The application of different Convolutional Neural Network architectures on a reduced Fruit 360 data set

Joel Kischkel

September 15, 2020

Abstract

With the usage of GPUs the success of convolutional neural networks in computer vision was born. Since the first modern implementation of Convolutional Neural Networks in 2012 various architectures were proposed in different image classification tasks. The goal of this paper is to introduce different architectures and show the impact of different convolutional operations on the image classification of fruits. As architectures we use the Interception architecture proposed by Google, the VGG architecture and a mix between the AlexNet and the architecture used by the researchers of the Fruit 360 data set. The technical implementation of the CNNs is done in Tensorflow. In our experiments we have shown that the VGG and Interception architecture outperforms architectures based on AlexNet. As well we have shown that smaller filter sizes can perform better and produce fewer computational costs.

1 Introduction

With the ImageNet challange 2012 the success of Convolutional Neural Networks in computer vision begun. Since then several different architectures were developed and proposed by researcher. The aim of this paper is to use different architectures to build successful models to classify different categories of fruits. The research is based in following on two key questions. How different architectures influence the result in image classification and which architecture is preferred so solve this problem. The paper is structured as followed. First we describe the data set we use. This includes an explanation of gray scaled and colored images and the normalization of the data. This chapter is followed by a short introduction into convolutional neural networks and its operations. As well in this chapter the activation and loss function is described. Also dropout and batch normalization is introduced. After this we introduce different architectures which we are using in our experiments. In the section results we compare the performance of the used architectures. The paper is finished with a short conclusion.

2 Methology

In the first step we build different CNN models, based on different architectures. These models are trained on the training data. We use 20% of the training data to construct a validation set. As loss function for the networks we use the categorical cross entropy. For computing the gradient we are using the adam algorithm. All models have to predict the labels of the test data set. To measure the test error we are using the zero-one loss.

3 Data set description

The original Fruit 360 data set contains 90483 images of 131 fruits and vegetables. [M. 17] For simplicity in the further context of this paper we will call it fruit data set. For our analysis we are using a subset of the original data. Only the 10 most common types will be considered. These are apple, banana, plum, pepper, cherry, grape, tomato, potato, pear and peach. The figure 2 shows a random sample of the fruit data. Each type of fruit has several sub types. So for example the fruit apple has sub types Pink Lady, Breaburn or Red. For our approach we generalize the sub types to the corresponding type. In total the training data set contains 32607 images. We apply the same operation on the test data. In total the test data set contains 10906 images. The figure 1 shows the distribution of the labels in the data set. We see that the labels are not perfectly distributed and that for example the type 0 or apple is very common while other types are more rare.

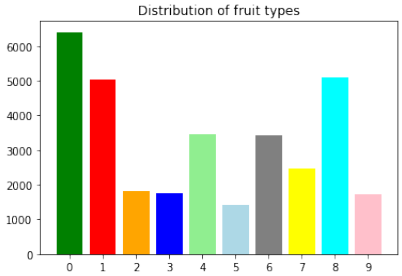


Figure 1: Distribution of the labels

Kaggle host the data as .jpg images. Each image has a is 100 pixels height and 100 pixels weight. For our approach we change the images into a 100 x 100 x 3 dimensional matrix. Each matrix has three channels. They correspondence to the color RGB, which stands for red, green and blue. The figure 3 shows how an color image can be represented through a matrix with three channels.[1] For computation purpose each fruit label is symbolized by a unique number. The numbers are in a range between 0 and 9. The following table 1 shows the number and the label of each fruit type. The figure 2 shows the ten first images of the training data with the label of the fruit. For our experiments we reduce the resolution of the images to 64 x 64 x 3.

In our research approach we will just focus on colored images. The main reason therefor is to reduce the number of trained models to perform all the experiments with images converted to gray scale again. Not normalized data can affect the computation of an artificial neural network in two ways. First it can affect the estimation error. Normalization allows in many cases to reduce the estimation error. Another effect is that the normalization can improve the computation speed. [S. 14, p. 316 - 318] This means to avoid that some large values will affect over proportional the activation function, we normalize the values in a range between [0, 1]. Formula 3 shows the equation for the max/min normalization of each value.

$$x_i = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}} \tag{1}$$

[1]For a better usage of the images I built an compressed .npz file which contains training and test data and the corresponding labels. This allows us to load the data much faster and es well we not have to convert always images into matrices. To avoid that a network just learn one type of fruit, each image matrix and the corresponding label are in a random order.

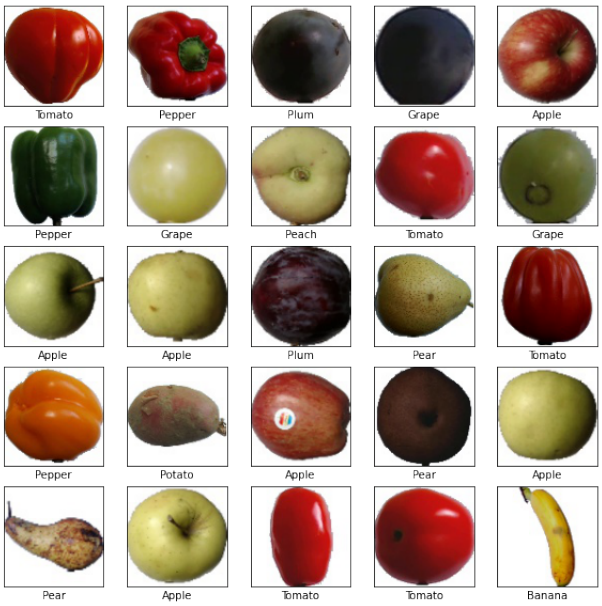


Figure 2: The ten first elements of the fruit training data

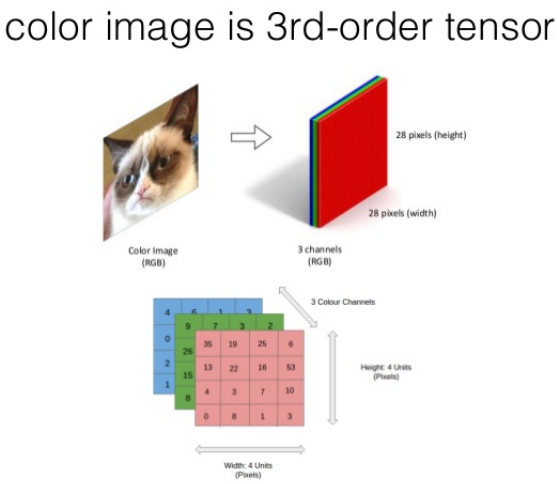


Figure 3: Tensor of a RGB image ($Source$: $https$: $//medium.com/@prince.canuma/beyond-neural-networks-bdc68980d474$)

