**Implementing Transfer Learning on Plant Village Dataset using multiple CNN-based Algorithms**

Abstract

In this project, multiple Convolution Neural Network based pre-defined models have been used for implementing transfer learning on images of multiple fruits (in both healthy and diseased state) from the Plant Village dataset. These pre-defined models were imported from the keras library. The final layer was created for each model according to number of classes in the dataset. The models were then implemented on images of the fruits determining if they were healthy or diseased, and, diagnose the class of disease. The best prediction accuracy and the f1 score were achieved by Resnet 50 algorithm for all the datasets.

Introduction

Motivation:

The motivation to doing this project was to understand the algorithms that can be used for implementing transfer learning by training and testing pre-defined algorithms on images of various fruits and to see how well they performed across different datasets.

Background:

I referred to The Data Science blog [[1](#Link_1)] to understand as to how Convolution Neural Network (CNN) works and help me in this project. I have also referred to following blogs to study the structures of various algorithms and, their methodology of transfer learning implementation.

1. Understanding various architectures of Convolutional Networks [[2](#Link_2)],
2. An Intuitive Guide to Deep Network Architectures [[3](#Link_3)],
3. A Simple Guide to the Versions of the Inception Network [[4](#Link_4)]

Convolutional Neural Network [[1](#Link_1)]:

Convolutional neural network is a class of deep, feed-forward artificial neural networks that have successfully been applied to analyzing visual imagery. A CNN consists of an input and an output layer, as well as multiple hidden layers.

The layers of a CNN typically consist of

1. Convolutional Layer

2. Pooling Layer

3. Classification (Fully connected Layer)

**Convolution Layer:** The primary purpose of the Convolution layer is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.

**Pooling Layer:** Spatial Pooling (also called subsampling or down-sampling) reduces the dimensionality of each feature map but retains the most important information.

**Fully Connected Layer:** The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset. Apart from classification, adding a fully-connected layer is also a (usually) cheap way of learning non-linear combinations of these features. Most of the features from convolutional and pooling layers may be good for the classification task, but combinations of those features might be even better.

Transfer Learning [[6](#Link_6)]:

Transfer learning is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks.

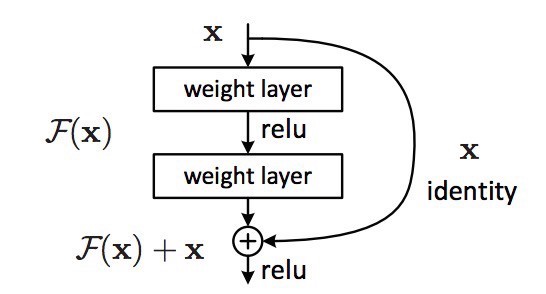
Literature Review of the Models Used:

Resnet 50 [[3](#Link_3)]:

ResNet was born from a simple observation: *why do very deep nets perform worse as you keep adding layers?* Intuitively, deeper nets should perform no worse than their shallower counterparts, at least at train time (when there is no risk of overfitting). As a thought experiment, let us say we have built a net with n layers that achieve a certain accuracy. At the minimum, a net with n+1 layers should be able to achieve the exact same accuracy, if only by copying over the same first n layers and performing an identity mapping for the last layer. Similarly, nets of n+2, n+3, and n+4 layers could all continue performing identity mappings and achieve the same accuracy. In practice, however, these deeper nets almost always degrade in performance.

The authors of ResNet boiled these problems down to a single hypothesis: *direct mappings are hard to learn*. And they proposed a fix: instead of trying to learn an underlying mapping from x to H(x), learn the *difference* between the two, or the “residual.” Then, to calculate H(x), we can just add the residual to the input.

Say the residual is F(x)=H(x)-x. Now, instead of trying to learn H(x) directly, our nets are trying to learn F(x)+x. This gives rise to the famous ResNet (or “residual network”) block you have probably seen:



Each “block” in ResNet consists of a series of layers and a “shortcut” connection adding the input of the block to its output. The “add” operation is performed element-wise, and if the input and output are of different sizes, zero-padding or projections (via 1x1 convolutions) can be used to create matching dimensions.

