# Plant disease detection with CNNs: effects of real-time data augmentation and hyperparameters

## Vision and Cognitive Services

Francesco Fontana  Lorenzo Mancini

03/09/2021

University of Padua

# Table of contents

# Introduction

# Goal

The goal of this project is to exploit computer vision resources in order to perform a detection for plant diseases and pests

**Figure 1:** Example of diseased plants: (from top left) potato, apple, strawberry, tomato

# Motivation

Plant automatic identification is a major land mark in agriculture:

- Disease identification positively impacts crop yield and quality
- It requires continuous monitoring by experts
- Historically, disease identification has been supported by agricultural extension organizations or other institutions, such as local plant clinics

# Overview

Our project is organized as follows:

- Comparison between networks with same hyperparameters;
- Performances with data augmentation;
- Effects of hyperparameters;
- Main results and conclusions

# Dataset

# PlantVillage-Dataset

In order to achieve our goal, we make use of a public dataset available online PlantVillage dataset [2] 38 different classes and 54303 images (and more images are being collected)
The dataset is divided into 3 directory:

- **color** : contains original images in RGB
- **grayscale** : contains grayscaled version of original images
- **segmented** : RGB images with just the leaf segmented and color corrected.

For our goal we use just the **color** directory.

# Dataset example

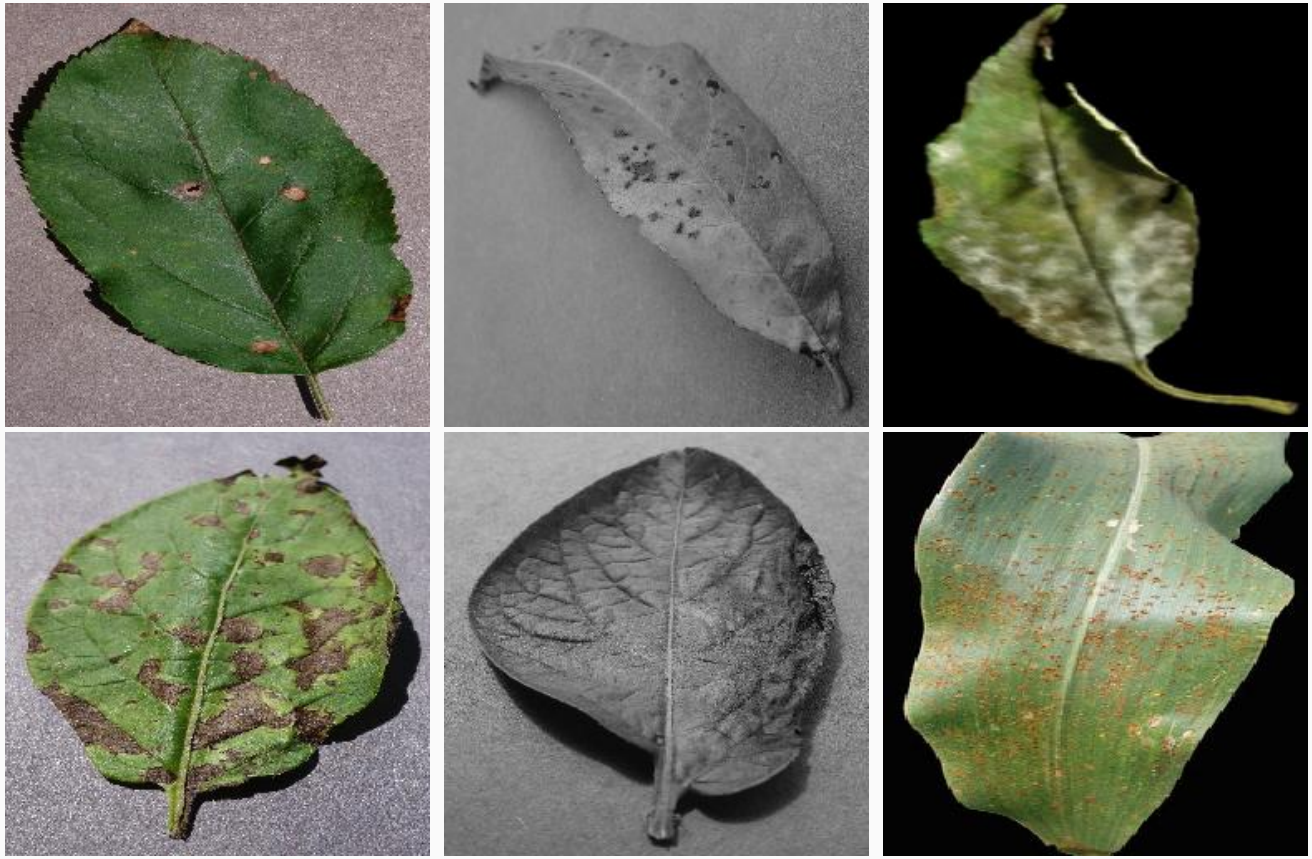


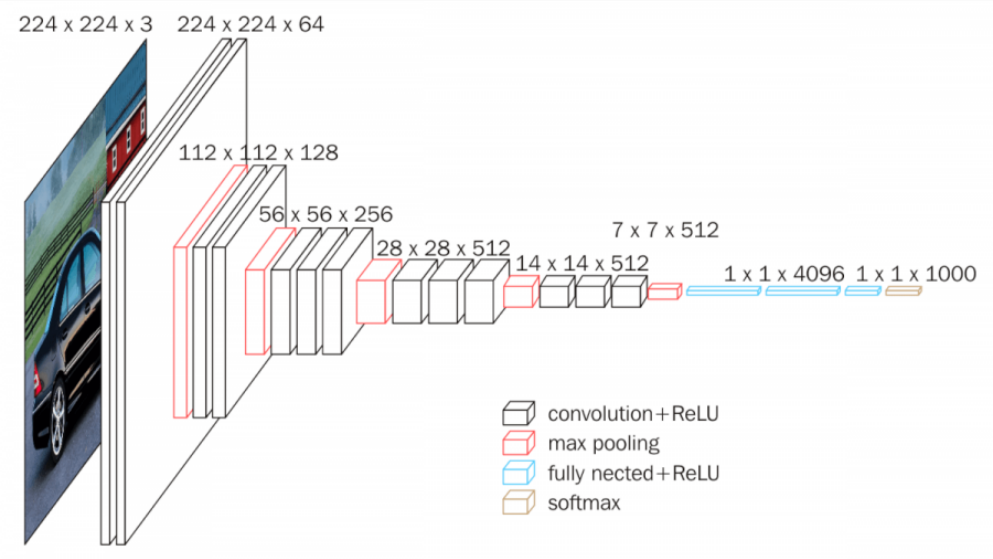
**Figure 2:** *Example of colored (left), grayscaled (middle) and segmented samples (right).*

