Convolutional Neural Networks and Image Classification: "Muffin vs Chihuahua"

Machine learning, statistical learning, deep learning and artificial intelligence

Rodion Horbunov
Matriculation number 988092
rodion.horbunov@studenti.unimi.it

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/us or any other person for assessment on this or any other course of study.

Abstract

During this project I developed different Convolutional Neural Networks for images classification of muffins and chihuahuas, binary classification problem. I experimented with 4 network architectures and training hyperparameters, and I investigated their influence on model performance regarding the loss and accuracy. Overall, I experimented 18 models. The final, risk estimate I computed with 5-fold cross validation technique. Model with three convolutional and max pooling layers, one drop out layer with 0.5 rate and one batch normalization layer with size of 64, showed the highest accuracy and lowest loss among other ones.

1 Introduction

Neural networks rebounded around 2010 with big successes in image classification. Around that time, massive databases of labeled images were being accumulated, with ever increasing numbers of classes. A special family of convolutional neural networks (CNNs) has evolved for convolutional neural network classifying images such as these, and has shown spectacular success on wide range of problems. [1] This report provide developing of deep learning solution to classify images using CNN. The goal of this report is to try different architecture and to find the best architecture to classify the image by assigning the right specific label to it, basing on “Muffin vs Chihuahua” dataset. In the second paragraph, I would describe the data and dataset pre-processing. In the third paragraph, I would describe theoretic behind the CNN layers with some image examples from the dataset of the project. Consequently, I would provide the description of developing and analysis of different network architectures in respect with chosen hyperparameters and hyperparameter’s values. Additionally, I would provide accuracy and loss for each architecture and model. Finally, I would provide the experimental results obtained and point out the conclusions.

2 Dataset and Data Preprocessing

The Dataset has 6,000 colored JPEG images of different size in total, scraped from google images. Duplicate images have been removed. The number of images with muffins - 2718 , number of images with chihuahua - 3199. The dataset is divided on test set and training set with a 20-80 split.

Images in the dataset were converted to the RGB format and rescaled the specific size (150 , 150) in order to obtain the pictures of the same size.

For computational purpose optimization that are performed in colab environment, images converted to grayscale, spaced tones ranging from black to white through shades of gray. Data that goes into neural networks should usually be normalized in some way to make it more amenable to processing by the network (i.e. It is uncommon to feed raw pixels into a ConvNet.)

For this case, I preprocessed the images by normalizing the pixel values to be in the [0, 1] range (originally all values are in the [0, 255] range).

Therefore, the chihuahua image is associated with label 0 and muffin with label 1.

The primary evaluation of architecture of the models would be performed basing on initial image separation provided at Kagle. The 5-fold cross validation would be performed on the best models from the primary assessment.

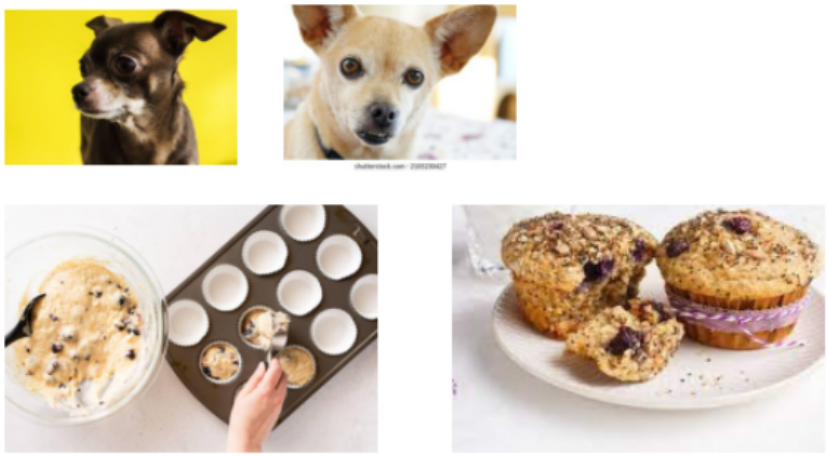


Figure 1: Sample of non-trasformed images.

