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

Francesco Fontana
francesco.fontana.10@studenti.unipd.it

Lorenzo Mancini
lorenzo.mancini.1@studenti.unipd.it

Abstract

Identification of plant disease and pests is a major land mark in agriculture. In this work we exploit Convolutional Neural Networks in order to find a model able to detect plant disease over the (restricted) PlantVillage Dataset: first we compare the performances of different pre-trained models (VGG19, Inception V3, VGG16 and ResNet101V2) and discuss their limitations. Then we study the impact of the hyperparameters and of the real-time data augmentation technique. We show how the latter gives the best improvements to the models especially in terms of generalization (less overfitting). Finally, tuning the dropout, we find our best model in the ResNet101V2 network that provides an accuracy of 93.4%.

1. Introduction

The identification of plant diseases is a crucial point in the agricultural sector. First of all, the early identification of a specific pest clearly impacts the crop yield and quality in a positive way. Furthermore, most rural areas are still based on the visual identification made by experts which is time consuming and expensive. The development of an efficient model accessible to everybody, through the smartphone for example, might represent a good step forward in agriculture and technology. For this reason, in the last years, researchers presented several solutions using Computer Vision methods and nowadays more and more institutions are working on this direction. For our task, we rely on deep learning methods, CNNs in particular, which seem to be one of the most promising and reliable techniques in the automatic learning sector. Other methods such as Support Vector Machines (SVM), Clustering (KNN for example), random forests (RF) are valid alternatives [5]. If CNNs provide very good and stable solution, one needs to remark the fact that deep learning methods often require a large amount of data with a consequent heavy hardware involvement. In our work, we try to exploit some pre-trained models with the aim of finding a satisfying solution for the disease and pest identification problem. These models are sensitive to some parameters which can be tuned in order to reduce overfitting and to decrease the loss. We go into some of those parameters and see how they impact on the network. Moreover we also perform the technique of real-time data augmentation (that does not involve memory) which results to be an excellent method to reduce overfitting and to improve the performance also with a restricted dataset.

2. Related Works

Since this kind of classification task is gaining more and more attention, it is easy to find interesting material on the web. Here we summarize some approaches used by different authors.
In [1] authors make use of six different pre-trained models, namely AlexNet, VGG16, VGG19, GoogLeNet, ResNet101V2 and DenseNet201. Here, authors manage to identify up to 10 different plant disease reaching an accuracy of 97.3% using the GoogleNet network.
In [5] authors exploit hisogram-based and geometric features from segmented tomato-diseased-leaves portions and apply a SVM classifier with different kernels. Hassan et al. [3] use different pre-trained networks (AlexNet, VGG, NASNet, ResNet etc.) and show the performances for different hyperparameters such as: the number of training epochs, the batch size, dropout and learning rate. The highest accuracy here is reached with EfficientNetB0 (99.56%).
In [2] authors use various CNN architectures reaching $\sim$ 99.5% accuracy with 58 different classes of disease.

3. Dataset

The web provides a lot of datasets that can be used for experimenting new models and machine learning techniques. In our case we make use of the public dataset "PlantVillage", available at [4]. It belongs to the PlantVillage.org platform born in 2013. The dataset contains more than 50.000 images of healthy and diseased plants. All the images present in the dataset were taken at experimental research stations associated with the Land Grant Universities in USA. The leaves were removed from the plant and then placed against a gray or black paper sheet. For each leaf,

4-7 images were recorded using a standard digital camera (Sony DSC - Rx100/13 20.2 megapixels). Some plants have leaves too large to be captured entirely in a single image: in this cases, images of different subsection of the leaf were taken. This results in 54.309 images of leaves that belong to 14 different crop species: Apple, Blueberry, Cherry, Corn, Grape, Orange, Peach, Bell Pepper, Potato, Raspberry, Soybean, Squash, Strawberry, Tomato. More in detail, it contains photos of 17 fungal diseases, 4 bacterial diseases, 2 mold (oomycete) diseases, 2 viral disease, and 1 disease caused by a mite [4]. For some species there are also photos of healthy leaves. In a few words, we have 38 different classes. Referring to the repository [4] of the dataset, we can find 3 different kind of the dataset:

- `color`: contains the original RGB version of the photos;
- `grayscale`: contains the gray-scaled version of the original images;
- `segmented`: contains RGB images with just the leaf segmented and color corrected.