# Methods

# CNNs

Deep Learning methods:

- very efficient (high accuracy) and promising in the extracting features task [1];
- a lot of pre-trained models are available on the web and simple to load and customize;
- requires a huge amount of data for the training → computational heavy.



# Transfer Learning

Due to our hardware limitations, we implement Transfer Learning method and in particular we make use of Feature Extraction procedure:

- import pre-trained model (ex. VGG16/19, InceptionV3, etc.) with relative weights trained on the famous dataset “ImageNet”, without the top layers.
- freeze all parameters already present in the imported model so that they won’t be trained again
- add top layers of the network adapted for our particular task (only these will be trained)

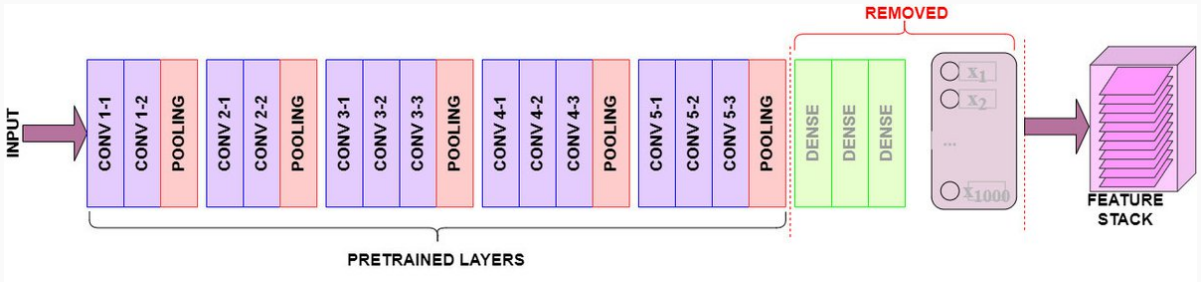



**Figure 3:** Example of *feature extraction* with VGG16’s architecture

# Used models

In our project we've compared the results obtained with the following pre-trained models, already implemented on the *Tensorflow* library:

| Model | n° of layers |
|---|---|
| VGG-16 | 16 |
| VGG-19 | 19 |
| InceptionV3 | 38 |
| ResNet101V2 | 101 |

All these models can be tuned according to some parameters, such as: the input size, epochs, the learning rate, the batch size, the dropout and others.

# Hyper-Parameters

For all the previous models we firstly consider the following hyper-parameters as the “default” ones:

| | |
|---|---|
| Epochs | 30 |
| Learning rate | 0.001 |
| Batch size | 32 |
| Dropout | 0.15 |
| Train – Test sizes | 0.8 – 0.2 |

# Input size i

- First, we feed the models with different input sizes:

64x64 128x128 256x256

- See how much *input size* influences the overall performance and results of the models (ex. plot below with VGG16)

# Input size ii

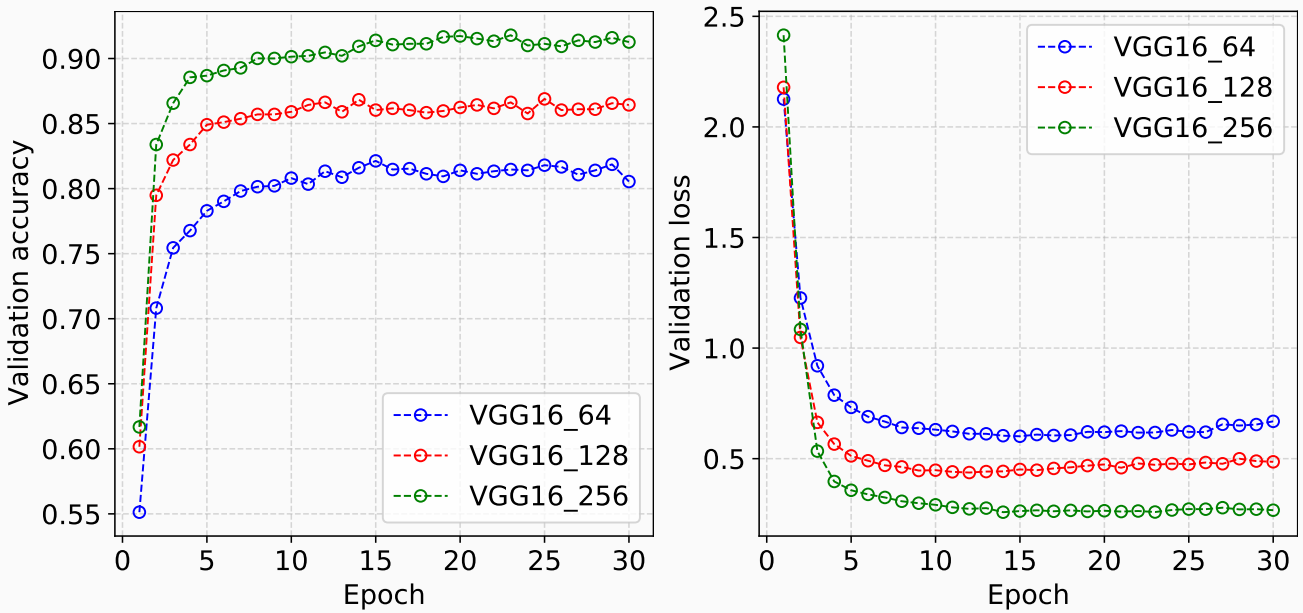



**Figure 4:** *Loss and accuracy plots of pre-trained VGG16 model with different image sizes in input*

- **Note:** *From now on all plots refers to 256x256 input size, since it provides the best results*