Figure 2: Sample of trasformed images.

3 Theoretical methodology

Deep learning, a subset of machine learning, employs artificial neural networks with multiple layers, also known as deep neural networks, to model and understand complex patterns in datasets. These neural networks, akin to the human brain, process input data through a series of interconnected nodes or "neurons", with each layer learning to transform its input data into a slightly more abstract representation. In this part I provide the meaning of main elements behind

Neural Networks and Convolutional Neural Networks, that are used mainly and referred to the my experiments.

3.1 Feedforward Neural Network

A feedforward neural network is a type of artificial neural network where the information flows in one direction, from the input layer through intermediate layers to the output layer. It is called "feedforward" because the data moves forward without loops or cycles.

The structure of a feedforward NN is a directed acyclic graph G = (V, E) where each node j (except for the input nodes) computes a function g(v) whose argument v is the value of the nodes i such that (i, j) ∈ E. The nodes in V are partitioned in three subsets:

$$V = V_{IN} \cup V_{hid} \cup V_{out}$$

Where V_{IN} (with $|V_{IN}|$ = d) are the input nodes which have no incoming edges, V_{out} (with |Vout| = n) are the output nodes which have no outgoing edge, and V_{hid} are the hidden nodes, which have both incoming and outgoing edges. The set of input nodes and the set of output nodes are respectively called the input layer and the hidden layer. The simplest form of feedforward NN is a multilayered NN, in which the nodes of V can be partitioned in a sequence of layers such that each node of a layer has incoming edges only from nodes in the previous layer and outgoing edges only to nodes of the next layer. The layers containing the hidden nodes are called hidden layers.

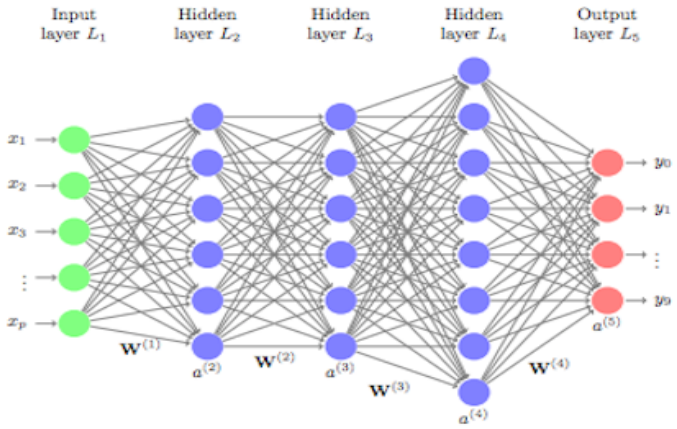


Figure 3: Example of Feedforward Neural Network

3.2 Convolutional Neural Networks

A special family of convolutional neural networks (CNNs) has evolved for classifying images such as these, and has shown spectacular success on a wide range of problems. CNNs mimic to some degree how humans classify images, by recognizing specific features or patterns anywhere in the image that distinguish each particular object class.

The network first identifies low-level features in the input image, such as small edges, patches of color, and the like. These low-level features are then combined to form higher-level features. Eventually, the presence or absence of these higher-level features contributes to the probability of any given output class.

3.2.1 Convolutional Layer

A convolution layer is made up of a large number of convolution filters, each of which is a template that determines whether a particular local feature is present in an image. A convolution filter relies on a very simple operation, called a convolution, which basically amounts to repeatedly multiplying matrix elements and then adding the results.

In a convolution layer, we use a whole bank of filters to pick out a variety of differently-oriented edges and shapes in the image. Using predefined filters (***Kernels***) in this way is standard practice in image processing. By contrast, with CNNs the filters are learned for the specific classification task. We can think of the filter weights as the parameters going from an input layer to a hidden layer, with one hidden unit for each pixel in the convolved image. During this layer user can decide some important parameters like size of the filter, number of filters, activation function etc.

As the example I provide the chihuahua convolution that enhance edges of image with 3x3 Kernel filter.