If we go back to our thought experiment, this simplifies our construction of identity layers greatly. Intuitively, it is much easier to learn to push F(x) to 0 and leave the output as x than to learn an identity transformation from scratch. In general, ResNet gives layers a “reference” point — x — to start learning from.

This idea works astoundingly well in practice. Previously, deep neural nets often suffered from the problem of vanishing gradient, in which gradient signals from the error function decreased exponentially as they back propagated to earlier layers. In essence, by the time the error signals traveled all the way back to the early layers, they were so small that the net could not learn. However, because the gradient signal in ResNets could travel back directly to early layers via shortcut connections, we could suddenly build 50-layer, 101-layer, 152-layer, and even (apparently) 1000+ layer nets that still performed well.

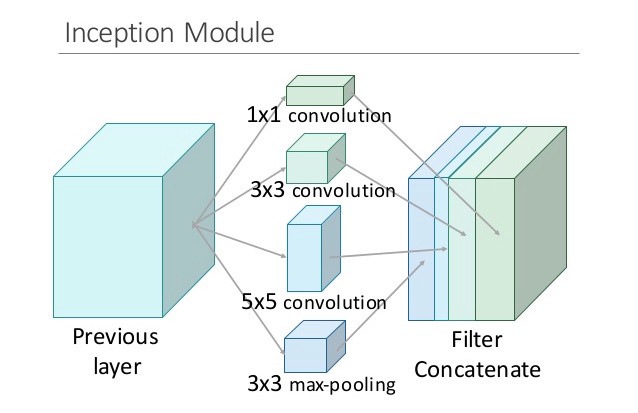
**ResNet50**, with 50 weight layers, is built on the above-mentioned micro-architecture modules also known as network-in-network architectures. Micro-architecture can also be viewed as the set of “building blocks” utilized for construction of a network. Updating the residual module for using identity mappings can help in obtaining better accuracy. ResNet50 being deeper than VGG16 & VGG19, the model size is remarkably smaller, as fully-connected layers have been replaced by the utilization of global average pooling— this significantly reduces the network size to 99MB for ResNet50.

Inception\_Resnet\_v2 [[3](#Link_3)], [[4](#Link_4)]:

**Inception**: If ResNet was all about going deeper, the Inception is all about going wider. In particular, the authors of Inception were interested in the computational efficiency of training larger nets. In other words: how can we scale up neural nets without increasing computational cost? The original paper focused on a new building block for deep nets; a block is now known as the “Inception module.” At its core, this module is the product of two key insights:

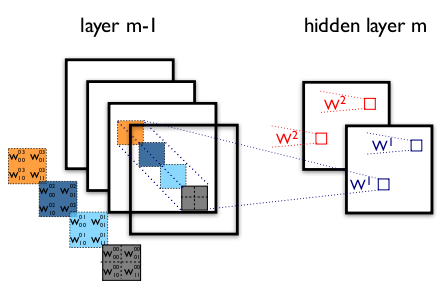
1. Why not let the model choose?

An Inception module computes multiple different transformation over the same input map in parallel, concatenating their results into a single output. In other words, for each layer, Inception does a 5x5 convolutional transformation, and a 3x3, and a max-pool. And the next layer of the model gets to decide if (and how) to use each piece of information.



The increased information density of this model architecture comes with one glaring problem: we’ve drastically increased computational costs. Not only are large (e.g. 5x5) convolutional filters inherently expensive to compute, stacking multiple different filters side by side greatly increases the number of feature maps per layer. And this increase becomes a deadly bottleneck in our model.