# Results (“default” parameters)

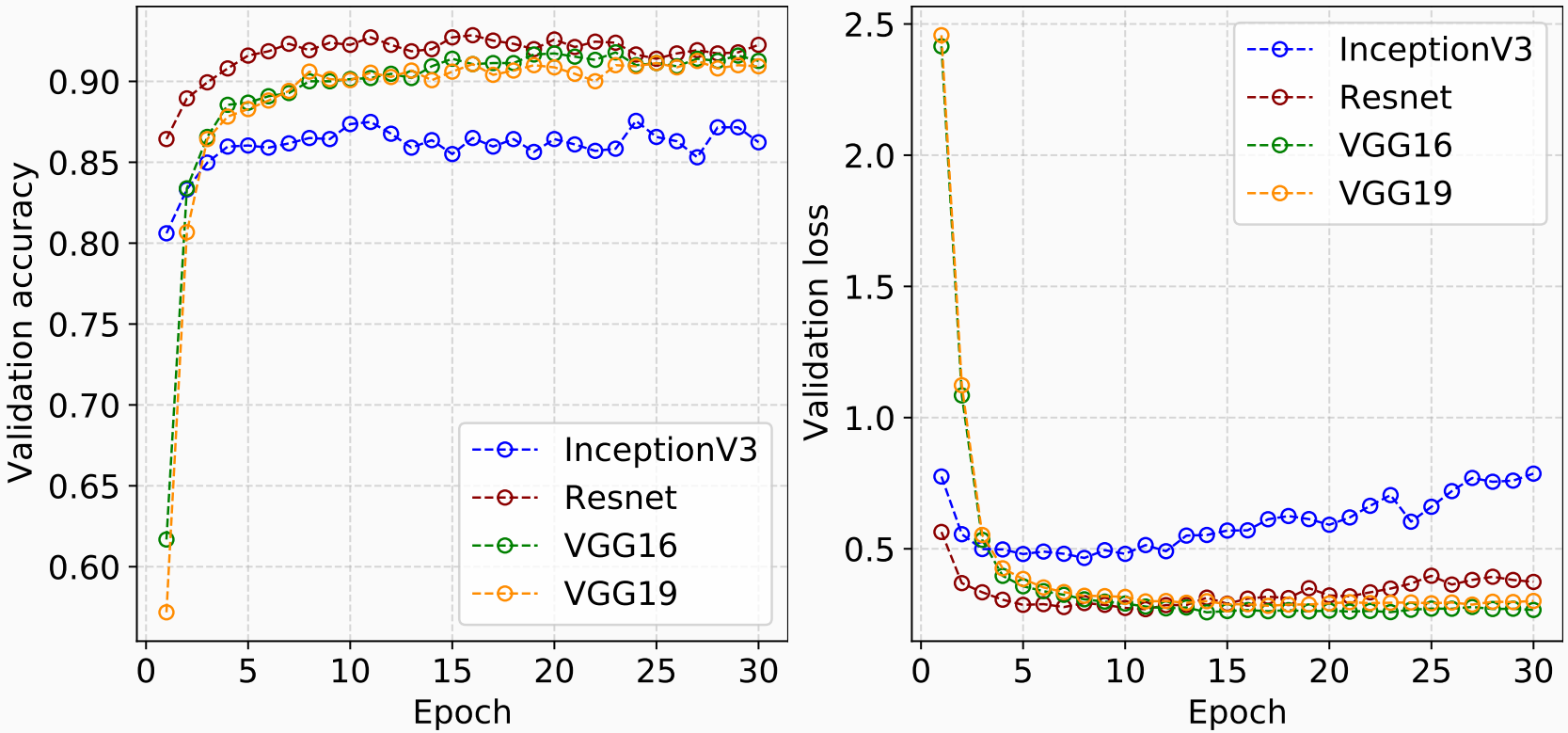



**Figure 5:** *Results obtained with the “default” training.*

# Overfitting (“default” parameters)

This standard approch leads in many cases to overfitting.