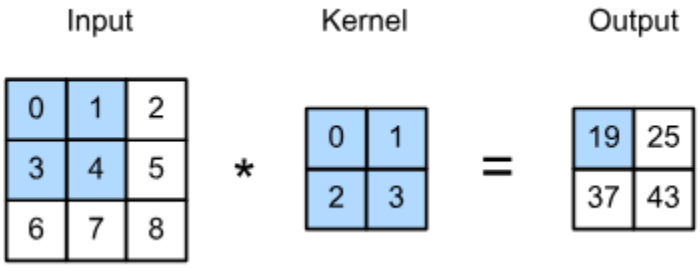


Figure 4: Convolution process

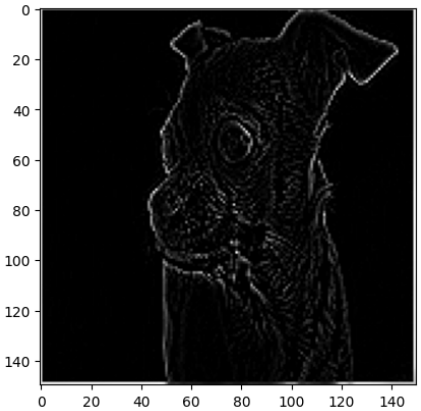


Figure 5: Example of chihuahua convolution

3.2.2 Pooling Layers

This is the layer which is designed to compress the image, while maintaining the content of the features that were highlighted by the convolution. Also, pooling layers are used in convolutional neural networks (CNNs) to decrease the spatial dimensions of feature maps. This reduction in dimensions helps to reduce the computational load during training, potentially improving training speed. Additionally, pooling can help control overfitting by reducing the number of weights in the network, thus promoting generalization. By specifying (2,2) for the MaxPooling, the effect is to quarter the size of the image. The idea is that it creates a 2x2 array of pixels, and picks the biggest one. Thus, it turns 4 pixels into 1. It repeats this across the image, and in doing so, it halves both the number of horizontal and vertical pixels, effectively reducing the image to 25% of the original image.

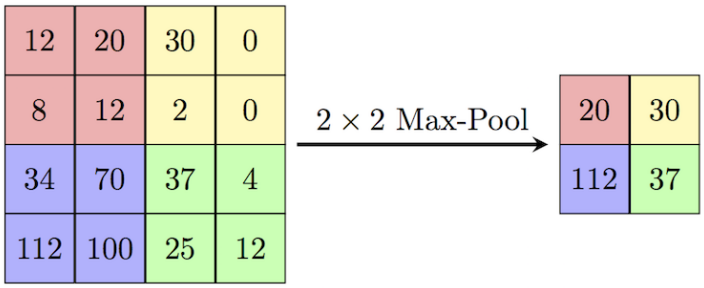


Figure 6: Example of pooling process

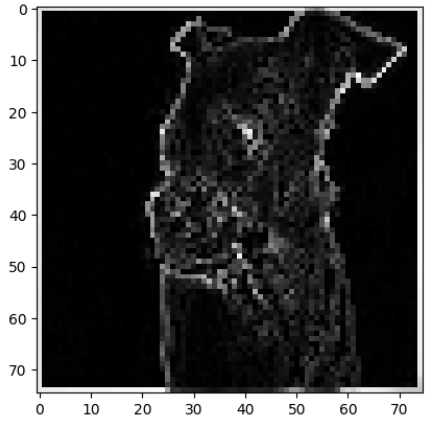


Figure 7: Example of chihuahua convolution with pooling

3.2.3 Data Augmentation Layer

Data Augmentation Layer is used to generate more training data from existing examples by applying random transformations such as rotation, scaling, or cropping. This helps improve model performance by providing more varied inputs.

