Statistical Methods For Machine Learning - Experimental Project on NN

Riccardo Graziosi

27 September 2020

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

1 Introduction

The aim of this report is to train a neural network that is able to recognize fruits and vegetables from images. Since there are no exact rules for training a neural network in the best way possible, I have experimented with different network architectures and hyperparameters choices, documenting their influence on the final predictive performance. So, another goal of this paper is to provide an overview of hyperparameter and topology choices and their impact on model predictive performance.

To achieve this goal, I have used convolutional neural networks, a particular type of neural network that performs especially well on image recognition tasks. Section 2 contains a brief overview of the main concepts of convolutional neural networks used in the report.

In order to implement, train and test a neural network, I have made use of the TensorFlow library from Google [1]. More information on TensorFlow and its use in this paper is provided in section 3.

The dataset used to train and test the neural network is called Fruits-360, and can be downloaded from reference [3]. It contains colored images of 131 varieties of fruits and vegetables. More information on the dataset and its modifications for this report is provided in section 4.

Section 5 contains a description of the numerical experiments conducted and their final results. Moreover, relations between certain design decisions (e.g. network architectures, hyperparameters, etc.) and predictive results are discussed.

Lastly, section 6 states whether the goal of the report has been achieved, contains pointers to design decisions not made in this report but that may lead to interesting results and a comparison with similar reports.

A repository with the scripts used to prepare the dataset, train/evaluate the neural networks, experiment with hyperparameters and topologies, and more, can be found at [5].

2 Convolutional neural networks

A convolutional neural network is a particular type of neural network which is usually composed of convolutional layers, pooling layers, Rectified Linear Unit (ReLU) layers, fully connected layers and loss layers [6]. Convolutional neural networks are based on three main ideas: local receptive fields, shared weights and pooling. See [7] for a brief explanation of these concepts.

I have decided to use convolutional neural networks because they take into consideration the spatial structure of the input, which is a well-suited feature for working with images. Moreover, it is broadly accepted that, in general, they lead to better predictive results than fully-connected neural networks for image recognition tasks. An example of that can be found in the experiments conducted against the famous MNIST database of handwritten digits: the best results were obtained using convolutional networks [8].

3 TensorFlow

TensorFlow is an end-to-end open source platform for machine learning [1].

The TensorFlow library has been used to load and pre-process data, and to train and evaluate the neural network. In particular, version 2 of TensorFlow has been employed, which provides, among other things, eager execution and more abstraction regarding graphs and sessions than version 1 [2].

TensorFlow provides an high-level API called Keras. Whenever possible, I tried to use Keras since it is easier and faster to develop with.

4 Dataset

The dataset used in this report is a modified version of the Fruits-360 dataset. For a more in-depth description on how the original Fruits-360 dataset was created and its contents, see the related paper [4].

Upon writing this report, the dataset version was 2020.05.18.0, so that is the version used in this paper.

The dataset contains 90483 images of fruits and vegetables labeled within a set of 131 possible labels. The training set size is 67692 and the test set size is 22688. Fruits and vegetables are labeled by variety, not by type (e.g. there is no label ”Apple” for the fruit type apple, but rather ”Apple Braeburn”, ”Apple Golden 1”, etc., that denote varieties of apples). All images are JPG of size 100x100 pixels, with RGB color channel and contain a single fruit or vegetable.

However, for the sake of the experiments in this report, the dataset has been slightly modified in the following way:

- Labels denotes *types* of fruits or vegetables, rather than varieties
- Only the 10 most frequent types are considered, which are: apple, banana, plum, pepper, cherry, grape, tomato, potato, pear and peach.

The resulting dataset has 32607 images in the training set, 10906 images in the test set and 10 labels.

In order to generate the modified dataset in a fast and reproducible way, I have created a Python script that automates the process. The script is called *prepare_dataset.py* and can be found following the link in reference [5], which redirects to a repository containing all the scripts used in this paper.

5 Numerical experiments

5.1 A consideration on the random nature of training

The training process, which is the procedure that consists in feeding a learning algorithm with a training set and getting back a model, is usually random in machine learning, and neural networks make no exception. Randomness may derive from both the training set (of course, different training sets lead to a different trained model) and the learning algorithm (which may be, by nature, stochastic).

In particular, learning algorithms of neural networks use randomness during the process of training, mainly in two ways: weights are randomly initialized and samples are randomly shuffled each epoch. As a consequence, training the same network (with same exact hyperparameters and topology) over the same training set, still leads to different models being trained each time.