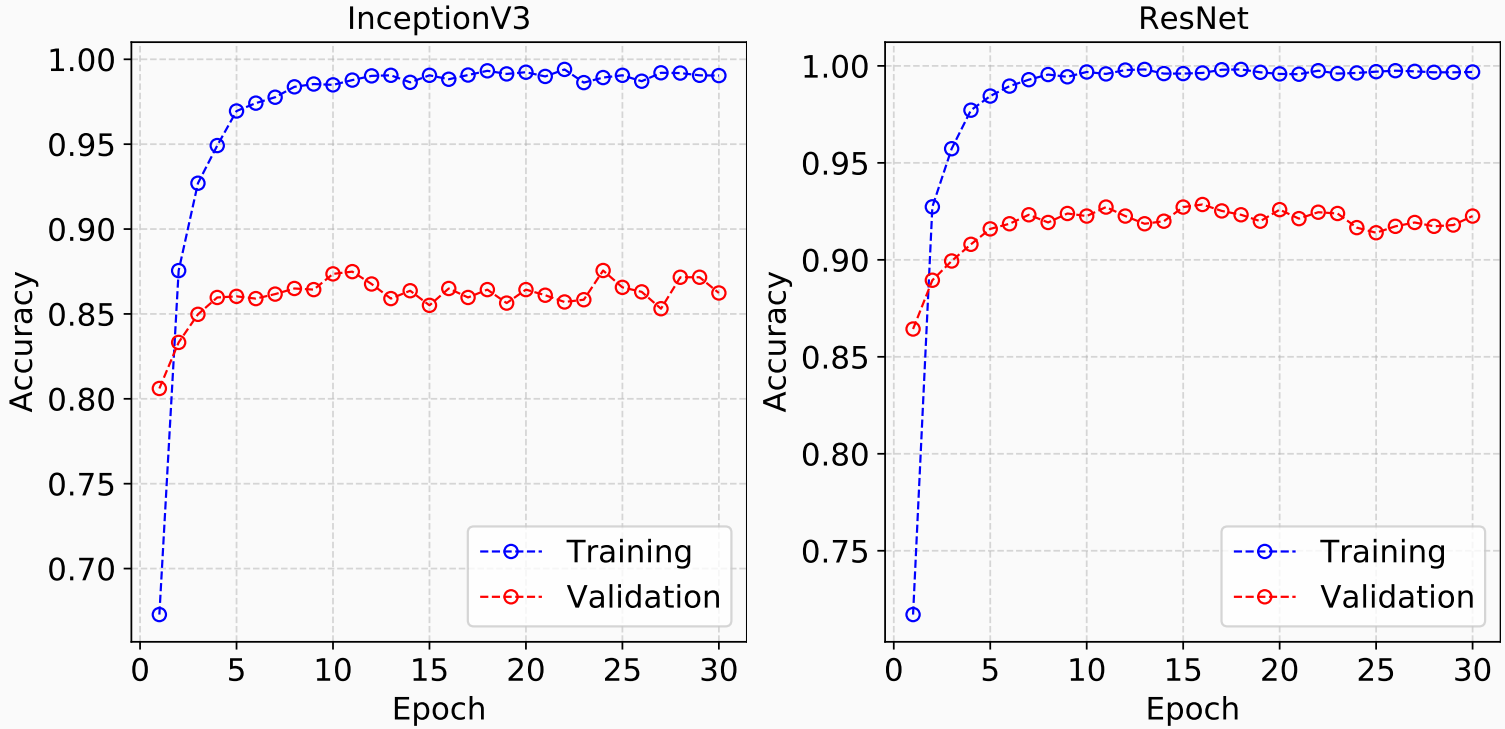



**Figure 6:** *Accuracy plot of InceptionV3 and ResNet101V2 with “default” parameters. Here can be seen an evident example of overfitting*

# Possible solutions

In order to solve the problem of overfitting, we try two different solutions:

- Real-time data augmentation;
- Tuning the hyperparameters.

# Experiments

# Data Augmentation i

Real-time data augmentation allows to train the networks with more data without storing it in the memory:

- very efficient and easy to implement;
- reduces overfitting and improves performance;
- it represent a very good compromise when the input dataset is not so large.

# Data Augmentation ii

It is implemented in the *Tensorflow* library:

```
aug = ImageDataGenerator(
            rotation_range=25,
            width_shift_range=0.2,
            height_shift_range=0.2,
            shear_range=0.2,
            zoom_range=0.3,
            horizontal_flip=True,
            fill_mode="nearest")
```

