What is this fruit? Neural network application for Vietnamese fruit recognition

Mateusz Baryła[1,*]

[1]Faculty of Applied Mathematics, Wroclaw University of Science and Technology

Abstract. We use deep learning for problems in computer vision, image recognition and classification. Deep learning methods for fruit recognition are built with methods where features (in our case fruits key features) are processed and sent through multiple layers where transformations and computations are done sequentially to form a prediction model. Deep learning algorithms draws inspiration from many fields especially applied maths fundamentals like linear algebra, probability, information theory and numerical optimization. To the best of our knowledge this is the first web application for fruit recognition. Thanks to that users will be able to recognize most of the Vietnamese fruits without knowledge of Vietnamese language. What is more they can get a short description for each fruit and a video how to eat it. In the paper we compare different models of convolutional neural networks in order to find the best possible model of CNN. This system will is fine-tuned, what means that it learns on examples provided by users of application. Having that we propose algorithm which detects non-fruits pictures uploaded by user.

1 Introduction

Vietnam is located in tropical zone and famous with many kinds of fruits. There is a list of Top Tropical Fruits in Vietnam- they are really popular and both foreigners and Vietnamese should try them. Because of my studies which are Applied Mathematics and Computer Science I want to combine my two majors and create a web application with usage of complex math which are in Convolutional Neural Networks. Convolutional neural networks are primarily used to solve image recognition problems. Due to their simple architecture they offer a simple method of getting started with ANNs. AI is not new it actually started over 65 years ago. Today we finally have a cost efficient way to store transfer and compute a massive amount of data in a way that was never possible before.

Over the last several years machine learning techniques, especially when applied to neural networks, have played an increasingly important role in the design of image recognition systems.One of the main reasons for developing *www.whatisthisfruit.info* web application is to help people in Vietnam with fruits recognition by applying artificial intelligence solutions. Using this solution user is does not have to know Vietnamese language in order to ge to know fruits name, actually he or she might use my application, takes the picture and then review results.

*e-mail: 230038@student.pwr.edu.pl

The paper is structured as follows. Firstly I discuss related work obtained for fruits recognition. Next I introduce readers to the mathematical background of artificial neural networks. Later I discuss CNNs and their architecture. After that I present dataset and preprocessing used in preparing images following with results achieved with my different models. I also explain way of fine-tuning model, and architecture of web-application. Last but not least I propose future work and sum up my results.

2 Related work

Recent work in deep neural networks has led to the development of many interesting fruit detection systems.

Paper [1] presents a novel approach to fruit detection using deep convolutional neural networks. It's aim is to build detection system which is accurate, fast and reliable because it is a vital element of an autonomous agricultural robotic platform.In this work model is adapted, through transfer learning, for the task of fruit detection using imagery obtained from two modalities: colour (RGB) and Near-Infrared (NIR). Early and late fusion methods are explored for combining the multi-modal (RGB and NIR) information. In addition to improved accuracy, this approach is also much quicker to deploy for new fruits, as it requires bounding box annotation rather than pixel-level annotation (annotating bounding boxes is approximately an order of magnitude quicker to perform).

Research paper [3] propose a new, high-qality dataset of images containing most popular fruits. The dataset can be downloaded directly from [4] or [5]. It contains 49461 images of 74 fruits. The dataset is special because it doesn't contain noisy background. Authors use C++ programming language to remove background and leave only fruit on a given image.

In other papers researchers tried to detect fruits based on color shape or texture. There is also a method [2] which uses k-nearest neighbor to increase accuracy of fruit recognition.

In my work I am focused on CNNs architecture with the real-life use-case and then making people's lives easier. I've tried many different models of CNN architecture in order to find the best one and then deploy it to public use as a web application. In my work I describe 5 of them which has given the best results. What is more my system is fine-tuned what means that it is going to give better results with time.

