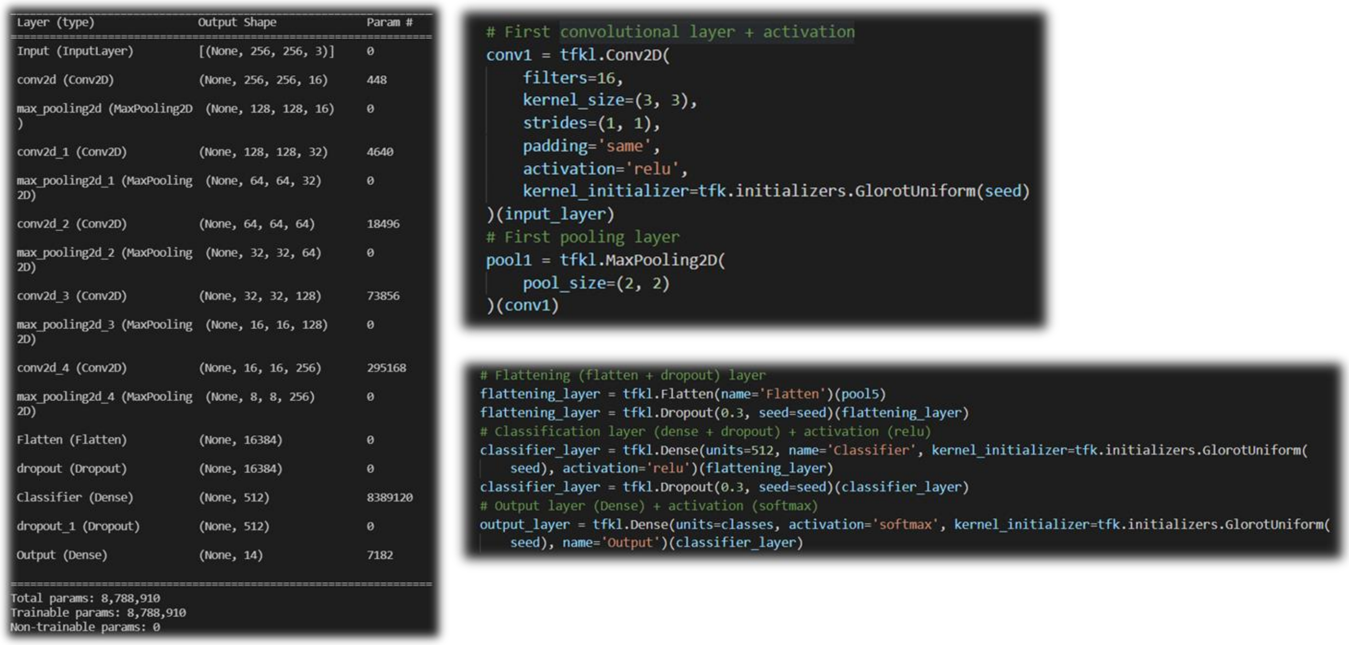
In this chapter are listed all the training experiments we made.

All the model used **early-stopping to avoid overfitting** as much as possible. For all models it was kept a patience of 10 epochs and the best epoch was kept. Except for the last one (InceptionResNetV2) we monitored for the early-stopping the **Validation Loss** which was chosen to be the Categorical Cross-entropy. We used the Adam optimization function in all our trained models with default parameters for Transfer Learning and Simple CNN, while for Fine-Tuning we set the Learning Rate to 10^{-4} in order to have a more precise evaluation at the end.

3.1 SIMPLE CNN

The first net we tried was designed as a **simple convolutional neural net**, that is, a convolutional part followed by a fully connected one. The dataset used for this first model was the original one, so all the images have been kept and **no under-sampling was applied** to solve the class-imbalance problem.