For our project, and for computational reasons, we just rely on the `color` dataset.

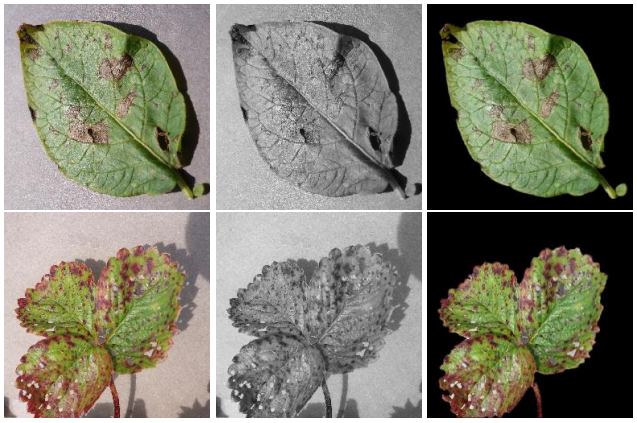


Figure 1: *An example of image extracted from, respectively, color, grayscale and segmented datasets.*

4. Methods

Convolutional Neural Networks represent one of the most popular and efficient tool in the field of classification. In particular, CNN can learn features from a dataset made up of images and the architecture can be easily tuned in order to reach very high performances. For those reasons we choose to exploit Deep Learning techniques for the plant disease identification. Furthermore, due to the huge amount of data and the consequent time that the training would require, we exploits some pre-trained networks. In particular, as anticipated, we first study the performances of the following models: VGG16, VGG19, InceptionV3 and ResNet101V2, all based on the results obtained with the same models reported in the scientific literature (pre-trained on "ImageNet" dataset). So far, we use the "Transfer learning" tecnique and in particular we implement the "Feature Extraction" procedure. It consists in freezing all the layers and the respective weights of the pre-trained models right before the last fully connected ones and manually adapt the top layer to our particular task. For all the networks we use the Adam algorithm as optimizer and the loss is computed with the "categorical cross entropy" loss function. Moreover, the activation function considered is the "softmax".

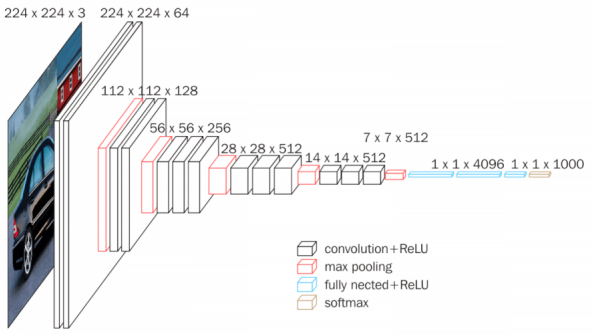


Figure 2: *An example of VGG16's structure with 224x224 RGB input size*

VGG16 model is composed of convolutions layers, max pooling layers, and fully connected layers. The total is 16 layers with 5 blocks and each block with a max pooling layer. Similar to VGG16, VGG19 has 19 layers with extra convolution layers in the last three blocks. VGG19 is slightly better but requires more memory. With deep layers, both VGG16 and VGG19 achieve great performance in the image competition. In comparison to VGGNet models, Inception Networks has been proved to be more computationally efficient, both in terms of the number of parameters generated by the network and the economical cost incurred (memory and other resources). In particular, InceptionV3 architecture is composed by factorized convolutions, small and asymmetric convolutional filter, auxiliary classifier (used as regularizer) and an efficient grid size reduction making use of parallels stride blocks. All this leads to a CNN that is 48 layers deep. ResNet network uses a technique called "residual mapping" that implements the "identity connection" between the layers, building the residual block, in order to protect the network from the vanishing gradient problem. In this particular case, we used ResNet101V2 that differs from the V1 models by the fact that it implements the pre-activation of weights layer instead of post-activation. The overall network is composed by 101-layer and even if the depth is increased, it has lower complexity than VGG-16/19 networks.