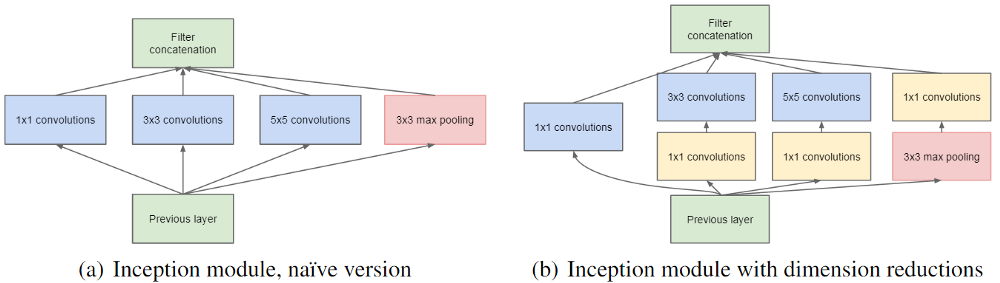
Think about it this way. For each additional filter added, we have to convolve over all the input maps to calculate a single output. See the image below: creating one output map from a single filter involves computing over every single map from the previous layer.



In other words, as the authors note, “any uniform increase in the number of [filters] results in a quadratic increase of computation.” This increase in computation was a drawback, and hence, the second insight.

1. Using 1x1 convolutions to perform dimensionality reduction.

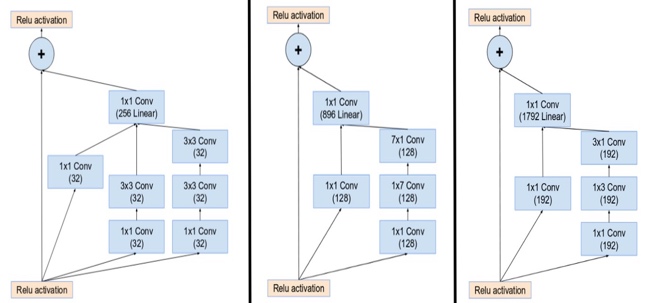
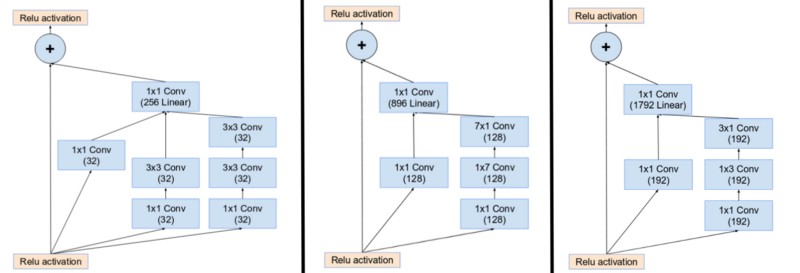
In order to solve the computational bottleneck, the authors of Inception used 1x1 convolutions to “filter” the depth of the outputs. A 1x1 convolution only looks at one value at a time, but across multiple channels, it can extract spatial information and compress it down to a lower dimension. For example, using 20 1x1 filters, an input of size 64x64x100 (with 100 feature maps) can be compressed down to 64x64x20. By reducing the number of input maps, the authors of Inception were able to stack different layer transformations in parallel, resulting in nets that were simultaneously deep (many layers) and “wide” (many parallel operations).



Inception rapidly became a defining model architecture. Most importantly, however, Inception demonstrated the power of well-designed “network-in-network” architectures, adding yet another step to the representational power of neural networks.

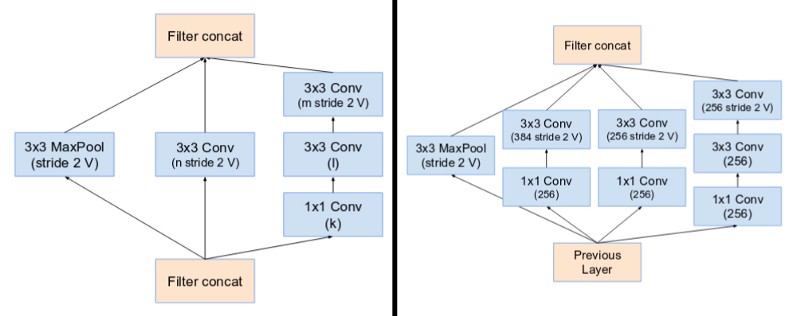
Inception-Resnet: Inspired by the performance of the ResNet, a hybrid inception module was proposed. The latest version of Inception, v4, threw in residual connections within each module, creating an Inception-ResNet hybrid. There are two sub-versions of Inception ResNet, namely v1 and v2.

