POLITECNICO
MILANO 1863

HOMEWORK 1

IMAGE CLASSIFICATION

GROUP: 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

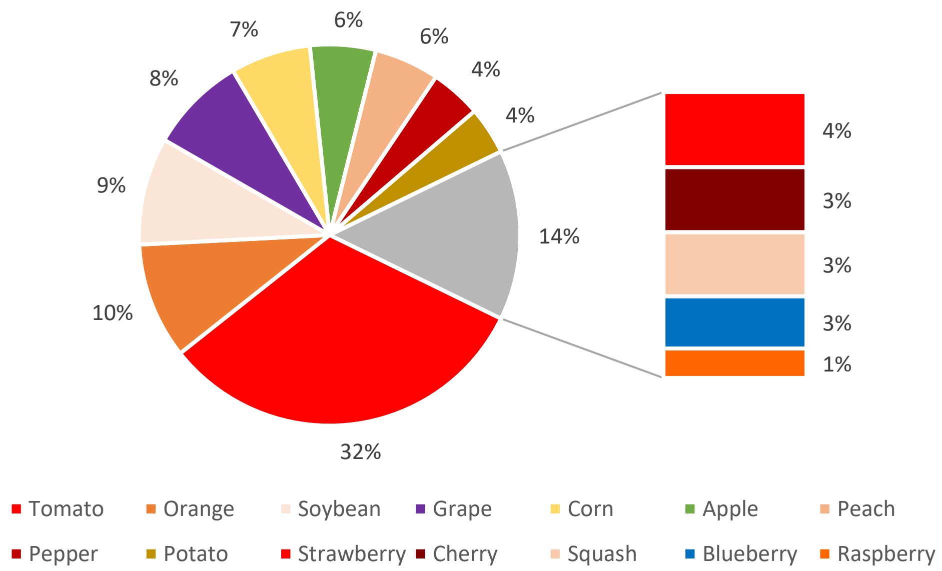

Table 1: class-imbalance problem

As is shown in *Table 1: class-imbalance problem*, some classes contain much 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.

```python
# 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.

2.2.3 AUGMENTATION TECHNIQUES

- **Horizontal and vertical shift** augmentation: a shift to an image means moving all pixels of the image in one direction, such as horizontally or vertically, while keeping the image dimensions the same.
- **Horizontal and vertical flip** augmentation: an image flip means reversing the rows or columns of pixels in the case of a vertical or horizontal flip respectively.
- **Random rotation** augmentation: a rotation augmentation randomly rotates the image clockwise by a given number of degrees from 0 to 360.
- **Random zoom** augmentation: a zoom augmentation randomly zooms the image in and either adds new pixel values around the image or interpolates pixel values respectively.

2.3 SOME OBSERVATION