In this section we investigate the performances of those

models adapted to our task. Then, in the next one, we look how the various parameters affect the performances and we discuss some extra techniques that can lead to better results. Furthermore, it is worth to remark that for computational and hardware limitations, we work on a reduced dataset, keeping only the first 200 images of each class.

Model	**n. of layers**
VGG16	16
VGG19	19
InceptionV3	48
ResNet	101

Table 1: *Number of layers of each Deep Neural Network*

All those models can be tuned according to some parameters, such as the input size, the number of epochs, the learning rate, the batch size, the dropout and others. In this section we keep the same parameters of Table 2 for all the models.

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

Table 2: *Parameters used as default for all the pre-trained models*

It is important to highlight the impact of the input size parameter: input image can be passed to the network with the original size (256x256) or they can be resized. Here, we show how this parameter affects the validation accuracy. We take three sizes: 64x64, 128x128 and the original one, 256x256. As one can expect a smaller resolution gives a lower performance, since it is harder for the network to distinguish the main aspects of the disease. In Figure 4 we show the results of the comparison between different input sizes for the VGG16 model. The same behaviour obviously occurs for all the other networks.

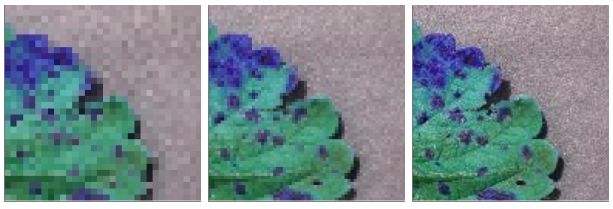


Figure 3: *Examples of different images resolutions, from left: 64x64, 128x128, 256x256 px.*

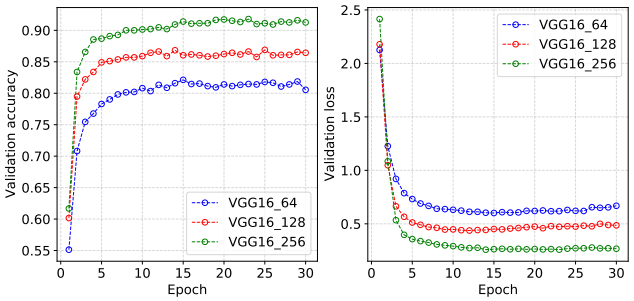


Figure 4: *Loss and accuracy plots of pre-trained VGG16 model with different image sizes in input*

At this point, we can now proceed with the training of all the models feeding the networks with original sized images: this, obviously at the price of a longer training time.

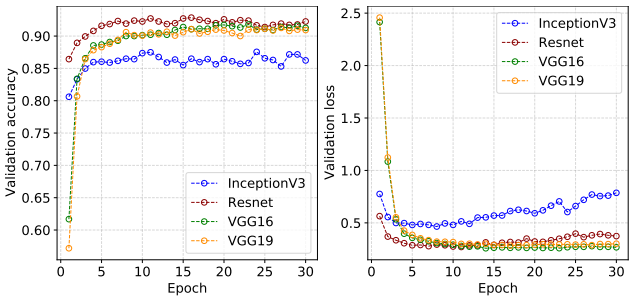


Figure 5: *Plots of loss and accuracy obtained during the training of the network for each model using the parameter in Table 2*

ResNet and VGG19 reach very good results: remind the fact that we're working with a reduced dataset and "default" parameters. Nevertheless an accuracy slightly higher than 90% is not so far from the ones obtained in the various papers. Clearly, this isn't the best way to compare performances, since a set of parameters can be the best choice for one precise network but not for the others. Moreover, this standard approach leads to overfitting as can be seen from the following graphs: as an example, Inception and ResNet have very high training accuracy but the validation one is quite smaller.