3 Artificial neural neurons

Artificial Neural Networks are computational processing systems of which are heavily inspired by way biological nervous systems (such as the human brain operate). ANNs collect their knowledge by finding patterns and relationship in data and learn through experience, not from programming. Every ANNs is formed from hundreds of single units, artificial neurons or processing elements, connected with weights which make the whole neural structure and are organized in layers. In 1943, neurophysiologist Warren McCulloch and mathematician Walter Pitts wrote a paper on how neurons might work. In order to describe how neurons in the brain might work, they modeled a simple neural network using electrical circuits, the first model of a "McCulloch-Pitts neuron". The McCulloch-Pitts model was an extremely simple artificial neuron. The inputs could be either a zero or a one. And the output was a zero or a one.It is a neuron of a set of inputs $I_1, I_2, I_3, ..., I_m$ and one output y. The linear threshold gate simply classifies the set of inputs into two different classes. Thus the output y is binary. Such

a function can be described mathematically using these equations:

$$Sum = \sum_{i=1}^{N} I_i W_i \tag{1}$$

$$y = f(Sum) \tag{2}$$

$W_1, W_2, W_3, \cdots, W_m$ are weight values normalized in the range of either $(0, 1)$ or $(-1, 1)$ and associated with each input line. Sum is the weighted sum, and T is threshold constant. The function is linear step function at threshold T.

The McCulloch-Pitts model of a neuron is simple yet has substantial computing potential. It also has a precise mathematical definition. However, this model is so simplistic that it only generates a binary output and also the weight and threshold values are fixed.

In the first stage, the linear combination of inputs is calculated. Each value of input array is associated with its weight value, which is normally between 0 and 1. Also, the summation function often takes an extra input value θ with weight value of 1 to represent threshold or bias of a neuron. The summation function will be then performed as

$$x = \sum_{i=1}^{N} A_i W_i + \Theta \tag{3}$$

The sum-of-product value is then passed into the second stage to perform the activation function which generates the output from the neuron. The activation function squashes the amplitude the output in the range of $(0, 1)$, or alternately $(-1, 1)$ [14]. The behavior of the activation function will describe the characteristics of an artificial neuron model.

One convenient form of such "semi-linear" function is the logistic sigmoid function, or in short, sigmoid function. As the input x tends to large positive value, the output value y approaches to 1. Similarly, the output gets close to 0 as x goes negative. However, the output value is neither close to 0 nor 1 near the threshold point. This function is expressed mathematically as follows:

$$y = \frac{1}{1 + exp(-x)} \tag{4}$$

Additionally, the sigmoid function describes the "closeness" to the threshold point by the slope. As x approaches to $-\infty$ or ∞, the slope is zero; the slope increases as x approaches to 0. This characteristic often plays an important role in learning of neural networks.

Perhaps, the most primary significance of a neural network is the ability to learn the incoming information and to improve the performance of processing information. The term learning refers to many concepts by various viewpoints, and it is difficult to agree on a precise definition of the term. In neural networks, we define learning as the following sequence of events[14]:

- Stimulation by an environment in which the network is embedded.
- Changes in free parameters of the network as the result of stimulation.
- Responses in a new way to the environment for improved performance.

A learning algorithm is a prescribed set of well-defined rules for learning of a neural network. There are many types of learning algorithms; the common goal of learning is the adjustment of connection weights.

There are two classes of learning: supervised and unsupervised learning. Supervised learning requires an external source of information in order to adjust the network. On the other hand, in unsupervised learning, there is no external agent that overlooks the process of learning. Instead, the network is adjusted through internal monitoring of performance. In this thesis, I deal with supervised learning.

Backpropagation neural networks employ one of the most popular neural network learning algorithms, the Backpropagation (BP) algorithm. Backpropagation is also the most suitable learning method for multilayer networks. Perhaps, the reason why the backpropagation made the major turning point is because the learning rule has a solid mathematical foundation and it is practical. The backpropagation processing unit should be in the form modified from a linear percepton so that the activation function is nonlinear and smoothed out at the threshold point. The suggested form of the activation function is the sigmoid function as mentioned previously. With sigmoid function, we can obtain not only output from the neuron but also information about how close we are to the threshold point using the slope of the sigmoid function. Mathematically, we can derive the slope from the equation as follows:

$$\frac{d}{dx}f(x) = \frac{exp(-x)}{(1+exp(-x))^{-2}} = \frac{1}{1+exp(-x)}\frac{exp(-x)}{1+exp(-x)} =$$
$$= \frac{1}{1+exp(-x)}\left[1 - \frac{1}{1+exp(-x)}\right] = f(x)[1-f(x)] \tag{5}$$

3.0.1 Backpropagation Learning Algorithm

The backpropagation algorithm trains a given feed-forward multilayer neural network for a given set of input patterns with known classifications. When each entry of the sample set is presented to the network, the network examines its output response to the sample input pattern. The output response is then compared to the known and desired output and the error value is calculated. Based on the error, the connection weights are adjusted. The backpropagation algorithm is based on Widrow-Hoff delta learning rule in which the weight adjustment is done through mean square error of the output response to the sample input. The set of these sample patterns are repeatedly presented to the network until the error value is minimized.