In order to handle the stochastic nature of training, in some experiments I trained the network N times, then I computed the average and the standard deviation. This should give a more realistic approximation of the predictive performance of the "average" model produced by a specific combination of parameters. Then, on this approximation it may make sense to bound the statistical risk to an interval with certain probability (these computations have not been done in the report, but can easily be calculated by knowing the number of test set images, which is 10906, as reported in section 4).

5.2 Structure of the neural network

As said in section 2, the neural network used for the experiments is a convolutional one. In this section, the structure of the neural network is explained in detail. The following architecture and hyperparameters choices, unless stated otherwise, have been consistently used for the hyperparameters experiments, which can be found in the section 5.4. In addition, experiments with different network architectures can be found in section section 5.5.

Note: all layers in this paper use ReLU as the activation function.

Layer type	**Dimensions**	**Output**
Convolutional	3x3x3	(32, 32, 8)
Max Pooling	2x2 – Stride: 2	(16, 16, 8)
Convolutional	3x3x8	(16, 16, 16)
Max Pooling	2x2 – Stride: 2	(8, 8, 16)
Convolutional	3x3x16	(8, 8, 32)
Max Pooling	2x2 – Stride: 2	(4, 4, 32)
Dropout	–	(4, 4, 32)
Flatten	–	512
Fully Connected	256	256
Fully Connected	10	10

Table 1: The basic architecture of the neural network used in hyperparameter experiments.

This structure is the typical one for convolutional neural networks: some convolutional layers, each one followed by a max pooling layer, and lastly some fully connected layers. In this case there are 3 convolutional layers (and so 3 pooling layers) and 2 fully connected layers (the last one has the same number of neuron as the number of classes to predict).

Other two layers are present in the architecture:

- The dropout layer has rate 0.4 and is applied in order to reduce overfitting. More information on dropout is available in section 5.4.6.
- The flatten layer is just used to transform the multidimensional output tensor of shape (4, 4, 32) to a flat tensor of shape 512.

A graphical representation of the network described in table 1 can be found in the following image (dropout and flatten layers not shown):

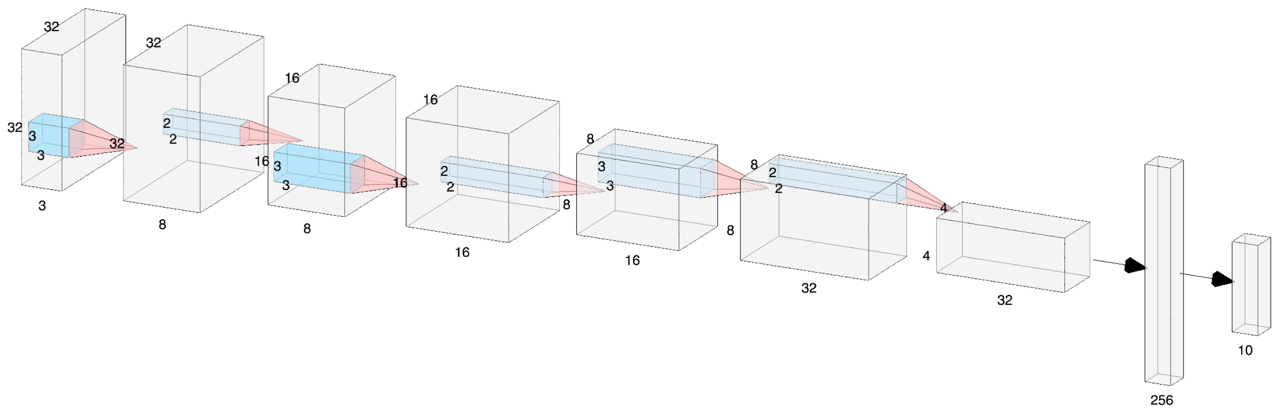

Figure 1: Graphical representation of the basic neural network architecture used in hyperparameter experiments. Note that dropout and flatten layers are not shown in the image.

An important thing to notice is that the number of filters of convolutional layers is always incremented in subsequent layers (in this case: 8, 16 and 32). That is because, heuristically, each filter learns how to recognize a feature from the image (e.g. in facial recognition, a first layer filter may recognize an eye as a feature); the next layer should be able to recognize sub-features of the features found in the previous layer (e.g. given the feature eye, the next layer may identify sub-features such as eyebrows, eyeslashes, iris etc.); and so on. So more filters are required in subsequent layers, because more features have to be found.