Rotation and shift will leave areas of the frame with no pixel data, that were **filled with black pixels**. In fact, we left the default fill mode as the original images have a black background and the border, so other fill modes would have given us augmented images too different to the originals.

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 remained for all Categorical Cross-entropy.

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 (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_2 (Conv2D)	(None, 64, 64, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 64)	0
conv2d_3 (Conv2D)	(None, 32, 32, 128)	73856
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 128)	0
conv2d_4 (Conv2D)	(None, 16, 16, 256)	295168
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 256)	0
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.

3.1.1 ACCURACY

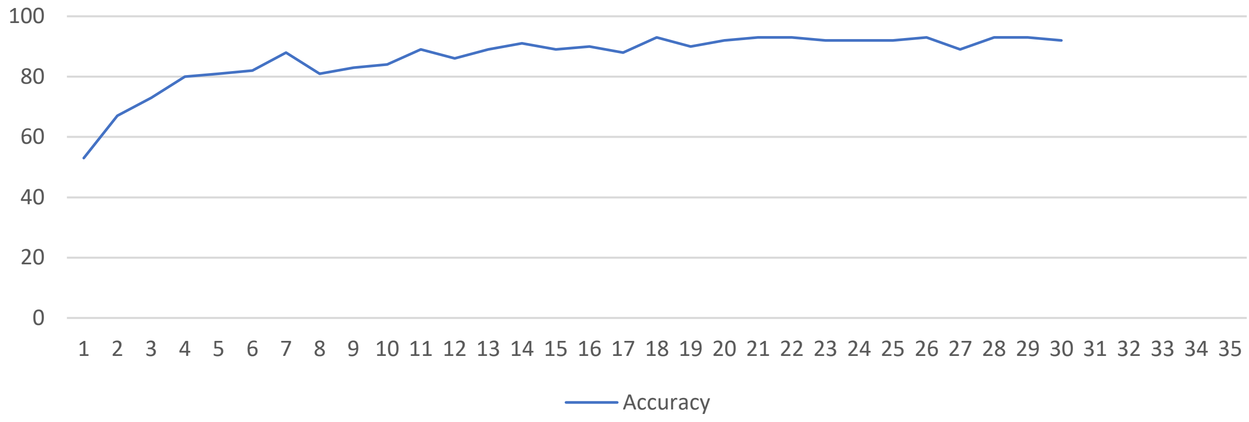

Table 2: Accuracy of Simple CNN

3.1.2 CATEGORICAL CROSS-ENTROPY

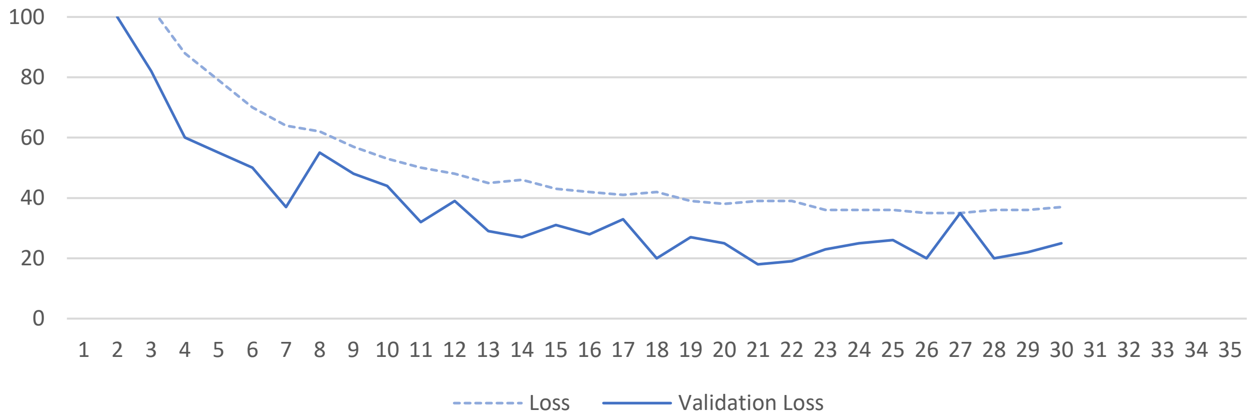

Table 3: Categorical Cross-entropy of Simple CNN

3.1.3 CONFUSION MATRIX

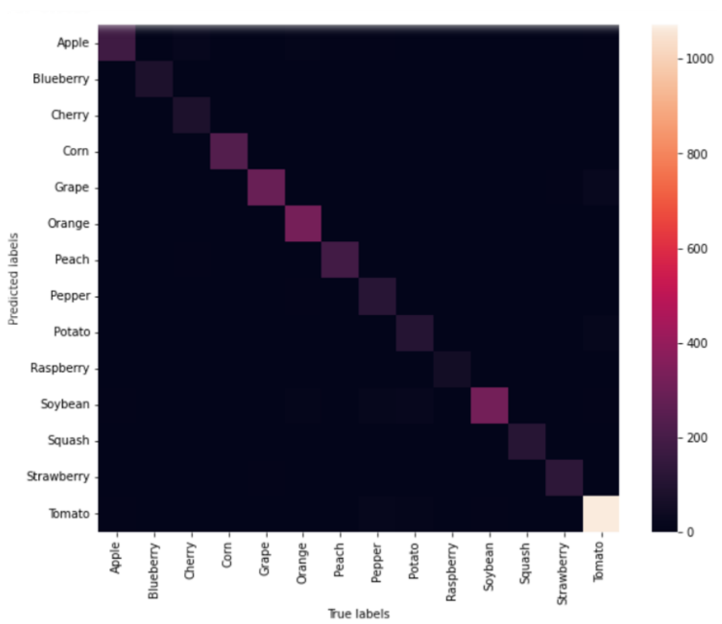

Figure 4: Confusion Matrix of Simple CNN

3.2 VGG16

Our second model is a VGG16 based CNN. We used VGG as supernet and 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 be more precise in its generalization ability.

