

Fruit recognition from images using deep learning

Definition

Project Overview

This project comes from the following Kaggle Fruits dataset : <https://www.kaggle.com/moltean/fruits>

There are an increasing number of robotics applications aimed at detecting fruits from images or videos. Although various research efforts have been made in this field, challenges still remain for complex scenes with varying lighting conditions, low contrast between fruits and leaves, foreground occlusions and cluttered backgrounds. Most of these applications have been to find the fruits for automatic harvesting. A recently new direction is to find the fruits for plant breeding purposes to automatically recognize, count and measure the fruits in order to assess the differences in quality of the genetic material. When the measurements are made by a computer, this is often referred to as digital phenotype and the field is growing in importance.

Fruits have certain categories that are hard to differentiate, like the citrus genus, that contains oranges and grapefruits.

Historical information relevant to the project is that this project started with small dataset about five hundred images but now it become large about 50000 images, it also implemented by ACTA UNIV. SAPIENTIAE , INFORMATICA.

The project that has the target of obtaining a classifier that can identify a much wider array of objects from images. This fits the current trend of companies working in the augmented reality field. During its annual I/O

conference, Google announced that is working on an application named Google Lens which will tell the user many useful information about the object toward which the phone camera is pointing. First step in creating such application is to correctly identify the objects. The software has been released later in 2017 as a feature of Google Assistant and Google Photos apps. Currently the identification of objects is based on a deep neural network [[link above](#)].

Such a network would have numerous applications across multiple domains like autonomous navigation, modeling objects, controlling processes or human-robot interactions. The area we are most interested in is creating an autonomous robot that can perform more complex tasks than a regular industrial robot. An example of this is a robot that can perform inspections on the aisles of stores in order to identify out of place items or understocked shelves. Furthermore, this robot could be enhanced to be able to interact with the products so that it can solve the problems on its own. An other area in which this research can provide benefits is autonomous fruit harvesting. While there are several papers on this topic already, from the best of our knowledge, they focus on few species of fruits or vegetables.

Problem Statement

this project aims to classify the types of 81 different fruits by there images using convolution neural network (CNN). Thus we want to see how well can an artificial intelligence complete the task of classifying them. Another reason is that fruits are very often found in stores, so they serve as a good starting point for the previously mentioned project.

the solution is to use deep learning technique by using convolution neural network (CNN). Each level learns to transform its input data into a slightly more abstract and composite representation. deep neural networks specifically convolutional neural networks have been proved to obtain great results in the field of image recognition.

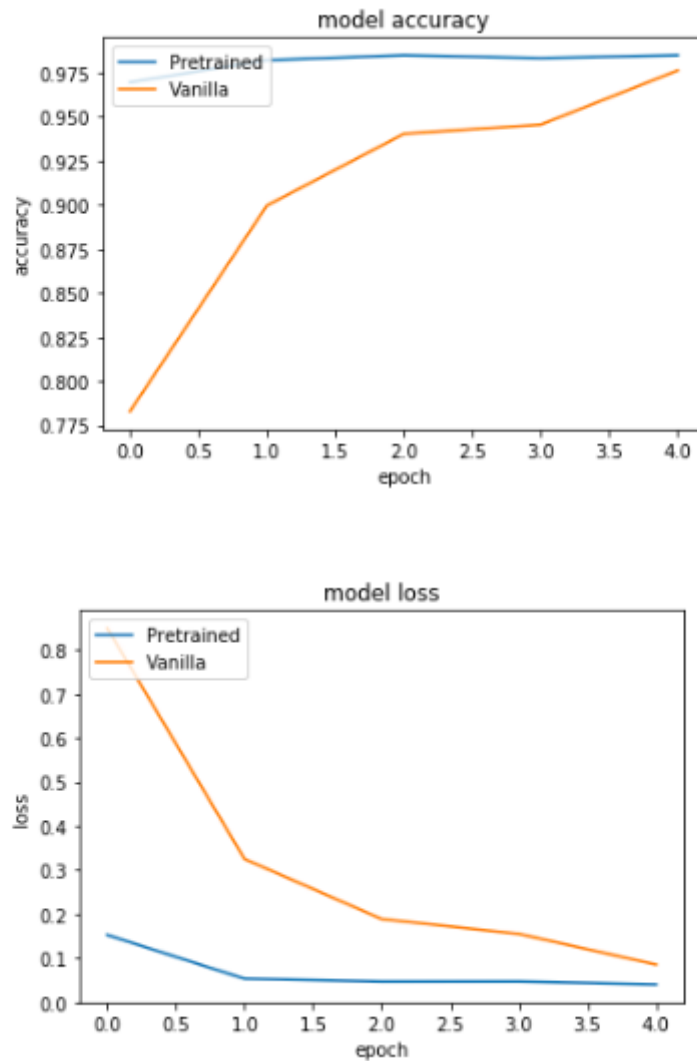
Metrics

This problem is a multi classification problem, Accuracy will be used as the evaluation metric between the solution model and the benchmark model. This problem is a multi classification problem, Accuracy will be used as the evaluation metric between the solution model and the benchmark model because the used dataset has on balance. Accuracy is defined by

“Accuracy = number of correctly identified instances ÷ all instances”

I depend on the result of the Evaluation metrics

Activity	Benchmark	Solution statement
Size of data	About 50000 images	About 53250 images
Quantify the performance of both	The accuracy reach to over 94.54 %	Hope to reach the accuracy over 94%
Number of epochs	Use four epochs	Use about five epochs



Analysis

Data Exploration

The dataset has about 53250 images of fruits spread across 81 labels.

The data set is available on GitHub and Kaggle. The dataset is balanced and the project doesn't have any unbalanced dataset because of the equality of the data and each-other all types of data have the same number of images which enter the model to be trained and tested.

the dataset are available and open source on [[Fruit-360](#)] ,i can work on kaggle workbench, it gives 6 hours per session so can work on it and implement the dataset from it. The characteristics of the dataset that every

type of the fruits have at least 420 images to train and 160 images to test from all directions and different kind of the same fruit. The dataset is the main component in the project it's divided in two sections train data and test data so their use is appropriate given the context of the problem.

Some historical information about dataset (The images were obtained by filming the fruits while they are rotated by a motor and then extracting frames. Fruits were planted in the shaft of a low speed motor (3 rpm) and a short movie of 20 seconds was recorded. Behind the fruits we placed a white sheet as background.

In the figure above :

Left-side: original image. Notice the background and the motor shaft.

Right-side: the fruit after the background removal and after it was scaled down to 100x100 pixels. The images then scaled them to 100x100 pixels images.



The labels and the number of images for training are given in Table1.