In particular, we used **5 convolutional layers** (+ activation + pooling) followed by a flatten layer, a classification layer, and an output layer.



```
Layer (type)                Output Shape              Param #
=================================================================
Input (InputLayer)          [(None, 256, 256, 3)]     0

conv2d (Conv2D)             (None, 256, 256, 16)      448

max_pooling2d (MaxPooling2D  (None, 128, 128, 16)     0
)

conv2d_1 (Conv2D)           (None, 128, 128, 32)      4640

max_pooling2d_1 (MaxPooling  (None, 64, 64, 32)       0
2D)

conv2d_2 (Conv2D)           (None, 64, 64, 64)        18496

max_pooling2d_2 (MaxPooling  (None, 32, 32, 64)       0
2D)

conv2d_3 (Conv2D)           (None, 32, 32, 128)       73856

max_pooling2d_3 (MaxPooling  (None, 16, 16, 128)      0
2D)

conv2d_4 (Conv2D)           (None, 16, 16, 256)       295168

max_pooling2d_4 (MaxPooling  (None, 8, 8, 256)        0
2D)

Flatten (Flatten)           (None, 16384)             0

dropout (Dropout)           (None, 16384)             0

Classifier (Dense)          (None, 512)               8389120

dropout_1 (Dropout)         (None, 512)               0

Output (Dense)              (None, 14)                7182

=================================================================
Total params: 8,788,910
Trainable params: 8,788,910
Non-trainable params: 0
```

```python
# First convolutional layer + activation
conv1 = tfkl.Conv2D(
    filters=16,
    kernel_size=(3, 3),
    strides=(1, 1),
    padding='same',
    activation='relu',
    kernel_initializer=tfk.initializers.GlorotUniform(seed)
)(input_layer)
# First pooling layer
pool1 = tfkl.MaxPooling2D(
    pool_size=(2, 2)
)(conv1)
```

```python
# Flattening (flatten + dropout) layer
flattening_layer = tfkl.Flatten(name='Flatten')(pool5)
flattening_layer = tfkl.Dropout(0.3, seed=seed)(flattening_layer)
# Classification layer (dense + dropout) + activation (relu)
classifier_layer = tfkl.Dense(units=512, name='Classifier', kernel_initializer=tfk.initializers.GlorotUniform(
    seed), activation='relu')(flattening_layer)
classifier_layer = tfkl.Dropout(0.3, seed=seed)(classifier_layer)
# Output layer (Dense) + activation (softmax)
output_layer = tfkl.Dense(units=classes, activation='softmax', kernel_initializer=tfk.initializers.GlorotUniform(
    seed), name='Output')(classifier_layer)
```

Figure 3: Simple CNN Summary and layers

The metrics obtained by the fitted model on the validation set are:

- **Accuracy**: **92.06%**
- **Precision**: **91.18%**
- **Recall**: **89.89%**
- **F1**: **90.16%**

We obtained an **Accuracy** of **56.22%** on the hidden test set during the Development Phase.

The discrepancy between the Validation Accuracy and the Accuracy on the test set made us think that the model was too simple to cope with the given problem. For this reason, we decided to experiment some other training paradigms.