The premise for Inception-ResNet hybrid was to introduce residual connections that add the output of the convolution operation of the inception module, to the input. For residual addition to work, the input and output after convolution must have the same dimensions. Hence, we use 1x1 convolutions after the original convolutions, to match the depth sizes (Depth is increased after convolution).



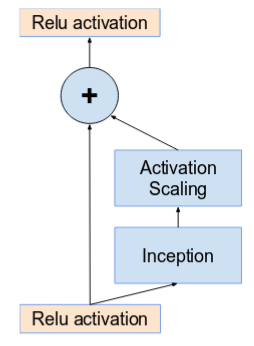
*(From left) Inception modules A,B,C in an Inception ResNet. Note how the pooling layer was replaced by the residual connection, and also the additional 1x1 convolution before addition.*

The pooling operation inside the main inception modules was replaced in favor of the residual connections. However, you can still find those operations in the reduction blocks. Reduction block A is same as that of Inception v4.

**

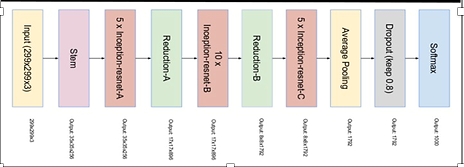
*(From Left) Reduction Block A (35x35 to 17x17 size reduction) and Reduction Block B (17x17 to 8x8 size reduction). Refer to the paper for the exact hyper-parameter setting (V,l,k).*

Networks with residual units deeper in the architecture caused the network to “die” if the number of filters exceeded 1000. Hence, to increase stability, the authors scaled the residual activations by a value of around 0.1 to 0.3.



Findings:

* Activations are scaled by a constant to prevent the network from dying.
* It was found that Inception-ResNet models were able to achieve higher accuracies at a lower epoch.
* The final network layout for both Inception v4 and Inception-ResNet are as follows:



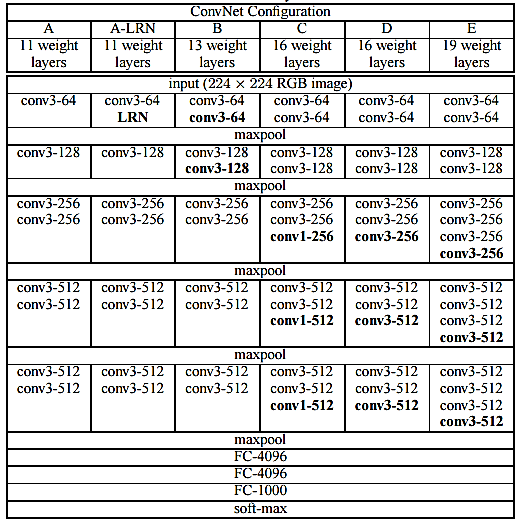
Properties:

* Modules A, B and C are similar.
* Each branch starts with a 1x1 convolution on the input.
* All branches merge into one 1x1 convolution (which is then added to the original input, as usually in residual architectures).
* Module A uses 3x3 convolutions, B 7x1 and 1x7, C 3x1 and 1x3.
* The reduction modules also contain multiple branches. One has max pooling (3x3 stride 2), the other branches end in convolutions with stride 2.

VGG 16 & 19 [[2](#Link_2)]:

This architecture is from VGG group, Oxford. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3X3 kernel-sized filters one after another. With a given receptive field (the effective area size of input image on which output depends), multiple stacked smaller size kernel is better than the one with a larger size kernel because multiple non-linear layers increases the depth of the network which enables it to learn more complex features, and that too at a lower cost.

The network architecture is given in the table.

