Report

First phase (Mean accuracy on test set: 24%)

Improvements

As the first model, we decided to start building a very simple model without the optimizations seen in the lectures. Specifically, we did not include any possible improvement as *early stopping*, *weight decay*, *data augmentation* or other techniques. Our objective was to build the most basic model possible. We only normalize each of them dividing their value by 255 in the notebook and in the model.py file. In the model compile we used *Categorical_Crossentropy and Adam optimizer.*

Model, Dataset and Results

The structure of the model was very similar to the one described in the last exercise. So we considered five *Convolution2D* and *MaxPooling2D* layers, then a *Flatten* layer followed by a *Dense* layer composed of 512 units. At the end a *Dense* Layer with 14 units was added.
Since the beginning, we noticed a remarkable imbalance between the classes: specifically the *Tomato* class included too many samples with respect to the other classes.
As we expected, the model overfitted the data given and the output on the test set was very bad (mean accuracy: 0.24).

Second phase (Mean accuracy on test set: 46%)

Improvements

In our second model we used *data augmentation* of *ImageDataGenerator:* zoom out/in, rotation, horizontal and vertical shift, modified brightness, flip with nearest fill mode. We also used a *callback function* that prevents overfitting thanks to *early stopping*, to keep track of our network learning and to save periodically progress with *checkpoints*.
Here we also used the following formula to calculate the weights for each class:

wj = n_samples / (n_classes * n_samplesj)

Where:

- wj: is the weight for each class
- n_samples: is the total number of samples
- n_classes: is the total number of unique classes (14 in our specific case)
- n_samplesj: is the total number of rows for each class.

The parameter *class_weight* of *model.fit* gives more importance to some samples than others: given those weights we tried to balance the learning process of each class considering the unbalanced dataset.

Model

In this second step we used a slightly different model, here we had:
five *Convolution2D* and *MaxPooling2D* layers, as above, then a *Flatten* layer, a *Dropout* layer followed by a *Dense* layer and another *Dropout* layer, to conclude with the last *Dense* layer made up of 14 units. We used a *Glorot Uniform Kernel Initializer.*

Dataset and Results

The dataset we used here to train our NN was the same as the one above but enlarged using the *Data Augmentation* technique.
Our results in this second submission improved a little bit on the test set (mean accuracy: 0,46).

Third Phase (Mean accuracy on test set: 60%)

Improvements

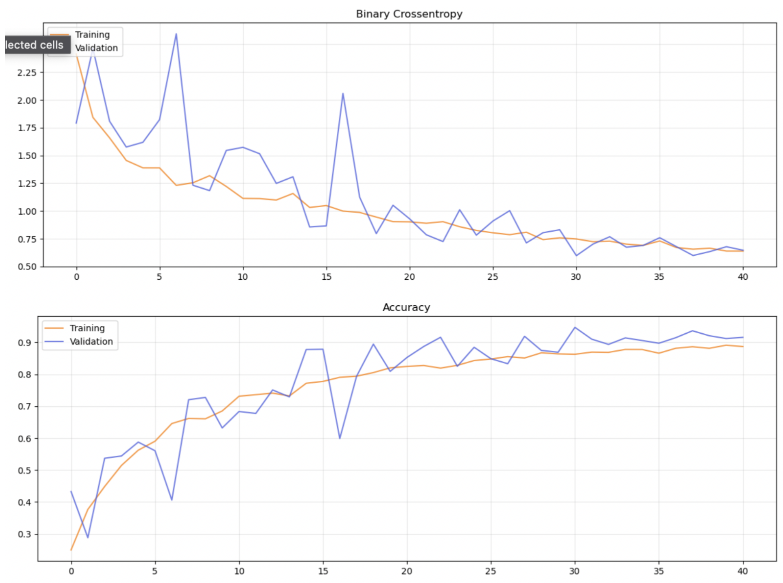
Starting from the previous model we thought about some new layers and relative parameters that could increase the performances. Specifically we added:

- A *Gaussian* Layer Noise at the beginning of the model in order to make it more robust
- *Batch Normalizer* Layer between *Convolutional* Layers and *Dense* Layers
- A specific *Learning Rate*
- *Kernel Regularizers* in the Dense Layers (*L2 regularization penalty, Weight Decay*)

Models and result

In that specific phase we tried very different model shapes in order to find the best one.

- The first one involved only one *Dense* layer before the output.
- The second one introduced two *Dense* layers in the middle: the first one made of 128 units and a second one with 64 units.
- We also substituted the *Flatten* layer with the *GAP* one even if this does not significantly change the performance.
- The last one was designed following this [*scientific paper*]() designed for the plant classification that provided similar results (attached it in the email).