0	Apple
1	Pear
2	Potato
3	Plum
4	Cherry
5	Banana
6	Grape
7	Pepper
8	Tomato
9	Peach

Table 1: Class features and the corresponding name

4 Foundation of Convolutional Neural Networks and Metrics

4.1 Introduction

An Artificial Neural Network is a collection of neutrons which are linked to each other. These links are weighted. A network has an input layer, one or several hidden layer and an output layer. The Image 4 shows an ANN with two hidden layer. Each node represent a neutron. Neutrons are a function which sums all incoming weighted connection and transfers this value through a function to output a new value. [Kub17, p. 91-92] The figure 5 shows a neutron. In mathematical terms a neutron is just the dot product of the input values x and the weight values w plus additional a bias. This can be seen in the following formula 4.1. Additionally the value z can be passed through a specific function known as activation function. Convolutional Neural Networks (CNN) are a special type of neural networks used in computer vision. This network architecture belongs to the group of deep neural networks (DNN). The term DNN refers to the circumstance that these networks have several hidden layers, which make them "deep". [Kuo16] Despite the difference that CNN are having a deep architecture the main difference to classical Artificial Neural Networks are the convolutional operations. Before passing values in the network these operation extract features from images through filtering or pooling values together. [YNDT18] The training is done by passing the values forward through the network. [Kub17, p. 93] Weights are updated with the back propagation algorithm. [Bus98]

$$z = f(b + x * w) = f(\sum_{i=1}^{n} x_i w_i) \tag{2}$$

4.2 Convolutional Layer

The convolutional layer extract features from the image. In this layer each neutron is called a kernel or filter. The kernel extracts the features from the input image. This is done by slicing the image into smaller parts. A kernel is normally a square with the dimension $K_i \times K_i \times d_i$. The convolutional layer can be formally defined as followed, where w is a tensor with three dimensions on the position r,s,k and acts as a kernel in the layer q as filter p. The feature is represented by the in this case three dimensional tensor h. [Agg18, p. 318-320] In simple words, the convolutional

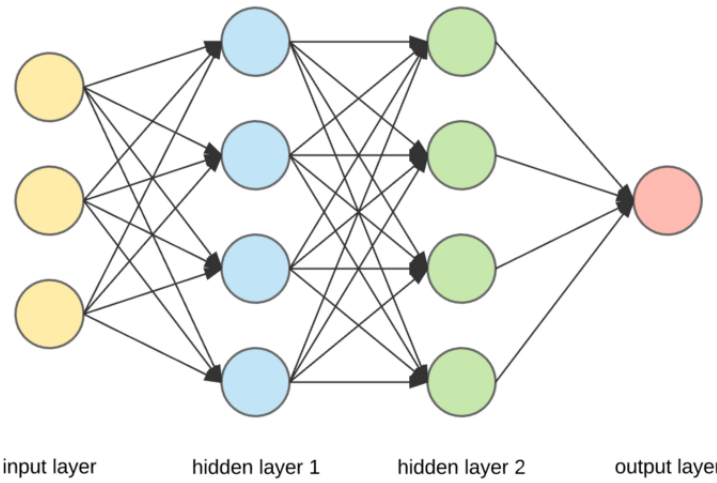


Figure 4: ANN with an input, two hidden and an output layer (*Source* : *https* : *//intra.ece.ucr.edu/ hyoseung/pdf/rtss*19 − *dart* − *slides.pdf*, *modified*)

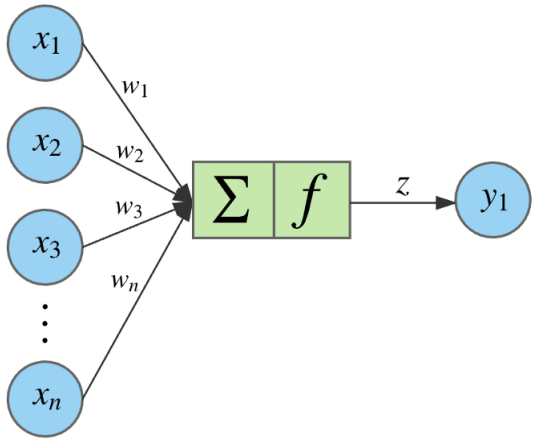


Figure 5: An abstract version of a neutron (*Source* : *https* : *//www.uniquenewsonline.com/neural* − *network* − *model* − *brief* − *introduction* − *glossary* − *backpropagation/*)

layer is the sum of the element wise product between the sliced block and the kernel. A visual understanding can be seen in figure 6, while the formula 4.2 shows the mathematical formula for the convolutional operation.

$$h_{ijp}^{(q+1)} = \sum_{r=1}^{F_q} \sum_{s=1}^{F_q} \sum_{k=1}^{d_q} w_{rsk}^{(p,q)} * h_{I+r-1,j+s-1,k}^{q} \tag{3}$$

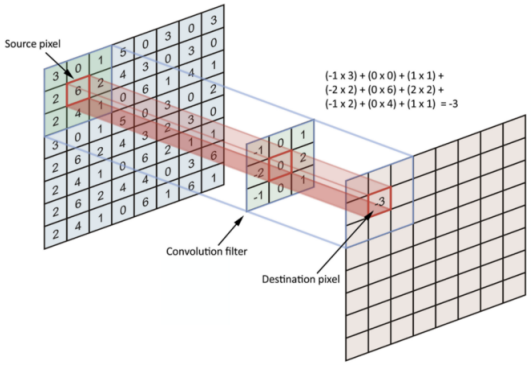


Figure 6: Visualization of a convolutional filter (*Source* : *https* : *//www.freecodecamp.org/news/an* − *intuitive* − *guide* − *to* − *convolutional* − *neural* − *networks* − 260*c*2*de*0*a*050/)