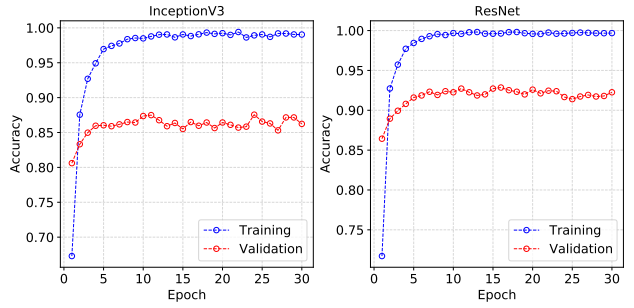


Figure 6: *Accuracy plot of InceptionV3 and ResNet101V2 with "default" parameters. Here can be seen an evident example of overfitting*

In the next section we present some experiments on the parameters and on the input dataset (data augmentation) that generalizes the results (less overfitting) and, in some cases, leads to better performances.

5. Experiments

5.1. Data augmentation

One idea is to exploit the real-time data augmentation method. Since it is not efficient to store the augmented dataset into memory (it would be also meaningful in our case since we're working with a reduced dataset) this technique is a good candidate for our goal. We rely on the `ImageDataGenerator` class of `Tensorflow`. In a few words, it generates batches of tensor image data with real-time data augmentation.

```
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")
```

Listing 1: *Python code for the* `ImageDataGenerator` *and its parameters.*

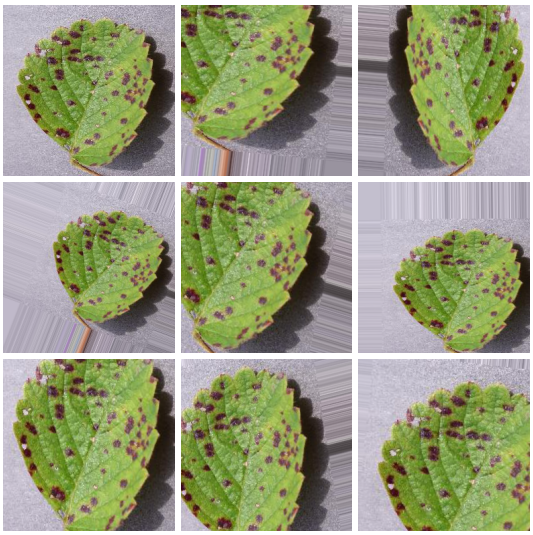


Figure 7: *Examples of random transformations applied on-the-fly to images in order to perform data augmentation (top left image is the original one)*

Feeding the networks with the same parameters as before, but with data augmentation, this is what we get as results:

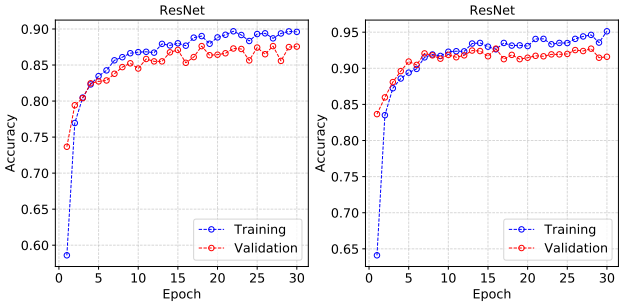


Figure 8: *Same plots of [Figure 6] using data augmentation during the training of the network*

We can have a global look of this behaviour by computing the absolute value of the mean difference between the training accuracy (last 10 values) and the validation one (last 10 values):

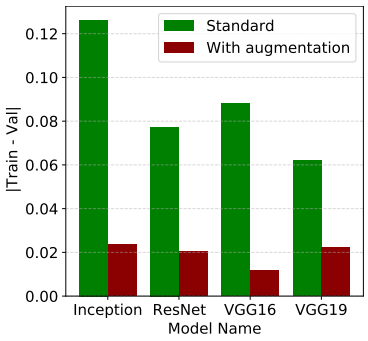


Figure 9: *Average differences between training and validation accuracy considering the last 10 epochs for all the models. As one can see, data augmentation procedure decrease significantly the overfitting*

As one can notice, with the procedure of data augmentation, the model is much more "generalized" in terms of overfitting: the validation and the training losses are now much closer than before. Now, what remains is to study the impact of the other hyperparameters.