Algorithm of backpropagation [15]:

1. Initialize connection weights into small random values
2. Present the pth sample input vector ofpattern $X_p = (X_{p1}, X_{p1}, \cdots, X_{pN_0})$ and the correspodning output target $T_p = (T_{p1}, T_{p2}, \cdots, T_{pN_M})$ to the network
3. Pass the input values to the first layer, layer 1. For every input node i in layer 0, perform: $Y_{0i} = X_{p_i}$
4. For every neuron i in every layer $j = 1, 2, \cdots, M$ from input to output layer find the output from the neuron

$$Y_{ji} = f\left(\sum_{k=1}^{N_{j-1}} Y_{(j-1)k} W_{jik}\right) \tag{6}$$

where

$$f(x) = \frac{1}{1+exp(-x)}$$

5. Obtain output values, for every output node i in layer M, perform:

$$O_{pi} = Y_{Mi}.$$

6. Calculate error value δ_{ji} for every neuron i in every layer in backward order $j = M, M-1, ..., 2, 1$, from output to input layer, followed by weight adjustments. For the output layer, the error value is:

$$\delta_{Mi} = Y_{Mi}(1 - Y_{Mi})(T_{pi} - Y_{Mi}), \tag{7}$$

and for hidden layers:

$$\delta_{ji} = Y_{ji}(1 - Y_{ji}) \sum_{k=1}^{N_{j+1}} \delta_{(j+1)k} W_{(j+1)ki} \tag{8}$$

The weight adjustment can be done for every connection from neuron k in layer $(i-1)$ to every neuron i in every layer i:

$$W_{jik}^{+} = W_{jik} + \beta \delta_{ji} Y_{ji} \tag{9}$$

where β represents weight adjustment factor normalized between 0 and 1.

The actions in steps 2 through 6 will be repeated for every training sample pattern p, and repeated for these sets until the root mean square (RMS) of output errors is minimized. Now attempt to derive the error and weight adjustment equations shown above. Let's begin with the Root Mean Square (RMS) of the errors in the output layer defined as:

$$E_p = \frac{1}{2} \sum_{j=1}^{N_M} (T_{pj} - O_{pj})^2, \tag{10}$$

for the pth sample pattern. In generalized delta rule [15] the error value δ_{ji} associated with the ith neuron in layer j is the rate of change in the RMS error E_p respect to the sum-of-product of the neuron:

$$\delta_{ji} = -\frac{\partial E_p}{\partial net_{ji}}, \tag{11}$$

where net_{ji} represents the sum-of-product value. With the chain rule, we can obtain the rate of change in the RMS error E_p in response to weight change:

$$\frac{\partial E_p}{\partial W_{jik}} = \frac{\partial E_p}{\partial net_{ji}} \frac{\partial net_{ji}}{\partial W_{jik}} = -\delta_{ji} \frac{\partial}{\partial W_{jik}} \left[Y_{(j-1)0} W_{(j-1)i0} + \cdots + Y_{(j-1)} k W_{(j-1)ik} + \cdots \right] =$$
$$= -\delta_{ji} \frac{\partial}{\partial W_{jik}} Y_{(j-1)ik} W_{(j-1)ik} = -\delta_{ji} Y_{(j-1)ik} \tag{12}$$

The weight change is proportional to value above [15].

$$\Delta W_{jik} = \beta \delta_{ji} Y_{(j-1)k} \tag{13}$$

where β is a constant. Because of that, weight change can be performed as:

$$W_{jik}^{+} = W_{jik} + \Delta W_{jik} \tag{14}$$

Now get back to the equation (14) to find an error value associate with the neuron. Again, using the chain rule, we get:

$$\delta_{ji} = -\frac{\partial E_p}{\partial Y_{ji}} \frac{\partial Y_{ji}}{\partial net_{ji}} \tag{15}$$