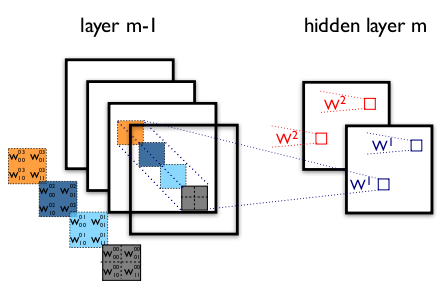


You can see that in VGG-D and VGG-E, there are blocks with same filter size applied multiple times to extract more complex and representative features. This concept of blocks/modules became a common theme in the networks after VGG.

The VGG convolutional layers are followed by 3 fully connected layers. The width of the network starts at a small value of 64 and increases by a factor of 2 after every sub-sampling/pooling layer. It achieves the top-5 accuracy of 92.3 % on ImageNet. The numbers ‘16’ and ‘19’ that are included in the names, represent the number of weight layers in the network. Deployment of VGG networks is tedious due to inherently slow training speeds and large architecture weights.

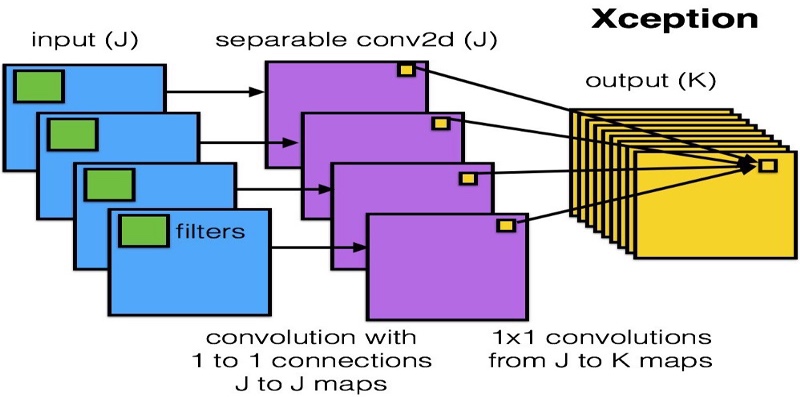
Xception [[3](#Link_3)]:

Xception stands for “extreme inception.” And, as the name suggests, it takes the principles of Inception to an extreme. The hypothesis is: “cross-channel correlations and spatial correlations are sufficiently decoupled that it is preferable not to map them jointly.” In a traditional convolution network, convolutional layers seek out correlations across both space and depth. Let us take another look at our standard convolutional layer:



In the image above, the filter simultaneously considers a spatial dimension (each 2x2 colored square) and a cross-channel or “depth” dimension (the stack of four squares). At the input layer of an image, this is equivalent to a convolutional filter looking at a 2x2 patch of pixels across all three RGB channels. Here’s the question: is there any reason we need to consider both the image region and the channels at the same time?

In Inception, we began separating the two slightly. We used 1x1 convolutions to project the original input into several separate, smaller input spaces, and from each of those input spaces, we used a different type of filter to transform those smaller 3D blocks of data. Xception takes this one step further. Instead of partitioning input data into several compressed chunks, it maps the spatial correlations for each output channel separately and then performs a 1x1 depth wise convolution to capture cross-channel correlation.



The author notes that this is essentially equivalent to an existing operation known as a “depth wise separable convolution,” which consists of a depth wise convolution (a spatial convolution performed independently for each channel) followed by a pointwise convolution (a 1x1 convolution across channels). We can think of this as looking for correlations across a 2D space first, followed by looking for correlations across a 1D space. Intuitively, this 2D + 1D mapping is easier to learn than a full 3D mapping.

Xception exercises the smallest weight serialization at only 91MB. Xception, when trained on ImageNet dataset for image classification significantly outperformed InceptionV3, due to the efficient usage of model parameters and incorporation of depthwise separable convolution operation instead of Inception modules. Most importantly, it has the same number of model parameters as Inception, implying a greater computational efficiency.