The usage of filters reduces the size of an image. To avoid this padding can be applied. Padding increases the size of an input image by adding additional values around the border of the image. Typically zero values are used. [Kuo16] The output size of an image can be calculated with the following formula 4.2, where i stands for the size of the image and k for the size of the filter.

$$o = \left\lfloor \frac{i + 2p - k}{s} + 1 \right\rfloor \tag{4}$$

Another variable is the stride. It defines the number of steps a filter moves on the image. If we look back at the figure 4.2 we can think about about strides as the number of fields moving ahead after a convolutional operation. In practice this influence the amount of overlapping between filters. [ON15, p. 7] As an example. The images we use are 100 pixels x 100 pixels which makes i = 100. If the kernel is k = 5 and stride s = 1, without padding the output size would be o = 96.

$$o = \left\lfloor \frac{100 + 2 * 0 - 5}{1} + 1 \right\rfloor, o = \left\lfloor \frac{95}{1} + 1 \right\rfloor, o = 96$$

Instead if we would like to maintain the size we would need a padding size of p = 2.

$$100 = \left\lfloor \frac{100 + 2 * p - 5}{1} + 1 \right\rfloor, 99 = \left\lfloor \frac{95 + 2p}{1} \right\rfloor, 2p = 4, p = 2$$

4.3 Pooling

The idea of the pooling layer is very simple. The pooling layer works by extracting a value inside a Matrix m x n used on the input matrix j x i. Which value is chosen depends on a rule, which can be for example the maximum value inside this matrix. Again pooling can have different strides. Typically the pooling size is 2x2 with a stride of 2. With the pooling the maximum or average value can be extracted. [Agg18, p. 326 - 327] In our used architectures we will use average and max pooling.

4.4 Fully Connected Layer

Like traditional feed forward networks also CNN have fully connected layer. The fully connected layer allows to separate the data by learning a nonlinear function. Depending on the architecture several fully connected layer can be used. [Agg18, p. 327 - 328]

4.5 Functions

The loss function for the training process is the categorical cross entropy.

$$CE = -log(\frac{e^{s_p}}{\sum_j^C e^{s_p}}) \tag{5}$$

As activation function for the network we are using ReLU. In the literature it is well known that that the ReLU function performs with respect to speed and accuracy well in deep learning applications. The function is applied to each layer of the convolutional layers. [Agg18, p. 325]

$$R(x) = max(0, x) \tag{6}$$

The last fully connected layer has the softmax function for the multi class classification. The function returns the probability of a value to be in a special class. [C. 17, p. 8]

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{7}$$

4.6 Optimizer

To perform the gradient decent optimization we are using the adam algorithm. In Tensorflow we use the default conditions of this optimization algorithm.[2] Various experiments have been shown that Adam is a robust algorithm in the context of deep learning with large data sets. [KB15]

[2] $https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam(Access: 2020-08-12)$

4.7 Measuring accuracy / error

For measuring the training and validation accuracy we use the Tensorflow build in method "Accuracy" for measuring it.[3] The test error will be measured with the zero-one loss, which basically sums up all miss classification with one.

4.8 Dropout

To improve the generalization of our models we will train models where we include dropouts in the architecture. Dropouts remove neutrons from a layer. With this approach all all connections from and to this node are eliminated. Typically the dropout rate is between 20% and 50%. [Agg18, p. 188 - 189]

4.9 Batch normalization

Another method to improve the accuracy of our models is the usage of batch normalization. The idea of this is that after each convolutional operation before using the activation function each input data inside a batch is again normalized. This makes computation easier for the activation function. [Agg18, p. 152-153]

[3] $https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Accuracy (Access: 2020-08-12)$

5 Convolutional Neural Network Architectures

5.1 AlexNet based architecture

The first architecture we use is based on the CNN proposed by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton for the ImageNet contest in 2012. [KSH12] For our experiments on the fruit data set we modify the architecture. First of all we will train the model on just one GPU, while the creators of the AlexNet used two GPUs. Another important change is that we will use different values for the number of filters and size of the kernels. H. Mureșan and M. Oltean who created the data set used for there research a similar CNN to the AlexNet. For our approach we are using a mix of both models. On the one side we orientate on the CNN used in the original fruit data set by using the same numbers and sizes of filters. [MO18] On the other side we take inspiration from the AlexNet architecture by adding dropouts to our CNN. These dropout layers are added between the fully connected layers and have a rate of 50%. [KSH12, p. 6] In detail the network has a convolutional layer with 16 filters of size 5 followed by a max pool layer with size 2 and stride 2. This combination is repeated but now with 32 filters. This is followed by a convolutional filter of 64 and 128 filters with each them has a kernel size of 5. The following max pool layer has a pool size of 3 and stride of 2. After it is flattern the two fully connected layer have each 128 nodes and after each of them a dropout layer with 50%. The last layer has 10 neutrons and a softmax activation function. In total this model has 1,090,986 parameters to train. The figure 7 shows the architecture used.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 64, 64, 16)        1216
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        12832
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        51264
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 128)       204928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 128)               802944
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
```