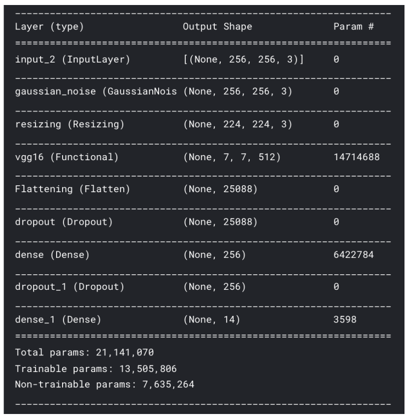
Fourth Phase (Mean accuracy on test set: 85%)

Improvements

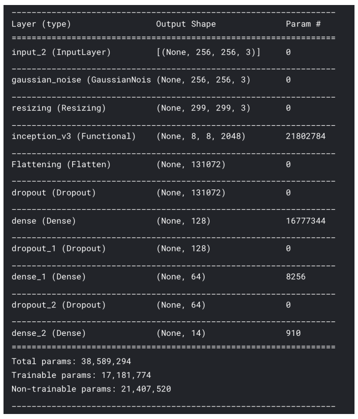
Finally, we used *Transfer Learning* to exploit knowledge gained by other existing models that have achieved state-of-the-art results for image classification tasks. We compared *InceptionV3, Resnet50, Xception and VGG16*. We focused on *VGG16* as it seemed to be the most promising as suggested by this [article of ITAT 2020](). It allowed us to achieve more than 85% accuracy on the test set in the first stage of the challenge. We did several experiments using as base network different numbers of layers. From 11 to 16, the best one appeared to be 13. We *fine-tuned* the remaining together with the fully connected final part of the model. Also here we experiment different solutions in terms of number of dense layers and neurons for each layer. We always used the preprocess_input function.

Model

Comparison between *VGG16* (left) and *InceptionV3* (right) models:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 256, 256, 3)]     0
_________________________________________________________________
gaussian_noise (GaussianNois (None, 256, 256, 3)       0
_________________________________________________________________
resizing (Resizing)          (None, 224, 224, 3)       0
_________________________________________________________________
vgg16 (Functional)           (None, 7, 7, 512)         14714688
_________________________________________________________________
Flattening (Flatten)         (None, 25088)             0
_________________________________________________________________
dropout (Dropout)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 256)               6422784
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 14)                3598
=================================================================
Total params: 21,141,070
Trainable params: 13,505,806
Non-trainable params: 7,635,264
_________________________________________________________________
```



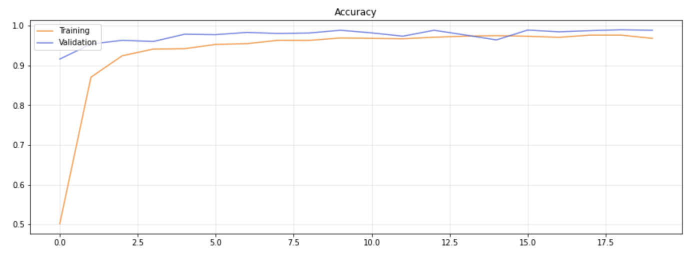
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 256, 256, 3)]     0
_________________________________________________________________
gaussian_noise (GaussianNois (None, 256, 256, 3)       0
_________________________________________________________________
resizing (Resizing)          (None, 299, 299, 3)       0
_________________________________________________________________
inception_v3 (Functional)    (None, 8, 8, 2048)        21802784
_________________________________________________________________
Flattening (Flatten)         (None, 131072)            0
_________________________________________________________________
dropout (Dropout)            (None, 131072)            0
_________________________________________________________________
dense (Dense)                (None, 128)               16777344
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 14)                910
=================================================================
Total params: 38,589,294
Trainable params: 17,181,774
Non-trainable params: 21,407,520
_________________________________________________________________
```