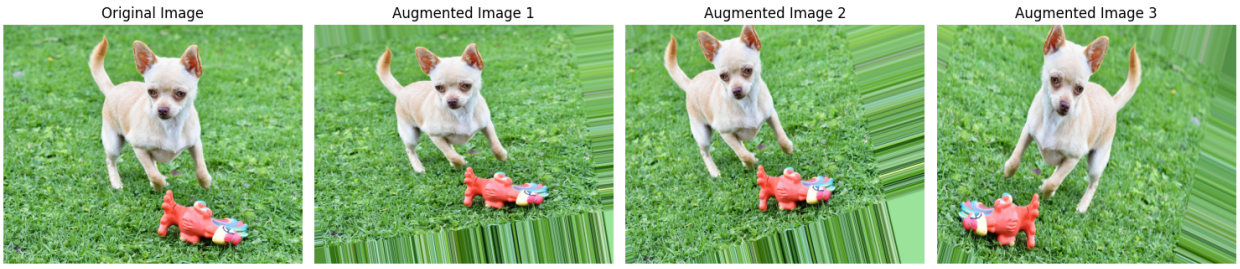


Figure 8: Example of augmentation of chihuahua

3.2.4 Flatten Layer and Dense Layer

The Flatten Layer is used to convert the final feature maps into a single one-dimensional vector, necessary for transitioning between convolutional layers and dense layers after a series of Convolutional and MaxPooling layers. This flattened vector can then be fed into Dense layers, also known as Fully Connected layers. These dense layers are capable of learning intricate relationships among the high-level features extracted by the earlier convolutional and max-pooling layers. This allows the model to comprehend images by establishing information flow between each input pixel and each output class. The hyperparameters of the dense layers include the number of nodes and the activation function. Ultimately, the one-dimensional vector generated by the flatten layer passes through the dense layers to perform the classification task, producing probabilities for each image's association with each class.

3.2.4 Dropout Layer

Dropout layers are typically positioned after the pooling layer in a neural network architecture. They act as masks that selectively nullify the contributions of certain neurons towards the next layer, while leaving the rest unaffected. Dropout layers can be applied to both the input vector, where they nullify specific features, and hidden layers, where they nullify hidden neurons. These layers play a crucial role in training convolutional neural networks as they help prevent overfitting by reducing the disproportionate influence of the first batch of training samples. Dropout can also be applied after a dense layer to further enhance regularization. So, during training, randomly selected neurons are ignored, reducing the dependency on any one neuron and encouraging a more robust network.

3.2.5 Batch Normalization Layer

Batch normalization layers are commonly inserted after convolutional layers and before pooling layers in a neural network architecture. They are utilized, as a technique to enhance the coordination between multiple layers by scaling the layer's output. Specifically, batch normalization standardizes the activations of each input variable within a mini-batch, ensuring that assumptions made by subsequent layers regarding input distribution during weight updates

remain stable. This stabilization effect not only speeds up the training process but also improves its overall efficiency.

3.3 Activation functions

Activation Functions are mathematical equations that determine the output of a neural network. These functions help to decide if a neuron should be activated or not, introducing non-linear properties to the model. The most popular functions:

1. **Sigmoid function.**

Also known as the logistic function, maps the input value to a range between 0 and 1. It is given by the formula: $f(x) = 1 / (1 + exp(-x))$ The sigmoid function is useful for binary classification problems, as it can interpret the output as a probability. However, it suffers from the vanishing gradient problem, limiting its effectiveness in deep neural networks.

2. **ReLU function**

The rectified linear unit (ReLU) function sets the output to zero for negative inputs and leaves positive inputs unchanged. It is defined as $f(x) = max(0, x)$. ReLU is widely used in deep learning due to its simplicity and ability to mitigate the vanishing gradient problem. It provides faster convergence during training and helps in learning complex features.

3.4 Optimizers

Optimizers are algorithms or methods used to adjust the attributes of the neural network, such as weights and learning rate, to reduce losses. The most popular optimizers:

1. SGD (Stochastic Gradient Descent)

SGD is a variation of Gradient Descent that updates model parameters after computing the loss for each training example. It aims to overcome the slow convergence of standard Gradient Descent by calculating the gradient for a randomly chosen observation at each step. However, both SGD and Gradient Descent can get stuck at local minimals, leading to the preference for other optimizers.