The following table contains the default hyperparameters values and other configuration used in the next section, unless stated otherwise.

Image size	32x32 pixels
Validation split	0.2
Batch size	32
Epochs	1000
Early stopping patience param	10
Optimizer	Adam
Learning rate	0.001

Table 2: Default values for hyperparameters and other configs used in the hyperparameter experiments section.

5.3 Loss functions

The loss functions used throughout the experiments are:

- Sparse categorical cross-entropy for training
- Zero-one loss for multiclass classification for testing

5.4 Hyperparameters experiments and considerations

5.4.1 A consideration on hyperparameters tuning

In most of the following experiments, hyperparameters are tuned against the validation set rather than the test set. This approach is employed because, if hyperparameters are set based on evaluations of the test set, there is a high chance that hyperparameters end up overfitting the test set.

5.4.2 Image size

Images have been scaled down from 100x100 pixels to 32x32 pixels in all experiments. In this way, the number of parameters of the network is smaller and so the training process takes less time. However, since the images lose details, it is also harder to learn from them and to predict them.

In fact, an experiment performed on two networks with exact same topology and hyperparameters, but different input image sizes, has yielded the results in the following table. The network has been trained and evaluated 10 times for each image size (16x16, 32x32, 64x64) to mitigate the intrinsic randomness of the training process.

Image size	**16x16**	**32x32**	**64x64**
Avg test error (*10^-3)	11.7	8.7	7.0
Test error standard deviation (*10^-3)	2.03	3.11	1.59
Avg epoch duration (s)	5.4	14.4	47.5
Avg number of epochs	26.5	30.8	36.3

Table 3: Results of the experiments on image size.

The previous table clearly shows a direct correlation between image size and predictive performance: the higher the resolution, the better the predictive performance.

Moreover, the results also confirm the hypothesis that bigger images makes the training phase duration longer. With images of bigger size, not only does each epoch take a lot more time to complete, but it also takes more epochs before the training stops (due to early stopping technique, see table 2). This latter consideration can be justified by the fact that, since images have better resolution and more details, the network has more complex features to learn, and so it needs more epochs to fully learn them.

While it is true that the average number of epochs is directly correlated with image size, it is also true that validation loss decreases faster with bigger images, as can be seen in the following plot:

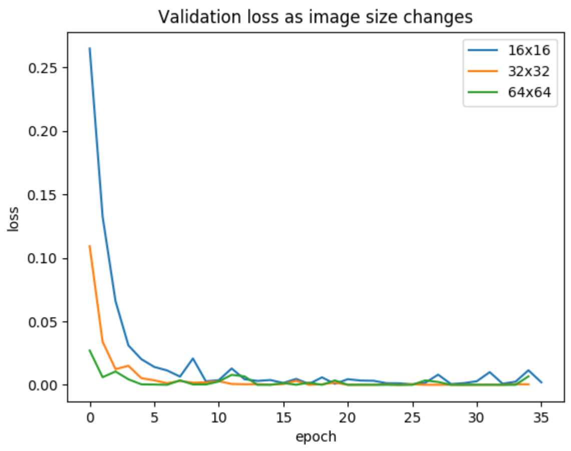

Figure 2: Validation loss as image size changes. Y axis: validation loss, X axis: epoch number

5.4.3 Batch size

During the training phase, the batch size is a number that indicates how many input must be processed by the network before the training weights are updated. A small batch size means that the weights are updated often, so it makes the model converge more precisely and faster than a larger batch size. On the other hand, a batch size that is too small does not use the full potential of computer resources, so can be slower than a bigger batch size.

An experiment has been performed in order to understand how batch size influences training time. The network has been trained 5 times for each batch size and epochs duration and number have been recorded for each run. The results are shown in the table below:

Batch size	**8**	**16**	**32**	**64**	**128**	**256**	**512**
Avg epoch duration (s)	21.4	16.9	14.1	12.7	11.9	11.1	10.6
Avg number of epochs	28.4	25.0	35.4	46.6	58.4	75.4	98.2
Avg training time (s)	607	422	499	591	695	837	1041

Table 4: Batch size experiments on training time.