For output layer $j = M$ and $Y_{Mi} = O_{pi}$. Thus

$$\delta_{Mi} = -\frac{\partial E_p}{\partial O_{pi}} \frac{\partial Y_{Mi}}{\partial net_{Mi}} = -\frac{\partial}{\partial O_{pi}} \left[\frac{1}{2} \left[(T_{P1} - O_{p1})^2 + \cdots + (T_{Pi} - O_{pi})^2 + \cdots \right] \right] \frac{\partial}{\partial net_{M_i}} f(net_{Mi}) =$$

$$= -\frac{\partial}{\partial O_{pi}} \left[\frac{1}{2} \sum_{i=1}^{K} \left(T_{P1} - O_{p1} \right)^2 \right] f'(net_{Mi}) \tag{16}$$

Using equation

$$\delta_{Mi} = (T_{pi} - O_{pi}) \left[f(net_{mi}) \left[1 - f(net_{mi}) \right] \right] = (T_{pi} - O_{pi})(O_{pi})(1 - O_{pi}). \tag{17}$$

For error values associated with the hidden layer neurons, we cannot use target values. For this reason, the part $\frac{\partial E_p}{\partial Y_{ji}}$ in equation needs to be found using a different approach. Next the chain rule applied to the sum-of-product values of neurons in the layer $j + 1$ (front layer)

$$\begin{aligned}
\frac{\partial E_p}{\partial Y_{ji}} &= \frac{\partial E_p}{\partial net_{(j+1)1}} \frac{\partial net_{(j+1)1}}{\partial Y_{ji}} + \frac{\partial E_p}{\partial net_{(j+1)2}} \frac{\partial net_{(j+1)2}}{\partial Y_{ji}} + \cdots = \\
&= \sum_{a=1}^{N_{j+1}} \left[\frac{\partial E_p}{\partial net_{(j+1)a}} \frac{\partial net_{(j+1)a}}{\partial Y_{ji}} \right] = \\
&= \sum_{a=1}^{N_{j+1}} \left[-\delta_{(j+1)a} \frac{\partial}{\partial Y_{ji}} (W_{(j+1)a0} Y_{j0} + \cdots + W_{(j+1)ai} Y_{ji} + \cdots) \right] \\
&= \sum_{a=1}^{N_{j+1}} \left[-\delta_{(j+1)a} \frac{\partial}{\partial Y_{ji}} Y_{ji} W_{(j+1)ai} \right] = \sum_{a=1}^{N_{j+1}} \left[-\delta_{(j+1)a} W_{(j+1)ai} \right]
\end{aligned} \tag{18}$$

Finally, combined with $\frac{\partial Y_{ji}}{\partial net_{ji}}$ the final form of error value associated with the ith neuron in layer j:

$$\begin{aligned}
\delta_{ji} &= \sum_{a=1}^{N_{j+1}} \left[-\delta_{(j+1)a} W_{(j+1)ai} \right] \frac{\partial Y_{ji}}{\partial net_{ji}} = \\
&= Y_{ji}(1 - Y_{ji}) \sum_{a=1}^{N_{j+1}} \left[-\delta_{(j+1)a} W_{(j+1)ai} \right]
\end{aligned} \tag{19}$$

The backpropagation algorithm, as just described, employs gradient descent by following the slope of RMS error value E_p downward along with the change in all the weight values. The weight values are constantly adjusted until the value of E_p is no longer decreasing. Since the RMS error value is very complex function with many parameter values of weights, it is possible that the backpropagation network may converge into a local minima instead of the desired global minimum.

A trained backpropagation network is able to detect and classify an input pattern that has not been seen during learning. This feature is called generalization, borrowed from the psychology terms. Neural networks are known to be good at classifying noisy input patterns, but not at classifying a pattern that is intermediate between two solid patterns from the training samples. In other words, neural networks are good at interpolation but not extrapolation. Also, there may exist overfitted input data, the unseen input pattern such that it can be classified into one of the trained output response undesirably. Suggested solution includes modification of network architecture and more adequate training samples.

4 Convolutional neural networks

Multilayer Neural Networks trained with the backpropagation algorithm constitute the best example of a successful Gradient-Based Learning technique. Convolutional Neural Networks that are specifically designed to deal with the variability of 2D shapes are shown to outperform all other techniques. ConvNet architectures make the explicit assumption that the inputs are images which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network. Convolutional Neural Networks are very similar to ordinary Neural Networks they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a nonlinearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer and all the tips/tricks we developed for learning regular Neural Networks still apply.