# Data Augmentation iii

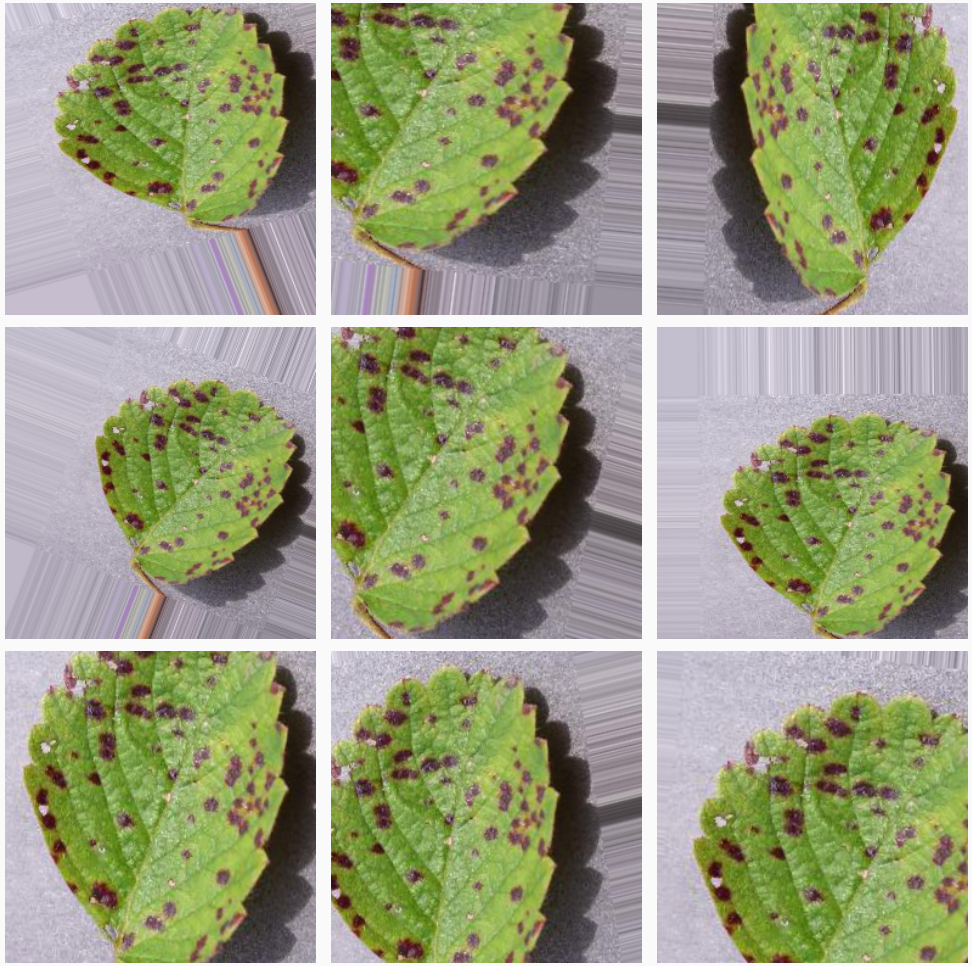


**Figure 7:** *Examples of random transformations applied on-the-fly to images in order to perform data augmentation.*

# Data Augmentation iv

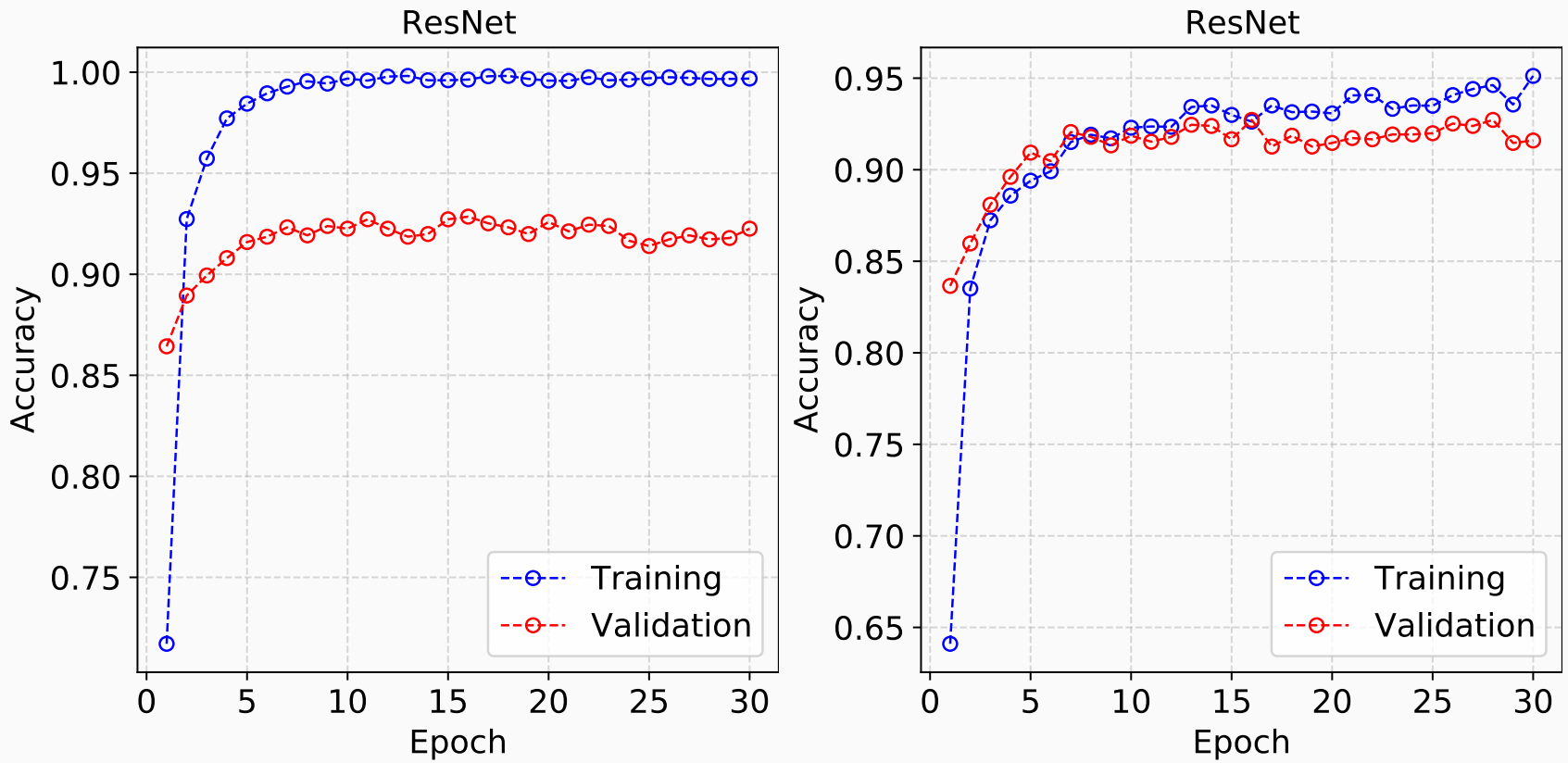



**Figure 8:** *Plots of loss and accuracy obtained during the training of the network for each model using the parameter*

# Data Augmentation v

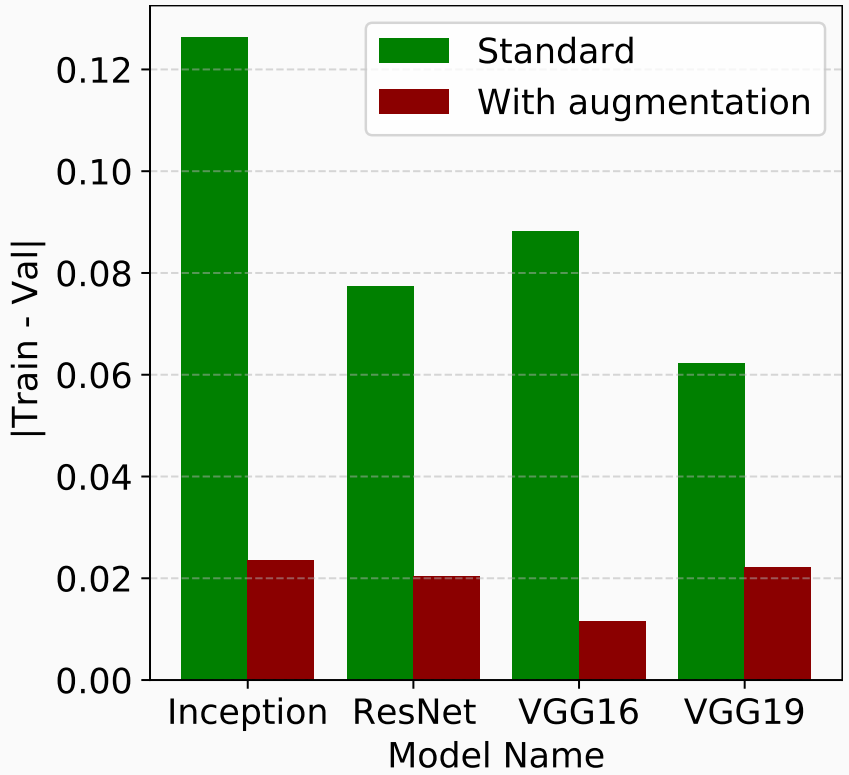



**Figure 9:** *Average difference between training and validation accuracy considering the last 10 epochs for all the networks.*

# Batch size

Batch size is the number of samples that feed the network at each step: for example, a batch size = 32 means that the networks is trained taking 32 samples at time.