3.1.1 ACCURACY

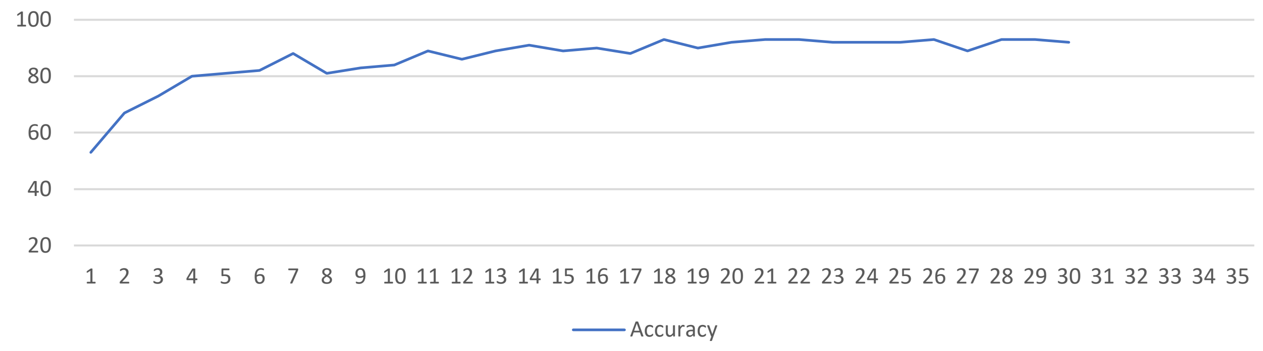


Table 2: Accuracy of Simple CNN

3.1.2 CATEGORICAL CROSS-ENTROPY

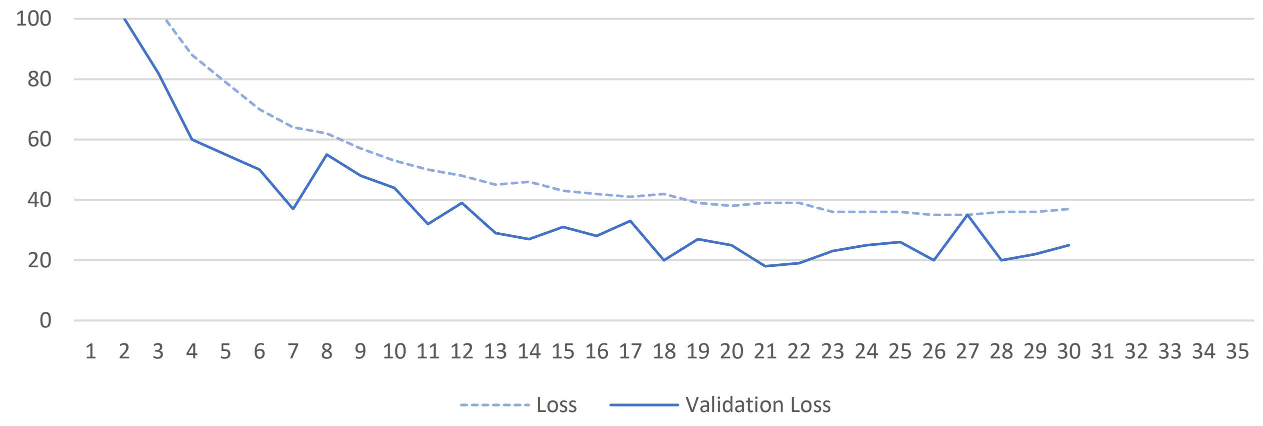


Table 3: Categorical Cross-entropy of Simple CNN

3.1.3 CONFUSION MATRIX

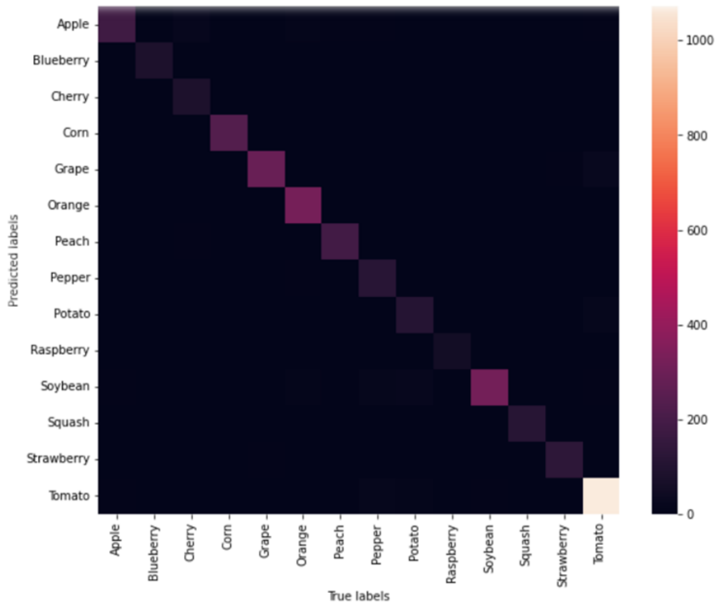
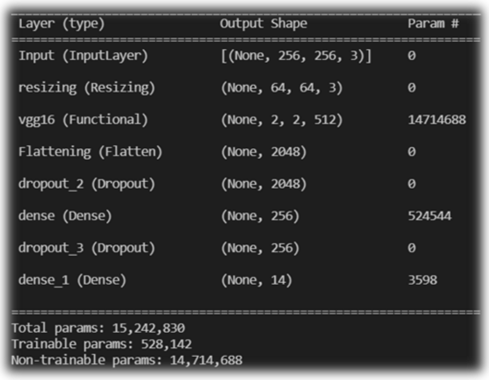