From the results above it seems like the perfect spot for the batch size is between 16 and 32. However, as stated above, the best batch size choice also depends on computer resources available, so the previous consideration must be taken with a grain of salt.

An experiment has also been conducted in order to understand if the batch size influences accuracy. For each batch size, the network has been trained 10 times and and its performance evaluated on the test set. Then, average and standard deviation have been calculated over these 10 test losses. The following table shows the results of this experiment:

Avg test error (*10ˆ-3)	**8**	**16**	**32**	**64**	**128**	**256**	**512**
Test error	6.97	6.46	6.24	3.68	3.96	3.73	3.48
Standard deviation (*10ˆ-3)	3.75	2.42	2.51	1.02	0.917	0.802	0.764

Table 5: Average test error over n=10 trainings as batch size changes. Variance is also reported.

From the results above it seems that higher batch size (at least up to a certain point) leads to better predictive performance. This is kind of counter-intuitive, since a lower batch size should result in more precise steps towards an optimal solution.

The result may be explained by the fact that a technique called early stopping has been used in order to stop the training when there are no performance improvements for p epochs (for more information on early stopping, see section section 5.4.5). In the experiment, p was equal to 10. A small batch size may make the model linger around some bad local optimum for more than 10 epochs, resulting in the training to stop; on the other hand, a bigger batch size may be able to escape these local bad optimums before the patience of the early stopping technique runs out. This hypothesis is also supported by the higher values of the standard deviation for small batch sizes.

Further work may involve tuning the patience parameter of early stopping in order to demonstrate the statement above.

5.4.4 Validation split

The validation set is used during training to validate the predictive performance of the model being trained. It is disjoint from the training set, to ensure that the model generalize beyond the training set. It is also used to tune the other hyperparameters, as stated in section 5.4.1.

A commonly used 80-20 split between training and validation sets has been adopted.

5.4.5 Epochs

To determine the number of epochs to train the network, I adopted a technique called early stopping. This method stops the training phase as soon as the validation loss stops decreasing for p epochs, where p is a parameter called patience. Then, the model is brought back to the epoch that had the smaller validation loss (weights and biases of that epoch were saved and are now restored).

As a consequence, with this technique it is no longer needed to tune the number of epochs, but rather the patience parameter p. No experiments have been performed in order to tune the patience parameter. Instead, a value of 10 has been adopted for two reasons:

- The model usually reaches good performance (validation accuracy higher than 99%) in about 5 epochs, so it learns fast and it is safe to assume that if it does not improve in 10 epochs it will probably no more improve in a meaningful way. While this is true in most cases, there can be exceptions, as noted in the second experiment of section 5.4.3.
- A bigger number for patience means more training time, which would have been impractical given the computing resources available.

5.4.6 Dropout

A dropout layer performs a form of regularization useful for reducing overfitting in neural networks. A dropout layer removes a random selection of output units by setting their activation to zero. The percentage of units to set to zero is determined by the dropout rate, a fractional number that can be set between 0 and 1. Dropout regularization is applied only during the training process: when evaluating on the test set, no dropout is applied.

Training the network with different dropout rates yielded the following results:

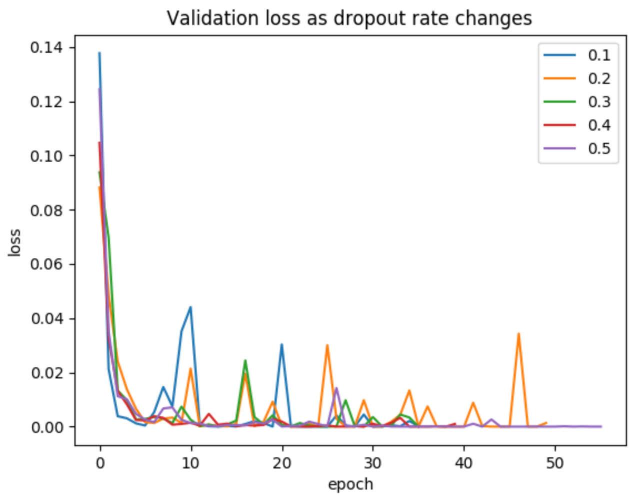

Figure 3: Validation loss as dropout rate changes. Y axis: validation loss, X axis: epoch number

With every dropout rate, validation loss reaches the same lower asymptotic bound.

The spikes in the plot are mainly produced by small dropout rates and are probably the result of overfitting, so it seems better to opt for an higher rate. Given this consideration, in the network topology experiments I decided to opt for a dropout rate of at least 0.4.