5.2. Batch size

This is the number of samples of the input dataset that are passed through the network. With a batch size of 32, the network takes the first 32 samples for the training. Then, it takes the second 32 samples and so on. For our goal, it might be interesting to see if important differences occur when the batch size is increased. We find that this parameter is not so much relevant, indeed, referring to the InceptionV3 network the performances are very similar:

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

Table 3: *Comparison of the validation accuracy between different batch sizes: for InceptionV3 we use 32 and 120 whereas for ResNet we use 32 and 64. Performances are very similar.*

Again, the same behaviour occurred for the other networks. Since there are not relevant changes, we keep on using as "standard" batch size value the value of 32.

5.3. Learning rate

The learning rate plays a major role in the loss optimization. It controls how quickly the model is adapted to the problem. Obviously, a larger learning rate will lead to a shorter training time (in terms of epochs) at the price of more unstable solutions. The converse happens for a small learning rate. We show the behaviour obtained with the ResNet model:

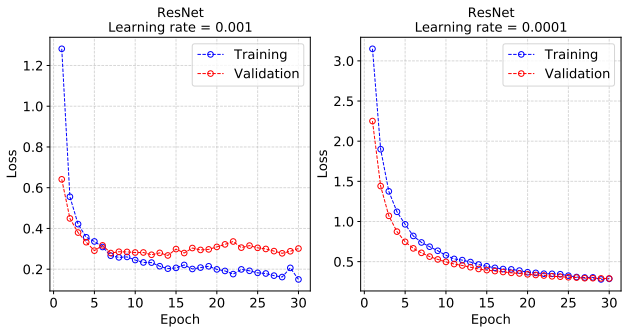


Figure 10: *Plots of training and validation loss using 2 different learning rates. With the lower one we can see a more stable behaviour and less overfitting. On the other hand the higher learning rate provides a very good performance taking into account the fact that the computational time is lower.*

.

Note the different scale on the y-axis of the two plots. On one hand we have that a smaller learning rate further reduces the overfitting and provides a very stable solution. At the same time, using the learning rate $lr = 0.0001$ requires more epochs to reach the minimum and this would entail a major use of computational resources (the training time for ResNet is ~ 3 hours with 30 epochs).

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 4: *Comparison of the validation accuracy between different learning rates: for all the models with learning rate 10^{-3} we set 30 epochs, instead for the others we used 50 epochs. It can be seen that there are no relevant differences and also in general with 10^{-3} we reach higher validation accuracy.*

5.4. Dropout

Dropout regularization, in short, consists in dropping out randomly selected neurons during the training. This results in the fact that the networks become less sensitive to the weights of single neurons [6]. Up to now, we've trained all the networks using a dropout coefficient equal to 0.15. With a dropout of 0.5 results are slightly lower whereas using an intermediate value of 0.3, we reach our best performance using the ResNet101V2 network:

Model Name	Dropout	Accuracy
ResNet101V2	0.15	93.2%
	0.30	93.4%
	0.5	88.6%

Table 5: *Comparison of performance for ResNet101V2 network with different dropouts.*

5.5. Different train and test sizes

Another thing that can be tested in order to further reduce the overfitting is to split the train and test sizes into a different way. Let'see what changes if we use 0.6 for the training and 0.4 for the test instead of $0.8 - 0.2$:

Model Name	Train – Test	Accuracy
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 6: *Validation accuracy for InceptionV3 and ResNet101V2 splitting the entire dataset in different partition between training and test sets.*

We try this different splitting of the dataset also for the other models (VGG16, VGG19) and we notice that in general the results achieved are very similar. Sometimes with a 0.6 - 0.4 (train/test split size) we obtain a slighly better accuracy and sometime viceversa. Nevertheless our best results up to now has been achieved with the "default" splitting size of 0.8 - 0.2 for ResNet101V2.