Table 1: Number of images for each fruit. There are multiple varieties of apples each of them being considered as a separate object. We did not find the scientific/popular name for each apple so we labeled with digits (e.g. apple red 1, apple red 2 etc).

Label	Number of training images	Number of test images
Avocado	427	143
Avocado ripe	491	166
Banana	490	166
Banana Red	490	166
Cactus fruit	490	166
Cantaloupe 1	492	164
Cantaloupe 2	492	164
Carambula	490	166
Cherry 1	492	164
Cherry 2	738	246
Cherry Rainier	738	246
Cherry Wax Black	492	164
Cherry Wax Red	492	164
Cherry Wax Yellow	492	164
Clementine	490	166
Cocos	490	166
Dates	490	166
Granadilla	490	166
Grape Pink	492	164
Grape White	490	166
Grape White 2	490	166
Grapefruit Pink	490	166
Grapefruit White	492	164
Guava	490	166
Huckleberry	490	166
Kaki	490	166
Kiwi	466	56
Kumquats	490	166

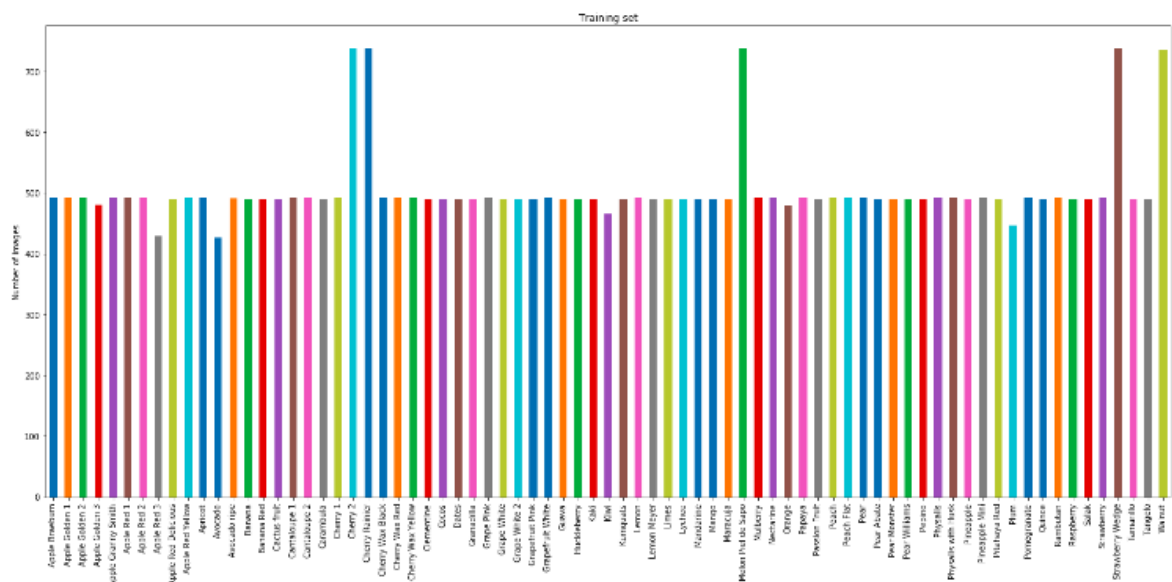
Lemon	492	164
Lemon Meyer	490	166
Limes	490	166
Lychee	490	166
Mandarine	490	166
Mango	490	166
Maracuja	490	166
Melon Piel de Sapo	738	246
Mulberry	492	164
Nectarine	492	164
Orange	479	160
Papaya	492	164
Passion Fruit	490	166
Peach	492	164
Peach Flat	492	164
Pear	492	164
Pear Abate	490	166
Pear Monster	490	166
Pear Williams	490	166
Pepino	490	166
Physalis	492	164
Physalis with Husk	492	164
Pineapple	490	166
Pineapple Mini	493	163
Pitahaya Red	490	166
Plum	447	151
Pomegranate	492	164
Quince	490	166
Rambutan	492	164
Raspberry	490	166
Salak	490	162
Strawberry	492	164
Strawberry Wedge	738	246

Tamarillo	490	166
Tangelo	490	166
Walnut	735	249

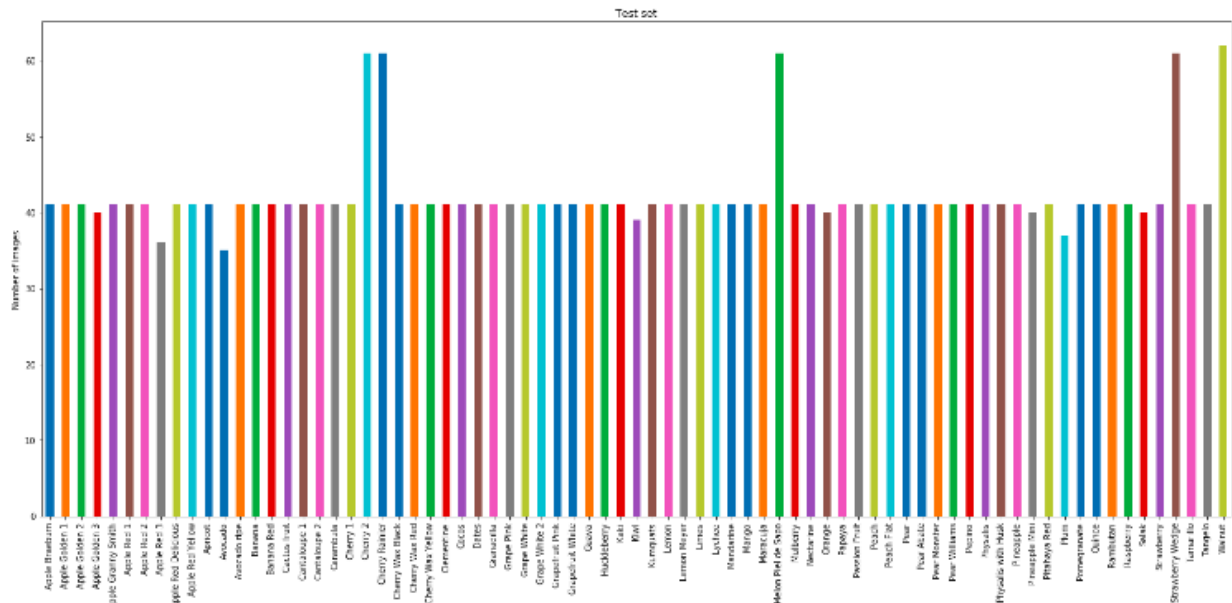
Exploratory Visualization

The relevant characteristic of the used dataset is all of them present the type of fruits and not any other object in the dataset exact fruits, secondly the feature or pattern in all dataset is that any image can contain only one fruit so it's easy to classify between them, third each type of fruits have a folder with it's images in the training set and test set, each folder of them contain the shape of the fruit from all direction to be completely described.