| Model Name | Batch size | Accuracy |
|---|---|---|
| InceptionV3 | 32 | 87.6% |
| | 120 | 87.5% |
| ResNet101V2 | 32 | 93.2% |
| | 64 | 92.7% |

**Table 1:** *Comparison of the validation accuracy between different batch sizes: for InceptionV3 we use 32 and 120, whereas for ResNet101V2 we use 32 and 64. Performances are very similar.*

# Learning Rate i

Learning rate (LR) plays a major role in the loss optimization since it controls both how quickly the model is adapted to the problem and also the stability of the solutions

- **Low LR** ( $\leq 10^{-4}$)
  - **Pro:** a more stable behaviour, less overfitting.
  - **Cons:** longer training time ( *more epochs required* )
- **High LR** ($\sim 10^{-2}/10^{-3}$)
  - **Pro:** Less computational time, good performance
  - **Cons:** more unstable solutions

# Learning Rate ii

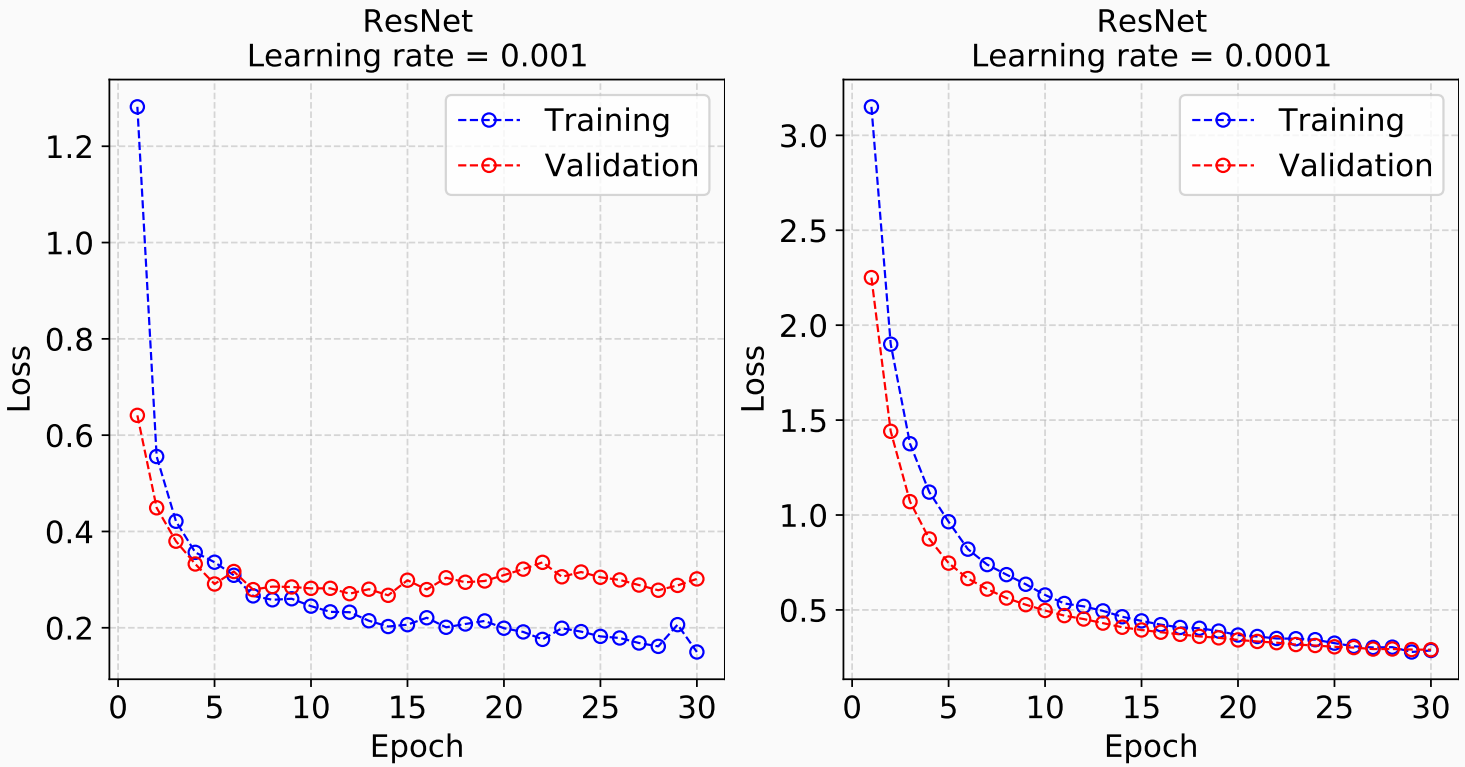



**Figure 10:** *Plots of training and validation loss using 2 different learning rates. With the lower one we can see a more stable behaviour and less overfitting. On the other hand the higher learning rate provides a very good performance taking into account the fact that the computational time is lower.*

# Learning Rate iii

| Model Name | Learning Rate | Accuracy |
|---|---|---|
| VGG16 | 1e-3 | 91.8% |
| | 1e-4 | 89.5% |
| VGG19 | 1e-3 | 91.3% |
| | 1e-4 | 89.6% |
| InceptionV3 | 1e-3 | 87.6% |
| | 1e-4 | 87.4% |
| ResNet101V2 | 1e-3 | 93.2% |
| | 1e-4 | 92.5% |

**Table 2:** *Comparison of the validation accuracy between different learning rate. ( $LR = 10^{-3} \longrightarrow$ 30 epochs; $LR = 10^{-4} \longrightarrow$ 50 epochs)*

# Dropout

To avoid the network to be a lot sensitive to each neurons' weight, we make use of Dropout regularization [3].

| Model Name | Dropout | Accuracy |
|---|---|---|
| VGG16 | 0.15 | 90.9% |
| | 0.5 | 91.1% |
| VGG19 | 0.15 | 90.5% |
| | 0.5 | 90.7% |
| InceptionV3 | 0.15 | 87.7% |
| | 0.5 | 87.1% |
| ResNet101V2 | 0.15 | 93.2% |
| | 0.30 | 93.4% |
| | 0.5 | 88.6% |