4.1 Convolution

The Conv layer is the core building block that does most of the convolutional heavy lifting. Convolution is a mathematical operation on two functions to produce a third function that is typically viewed as a modified version of one of the original functions, giving the integral of the pointwise multiplication of the two functions. Basically every small spatially (along width and height) filter is taken and slide this filter over the whole image. For example a typical filter of a ConvNet might have size 5x5x3(i.e. 5 pixels width and height, and 3 because images have depth 3, the color channels). Pixel values of the image are then multiplied with the pixel values in the filter. Intuitively, the network will learn filters that activate when they see some type of visual feature. Thanks to Stochastic Gradient Descent there is no need to specify them, because they can be learned themselves. Important features of classes can be found at any place because they are initialized randomly and are location-invariant.

4.1.1 Local Connectivity

When dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume. Instead each neuron connects only to a local region of the input volume.

4.1.2 Spatial arrangement

There are three hyperparameters control the size of the output volume: the depth, stride and zero-padding.

- **depth** of the output volume corresponds to the number of filters, each of them should be able to determine different features in image
- **stride** is the number of pixels with which thee filters move one step at a time
- **zero-padding** is a hyperparameter that allows to control the spatial size of the output volumes by padding the input volume with zeros around the border

It is able to compute the spatial size of the output volume as a function of the input volume size **W**, the receptive field size of the Conv Layer neurons (**F**), the stride with which they are applied (**S**), and the amount of zero padding used (**P**) on the border. The correct formula for calculating how many neurons will fit is given by $\frac{W-F+2P}{S+1}$.

4.1.3 Parameter Sharing

Parameter sharing scheme is used in Convolutional Layers to control the number of parameters. The number of parameters at the first ConvNet layer can be reduced by assumption that if the feature is useful to compute at some spatial position (x, y), then it should also be useful to compute at a different position (x_2, y_2). In other words, denoting a single 2-dimensional slice of depth as a depth slice

4.2 Pooling

It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture in order to control overfitting. It is a way to take large images and shrink them down while preserving the most important information in them. Thanks to Max Pooling filters can explore bigger parts of the image, additionally because of the loss in pixels, this is a common practice to increase the number of filters. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation.

4.3 ReLU

The ReLU function is $f(x) = max(0, x)$. Math behind this function is very simple wherever a negative number occurs swap it out for a 0. Purpose of it is usually to speed up training, because the computations are really simple.

4.4 Fully connected layers

Neurons in a fully connected layer take the high-level filtered images and translate them into votes. The output from the convolutional layers represents high-level features in the data. Of course that output can be easy connected to the output layer, but this is a common practice to add a fully connected layer which is a cheap way of learning non-linear combinations of these features and making the model end-to-end trainable. In this ConvNets transform the original image layer by layer from the original pixel values to the final class scores.

4.5 Loss function and training accuracy

The loss function is used to guide the training process of a neural network. For classification problem, mean squared error and cross entropy loss are widely used. Cross entropy loss is a loss function that commonly used in classification or regression problems. Cross entropy

is more advanced than mean squared error. The introduction of cross entropy comes from maximum likelihood estimation in statistics. The formula of cross entropy is

$$H_{y'}(y) = -\sum_i y_i' log(y_i).$$

Where y_i' is the ground truth label of ith training instance and y_i is the prediction result of classifier. Cross entropy is probably the most important loss function in deep learning, it can be seen almost everywhere, but usage of cross entropy can be very different.

5 Architecture of the proposed CNN

Most of the architectures of Convolutional Neural Networks are made up of only three layer types: CONV, POOL and FC. Most ConvNet architecture follows the pattern:

$$INPUT -> [[CONV -> RELU] * n -> POOL] * M -> [FC -> RELU] * k -> FC$$

where * indicates repetition and the $POOL$? indicates an optional pooling layer. Assumption:

$$N >= 0, M >= 0, K >= 0$$

Model 1			Model 2		
layer type	kernel size/ pool size	filters/ strides	layer type	kernel size/ pool size	filters/ strides
Convolutional	(5,5)	16	Convolutional	(5,5)	32
MaxPooling	(2,2)	(2,2)	MaxPooling	(2,2)	(2,2)
Convolutional	(5,5)	32	Convolutional	(5,5)	64
MaxPooling	(2,2)	(2,2)	MaxPooling	(2,2)	(2,2)
Convolutional	(5,5)	64	Convolutional	(5,5)	128
MaxPooling	(2,2)	(2,2)	MaxPooling	(2,2)	(2,2)