6. Results

Summarizing what we obtained, the best performance is the one of ResNet101V2 but other networks are not so far from it:

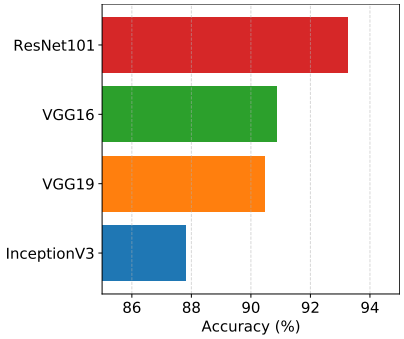
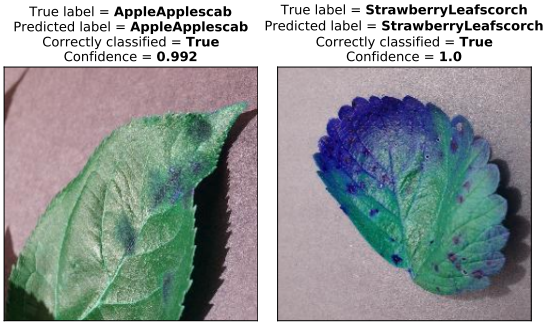


Figure 11: *Final results considering best performance for each model*

Model Name	Loss	Accuracy
ResNet101V2	0.269	93.2%
VGG16	0.293	90.9%
VGG19	0.311	90.4%
InceptionV3	0.393	87.8%

Table 7: *Comparison of Loss and Accuracy for each model: we consider the best results obtained with learning rate* = 10^{-3} *for each model, batch size* = 32 *for each model, dropout* = 0.15 *for VGG16, VGG19, Inception and dropout* = 0.3 *for ResNet101V2.*



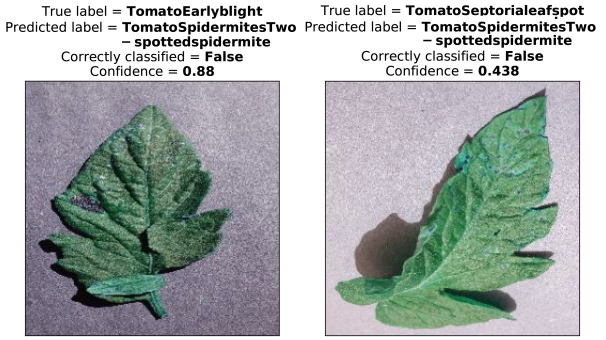


Figure 12: *Example of two correctly classified samples and two not correctly classified samples using ResNet101V2 model*

7. Conclusion

In the present work we try to adapt some pre-trained Convolutional Neural Networks in order to perform a classification task, that is the plant disease and pest identification. To this end we make use of a restricted portion of the public dataset (PlantVillage). We look at the performances of those networks and we study the impact of real time data augmentation and hyperparameters such as:

- input image size;
- batch size;
- learning rate;
- dropout;
- train and test sizes.

Performances are significantly better when real-time data augmentation is performed: the classification is much more generalized and thus overfitting is reduced. On the other hand, the input image size results to be the most relevant parameters in terms of performance: with the original size, all the networks achieve very good results, whereas with smaller sizes the classification is not performed correctly. Other parameters do not affect the networks in a relevant way: for the learning rate, choosing a smaller one would require much more training time (or advanced computational resources). Anyway, the best performance is reached with a slightly higher (compared to the initial one) value of dropout, 0.3 with the ResNet101V2 network: accuracy = 93.3%.

References

[1] Krishnaswamy Rangarajan Aravind and Purushothaman Raja. Automated disease classification in (selected) agricultural crops using transfer learning. *Automatika*, 61(2):260–272, 2020.

[2] Konstantinos P. Ferentinos. Deep learning models for plant disease detection and diagnosis. *Computers and Electronics in Agriculture*, 145:311–318, 2018.

[3] Sk Mahmudul Hassan, Arnab Kumar Maji, Michał Jasiński, Zbigniew Leonowicz, and Elżbieta Jasińska. Identification of plant-leaf diseases using cnn and transfer-learning approach. *Electronics*, 10(12), 2021.

[4] David P. Hughes and Marcel 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.

[5] Usama Mokhtar, Mona Ali, Aboul Ella Hassanien, and Hesham Hefny. *Identifying Two of Tomatoes Leaf Viruses Using Support Vector Machine*, volume 339, pages 771–782. 01 2015.

[6] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.