```
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,090,986
Trainable params: 1,090,986
Non-trainable params: 0
_________________________________________________________________
```

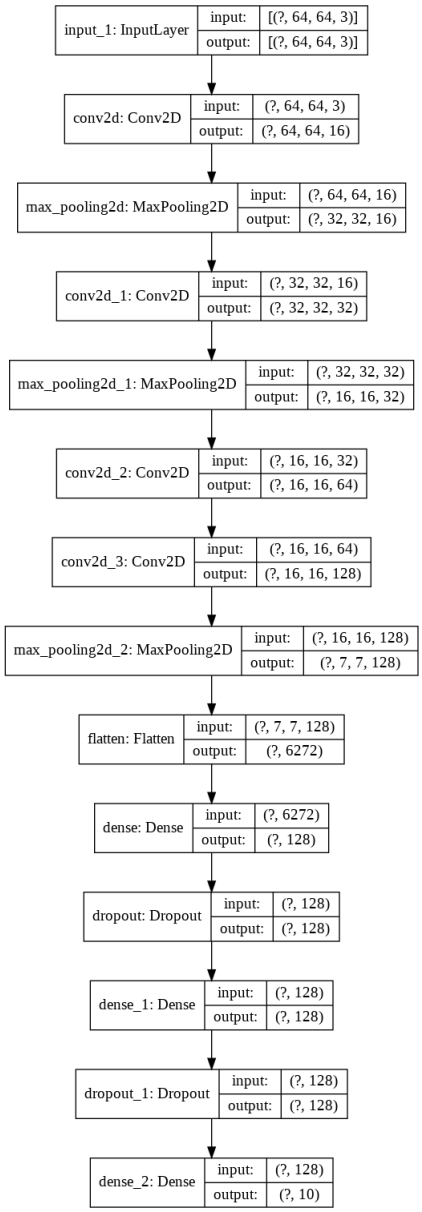


Figure 7: Architecture based on alexnet

5.2 VGG based architecture

Another architecture is the VGG. The VGG architecture was developed by K. Simonyan and A. Zisserman for the ImageNet challange of 2014. The VGG net is much deeper than the AllexNet. For our approach we not use exact the same architecture, because the original network was designed for data with much higher resolution and more objects to detect and more complicated structure of objects. Instead we simplify the architecture in the way that we use a smaller value of filters and partly less convolutional operations. We use a architecture of three blocks. Each block contains two convolutional and one max pooling layer. In each block the convolutional filter have the same number of filters and the same filter size. The first block has twice 32 filters with a size of 3x3. The max pooling layer has a pool size of 2 with stride of 2. The second block has 64 filters with 3x3 filter size. Again we are using max pooling with a size of 2 and stride of 2. The last block has 128 filters with a size of 3x3. Also again we are using the max pooling with size of 2 and stride of 2. After we have flattern the tensor we are using two fully connected layer each with a size of 128 neutrons. The last layer has 10 nodes and a softmax activation function. In total this network has 1,353,514 parameters. The VGG architecture based on blocks. This allows us easily to expand the architecture. We use a second model which is build of four blocks. To expand the architecture we add a fourth block with two convolutional filters with each 256 channels and a kernel of size 3 and again at the end a max pooling layer of size 2 and stride 2. Compared to the previous three block architecture this CNN has now 1,714,474 parameters. Also we constructed a model with five blocks. This model has more than 4,992,298 parameters. Th figure 8 shows the architecture of the VGG 3 model. It is easy to see that this architecture can be seen as a combination of blocks. The other VGG architectures we introduced simply expand this architectures with more blocks.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 64, 64, 32)        896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 64, 64, 32)        9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 128)       147584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
```

```
_________________________________________________________________
dense (Dense)                (None, 128)               1048704
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,353,514
Trainable params: 1,353,514
Non-trainable params: 0
_________________________________________________________________
```

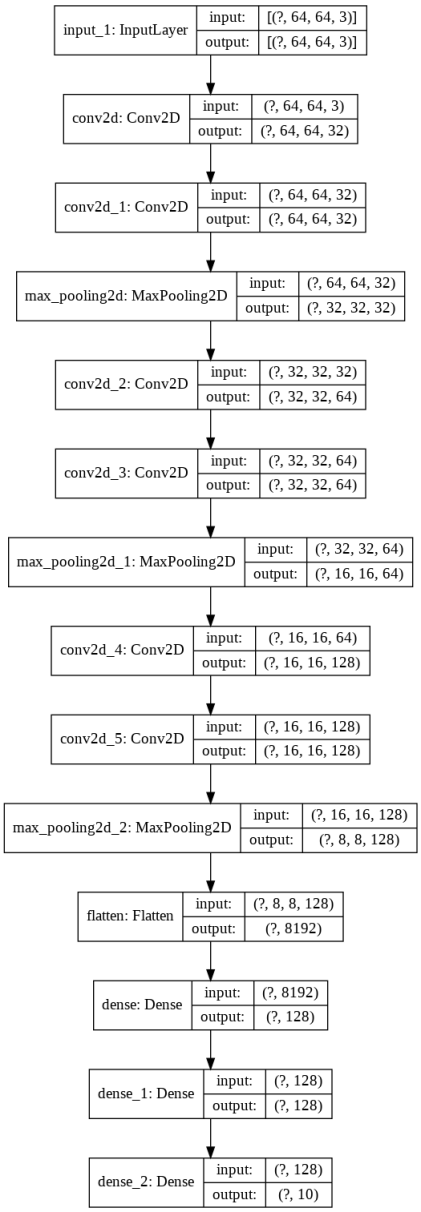


Figure 8: Architecture based on VGG

5.3 GoogLeNet based

Winner of the ImageNet challange 2014 was the GoogLeNet CNN. The main difference to the previous architectures is that in the GoogLenet not layer is followed by layer, instead the same input is distributed on the same time over several layers and later concatenate together. The Image 9 shows an interception with convolutional and max pooling layer. The interception module has two advantages. First the interception allows to extract on the same time different features. Second through the usage of 1x1 convolutional filters inside the interception module the number of parameters can be reduced which results in less computational effort. [SLJ+15] For our data set we take inspiration from the GoogLeNet but we will simplify the architecture. Mainly because our data has a different shape and the images are less complicated. The figure 10 shows the used architecture. The CNN has a convolutional layer followed by a max pool and then again by a convolutional layer. After this the network has two interception blocks which are followed by an average pooling and then a last convolutional layer. The tensor is flattern and after the dense layer the architecture includes a dropout layer before the last layer with 10 nodes and a softmax function is used. Even this network looks very deep it has just 594,394 parameters. This is much less than the VGG CNN before has.