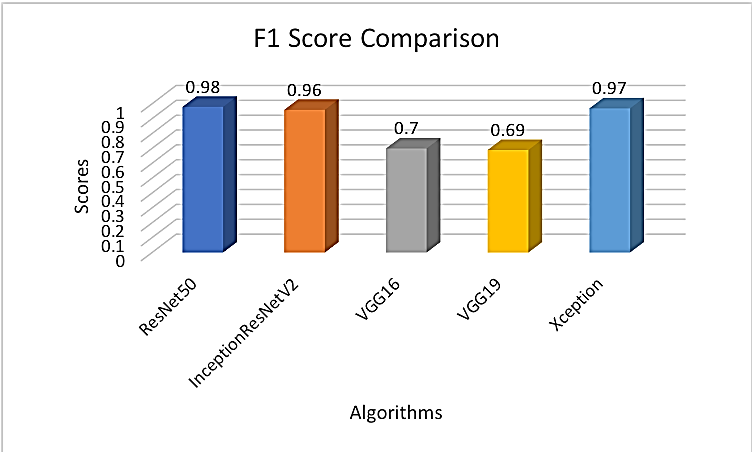
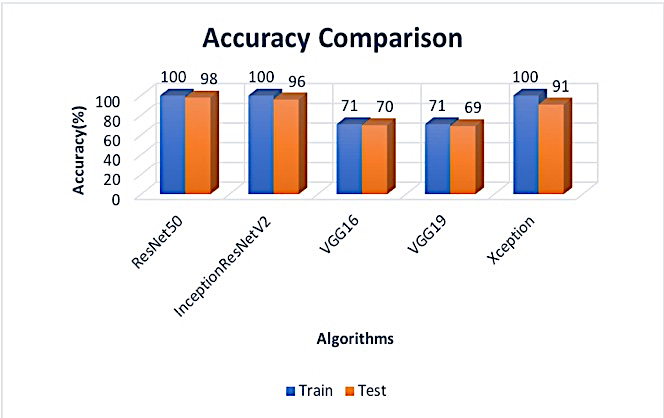
Methods:

For Transfer learning, I imported the pre-defined ResNet50, InceptionResNetV2, VGG16, VGG19, Xception algorithms from the keras library and trained them on the datasets. Then I created the final layer according to the number of classes for each dataset and fit the models on the dataset to get the final results for them. The implementation can be seen on this link: https://github.com/adbhutsangal/CSYE7245

Results:

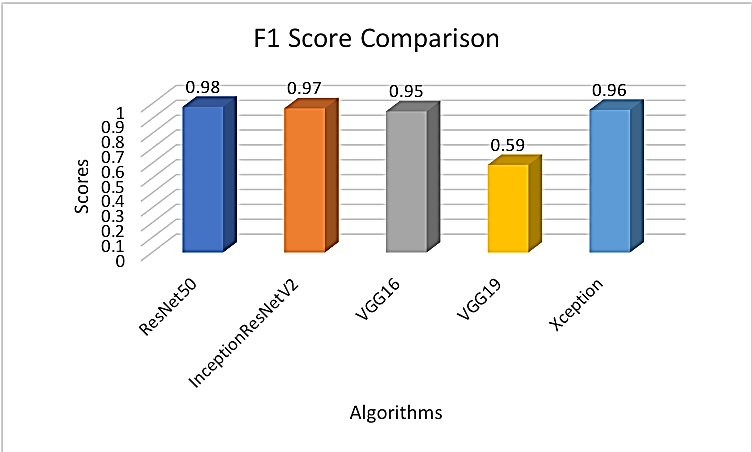
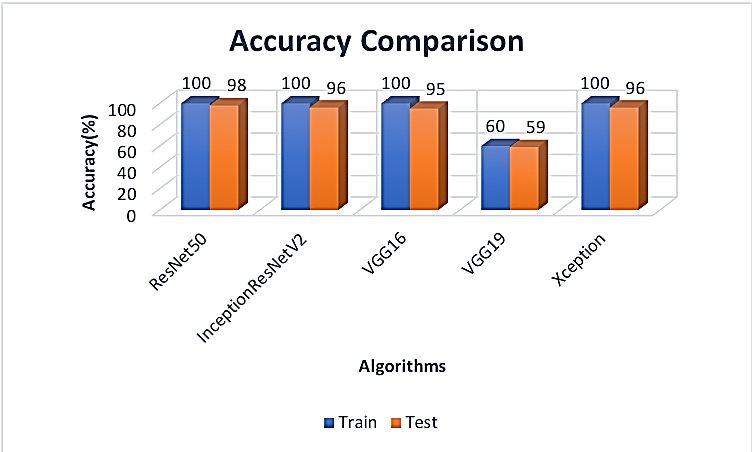
Apple Dataset:



****

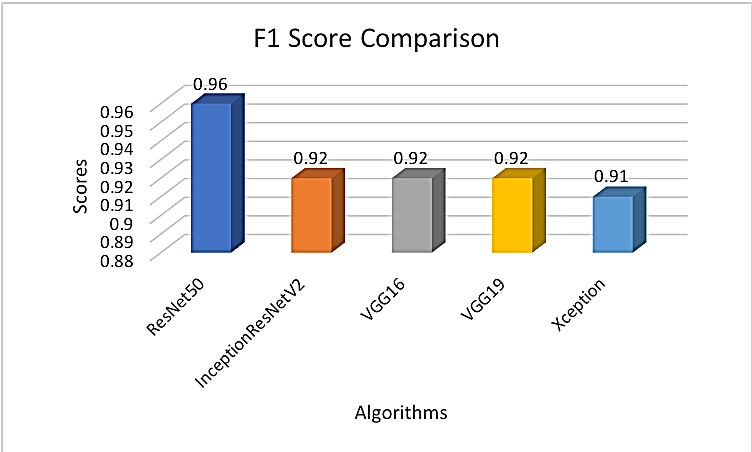
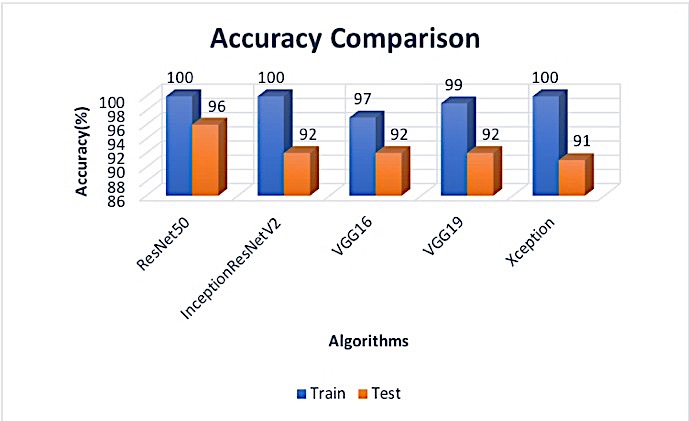
Grapes Dataset:



****

**Tomato Dataset:**



****

Discussion:

In this way we can implement transfer learning by importing the pre-defined models, training them on our datasets, creating final layer according to the number of classes and getting the results. By analyzing our results, we can choose which model will be best for transfer learning and implement that on other datasets.

In our study, the ResNet50 model has the better accuracy and F1 score. This is because, the addition of the residual network blocks and the shortcut connections helped reduce the vanishing gradient problem, which makes it effective. Moreover, we can also implement Inception\_ResNet\_V2, which has given similar results, but because of additional computation, it may take a little more time, and there is a probability of overfitting, especially on the larger datasets. VGG16 and VGG 19 are both heavy yet not as effective because of the comparatively less number of layers. Xception model has given great results on the smaller datasets but wasn’t as effective on the larger datasets. This is because, like Inception it is doing additional computation by performing multiple convolutions, which results in decrease in efficiency.