The third point is the most important on because this act the fruit like a 3D so it make the model be more accurate when detect the type of the fruit. so if the plot is provided form any folder of the dataset it can clearly defined the axis of any image are the same and the title of the fruit clearly. We will make bar plots of the number of fruits in each class for each set. First, we plot the number of fruits in each class for the training set.



Next, we plot the number of fruits in each class for the test set.



The figure shows that in the training set there are approximately 160 images for each class. Similar to the training there are a few exceptions.

Algorithms and Techniques

Deep learning

In the area of image recognition and classification, the most successful results were obtained using artificial neural networks. These networks form the basis for most deep learning models. Deep learning is a class of machine learning algorithms that use multiple layers that contain nonlinear processing units. Each level learns to transform its input data into a slightly more abstract and composite representation. Deep neural networks have managed to outperform other machine learning algorithms. They also achieved the first superhuman pattern recognition in certain domains. This is further reinforced by the fact that deep learning is considered as an important step towards obtaining Strong AI. Secondly, deep neural networks - specifically convolutional neural networks - have been proved to obtain great results in the field of image recognition.

Convolutional neural networks

Convolutional neural networks (CNN) are part of the deep learning models. Such a network can be composed of convolutional layers, pooling layers, ReLU layers, fully connected layers and loss layers. In a typical CNN architecture, each convolutional layer is followed by a Rectified Linear Unit (ReLU) layer, then a Pooling layer then one or more convolutional layer and finally one or more fully connected layer. A characteristic that sets apart the CNN from a regular neural network is

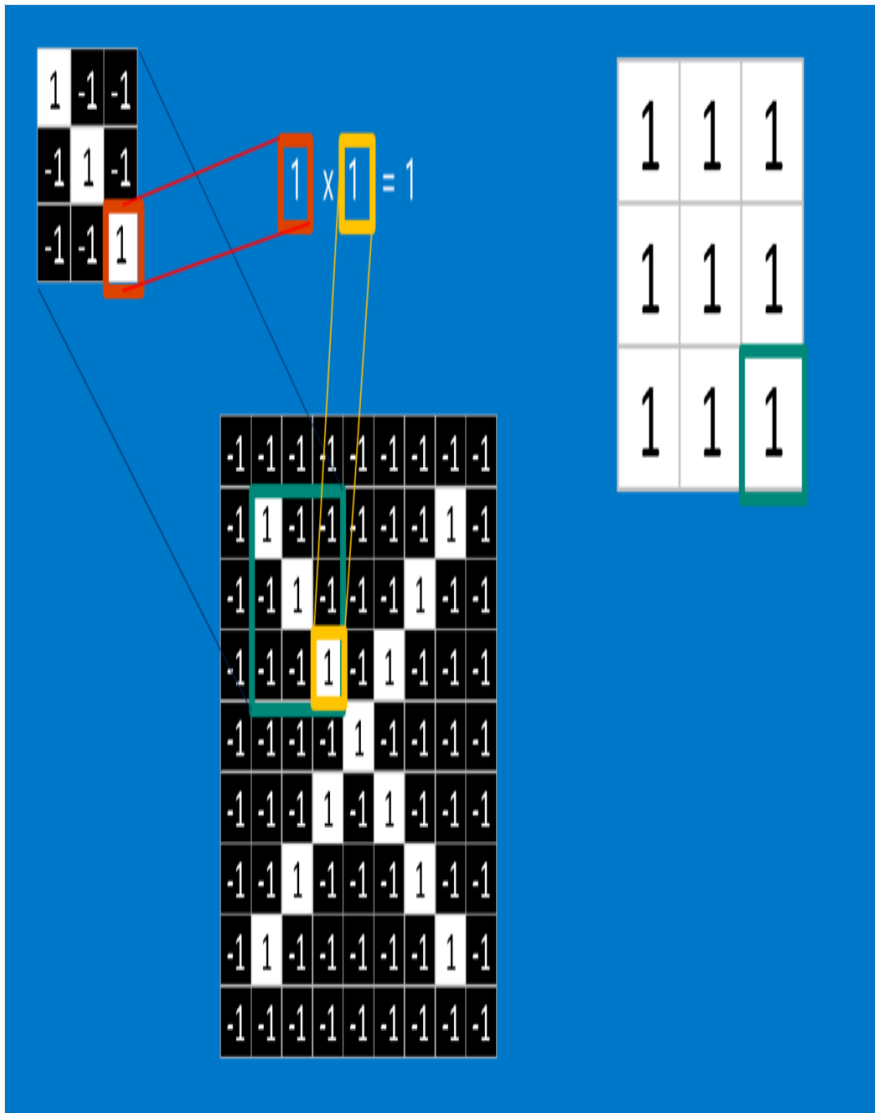
taking into account the structure of the images while processing them. Note that a regular neural network converts the input in a one dimensional array which makes the trained classifier less sensitive to positional changes. Among the best results obtained on the MNIST dataset is done by

using multi-column deep neural networks, they use multiple maps per layer with many layers of non-linear neurons. Even if the complexity of such networks makes them harder to train, by using graphical processors and special code written for them. The structure of the network uses winner-take-all neurons with max pooling that determine the winner neurons.

To a computer, an image looks like a two-dimensional array of pixels (think giant checkerboard) with a number in each position. In our example a pixel value of 1 is white, and -1 is black. When comparing two images, if any pixel values don't match, then the images don't match, at least to the computer.

How the CNN work ?