5.4.7 Optimizer

Optimizers are specific implementations of the gradient descent algorithm. They are used to compute the gradient of loss with respect to weights and biases and, therefore, to iteratively adjust weights and biases to reduce that error.

In the following experiment, three of the most common optimizers have been tested: Adam, Adamax and SGD. Learning rates was, respectively, 0.001, 0.001 and 0.01. The aim was to find the best optimizer for the fruits dataset, that is the one that minimizes validation loss the most and as fast as possible. The results of the experiment can be summarized by the plot of fig. 4.

It turns out that SGD is the fastest to converge, whereas all optimizers provide nearly the same prediction accuracy in the long run.

SGD is probably the fastest to converge because of the default learning rate assigned to it (see section 5.4.8), which is smaller than the ones assigned to Adam and Adamax.

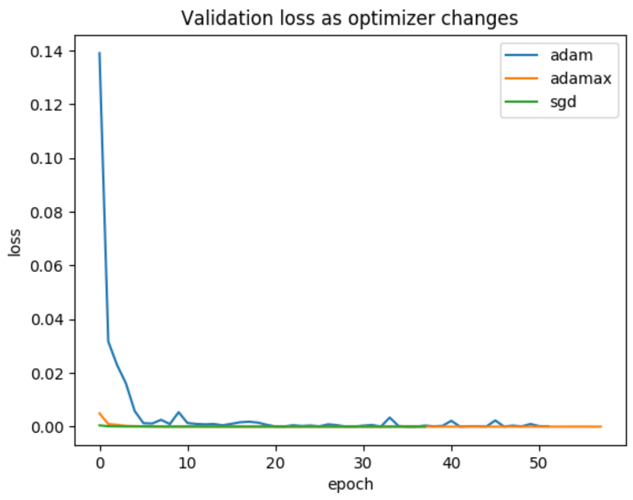

Figure 4: Validation loss as optimizer changes. Y axis: validation loss, X axis: epoch number

5.4.8 Learning rate

The learning rate is a number that, during the execution of the gradient descent algorithm, indicates how big the steps towards the direction of the gradient must be. Up to a certain point, the higher the learning rate, the faster the training. However, if learning rate is too big, the gradient descent may never reach a good minimum, but rather jump around it, and that is because of the big steps taken by the algorithm (which are directly related to the learning rate).

No formal experiments have been conducted on learning rate (and it should be pointed out that, in the optimizers used above, the learning rate is adaptive, meaning that it changes during training; the value we supply as the learning rate just becomes the upper-bound for the learning rate generated by the adaptive algorithm). Anyway, satisfying results were obtained using common values for it, which are: 0.001 for Adam and Adamax and 0.01 for SGD. Some of these optimizers also require other parameters such as momentum, beta, epsilon, etc. The values of these parameters, in the same way as the learning rate, have been chosen based on common usage, rather than by empirical observations.

5.5 Network topology experiments and considerations

5.5.1 Experiments summary

The same hyperparameters of table 2 have been used, with the only differences of the optimizer, which is now SGD instead of Adam, and learning rate, which is now 0.01 instead of 0.001.

Nr.	**Network topology**	**Parameters #**	**Test error**	**Test accuracy**
1	Convolutional 8 Max Pooling Convolutional 16 Max Pooling Convolutional 32 Max Pooling Dropout 0.4 Fully Connected 256 Fully Connected 10	139,930	0.00877	99.12%
2	Convolutional 8 Max Pooling Convolutional 16 Max Pooling Convolutional 32 Max Pooling Convolutional 64 Max Pooling Dropout 0.4 Fully Connected 256 Fully Connected 10	92,890	0.0261	97.39%
3	Convolutional 32 Max Pooling Convolutional 64 Max Pooling Convolutional 128 Max Pooling Dropout 0.4 Fully Connected 256 Fully Connected 10	620,362	0.00695	99.31%
4	Convolutional 8 Convolutional 8, s=2 Convolutional 16 Convolutional 16, s=2 Convolutional 32 Convolutional 32, s=2 Dropout 0.4 Fully Connected 10	23,314	0.0243	97.57%
5	DA: Flip h and v DA: Rotation 0.5 DA: Contrast 1.0 Convolutional 8 Max Pooling Convolutional 16 Max Pooling Convolutional 32 Max Pooling Dropout 0.4 Fully Connected 256 Fully Connected 10	139,930	0.0205	97.95%

Each network has been trained and evaluated against the test set 10 times. Then, average test loss and standard deviation have been calculated. Standard deviations, which are not present in the table above, are the following (*10ˆ-3): 2.33, 3.02, 1.12, 3.22 and 6.57, respectively.