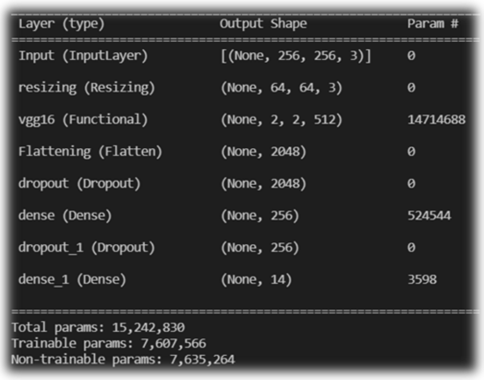
Figure 4: Confusion Matrix of Simple CNN

3.2 VGG16

Our second model is a **VGG16 based CNN**. We used VGG as supernet and we fine-tuned the fully connected part to it. The intention was to use the already trained features of this supernet to help our model to better generalize.



```
Layer (type)                Output Shape              Param #
=================================================================
Input (InputLayer)          [(None, 256, 256, 3)]     0

resizing (Resizing)         (None, 64, 64, 3)         0

vgg16 (Functional)          (None, 2, 2, 512)         14714688

Flattening (Flatten)        (None, 2048)              0

dropout_2 (Dropout)         (None, 2048)              0

dense (Dense)               (None, 256)               524544

dropout_3 (Dropout)         (None, 256)               0

dense_1 (Dense)             (None, 14)                3598

=================================================================
Total params: 15,242,830
Trainable params: 528,142
Non-trainable params: 14,714,688
```



```
Layer (type)                Output Shape              Param #
=================================================================
Input (InputLayer)          [(None, 256, 256, 3)]     0

resizing (Resizing)         (None, 64, 64, 3)         0

vgg16 (Functional)          (None, 2, 2, 512)         14714688

Flattening (Flatten)        (None, 2048)              0

dropout (Dropout)           (None, 2048)              0

dense (Dense)               (None, 256)               524544

dropout_1 (Dropout)         (None, 256)               0

dense_1 (Dense)             (None, 14)                3598

=================================================================
Total params: 15,242,830
Trainable params: 7,607,566
Non-trainable params: 7,635,264
```

Figure 5: Transfer Learning and Fine-Tuning Summary of VGG16

Metrics obtained after the **Transfer Learning** phase by the fitted model on the validation set:

- **Accuracy**: **87.59%**
- **F1**: **86.27%**

Metrics after the **Fine-Tuning** on the validation set:

- **Accuracy**: **96.77%**
- **F1**: **96.84%**

We obtained an **Accuracy** of **63.77%** on the hidden test set by submitting this model.