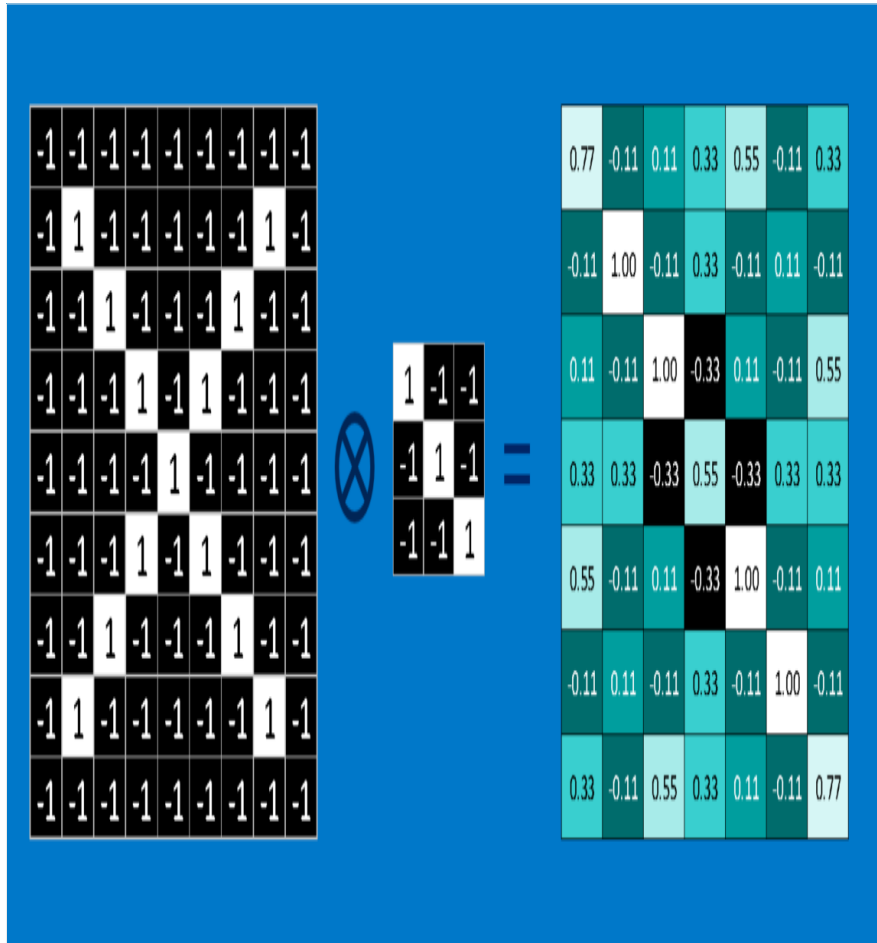
Convolution



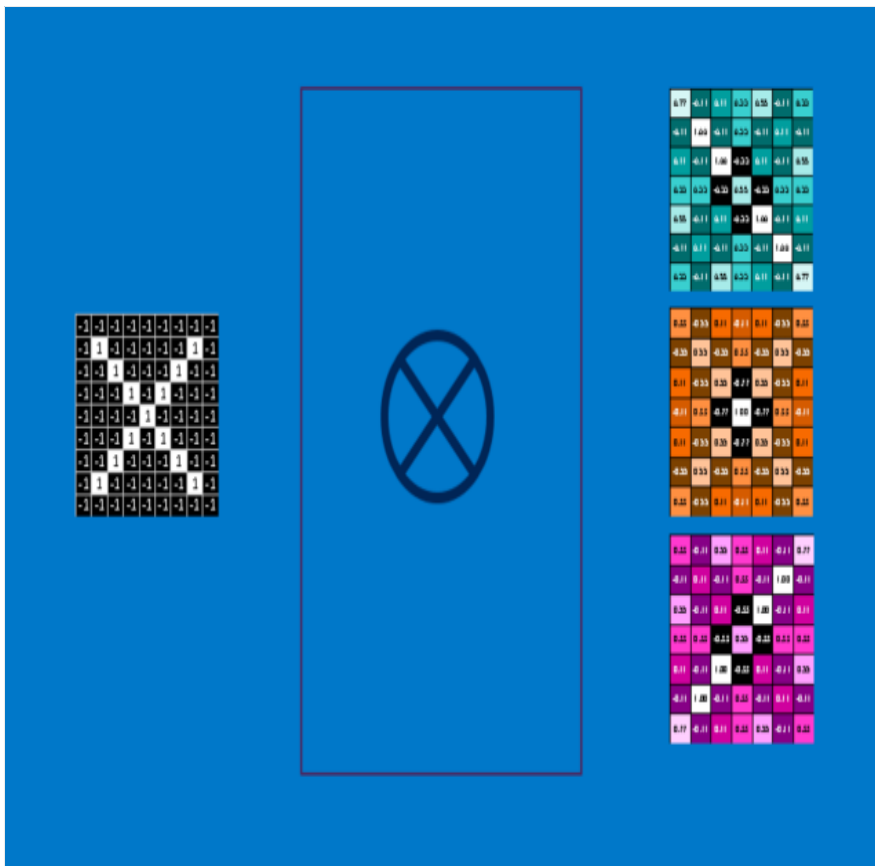
When presented with a new image, the CNN doesn't know exactly where these features will match so it tries them everywhere, in every possible position. In calculating the match to a feature across the whole image, we make it a filter. The math we use to do this is called convolution, from which Convolutional Neural Networks take their name.

The math behind convolution is nothing that would make a sixth-grader uncomfortable. To calculate the match of a feature to a patch of the image, simply multiply each pixel in the feature by the value of the corresponding pixel in the image. Then add up the answers and divide by the total number of pixels in the feature. If both pixels are white (a value of 1) then $1 * 1 = 1$. If both are black, then $(-1) * (-1) = 1$. Either way, every matching pixel

results in a 1. Similarly, any mismatch is a -1. If all the pixels in a feature match, then adding them up and dividing by the total number of pixels gives a 1. Similarly, if none of the pixels in a feature match the image patch, then the answer is a -1.