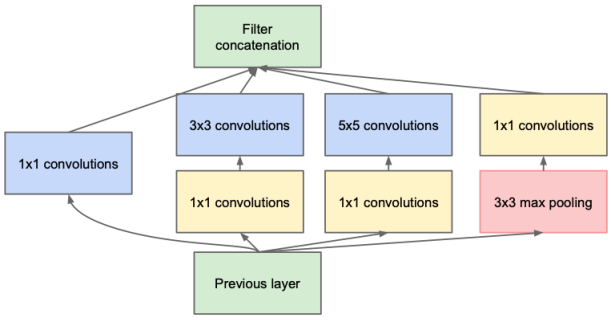


Figure 9: Example of an interception (*Source* : *https* : *//static.googleusercontent.com/media/research.google.com/de//pubs/archive/*43022*.pdf*, *p.*4)

Figure 10: Architecture of the interception network used in this paper

CNN name	Training accuracy	Validation accuracy	zero-one loss
FruitNet	0.9334	0.9902	0.0221
VGG 3	0.9861	0.9991	0.0107
VGG 4	0.9719	0.9786	0.0193
VGG 5	0.955	0.9833	0.0457
Interception	0.954	0.9898	0.0062
FruitNet (Dropout/Batch norm.)	0.4233	0.5406	0.4328
VGG 3 (Dropout/Batch norm.)	0.9492	0.9736	0.0023
VGG 4 (Dropout/Batch norm.)	0.9391	0.9276	0.0072
VGG 5 (Dropout/Batch norm.)	0.9456	0.9695	0.0044
Interception (Dropout/Batch norm.)	0.9599	0.973	0.0097

Table 2: Results of training different networks on the fruits 10 data set

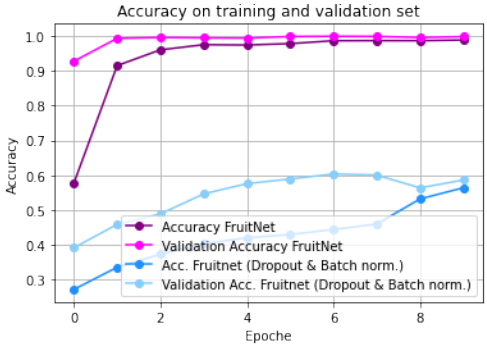
6 Experiments and results

Table 2 shows the accuracy of the training and validation set. Also the table contains the zero-one loss on the test data. Without dropout and batch normalization the best result is performed by the Interception net which is based on GoogLeNet. The Interception net achieved a zero-one loss of 0.0062. It is followed very closely by the VGG 3 performed a loss of 0.0107. The other architectures as FruitNet and VGG 4 achieved a zero-one loss of 0.0221 and 0.0193. A very bad result is performed by the VGG 5 architecture. The zero-one loss is 0.0457 which is more than seven times higher than the result of the Interception net.

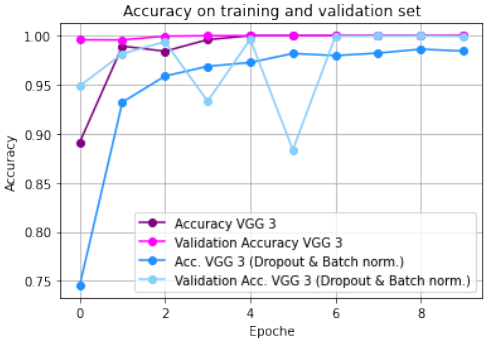
A different result is obtained if the models are trained with additional dropouts and batch normalization. The lowest zero-one loss is now achieved by the VGG 3 architecture. It is closely followed by the VGG 5 which achieved a zero-one loss of 0.0044. Both models performed better than the Interception net without dropout and batch normalization. The worse result is obtained with the FruitNet. The zero-one loss is 0.4328.

We can check the development of the training and validation accuracy in figure 11. As we have seen already the FruitNet does not perform better with the regularization. More interesting are the VGG architectures. By adding the regularization methods, the networks started to learn slower. We also see that using the regularization the network becomes unstable in the validation set.

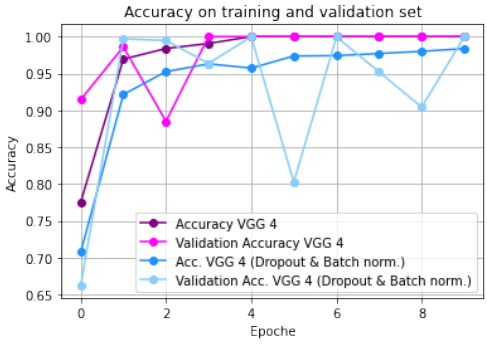
If we add dropouts and batch normalization to the models in the most cases we see an improvement in the zero-one loss. On the other side these methods lead to longer training time. If we check the result for the FruitNet we see in figure 11a that the accuracy is much lower. In experiments which we not show in this report we discovered that this architecture would require much higher number of epochs to achieve a higher result. Therefore we consider in this case the architecture as failed for dropout and batch normalization. A critical aspect we also discovered is that for the VGG architecture the validation accuracy becomes unstable. If we check the development of the loss in figure 12 we note that the FruitNet with introducing batch normalization and dropout shows a very constant decrease. The opposite is true for the VGG architectures. In that case the loss decreases in the beginning exponentially and then becomes stable. But the figure shows as well that the loss function has some outliers where it is suddenly jumping up and down. Interestingly for the Interception architecture changes are very small. We have already seen in table 2 that the differences between the two models are small. By adding dropout and batch normalization also the development of the loss is very similar.