5.5.2 Topology 1

This is the network topology used in section 5.4 for the hyperparameters experiments.

It yields good predictive performance while keeping the number of parameters pretty low.

5.5.3 Topology 2: adding one more convolutional layer

This topology is very similar to Topology 1. The only difference is that a new convolutional layer with 64 filters (and the usual max pooling layer after it) has been added to the network. As a consequence, the network should be able to recognize more features in images, and, given the fact that it is one level deeper, it should be able to build a more complex hierarchy of concepts (see penultimate paragraph of section 5.2 for a more detailed explanation of this heuristic).

Anyway, experiment results shows that the test error is higher for this network topology than for Topology 1. The reason behind this fact is that we are working with images of size 32x32. For each convolutional layer there is a max pooling layer with pool size 2x2 and each one of them reduce image dimension by a factor of 4. At the beginning, the image has dimension 32x32; after passing the first max pooling layer, the image has dimension 16x16; and so on. After three max pooling layers, the image has size 4x4: hence applying a convolutional layer with 64 filters is at best useless, at worst harmful, since that layer is very unlikely to be able to find 64 features in just 16 pixels.

So the takeaway is that, if we want to use more than 3 convolutional layers, we must also have bigger images. In fact, if we run the same exact network topology on images of size 64x64, we obtain a test accuracy of 99.50%, which is better than the accuracy of this topology and of Topology 1 for 32x32 images.

5.5.4 Topology 3: using convolutional layers with more filters

This network topology makes use of convolutional layers with an higher number of filters (32, 64 and 128, against 8, 16 and 32 of Topology 1).

With this topology the test error improves, probably meaning that there are more meaningful features to detect in the images than the number of filters used in Topology 1.

Further work may involve finding a network with enough filters to recognize all features of the images, but no more than that, in order to get the best predictive results while keeping the number of parameters of the network as small as possible.

5.5.5 Topology 4: an all convolutional network

This topology is a an all convolutional network, because it uses convolutional layers with strides 2x2 instead of max pooling layers with pool size 2x2. The idea of this kind of network has been taken from the paper cited at [9]. Note that, in this topology, the fully connected layer of 256 units is not present.

The number of parameters is very low with respect to the other topologies. On the other hand, predictive performance is not as good as the other networks.

5.5.6 Topology 5: data augmentation

This network topology makes use of data augmentation, which is a technique that increases the diversity and the size of the training set images by applying random transformations to them, such as rotation, flip, stretch, changes in contrast or brightness, and so on.

In this case, data augmentation has been used to apply the following random transformations to images:

- Horizontal and vertical flip
- Rotation
- Contrast

The test error worsens when data augmentation is applied. Experimenting with different variations of data augmentation (i.e. applying only flip and rotation transformations, or rotation only) has yielded similar results.

I believe the reason for these results can be found in the dataset itself. All the images of the dataset have been collected and elaborated using the same exact procedure. The result is that every image shares with all others some characteristics, such as brightness, background color, position of the fruit/vegetable in the image, etc. Applying random transformations to the training set through data augmentation may be useless if the model will be used against a test set collected using the same approach as the training set, because the transformations applied to the training images will not be found in the test images.

This does not mean that data augmentation is not useful. The model trained through data augmentation is more general and less overfitted to the training set. But because the training and test sets share so many similarities, this generalization of the model reduces the prediction accuracy rather than increasing it. In a sense, in this case, overfitting the training set yields better prediction performances on the test set.

However, if we were to evaluate the model against a new test set collected in a different way, it would make sense to apply data augmentation as it would probably increase the generalization ability of the model, hence prediction accuracy as well.

5.5.7 Wrong predictions of Topology 3

In this section are listed some of the images that Topology 3 wrongly predicts (Topology 3 has been chosen because it is the one with better performance).