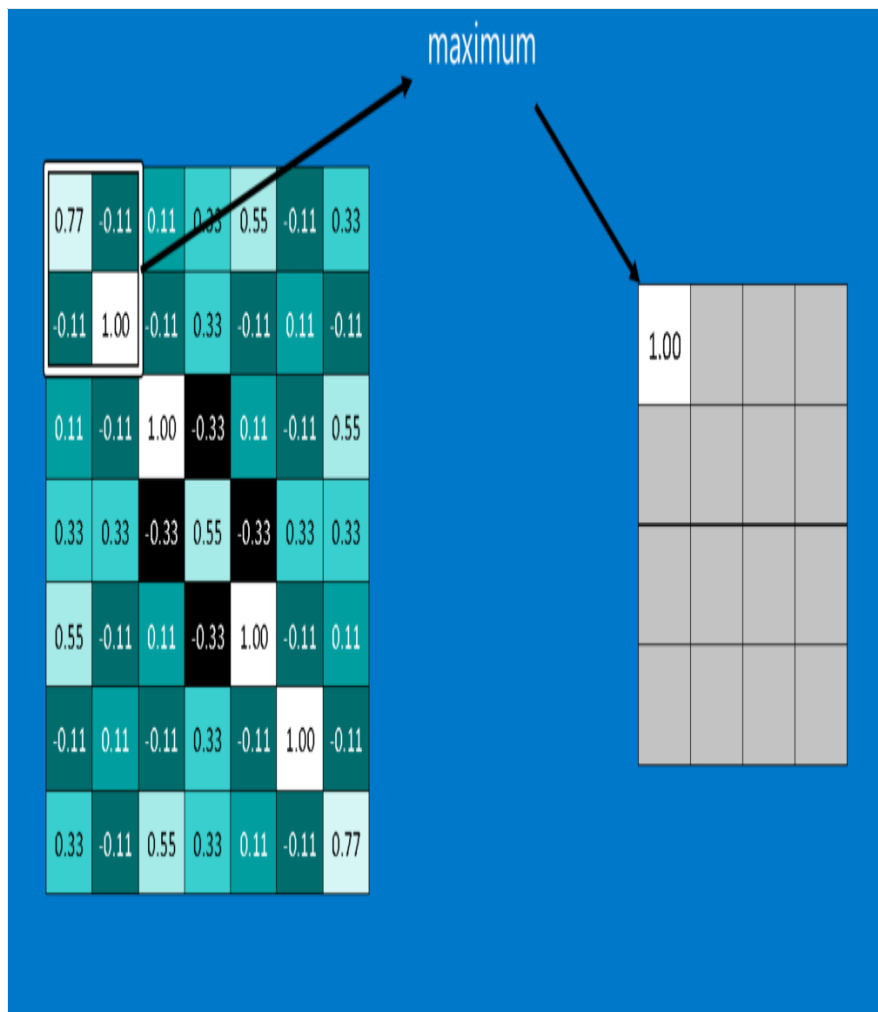
To complete our convolution, we repeat this process, lining up the feature with every possible image patch. We can take the answer from each convolution and make a new two-dimensional array from it, based on where in the image each patch is located. This map of matches is also a filtered version of our original image. It's a map of where in the image the feature is found. Values close to 1 show strong matches, values close to -1 show strong matches for the photographic negative of our feature, and values near zero show no match of any sort.



The next step is to repeat the convolution process in its entirety for each of the other features. The result is a set of filtered images, one for each of our filters. It's convenient to think of this whole collection of convolution operations as a single processing step. In CNNs this is referred to as a convolution layer, hinting at the fact that it will soon have other layers added to it.

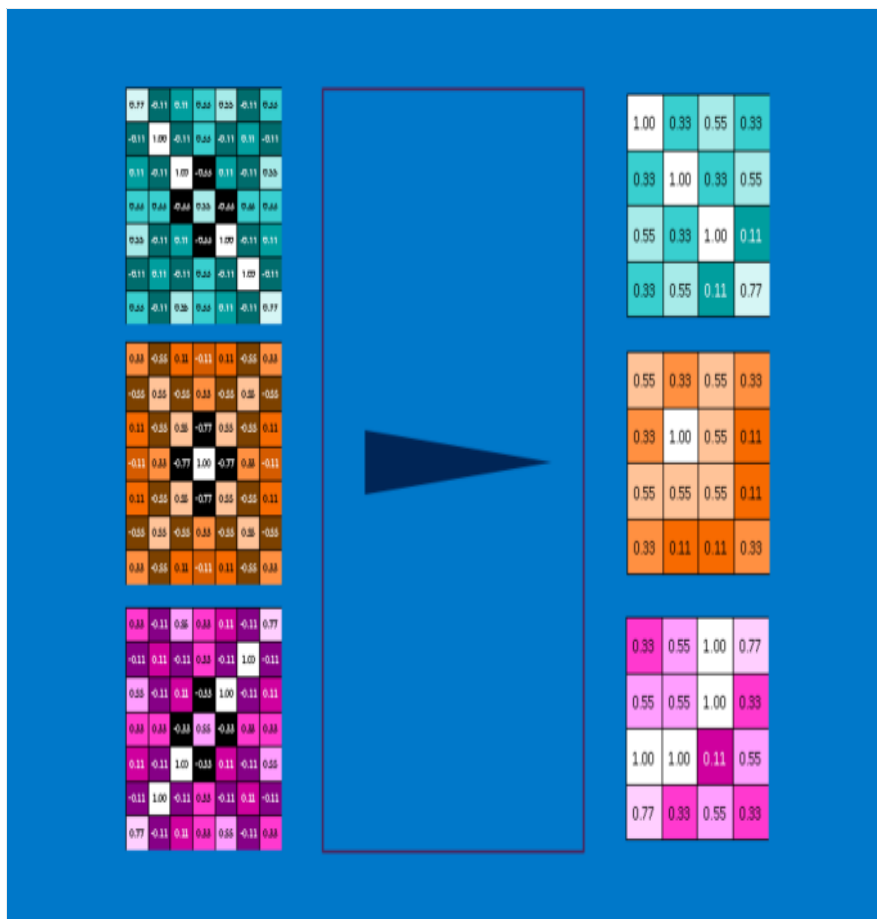
It's easy to see how CNNs get their reputation as computation hogs. Although we can sketch our CNN on the back of a napkin, the number of additions, multiplications and divisions can add up fast. In math speak, they scale linearly with the number of pixels in the image, with the number of pixels in each feature and with the number of features. With so many factors, it's easy to make this problem many millions of times larger without breaking a sweat. Small wonder that microchip manufacturers are now making specialized chips in an effort to keep up with the demands of CNNs.