**Table 3:** *Comparison of performance for all network with different dropouts*

# Different train / test sizes

In order to further reduce the overfitting we've splitted the train and test sizes into a different way.

| Model Name | Train – Test | Accuracy |
|---|---|---|
| VGG16 | 0.8 – 0.2 | 90.9% |
| | 0.6 – 0.4 | 91.1% |
| VGG19 | 0.8 – 0.2 | 90.5% |
| | 0.6 – 0.4 | 90.6% |
| InceptionV3 | 0.8 – 0.2 | 87.6% |
| | 0.6 – 0.4 | 88.1% |
| ResNet101V2 | 0.8 – 0.2 | 93.2% |
| | 0.6 – 0.4 | 92.9% |

**Table 4:** *Validation accuracy for InceptionV3 and ResNet101V2 splitting the entire dataset in different partition between training and test sets.*

# Results i

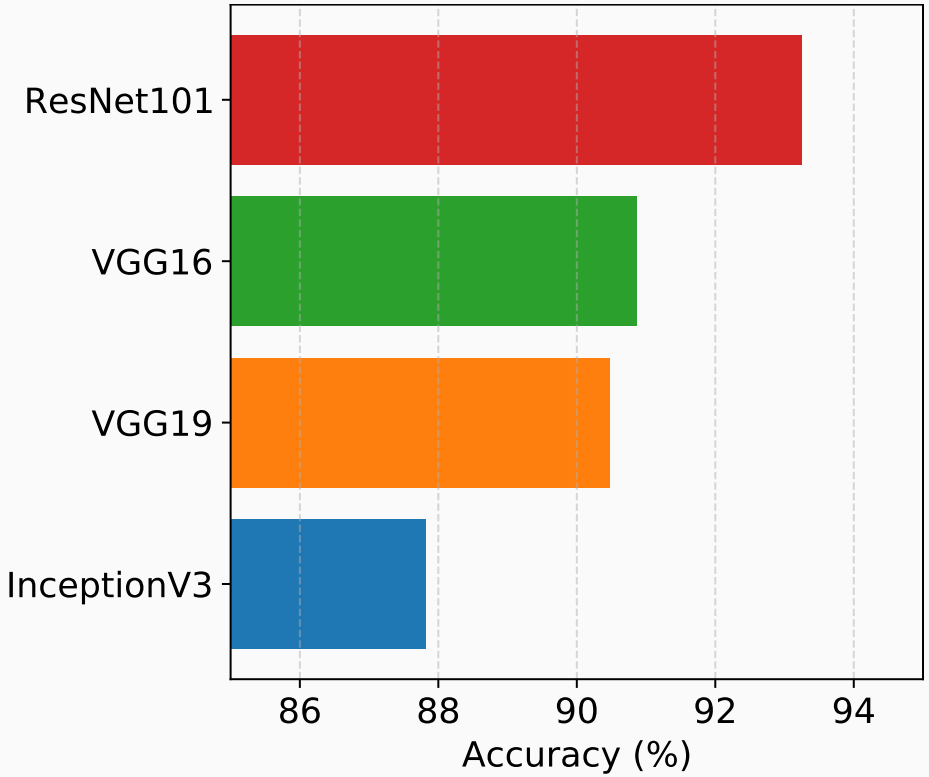



**Figure 11:** *Best performances obtained with each network. The highest accuracy (**93.4%**) is achieved by the ResNet model with data augmentation and dropout = 0.3.*