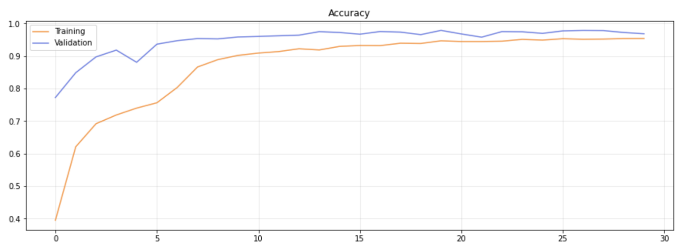
Dataset

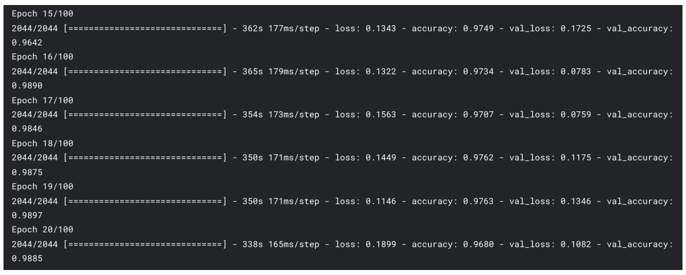
In this part we decided to balance the dataset. We used different copies of the same images to improve the size of folders like *Raspberry* and *Blueberry* and we removed additional images of larger folders like *Tomato*. The latter was a difficult choice as we didn't want to lose useful additional information about the dataset. To counteract overfitting due to duplication of the same images we exploited the Gaussian Noise Layer previously introduced. This one, in addition to the randomness during data augmentation, prevented the network from seeing the same images over and over.

Results

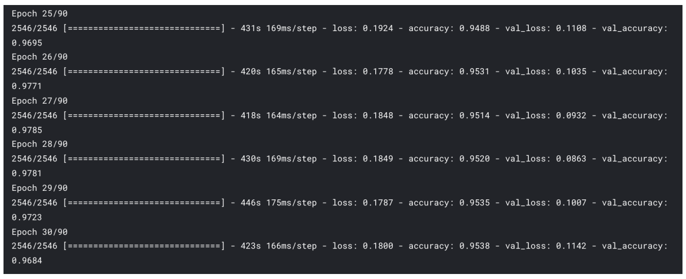
Comparison between *VGG16* (left) and *InceptionV3* (right):







```
Epoch 15/100
2044/2044 [==============================] - 362s 177ms/step - loss: 0.1343 - accuracy: 0.9749 - val_loss: 0.1725 - val_accuracy:
0.9642
Epoch 16/100
2044/2044 [==============================] - 365s 179ms/step - loss: 0.1322 - accuracy: 0.9734 - val_loss: 0.0783 - val_accuracy:
0.9890
Epoch 17/100
2044/2044 [==============================] - 354s 173ms/step - loss: 0.1563 - accuracy: 0.9707 - val_loss: 0.0759 - val_accuracy:
0.9846
Epoch 18/100
2044/2044 [==============================] - 350s 171ms/step - loss: 0.1449 - accuracy: 0.9762 - val_loss: 0.1175 - val_accuracy:
0.9875
Epoch 19/100
2044/2044 [==============================] - 350s 171ms/step - loss: 0.1146 - accuracy: 0.9763 - val_loss: 0.1346 - val_accuracy:
0.9897
Epoch 20/100
2044/2044 [==============================] - 338s 165ms/step - loss: 0.1899 - accuracy: 0.9680 - val_loss: 0.1082 - val_accuracy:
0.9885
```



```
Epoch 25/90
2546/2546 [==============================] - 431s 169ms/step - loss: 0.1924 - accuracy: 0.9488 - val_loss: 0.1108 - val_accuracy:
0.9695
Epoch 26/90
2546/2546 [==============================] - 420s 165ms/step - loss: 0.1778 - accuracy: 0.9531 - val_loss: 0.1035 - val_accuracy:
0.9771
Epoch 27/90
2546/2546 [==============================] - 418s 164ms/step - loss: 0.1848 - accuracy: 0.9514 - val_loss: 0.0932 - val_accuracy:
0.9785
Epoch 28/90
2546/2546 [==============================] - 430s 169ms/step - loss: 0.1849 - accuracy: 0.9520 - val_loss: 0.0863 - val_accuracy:
0.9781
Epoch 29/90
2546/2546 [==============================] - 446s 175ms/step - loss: 0.1787 - accuracy: 0.9535 - val_loss: 0.1007 - val_accuracy:
0.9723
Epoch 30/90
2546/2546 [==============================] - 423s 166ms/step - loss: 0.1800 - accuracy: 0.9538 - val_loss: 0.1142 - val_accuracy:
0.9684
```

PlantVillage Dataset [Mean accuracy on test set > 90%]:

[https://github.com/spMohanty/PlantVillage-Dataset]

All said so far is the result of the dataset provided in the challenge. Out of mere curiosity to see the best achievable results we used the full PlantVillage dataset. Using the complete dataset we obtain a counterintuitive of about 78% accuracy on the test set improved to >90% balancing the dataset using the same procedure described before.