3.2.1 ACCURACY

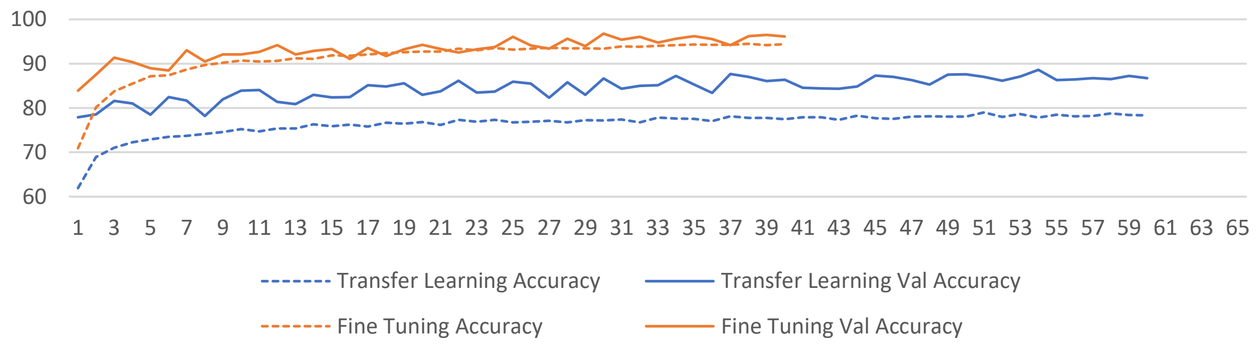


Table 4: Accuracy of VGG16 CNN

3.2.2 CATEGORICAL CROSS-ENTROPY

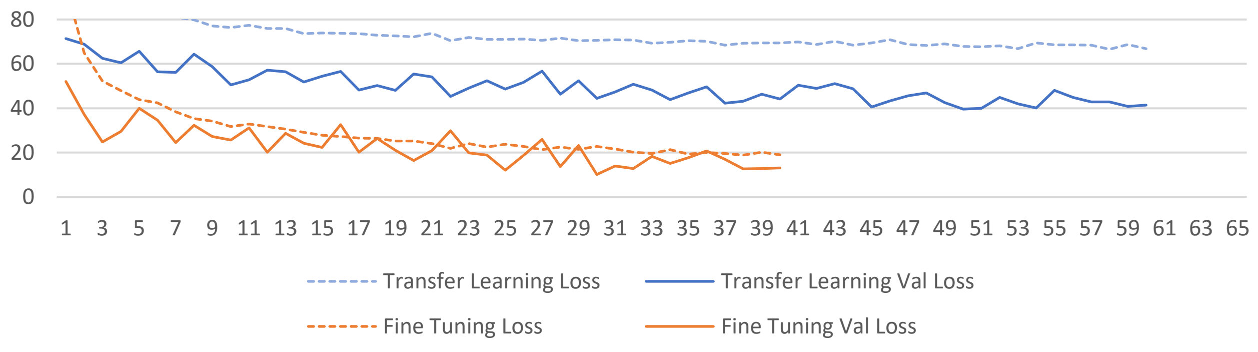


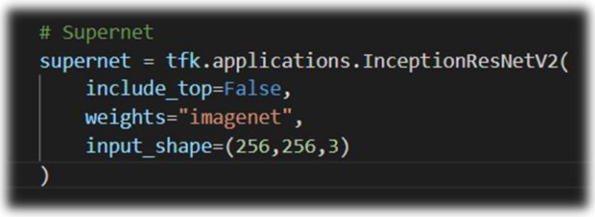
Table 5: Categorical Cross-entropy of VGG16 CNN

3.3 INCEPTION RESNET V2

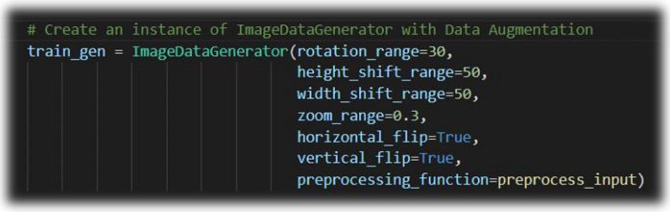
The third model we tried was **InceptionResNetV2 supernet** based.

This type of Transfer Learning, like the one based on VGG16, uses a supernet to optimize training and to increase the final accuracy. To better exploit the power of this supernet, we decided **not to resize the input images**, instead we chose to pre-process them using the standard Inception pre-processing function.

For this model, we used **Validation Accuracy** as **early-stopping parameter**.