Model 3			Model 4		
layer type	kernel size/ pool size	filters/ strides	layer type	kernel size/ pool size	filters/ strides
Convolutional	(5,5)	16	Convolutional	(5,5)	16
MaxPooling	(2,2)	(2,2)	MaxPooling	(2,2)	(2,2)
Convolutional	(5,5)	32	Convolutional	(5,5)	32
MaxPooling	(2,2)	(2,2)	MaxPooling	(2,2)	(2,2)
Convolutional	(5,5)	64	Convolutional	(5,5)	64
MaxPooling	(2,2)	(2,2)	MaxPooling	(2,2)	(2,2)
Convolutional	(5,5)	128			
MaxPooling	(2,2)	(2,2)			

Model 5		
layer type	kernel size/ pool size	filters/ strides
Convolutional	(5,5)	32
MaxPooling	(2,2)	(2,2)
Convolutional	(5,5)	64
MaxPooling	(2,2)	(2,2)

There are no hard guidelines how perfect architecture should look like for given problem. The basic principle which I follow is to keep the feature space wide and shallow in the initial stages of the network and then make it narrower and deeper towards the end. I am using small

filters because I expect small and local feature in picture uploaded by user. I start by using small filters in order to collect as much local information as possible. Then I increase the filter width to reduce the generated feature space width to represent more global information. I don't want to shrink my image too much so the stride (2,2) is used. I keep adding new layers until my model start overfit to the data (model 3). I was trying to use dropout for this model but it hasn't improved results much. I follow advice of classic layers like Conv-Pool Conv-Pool and keeping trend in the number of filters 16-32-64 or 32-64-128.

For model 1 less preprocessing is done. I perform in it only shear ranging, zooming and horizontal flipping. For other model preprocessing described in section 6. The general algorithm of training looks as follows:

- pass an input image to the first convolutional layer. Apply filters in the convolutional layer and extract relevant features from the input image to pass them further
- pass convoluted output as an activation map
- if the image needs to be retained use zero padding
- add pooling layer
- the output layer in CNN is fully connected layer with the number of classes desired by the network
- generate output through the output layer and compare to the output layer for error calculation. Use loss function to compute MSE, then calculate gradient descent
- perform backpropagation

6 Dataset and preprocessing

The first step in process is collecting data for the model to be trained. Having a high-quality data set is essential for obtaining good results from datasets. There are some free great free and paid standard datasets. I used ImageNet unfortunately it contains not only fruits but also noisy background which often leads to misleading classifier. The dataset is completed with dataset from Kaggle [5] in order to collect images of Vietnamese fruits. I preprocess manually every picture from dataset by cropping it to only one fruit. What is more I use Keras library for data augmentation for example for shearing, zooming, flipping both horizontal and vertical, rotating but I don't save them, I only store them in memory during program execution, and as soon as program finishes garbage collector delete them all. Data augmented images are not included in the below table. The dataset was split in 2 parts: training set and testing set.

Figure 1. Apricot

Figure 2. Pineapple

Figure 3. Dragon fruit

Figure 4. Apricot

Figure 5. Pineapple

Figure 6. Dragon fruit

Fruit	Number of training images	Number of test images
Apricot	560	139
Avocado	606	159
Carambola	528	132
Clementine	555	138
Cocos	563	141
Dates	468	182
Dragon Fruit	331	154
Durian	820	205
Guava	567	141
Kaki	735	182
Kiwi	617	154
Longan	281	70
Melon	659	164
Papaya	589	129
Pineapple	727	182
Quince	575	143
Rambutan	167	40
Tamarillo	150	37