Pooling



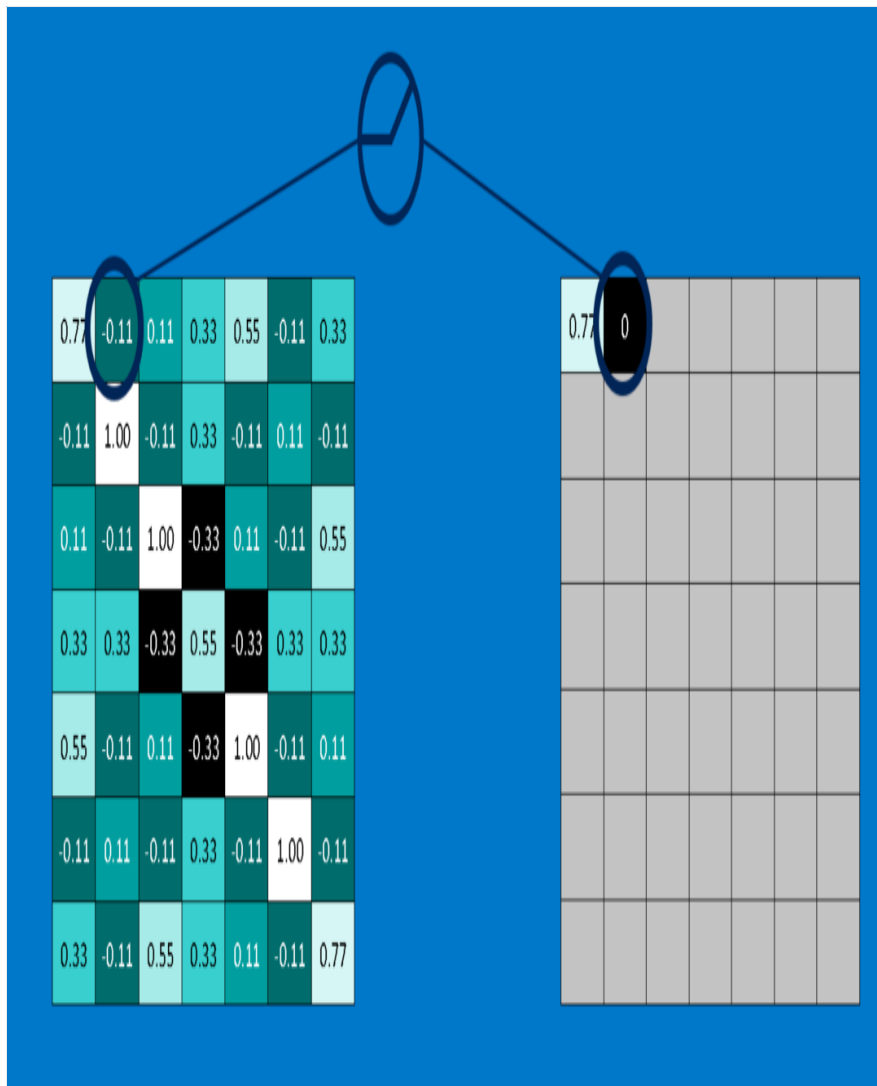
Another power tool that CNNs use is called pooling. Pooling is a way to take large images and shrink them down while preserving the most important information in them. The math behind pooling is second-grade level at most. It consists of stepping a small window across an image and taking the maximum value from the window at each step. In practice, a window 2 or 3 pixels on a side and steps of 2 pixels work well.

After pooling, an image has about a quarter as many pixels as it started with. Because it keeps the maximum value from each window, it preserves the best fits of each feature within the window. This means that it doesn't care so much exactly where the feature fit as long as it fit somewhere within the window. The result of this is that CNNs can find whether a feature is in an image without worrying about where it is. This helps solve the problem of computers being hyper-literal.

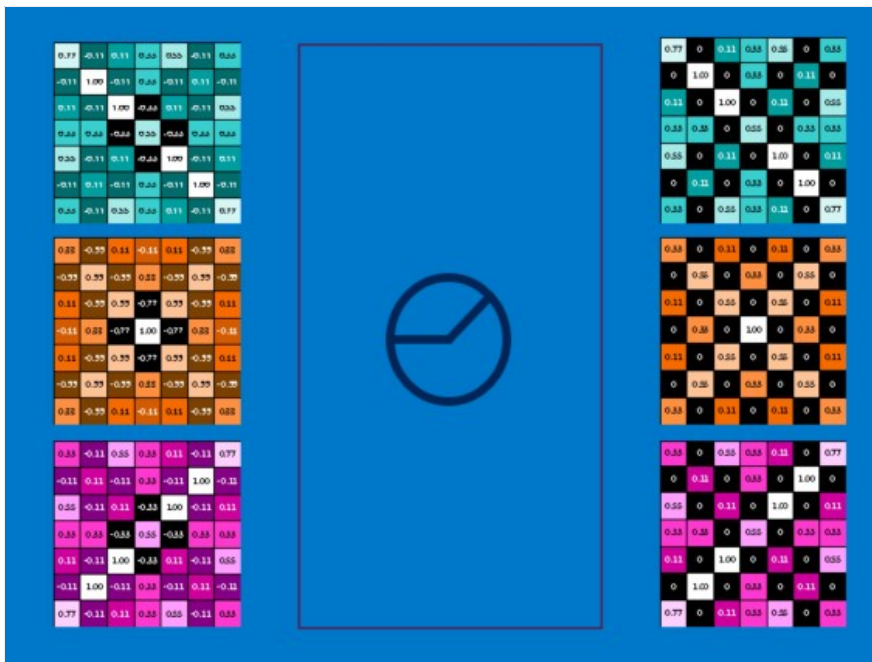


A pooling layer is just the operation of performing pooling on an image or a collection of images. The output will have the same number of images, but they will each have fewer pixels. This is also helpful in managing the computational load. Taking an 8 megapixel image down to a 2 megapixel image makes life a lot easier for everything downstream.

Rectified Linear Units



A small but important player in this process is the Rectified Linear Unit or ReLU. It's math is also very simple—wherever a negative number occurs, swap it out for a 0. This helps the CNN stay mathematically healthy by keeping learned values from getting stuck near 0 or blowing up toward infinity. It's the axle grease of CNNs—not particularly glamorous, but without it they don't get very far.



The output of a ReLU layer is the same size as whatever is put into it, just with all the negative values removed.