# Results ii

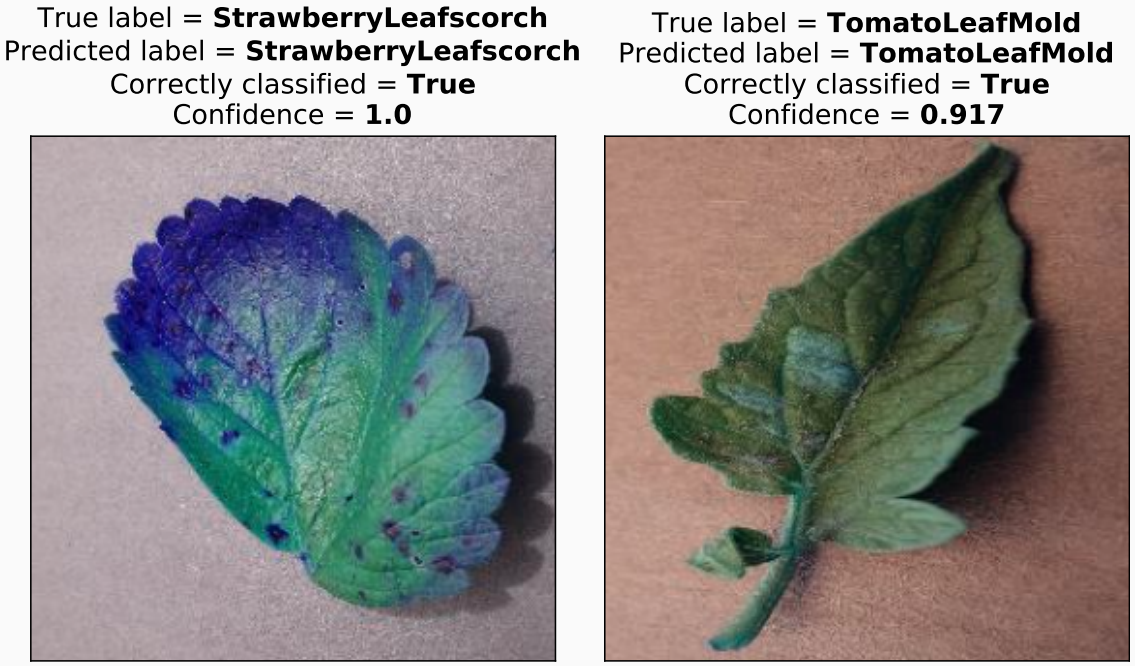



**Figure 12:** *Example of two correctly classified samples using ResNet101V2 model*

# Results iii

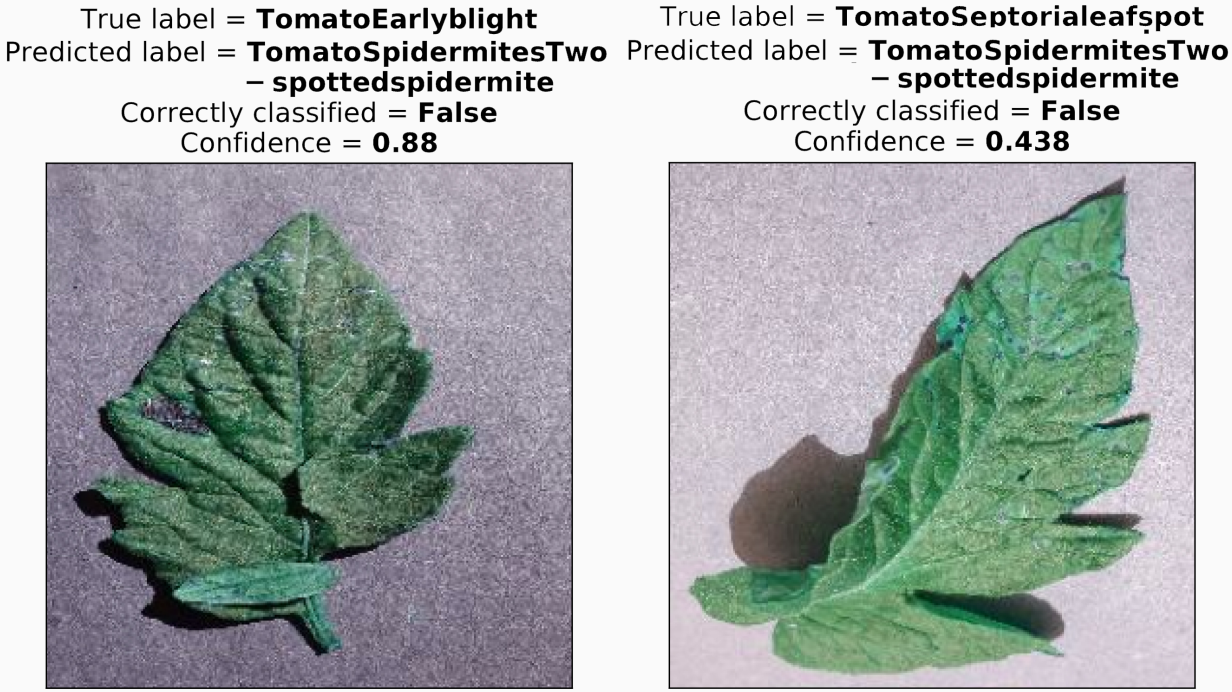



**Figure 13:** *Example of two not correctly classified samples using ResNet101V2 model*

# Conclusions

# Summary

In our work we make use of the public dataset PlantVillage in order to find a model able to predict diseases and pests. In particular we:

- exploit some pre-trained networks and study their performances and limitations;
- apply real-time data augmentation with the aim of reducing overfitting and improving performances;
- tune hyperparameters and study their impact on the networks.

At the end, we end up with our best performance equal to 93.4% accuracy provided by the ResNet101V2 network using data-augmentation and a different value of dropout.

# Hardware limitations

For our work we used the following hardware resources (referring to our local computer):

- **CPU:** AMD Ryzen 9 , 12 core, 24 threads (used multiprocessing of Tensorflow)
- **RAM memory:** 32 GB 3200 MHz

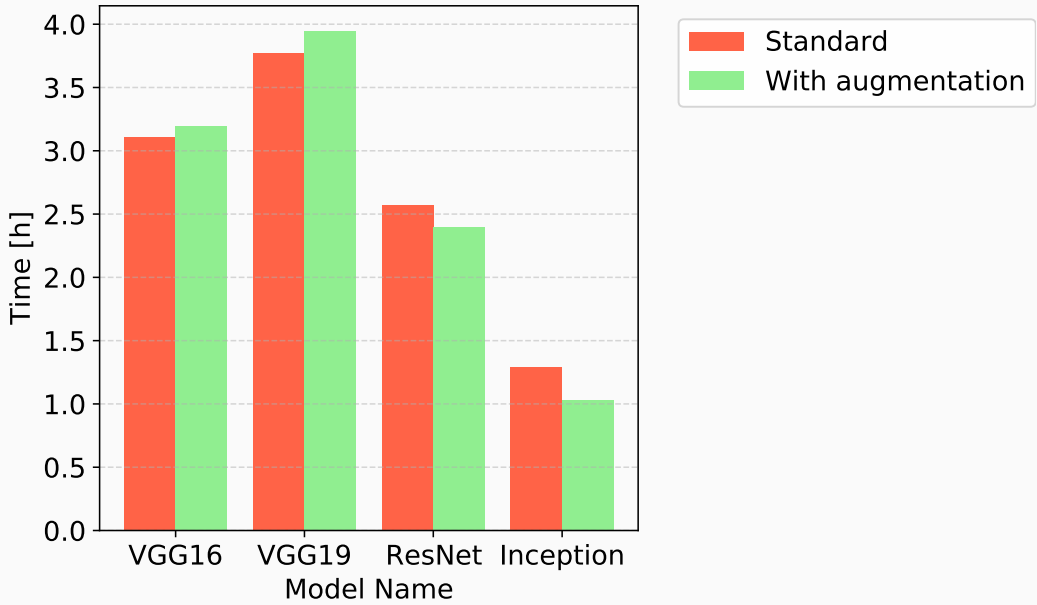



**Figure 14:** Training time for each network

# References i


S. M. Hassan, A. K. Maji, M. Jasiński, Z. Leonowicz, and E. Jasińska.
**Identification of plant-leaf diseases using cnn and transfer-learning approach.**
*Electronics*, 10(12), 2021.

D. P. Hughes and M. Salath'e .
**An open access repository of images on plant health to enable the development of mobile disease diagnostics through machine learning and crowdsourcing.**
*CoRR*, abs/1511.08060, 2015.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov.
**Dropout: A simple way to prevent neural networks from overfitting.**
*Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

# Thanks for your attention!