```
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

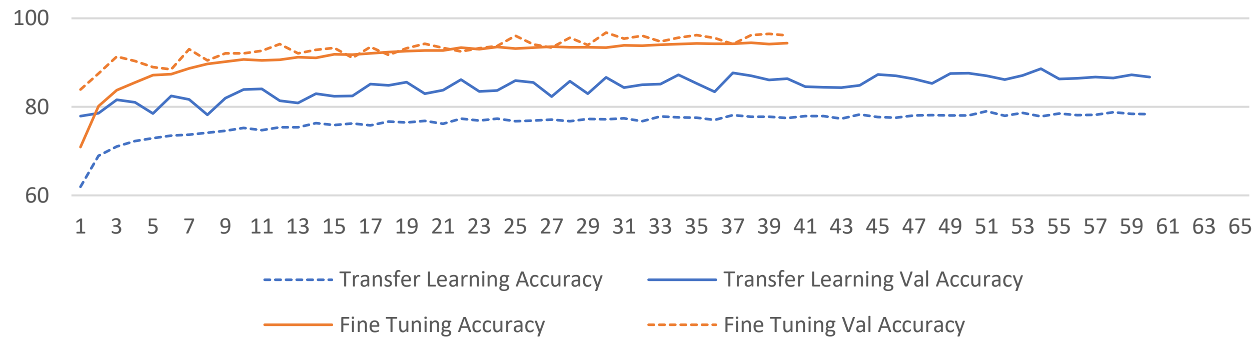

Table 4: Accuracy of VGG16 CNN

3.2.2 CATEGORICAL CROSS-ENTROPY

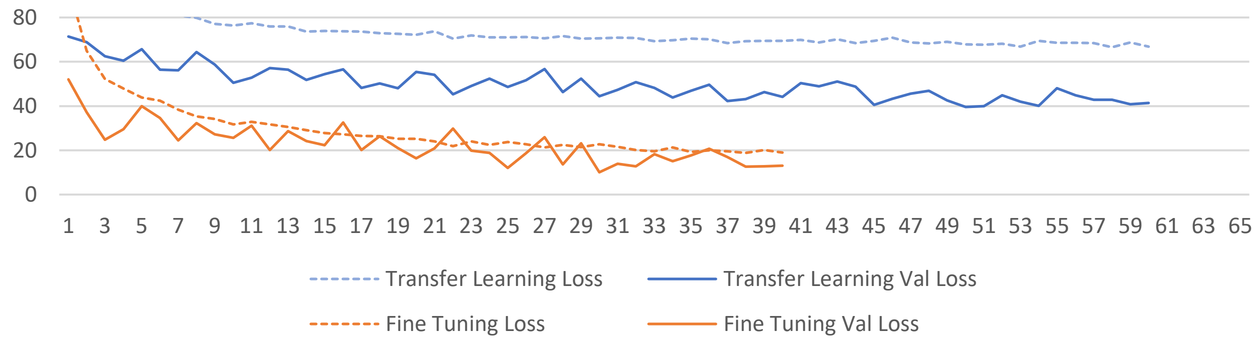

Table 5: Categorical Cross-entropy of VGG16 CNN

3.3 INCEPTION RESNET V2

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

This type of Transfer Learning, such as that 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**, which are still **pre-processed by 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 on the validation set are:

- **Accuracy**: **92.06%**
- **F1**: **90.16%**

We obtained an **Accuracy** of **89.43%** on the hidden test set.