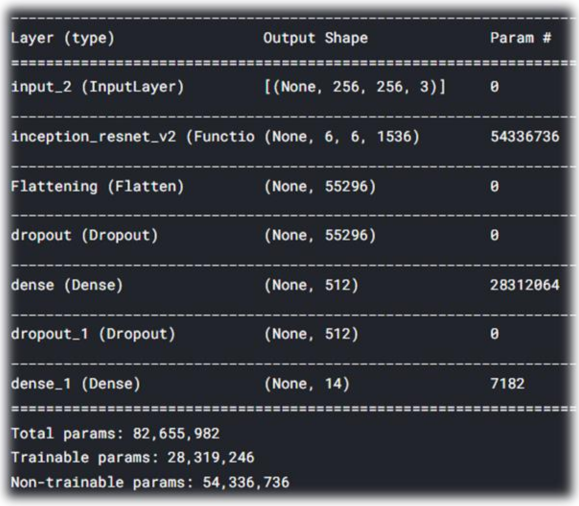


```
# Supernet
supernet = tfk.applications.InceptionResNetV2(
    include_top=False,
    weights="imagenet",
    input_shape=(256,256,3)
)
```

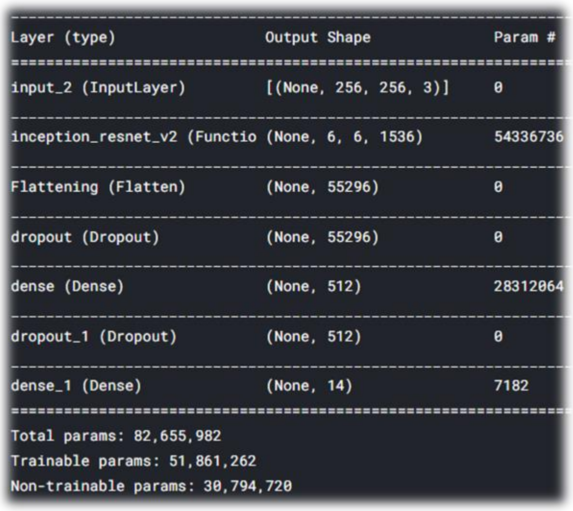


```
# Create an instance of ImageDataGenerator with Data Augmentation
train_gen = ImageDataGenerator(rotation_range=30,
                               height_shift_range=50,
                               width_shift_range=50,
                               zoom_range=0.3,
                               horizontal_flip=True,
                               vertical_flip=True,
                               preprocessing_function=preprocess_input)
```

Figure 6: InceptionResNetV2 supernet and augmentation with proprietary pre-processing function



```
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_2 (InputLayer)        [(None, 256, 256, 3)]     0

_________________________________________________________________
inception_resnet_v2 (Functio (None, 6, 6, 1536)       54336736
_________________________________________________________________
Flattening (Flatten)        (None, 55296)             0
_________________________________________________________________
dropout (Dropout)           (None, 55296)             0
_________________________________________________________________
dense (Dense)               (None, 512)               28312064
_________________________________________________________________
dropout_1 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_1 (Dense)             (None, 14)                7182
=================================================================
Total params: 82,655,982
Trainable params: 28,319,246
Non-trainable params: 54,336,736
```



```
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_2 (InputLayer)        [(None, 256, 256, 3)]     0
_________________________________________________________________
inception_resnet_v2 (Functio (None, 6, 6, 1536)       54336736
_________________________________________________________________
Flattening (Flatten)        (None, 55296)             0
_________________________________________________________________
dropout (Dropout)           (None, 55296)             0
_________________________________________________________________
dense (Dense)               (None, 512)               28312064
_________________________________________________________________
dropout_1 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_1 (Dense)             (None, 14)                7182
=================================================================
Total params: 82,655,982
Trainable params: 51,861,262
Non-trainable params: 30,794,720
```

Figure 7: Transfer Learning and Fine-Tuning Summary of InceptionResNetV2

In this case, we used an **under-sampled dataset**, trying to solve partially the class-imbalance problem.