Table 1. Dataset

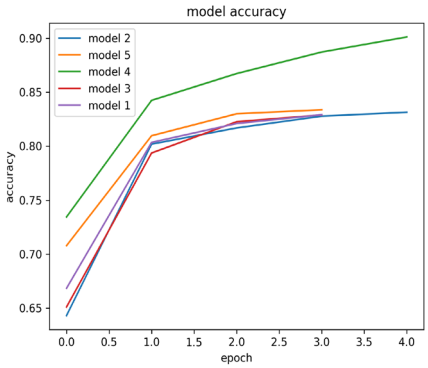


Figure 7. Model accuracy for dataset

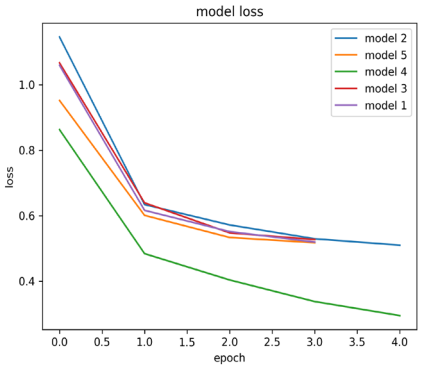


Figure 8. Model accuracy for dataset

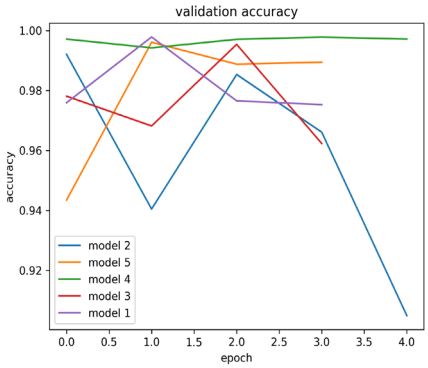


Figure 9. Validation accuracy for dataset

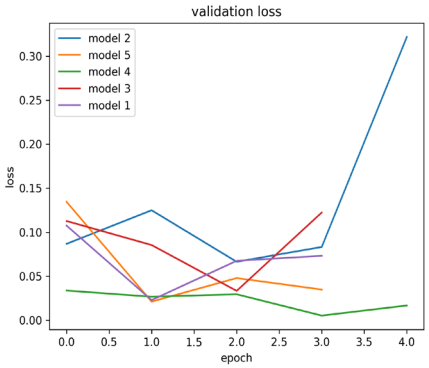


Figure 10. Validation loss for dataset

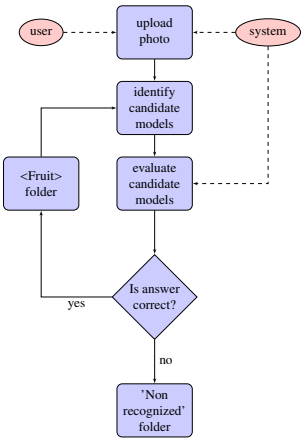
It is said that the lower the cost, the better the model. On the other hand the model shouldn't overfit to the training data (when model memorizes only the training examples and is not able to assign correct labels for the test set) which can be easily identify by when validation accuracy is much lower than training accuracy.

After each epoch the current model and validation accuracy and loss are saved. Results can be seen in Figures 7,8,9,10. It is clear that model 4 is the most reliable one with the best parameters. The comparison with other models shows that even though number of layers is similar more processing gives far better results than first model. On the other hand model perform best using pictures without noisy background and user of web application is encouraged to to take pictures in that manner.

7 Fine tuning model

When it comes to an online training we want the best quality models using as little time and computational power as possible. Instead of computing gradient on the whole dataset, I estimate it on a very small subset , with a batch size that allows me to stream the training samples. Online model consists in estimating the parameters of the model iteratively by means of stochastic gradient descent (SGD). Batch gradient descent performs redundant computations for large datasets as it recomputes gradients for similar examples before each parameter update. SGD does away with this redundancy by performing one update at a time. Is therefore much faster and can also be used to learn online. SGD enables it to jump to new and potentially better local minimum. It has been shown that with a slowly decreasing learning rate SGD almost certainly converges to a local or the global minimum for non-convex and convex optimization.

I fed my CNN with images of a new image uploaded by the user. After getting response from webserver user can help with updating model by answering question "Do you think that the answer is correct?" with possible answers "Yes" or "No".