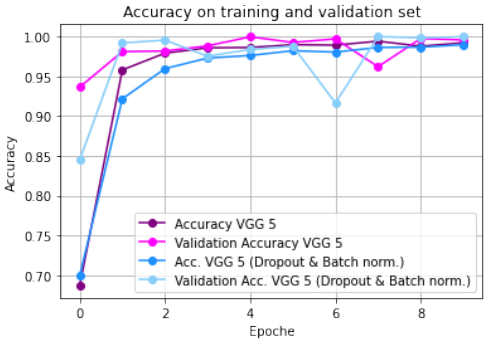
(a) FruitNet



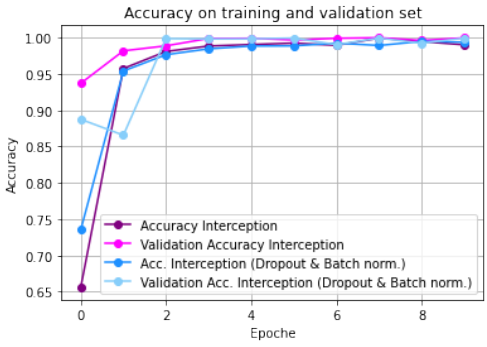
(b) VGG3



(c) VGG4

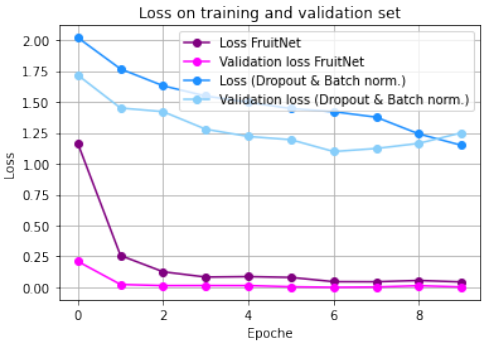


(d) VGG5

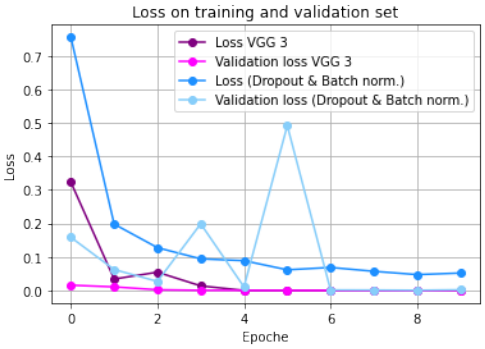


(e) Interception

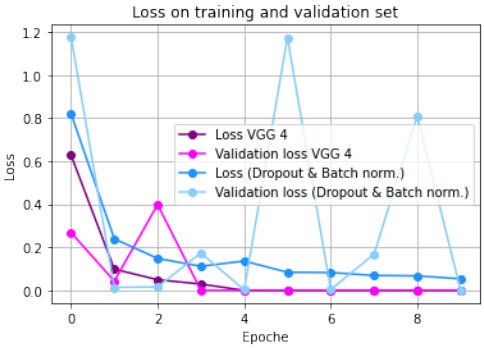
Figure 11: Comparison of the development of the error of architectures with and without dropout and batch normalization



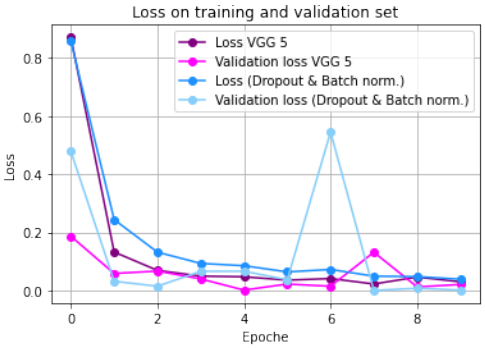
(a) FruitNet



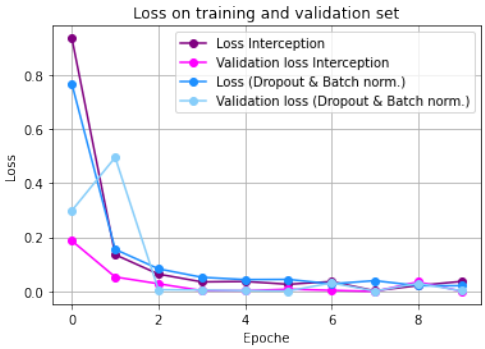
(b) VGG3



(c) VGG4



(d) VGG5



(e) Interception

Figure 12: Comparison of the development of the loss of architectures with and without dropout and batch normalization

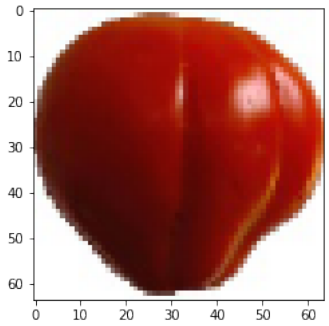
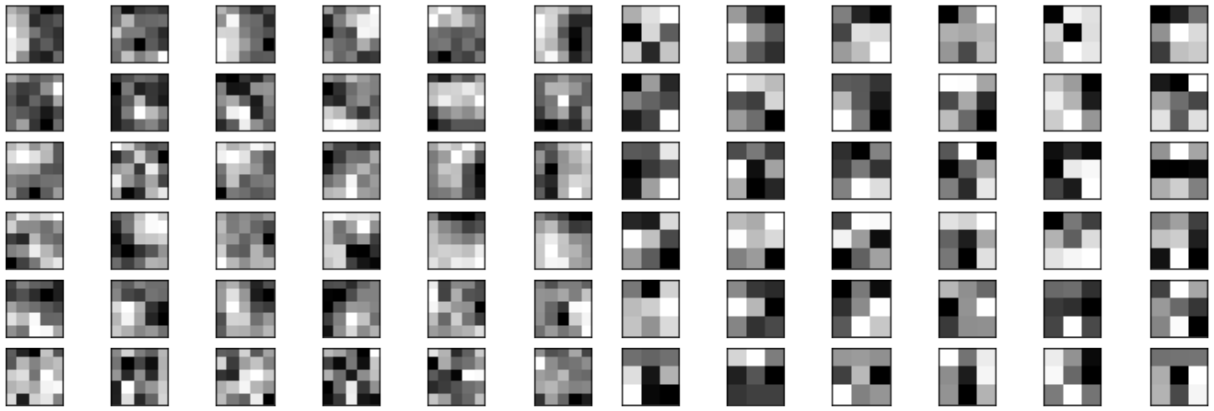


Figure 13: Picture of a plum

The figure 13 shows a plum. During the convolutional operation this plum will look different. We now check the impact of the different filters on the image. From the architecture we see that the network based on the AlexNet architecture is using kernel sizes of 5x5. The VGG architecture is just using a size of 3x3. The figure 14 shows the different filters in the layer 4 of both networks. Each column is a single channel of the convolutional layer. While each row is a filter used inside the channel. Since it is not meaningful to plot all the filters we just show the first five filters of the first six channels. In the FruitNet, figure 14a, architecture we see the greater filter size while the VGG in figure 14b has smaller kernel sizes. These different sizes have an actual impact on the networks how it sees the fruits. The figure 15 shows the convolutional layer of the FruitNet and the VGG of the same layer as the filters from the previous image. Besides the fact that the VGG has a higher resolution at this stage we see that the VGG can recognize the shapes of the fruit much better. At this stage both models are using 64 filters. Each image is the summary of the filters of each channel. The effectiveness of the VGG architecture can be seen if we visualize the last convolutional layer before flattern occurs. The figure 16 shows us that the VGG CNN still contains a lot of information compared to the FruitNet. From the zero-one loss we know that the Interception architecture can outperform the VGG.