The metrics obtained by the fitted model (Fine-Tuned) on the validation set are:

• **Accuracy**:	**99.47%**
• **Precision**:	**99.44%**
• **Recall**:	**99.37%**
• **F1**:	**99.40%**
• **Loss**:	**1.42%**

We obtained an **Accuracy** of **89.43%** on the hidden test set during the Development Phase and an **Accuracy** of **88.30%** during the Final Phase.

3.3.1 ACCURACY

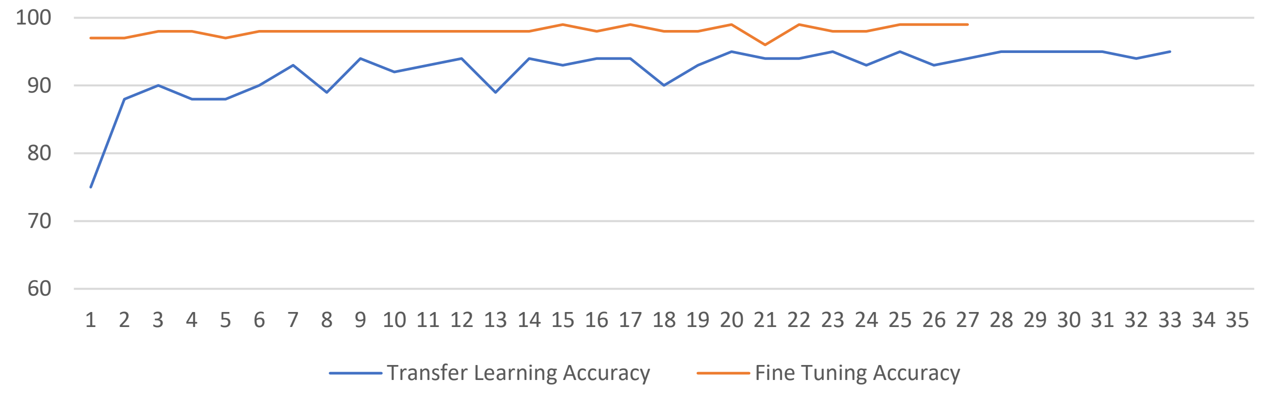


Table 6: Accuracy of InceptionResNetV2

3.3.2 CATEGORICAL CROSS-ENTROPY

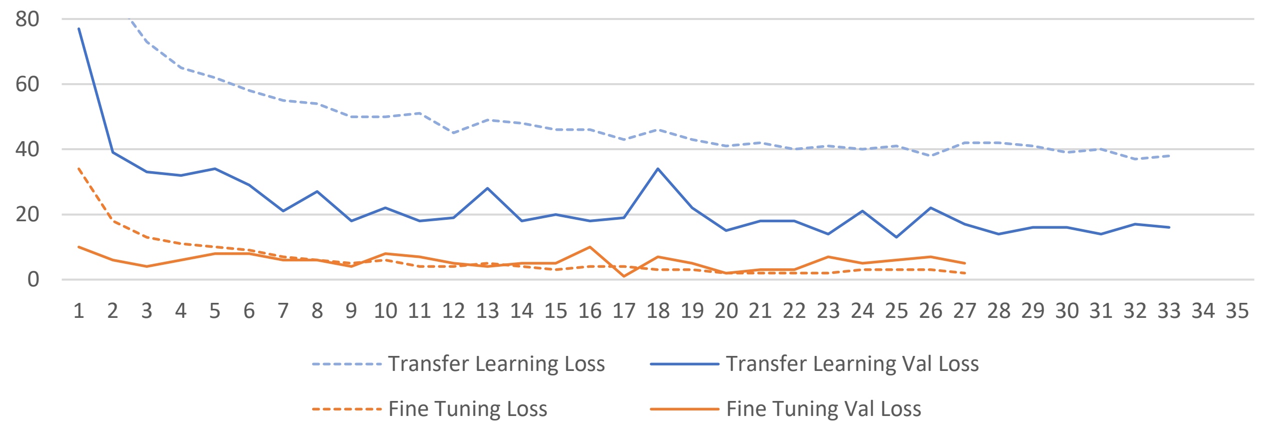


Table 7: Categorical Cross-entropy of InceptionResNetV2

3.3.3 CONFUSION MATRIX

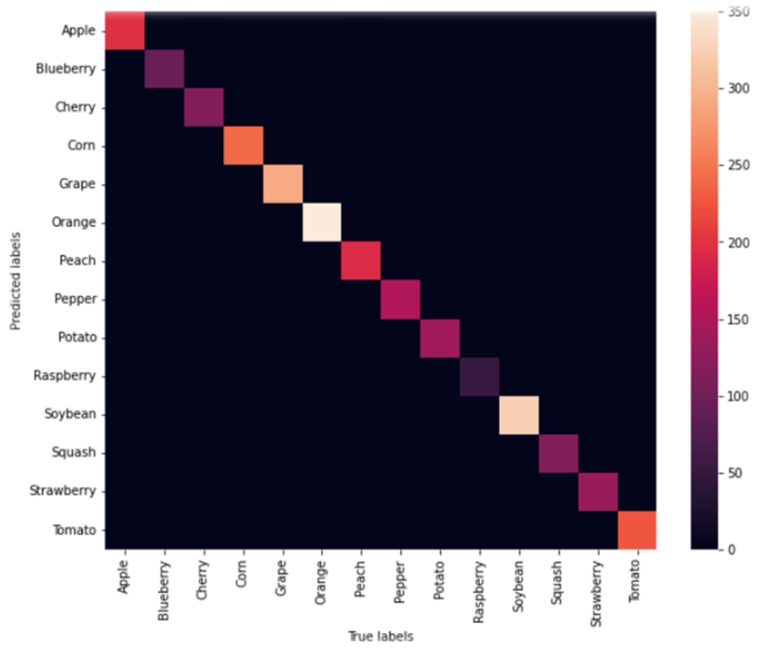


Figure 8: Confusion Matrix of InceptionResNetV2

4 ENSEMBLE

To reduce the variance of our final neural network model we decided to **ensemble multiple models** and let the **ensemble give the prediction**. This has been done by **dividing** the already under-sampled dataset **in 5 different splits**.

In order to obtain the splits, we used the Sklearn class **KFold**. Each set has each class divided differently among the 5 datasets obtained in this way, so that for each class the validation part does not overlap among the five. The way the ensemble works is to compute the argmax of the sum of the predictions of each model, at first reducing vectors of dimension (1, 1, 14) to a single one of the same dimensions. Then the argmax of this is taken to obtain the predicted class for the image.