3.3.1 ACCURACY

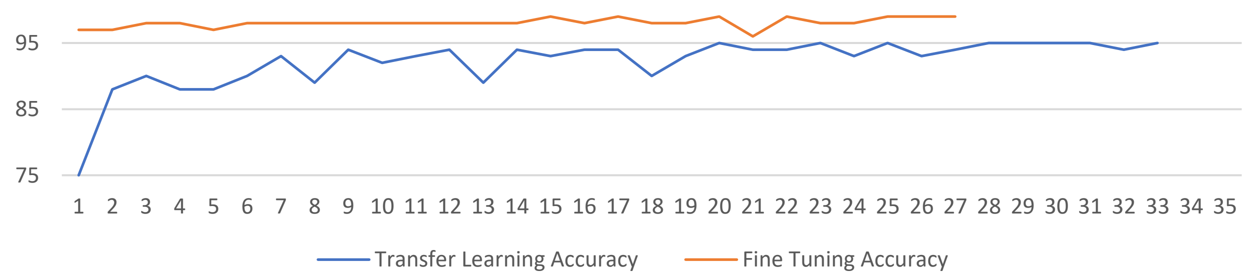

Table 6: Accuracy of InceptionResNetV2

3.3.2 CATEGORICAL CROSS-ENTROPY

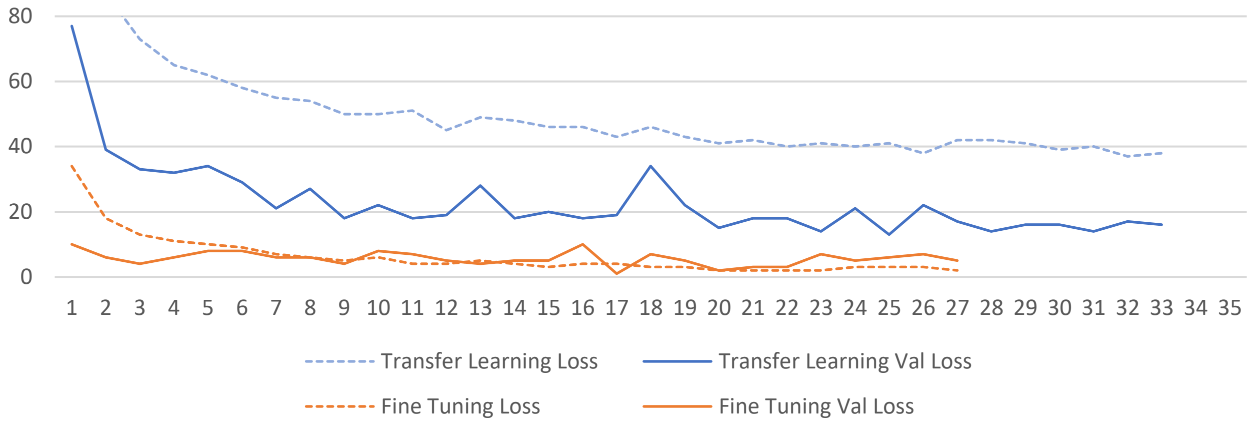

Table 7: Categorical Cross-entropy of InceptionResNetV2

3.3.3 CONFUSION MATRIX

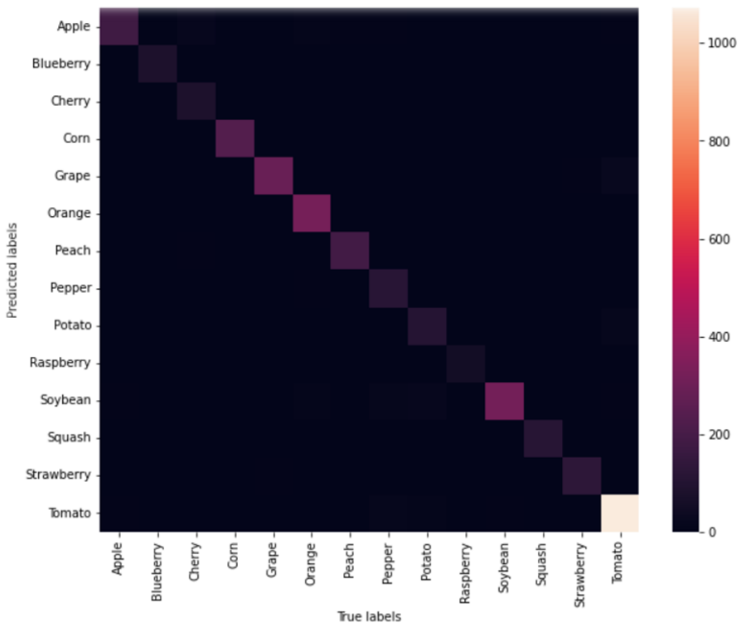

Figure 8: Confusion Matrix of InceptionResNetV2

4 ENSEMBLE