Then server administrator has to check Non recognized fruits in order to decide what were pictures unrecognized by the system. For fine tuning I change ADAM optimizer to a stochastic gradient descent with some specified batch size. Running SGD asynchronously is faster, additionally it is able to parallelize SGD on one machine without the need for a large computing cluster. Tensorflow is architecture which is used for optimizing parallelized and distributed SGD. The distributed version relies on a computation graph that is split into subgraph for every device while communication takes place using Send/Receive node pairs. Generally

in order to avoid providing training examples in a meaningful order to my model as this may bias the optimization algorithm. It is often a good idea to shuffle the training data after every epoch. I also monitor error on a validation set during training and do not update model if my validation error does not improve enough. Model is fine-tuned every Sunday in the night. The new weights are assigned and then model is updated.

8 Keras, the Python Deep Learning library

Keras [7] is a high-level neural networks API written in Python and capable of running on top of Tensorflow[8] which I use to to train models. This library supports both convolutional networks and recurrent networks as well as combinations of the two and runs seamlessly on CPU and GPU. A single model is presented as a sequence object or a graph of standalone. It is possible to combine neural layers, cost functions, optimizers, initialization schemes, activation functions and regularization schemes.

9 Behind web application

In order to deploy my web application for users I used PythonAnywhere which is online integrated development environment (IDE) based on the Python programming language and is recommended as a way of hosting machine learning-based web applications. Web applications hosted by the service can be written using and WSGI-based application framework. The Web Server Gateway Interface is a simple calling convention for web servers to forward request to web framework written in Python.

I used Flask web framework to create web application which takes image as an input and returns three predicted pictures with labels and levels of prediction. Flask is a web framework that provides you with tools, libraries and technologies that allow you to build a web application in python. Flask is a part of the categories of the micro-framework. Micro-framework are normally framework with little to no dependencies to external libraries. This has pros and cons. Pros would be that the framework is light, there are little dependency to update and watch for security bugs, cons is that some time you will have to do more work by yourself or increase yourself the list of dependencies by adding plugins.

10 Future work

The system will be fed through **the YOLO(you only look once)** real time object detection system which produces a number of detections depending on the number of fruits in the image. The YOLO network is going to be modified to only find one class, fruits detected by model. The next thing is to work more on a project which can be able to detect fruit with noisy background. It is vital to spend more time on working on a model of CNN which will be able to predict better fruit when the picture with noisy background was taken.

11 Conclusions

I have created a web application which is going to help foreigners in Vietnam. It can be found highly useful especially in situations when seller doesn't know English name of fruit which is very usual in real situations during trip to Vietnam. During my Student Exchange Programme in Vietnam I faced many situations when I used my application for fruit recognition. There is also a promo video of this web application which shows how simply everyone can get to know what is this fruit.

Acknowledgements

I thank Dr Michał Przewoźniczek, from Wroclaw University of Science and Technology for comments that greatly improved the research paper.

References

[1] I. Sa, Z.Ge, F. Dayoub, B. Upcroft , T. Perez & C. McCool, DeepFruits: A Fruit Detection System Using Deep Neural Networks, **Vol. 16(8)**, pp. 1222-, (2016)

[2] P. Ninawe, S. Pandey, A Completion on Fruit Recognition System Using K-Nearest Neighbors Algorithm, International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) **Vol. 3 (7)** (2014)

[3] Horea Muresan, Mihai Oltean, Fruit recognition from images using deep learning, Acta Univ. Sapientiae, Informatica **Vol. 10**, Issue 1, pp. 26-42, 2018.

[4] Fruit 360 Dataset on GitHub. https://github.com/Horea94/Fruit-Images-Dataset

[5] Fruit 360 Dataset on Kaggle. https://www.kaggle.com/moltean/fruits

[6] http://www.image-net.org/

[7] https://keras.io/

[8] https://www.tensorflow.org/

[9] https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

[10] https://cs231n.github.io/convolutional-networks/

[11] Hassoun, Mohamad H. *Fundamentals of Artificial Neural Networks.* (The MIT Press, Bambridge, MA, 1995)

[12] Zurada, Jacek M. *Introduction to Artificial Neural System.* (West Publishing Company, St. Paul, MN, 1992)

[13] Bose, N. K. and Liang, P. *Neural Network Fundamentals with Graphs, Algorithms, and Applications* (McGraw-Hill, New York, NY, 1996)

[14] Haykin, Simon. *Neural Networks: A Comprehensive Foundation, second edition* (Prentice-Hall, Upper Saddle River, NJ, 1999)

[15] Beale, R. and Jackson, T. Hilger, *Neural Computing: An Introduction* (Philadelphia, PA, 1991)