4.1 VGG16 ENSEMBLE

The results we obtained ensembling two or more VGG16 models, on the hidden test set, were actually superior to the one obtained by the single model.

The final ensemble consists of 5 models where **Accuracy** scored ~**75%** against the **63.77%** of the single original model.

4.2 INCEPTION RESNET V2 ENSEMBLE

Starting from a model having **89.43%** we incrementally ensembled models in number obtaining an **Accuracy** on the hidden test set of ~**91%** with an ensemble of 2 models and ~**93%** with an ensemble of 3.

Our final and best submitted model was at last obtained ensembling only the best 3 models in terms of Validation Accuracy among the five trained mentioned. This gave us an **Accuracy** in the Development Phase of **93.58%** and **92.83%** in the Final Phase.

Ensembled submission with more than 3 models would have resulted in **exceeding the maximum run time**, so they were discarded.

5 CONCLUSIONS

To successfully execute our code, it is necessary to download the split and under-sampled dataset: riccamper/ANN-Homework-1 (github.com).

We mainly used the lecture material, but we also looked at the following articles and documentation:

- How to Develop an Ensemble of Deep Learning Models in Keras (machinelearningmastery.com)
- 7 Techniques to Handle Imbalanced Data - KDnuggets
- Keras API reference
- Neural Network Based Undersampling Techniques (arxiv.org)