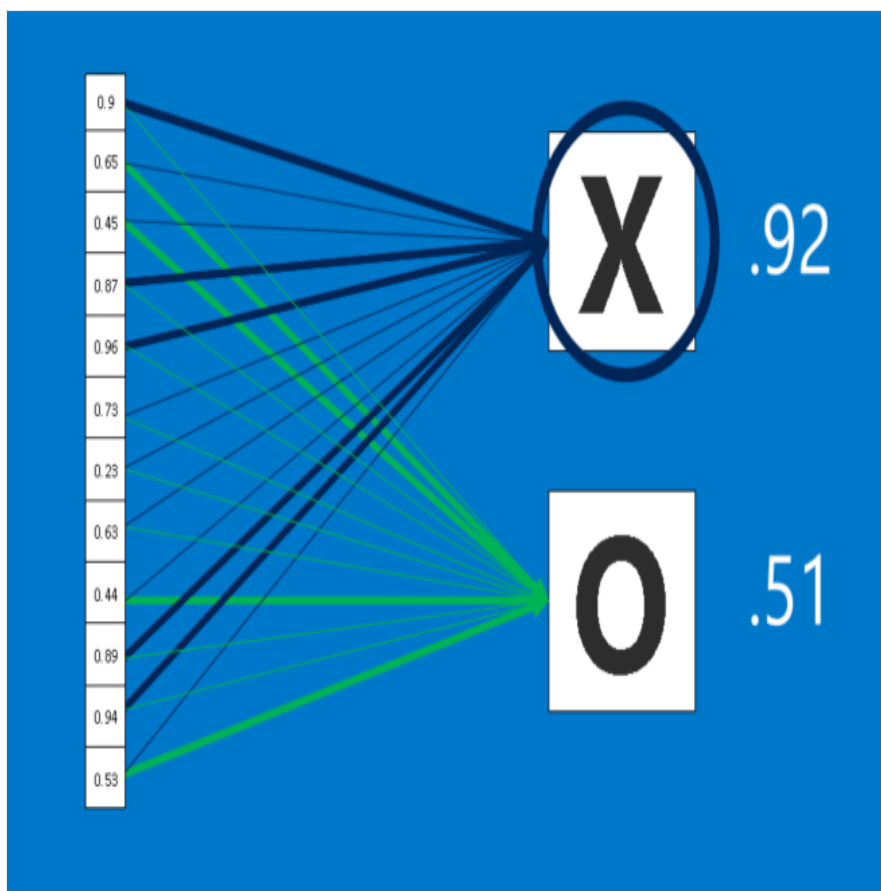
Deep learning



You've probably noticed that the input to each layer (two-dimensional arrays) looks a lot like the output (two-dimensional arrays). Because of this, we can stack them like Lego bricks. Raw images get filtered, rectified and pooled to create a set of shrunk, feature-filtered images. These can be filtered and shrunk again and again. Each time, the features become

larger and more complex, and the images become more compact. This lets lower layers represent simple aspects of the image, such as edges and bright spots. Higher layers can represent increasingly sophisticated aspects of the image, such as shapes and patterns. These tend to be readily recognizable. For instance, in a CNN trained on human faces, the highest layers represent patterns that are clearly face-like.

Fully connected layers



CNNs have one more arrow in their quiver. Fully connected layers take the high-level filtered images and translate them into votes.. Fully connected layers are the primary building block of traditional neural networks. Instead of treating inputs as a two-dimensional array, they are treated as a single list and all treated identically. These get larger votes than the others. These votes are expressed as weights, or connection strengths, between each value and each category.

When a new image is presented to the CNN, it percolates through the lower layers until it reaches the fully connected layer at the end. Then an election is held. The answer with the most votes wins and is declared the category of the input.

Fully connected layers, like the rest, can be stacked because their outputs (a list of votes) look a whole lot like their inputs (a list of values). In practice, several fully connected layers are often stacked together, with each intermediate layer voting on phantom “hidden” categories. In effect, each additional layer lets the network learn ever more sophisticated combinations of features that help it make better decisions.

Benchmark Model

I plan to compare the results of the CNN model which I implement with the result of the same project on kaggle ([linked to above](#)). I will compare the model accuracy/model loss and number of epochs used each other to see which is more effective, as well as compare the speed of the two techniques (after training, in the case of the CNN). In this project the accuracy reached to 94.54 %.

I will also consider the paper of the project is as a benchmark model to my project and it's accuracy. In the [paper](#) the accuracy reached to 96.19%.

Methodology

Data preprocessing

The targeted plants are peppers with fruits of complex shapes and varying colors similar to the plant canopy. The aim of the application is to locate and count green and red pepper fruits on large, dense pepper plants growing in a greenhouse. The training and validation data used consists of 28000 images of over 1000 plants and their fruits. The used method to locate and count the peppers is two-step :

in the first step, the fruits are located in a single image and in a second step multiple views are combined to increase the detection rate of the fruits. For this purpose the authors adapt a Faster Region-based convolutional network. The objective is to create a neural network that would be used by autonomous robots that can harvest fruits. The network is trained using RGB and NIR (near infra red) images. The combination of the RGB and NIR models is done in 2 separate cases: early and late fusion. Early fusion implies that the input layer has 4 channels :

3 for the RGB image and one for the NIR image. Late fusion uses 2 independently trained models that are merged by obtaining predictions from both models and averaging the results. The result is a multi modal network which obtains much better performance than the existing networks.

Implementation

The approach to find the pepper fruits in a single image is based on a combination of (1) finding points of interest, (2) applying a complex high dimensional feature descriptor of a patch around the point of interest and (3) using a so-called bag-of-words for classifying the patch.

The first of all we import the required libraries to be used, the libraries are 1- numpy : this library for linear algebra calculation in python, 2- pandas : this library for data processing, CSV file I/O (e.g. pd.read_csv), 3- matplotlib : this library for A numerical plotting library. It is very useful for any data scientist or any data analyzer.

Import of keras libraries (models – layers – optimizers – utils – preprocessing ,) This library: wraps Keras models requiring just **one line changed** to try out *Importance Sampling* , comes with modified Keras examples for quick and dirty comparison, is the result of ongoing research which means that *mileage may vary. The library uses composition to seamlessly wrap your Keras models and perform importance sampling behind the scenes. Then import the sklearn libraries which use in model selection.*

secondly the phase of load the data so write function to load the data from their folders and load them in arrays to be used and separate them to train set and test set then create the CNN model. The model use sequential model of CNN that can be composed of convolutional layers, pooling layers, ReLU layers, fully connected layers and loss layers. CNN architecture, each convolutional layer is followed by a Rectified Linear Unit (ReLU) layer, then a Pooling layer then one or more convolutional layer and finally one or more fully connected layer.

The initial model consisted of three layers, The first layer is consisted of 16 dense the second is 32 and the third is 64, adding **pooling** layers which are used on one hand to reduce the spatial dimensions of the representation and to reduce the amount of computation done in the network. The other use of pooling layers is to control overfitting. The most used pooling layer has filters of size 2×2 with a stride 2. This effectively reduces the input to a quarter of its original size.

Fully connected layers are layers from a regular neural network. Each neuron from a fully connected layer is linked to each output of the previous layer. The operations behind a convolutional layer are the same as in a fully connected layer. Thus, it is possible to convert between the two.

Loss layers are used to penalize the network for deviating from the expected output. This is normally the last layer of the network. Various lossfunction exist: softmax is used for predicting a class from multiple disjunct classes, sigmoid cross-entropy is used for predicting multiple independent probabilities (from the $[0, 1]$ interval). Then compile the model with loss (categorical cross entropy) , optimizer (Adam) and the metrics=['accuracy']. Then fit the model with the number of epochs which equal to 20 at first, the number of epochs is one of the complications which I face in this project and choose the optimizer also was one of the problems I can recover it by testing all number of them to avoid overfitting so the final model which I hope to reach is near to the benchmark model. Finally get the accuracy of the project.

Refinement

the first solution contained about 20 epochs, when we train it find the accuracy of the solution isn't acceptable it was less than 90% to be more accurate it was about 89.562% because of the over-fitting so in the final model we reduce the number of epochs to 5 epochs which fit will and get a great accuracy which Has become more influential and the accuracy rise to become 94.23% . Another change while compile change the optimizer from rms optimizer to adam which give the best fit and accuracy and rise the accuracy to reach 96.771636 %.