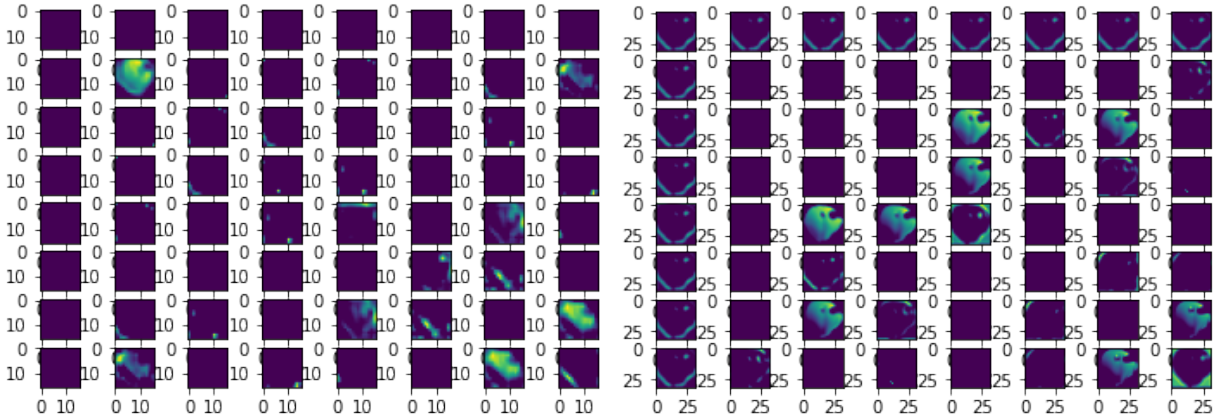
We now look closer to the interception architecture. We will skip the visualization of the filters and immediately look the combined output of the convolutional layer. The image 17 shows the visualization of the interception net layer. Similar to to the FruitNet the resolution is small, but the object can be visualized much better.

Finally figure 18 shows a random prediction based on the Interception net. Please note that one image is misclassified. The ground truth is an apple whereas the network interprets this as a peach.



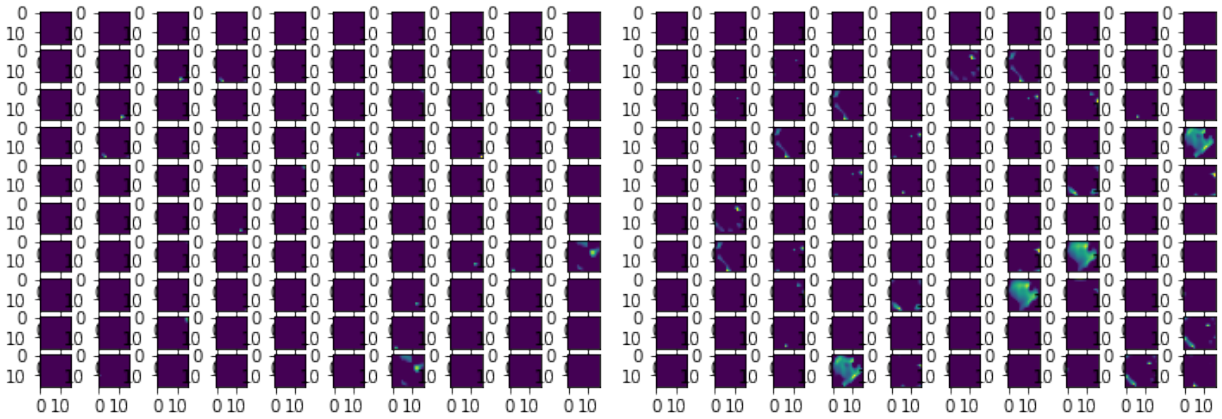
(a) Filter of FruitNet (b) Filter of VGG

Figure 14: Visualization of filters



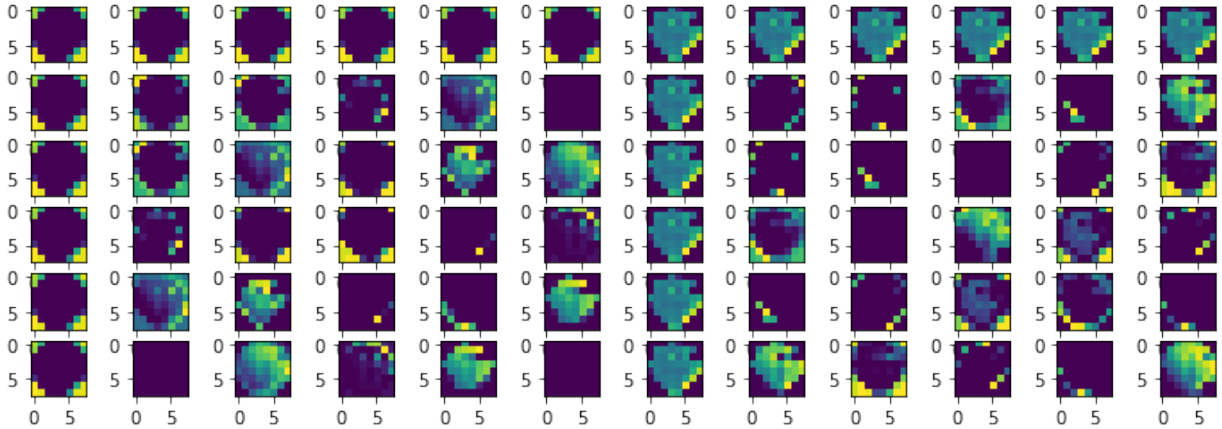
(a) Visualization Layer of the FruitNet (b) Visualization Layer of the VGG