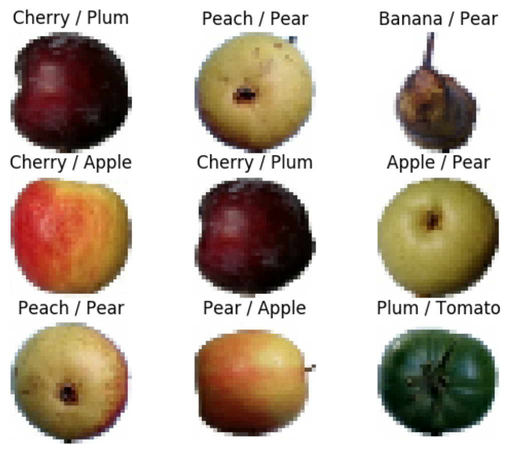

Figure 5: Some of the wrongly predicted images of Topology 3. The left label is the one predicted by the model, the other one is the correct label.

From the image above it is clear that the model has problems distinguishing between apples and pears, cherries and plums, and peaches and pears. However, it is true that some of these distinctions may be wrongly made by humans as well, since these types of fruits can be very similar to each other.

Further work may involve improving the model in a way that it understands how to distinguish between these types of fruits. Increasing image resolution may be the fastest way to achieve that.

6 Conclusions

The report has provided an overview of various hyperparameters and network topologies and their influence on predictive performance: in particular, some of them have been analyzed in a more theoretical way, others have been investigated through empirical experiments. Moreover, some experiments led to a satisfying interpretation of their results, others may require further work (see section 6.2) in order to be properly interpreted.

All in all, I believe this report achieves its goal of providing an overview of common design decisions in working with convolutional neural networks and reporting their impact on model predictive performance.

6.1 Comparison with other works

In some of the experiments of this paper, the network was able to predict more than 99% of the test set images correctly. Taking a look at experimental results of paper [4], which uses the same dataset, it seems like results are far inferior, because the best predictive performance reached is about 95%.

The reason for this is that the dataset used in this paper has been modified, as stated in section 4. So, in our dataset, there are far less classes (10 instead of 131) and we classify by type of fruit/vegetable, not variety. I think this latter consideration, in particular, has a lot of influence on the final predictive performance, since it is easier for the model to wrongly predict between varieties than between types (e.g. it is difficult to distinguish between two varieties of apples, since they look very similar; on the other hand, it is easier to distinguish between an apple and a banana).

The above consideration is also confirmed in paper [4] itself, at the end of section 7. An image with some examples of the wrongly classified images shows that, of 8 errors, 3 were due to wrong variety classification (the type was predicted correctly).

6.2 Further work

The aim of the report was just to provide an overview of the main hyperparameter and network topology choices and their influence on model performance, so there was no focus on a specific area. This means that a lot of the experiments conducted may be continued in various directions. However, in particular, the second batch size experiment of section 5.4.3, the network of Topology 3 of section 5.5.4 and the all convolutional network adopted in section 5.5.5 require further work, as mentioned in their respective sections.

Moreover, some parameters have not been taken into consideration at all, such as the activation functions of layers, certain parameters of optimizers (as noted in section 5.4.8), the convolutional window size of convolutional layers, and others.

References

[1] TensorFlow, Tensorflow. [online] Available at: https://www.tensorflow.org [Accessed 30 September 2020].

[2] TensorFlow, Tensorflow. [online] Available at: https://www.tensorflow.org/guide/effective_tf2 , section "A brief summary of major changes" [Accessed 1 October 2020].

[3] Oltean, M., and Muresan, H. Fruits 360 dataset on GitHub. [online] Available at: https://github.com/Horea94/Fruit-Images-Dataset [Accessed 30 September 2020].

[4] Muresan, H., and Oltean, M. Fruit recognition from images using deep learning, pp. 9-14.

[5] Graziosi, R. Repository with scripts used in this paper. [online] Available at: https://github.com/riggraz/smml-project [Accessed 30 September 2020].

[6] Wikipedia, Convolutional neural network article on Wikipedia. [online] Available at: https://en.wikipedia.org/wiki/Convolutional_neural_network [Accessed 30 September 2020].

[7] Nielsen, M., Neural Networks and Deep Learning, chapter 6. [online] Available at: http://neuralnetworksanddeeplearning.com [Accessed 30 September 2020].

[8] LeCun, Y., Cortes, C., and Burges, C. J., MNIST database of handwritten digits. [online] Available at: http://yann.lecun.com/exdb/mnist/

[9] Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. A., Striving for Simplicity: the All Convolutional Net. *Biosystems Engineering 118* (2014)