Results

Model Evaluation and Validation

The final model will be the refined CNN model which performed well on the testing set. The parameters of the final model ,We need to calculate the number of images in a batch for different sets. Along with the number of images in a batch, we also need to calculate 'steps per epoch'. This number should be 'total number of images in a set / number of images in a batch'. Lastly, the output of the last layer needs to be calculated. This should equal the number of classes. The Total parameters : 899,105, The Trainable parameters: 897,761 , Non-trainable parameters: 1,344. Number of images per epoch for training equal 41322 samples and validate on 13877 samples. For this project we used a convolutional neural network. As previously described this type of network makes use of convolutional layers, pooling layers, ReLU layers, fully connected layers and loss layers. In a typical CNN architecture, each convolutional layer is followed by a Rectified Linear Unit (ReLU) layer, then a Pooling layer then one or more convolutional layer and finally one or more fully connected layer.

Note again that a characteristic that sets apart the CNN from a regular neural network is taking into account the structure of the images while processing them. A regular neural network converts the input in a one dimensional array which makes the trained classifier less sensitive to

positional changes. The input that we used consists of standard RGB images of size 100 x 100 pixels.

Layer type	Dimensions	Output
Convolutional	5 x 5 x 4	16
Max pooling	2 x 2 — Stride: 2	-
Convolutional	5 x 5 x 16	32
Max pooling	2 x 2 — Stride: 2	-
Convolutional	5 x 5 x 32	64
Max pooling	2 x 2 — Stride: 2	-
Convolutional	5 x 5 x 64	128
Max pooling	2 x 2 — Stride: 2	-
Fully connected	5 x 5 x 128	1024
Fully connected	1024	256
Softmax	256	60

The first layer is a convolutional layer which applies 16 5 x 5 filters. On this layer we apply max pooling with a filter of shape 2 x 2 with stride 2 which specifies that the pooled regions do not overlap. This also reduces the width and height to 50 pixels each. The second convolutional layer applies 32 5 x 5 filters which outputs 32 activation maps. We apply on this layer the same kind of max pooling as on the first layer, shape 2 x 2 and stride 2. The third convolutional layer applies 64 5 x 5 filters. Following is



another max pool layer of shape 2 x 2 and stride 2. The fourth convolutional layer applies 128 5 x 5 filters after which we apply a final max pool layer. Because of the four max pooling layers, the dimensions of the representation have each been reduced by a factor of 16, therefore the fifth layer, which is a fully connected layer, has 5 x 5 x 16 inputs. This layer feeds into another fully connected layer with 1024 inputs and 256 outputs. The last layer is a softmax loss layer with 256 inputs. The number of outputs is equal to the number of classes.

To validate the robustness of this final model I computed the bottleneck features for the new modification in dataset and started training and testing on them. It got 0.9677 training accuracy score and 0.96771636 testing accuracy score which is suitable.

Justification

The solution model outperforms the benchmark model in accuracy of data with 9677% because the benchmark model accuracy is 9454% .

Conclusion

Free-Form Visualization

The dataset was split in 2 parts: training set - which consists of 37836 images of fruits and testing set - which is made of 12709 images.

The data was bundled into a TFRecords file (specific to TensorFlow).

The explanation above is the modification on dataset which I used in the project :









“ This is a binary file that contains protocol buffers with a feature map. In this map it is possible to store information such as the image height, width, depth and even the raw image. Using these files we can create queues in order to feed the data to the neural network. By calling the method `shuf_fle_batch` we provide randomized input to the network. The way we used this method was providing it example tensors for images and labels and it returned tensors of shape batch size x image dimensions and batch size x labels. This helps greatly lower the chance of using the same batch multiple times for training, which in turn improves the quality of the network.

On each image from the batch we applied some preprocessing in order to augment the data set. The preprocessing consists of randomly altering the hue and saturation, and applying random vertical and horizontal flips. For the hue and saturation we use the TensorFlow methods: random hue and random saturation. To further improve the accuracy of the network we converted each image from the batch to grayscale and concatenated it to the image. Thus the data that is fed into the network will have the size 100 x 100 x 4.”

In order to be able to detect fruits from images I used the previously18

described neural network which was trained over 50000 iterations with batches of 50 images selected at random from the training set. Every 50 steps we calculated the accuracy using cross-validation. This showed steady improving of the network until reaching 100% accuracy on cross-validation.

For the testing phase, we used the testing set and the calculated accuracy was 96.771636 %. Some of the incorrectly classified images are given blow Some of the images that were classified incorrectly. On the top we have the correct class of the fruit and on the bottom we have the class that was given by the network and its associated probability

Apple Golden 2 	Apple Golden 3 	Braeburn(Apple) 	Peach 
Apple Golden 3 96.54%	Granny Smith (Apple) 95.22%	Apple Red 2 97.71%	Apple Red Yellow 97.85%
Pomegranate 	Peach 	Pear 	Pomegranate 
Nectarine 94.64%	Apple Red 1 97.87%	Apple Golden 2 98.73%	Braeburn(Apple) 97.21%

Reflection

The entire end-to-end problem solution:

1. Choose the problem and a valid dataset which is already divided among training and testing.
2. Do image preprocessing on the chosen dataset.

3. choose the benchmark model from two approaches first based on the paper of the project, second on the one of the solutions on Kaggle which clearly defined on the benchmark model section.

4. Extract the bottleneck features from the CNN model.

5. Modify the initial model and get higher accuracy

The more interesting and challenging aspects of the project is to apply deep learning techniques and to solve the problem with accuracy higher than the accuracy in the paper of the project.

Difficult aspects in the project was preparing the dataset to be ready, it is already prepared but understanding what happens to it to be ready for use and the algorithms which applied on it was very difficult part. One of the challenges is to choose a strong benchmark model to compare it with the solution model and refining the final model to get best results.

Improvement

I described a new and complex database of images with fruits. Also I made some numerical experiments by using TensorFlow library in order to classify the images according to their content. From my point of view one of the main objectives for the future is to improve the accuracy of the neural network. This involves further experimenting with the structure of the network. Various tweaks and changes to any layers as well as the introduction of new layers can provide completely different results.

Another option is to replace all layers with convolutional layers. This has been shown to provide some improvement over the networks that have fully connected layers in their structure. A consequence of replacing all layers with convolutional ones is that there will be an increase in the number of parameters for the network. Another possibility is to replace the rectified linear units with exponential linear units. This reduces computational complexity and adds significantly better generalization performance than rectified linear units on networks with more than 3

layers. We would like to try out these practices and also to try to find new configurations that provide interesting results.

In the near future i plan to create a mobile application which takes pictures of fruits and labels them accordingly.

Another objective is to expand the data set to include more fruits.