Figure 15: Visualization of filters



(a) Visualization Layer of the FruitNet (b) Visualization Layer of the VGG

Figure 16: Visualization of filters



(a) A (b) B

Figure 17: Visualization of filters

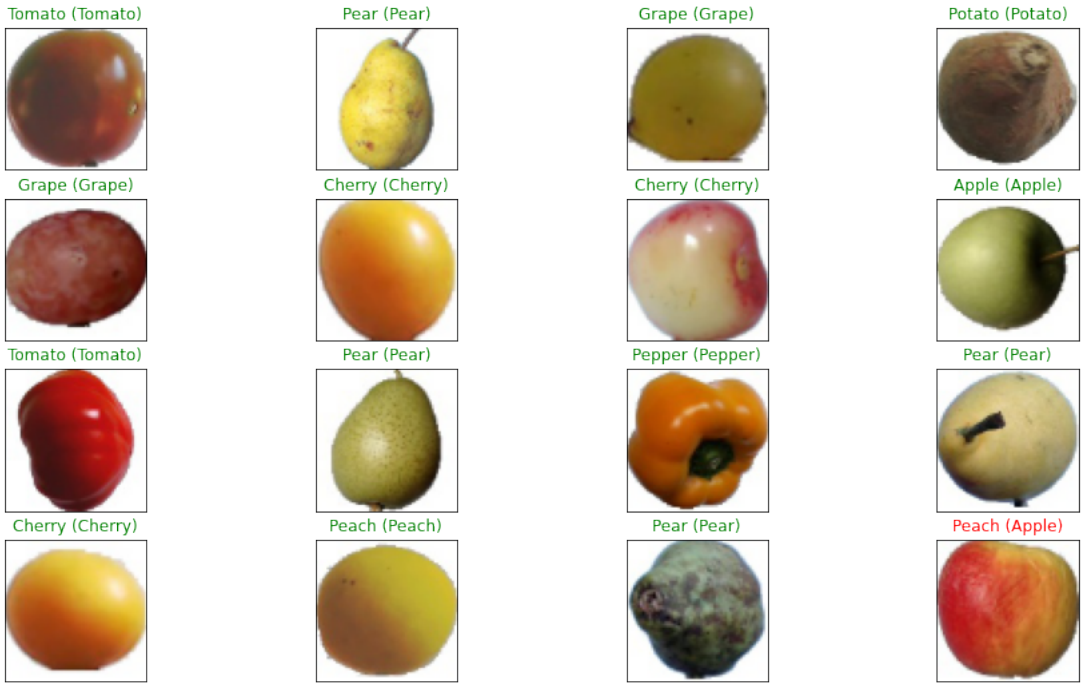


Figure 18: Randomly sampled images predicted by the interception net

7 Conclusion

In general all the models worked on the data and could identify the fruits correctly in most of the cases. We discovered that smaller filter sizes, as seen in VGG, can extract features better than greater sizes. A good way to reduce the number of parameters is shown in the Interception architecture by using a 1x1 convolutional filter before applying greater filter. In that cases we could not find a preferred architecture. In general the VGG and Interception architecture performed better than the architecture based on AlexNet. The only exception is, if we added to many layers on the VGG it performed worse. The main advantage of the VGG architecture is simplicity, while the Interception architecture is constructed in a more complicate way. On the other side the Interception architecture performed slightly better and is using less parameters. Another finding is that a deeper network not performs necessarily better. While in the beginning additional layers improve the model as seen in the FruitNet and VGG, adding too many layer has an opposite effect. Here again can be emphasized the advantage of the Interception architecture which allows to use different filters by not increasing the deep for the network.

The usage of regularization methods as dropout and batch normalization can improve the accuracy of the neural network. But it is important to check how high the impact is. We noted that the improvement with the Interception network was very small but the training process has taken much longer. In a real world application it is questionable if the benefit of a slight increase in accuracy would justify the additional computational costs. A negative aspect is, because the models have a lot of parameters to optimize it is computational expensive to apply grid search to optimize hyper parameters. A good strategy could be by sticking to key finding of other researchers to exclude from the beginning some parameters from the optimization.

To put it all in a nutshell, the architecture which is used should be fit to the data. In addition smaller filters should be applied to extract features and the usage of regularization methods is not always useful.

References

[Agg18] C. C. Aggarwal. *Neural Networks and Deep Learning.* Springer, 2018.

[Bus98] M. Buscema. Back propagation neural networks. *Substance Use & Misuse*, 33:233–270, 1998.

[C. 17] C. E. Nwankpa and W. Ijomah and A. Gachagan and S. Marshall. Activation Functions: Comparison of Trends in Practice and Research for Deep Learning, 2017. accessed: 2020-08-13.

[KB15] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. In *ICLR2015*, 2015.

[KSH12] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

[Kub17] M. Kubat. *An Introduction to Machine Learning*, volume 2. Springer, 2017.

[Kuo16] C.-C. Jay Kuo. Understanding convolutional neural networks with a mathematical model. *Journal of Visual Communication and Image Representation*, pages 406–413, November 2016.

[M. 17] M. Oltean and H. Muresan. Fruits 360, 2017. accessed: 2020-07-22.

[MO18] H. Muresan and M. Oltean. Fruit recognition from images using deep learning. *Acta Universitatis Sapientiae, Informatica*, 10:26–42, 2018.

[ON15] K. T. O'Shea and R. Nash. An introduction to convolutional neural networks, 2015.

[S. 14] S. Shalev-Schwartz and S. Ben-David. *Understanding Machine Learning - From Theory to Algorithms*, volume 1. Cambridge University Press, 2014.

[SLJ+15] C. Szegedy, W. Liu, Y. Jia1, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, and A. Arbor. Going deeper with convolutions. In *CVPR2015*. Computer Vision Foundation, 2015.

[YNDT18] R. Yamashita, M. Nishio, R. Kinh Gian Do, and K. Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9:611–629, 2018.