Conclusion:

ResNet50 comes out to be the best model as it has the better accuracy and F1 score as evident from the above results and graphs. In the future, more models like this can be implemented, as right now, only limited number of models are present in the keras library. Also, because of computational limitations, I only worked with limited number of activation functions (softmax) and optimizers (SGD). Hence, changes in hyperparameters (e.g., number of epochs, batch size, optimizers, etc.) can be explored for better results. Moreover, a mobile application can also be developed based on this project that can help the farmers detect the crop disease at an early stage and prevent it. This project can also be trained on datasets other than fruits and their disease identification, for example, identification of cats, dogs, etc. and their breeds.

References:

1. An Intuitive Explanation of Convolutional Neural networks: <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>
2. Understanding various architectures of Convolutional Networks:   
   <https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/>
3. An Intuitive Guide to Deep Network Architectures:  
   <https://towardsdatascience.com/an-intuitive-guide-to-deep-network-architectures-65fdc477db41>
4. A Simple Guide to the Versions of the Inception Network:  
   <https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>
5. Keras: The Python Deep Learning library:   
   <https://keras.io/>
6. Wikipedia:  
   <https://www.wikipedia.org>
7. **Code source:**

<https://github.com/YashG2002/Tomato-Disease-Detection>

**License:**

MIT License

Copyright (c) 2018 Adbhut Sangal

Permission is hereby granted, free of charge, to any person obtaining a copy

of this software and associated documentation files (the "Software"), to deal

in the Software without restriction, including without limitation the rights

to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

copies of the Software, and to permit persons to whom the Software is

furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

SOFTWARE.

Context:

For this project, I found the implementation of the ResNet50 algorithm on the Tomato dataset in the above-mentioned code source. I then had to optimize the code, as it had a lot of redundant code that needed to be removed and provided comments for the implementation of a lot of functions, explaining the code. Then I used this optimized code to create and implement code for other pre-defined algorithms (Inception\_ResNet\_V2, VGG16, VGG19, Xception). I then implemented all the five algorithms on the Apple, Grapes and Tomato datasets. Finally, using the results I got, I did an analysis on which algorithm was better for implementing transfer learning and what can be the possible reasons for it.

Terminologies:

1. **Batch size**: the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.
2. **Epochs**: number of epochs where one epoch = one forward pass and one backward pass of all the training examples.
3. **Activation function**: It’s just a thing (node) that you add to the output end of any neural network. It is also known as Transfer Function. It can also be attached in between two Neural Networks. It is used to determine the output of the neural network like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1, etc.(depending upon the function). e.g., relu, tanh, softmax, etc.
4. **Dropout**: Dropout is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks. The term "dropout" refers to dropping out units (both hidden and visible) in a neural network.
5. **Compile loss**: A loss function (or objective function, or optimization score function) is one of the two parameters required to compile a model, helps with the calculation of model losses. e.g., mse, cross-entropy, etc.
6. **learning rate**: Learning rate is a hyperparameter that controls how much we are adjusting the weights of our network with respect to the loss gradient. The lower the value, the slower we travel along the downward slope. While this might be a good idea (using a low learning rate) in terms of making sure that we do not miss any local minima, it could also mean that we’ll be taking a long time to converge — especially if we get stuck on a plateau region. Also, if we take a higher value of learning rate, it might result in overfitting of the model as less number of points will be considered.
7. **optimizer**: An optimizer is one of the two arguments required for compiling a model, helps with the calculation of gradients for the model. e.g., SGD, Adadelta, etc.
8. **Image Processing**: Image processing is the area of computer science that deals with the analysis, enhancement, and manipulation of digital images for feature extraction, recognition and classification purposes.
9. **Precision**: It is a statistical measure of random errors. It is the ratio of valid outputs also known as true positives to retrieved samples only.
10. **Recall**: Similar to precision, it is also a statistical measure. It is the ratio of valid outputs to total number of relevant samples.
11. **F1 Score**: Given Precision and Recall values, F1 score is computed as the harmonic mean of both.
12. **Plant Pathology**: It is the study of diseases in plants and their causal factors such as environmental conditions, pathogens etc., that affect the overall plant growth.