2. RMSprop (Root Mean Square Propagation)

Adapts the weights using gradient descent but employs a unique adaptive learning rate for each parameter. By scaling the rates with the moving average of squared gradients, RMSprop stabilizes and speeds up the learning process.

3. ADAM (Adaptive Moment Optimization)

Combines the benefits of SGD with momentum and RMSprop. It performs gradient descent using both scaling and adaptive learning rates, providing an effective approach for optimization. ADAM is recognized for its ability to achieve fast convergence and control various optimization challenges.

In this case, I am using the ***RMSprop optimization*** algorithm is preferable to stochastic gradient descent (SGD), because RMSprop automates learning-rate tuning for us. (other

optimizers, such as Adam and Adagrad, also automatically adapt the learning rate during training, and would work equally well here.)

3.5 Loss Function

Loss Functions, also known as cost functions, measure the inconsistency between predicted and actual outcomes. For binary classification problems, ***Binary Cross-Entropy*** is commonly used. The formula can be found below. Where y_i is the label associated to each point, while $p(y_i)$ is its predicted probability.

$$-\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Figure 9: Binary Cross-Entropy Formula

4 Architecture of models and experimental results

4.1 Architecture options

The further architectures would be considered:

1. Base Architecture with out Convolution and Max Pooling layers.
2. Base Architecture with Convolution and Max Pooling layers.
3. Base Architecture with Convolution, Max Pooling layers and Dropout.
4. Base Architecture with Convolution, Max Pooling layers, Dropout with Batch Normalization.

4.2 Model compilation setting

1. Epochs: 50, additionally I provided “callbacks” - the function that stops the model training in case if training accuracy reached 100% over the last Epoch.
2. Activation functions:
   For Convolution Layers : ReLU
   For last Dense Layer: Sigmoid
3. Optimizer: RMSprop
4. Max Pooling: 2x2
5. The size of the Convolution : 3x3
6. Metrics: Accuracy
7. Loss: Binary Crossentropy

4.3 Models and Hyperparamets

Basing on architectures of models above I would consider further models with specific hyperparameters tunning. Additionally going through architecture options, that were described bellow, only the best performed hyperparameter would be chosen.

model_1_32 , model_1_64, model_1_128, model_1_256, model_1_512 - the set of models with out Convolution and Max Pooling layers, with 32, 64, 128, 256 and 512 size of the output from the dense layer, respectively.

From the architecture 1, I choose the model “model1_128”, because it improve accuracy significantly among the previous models. Additionally, for computational time optimisation I didn’t choose the model “model1_256”, due to low increase of validation accuracy (0.4%) and high increase of time per epoch (57.1%) , comparing to “model1_128” model. The result of architecture 1, provided in Table 1.

Archt. №	Model	Training loss	Training accuracy	Validation loss	Validation accuracy	Validation accuracy change
1	model1_32	0.690	0.541	0.690	0.541	-
1	model1_64	0.732	0.529	0.732	0.529	-2.2%
1	***model1_128***	***0.666***	***0.627***	***0.666***	***0.627***	***18.5%***
1	model1_256	0.723	0.629	0.723	0.629	0.4%
1	model1_512	0.774	0.616	0.774	0.616	-2.1%

Table 1. Architecture 1 models result.

So, the base for Architecture 2 would be the model - “model1_128”.The further models were created and tested:

model2_32 - with 1 layer of Convolution and Max Pooling layers, with 32 filters computed.

model2_64 - with 2 layers of Convolution and Max Pooling layers, with 32 and 64 filters computed.

model2_128 - 3 layers of Convolution and Max Pooling layers, with 32, 64 and 128 filters computed.

model2_256 - 4 layers of Convolution and Max Pooling layers, with 32, 64, 128 and 256 filters computed.

The result of architecture 2, provided in Table 2.

Archt. №	Model	Training loss	Training accuracy	Validation loss	Validation accuracy	Validation accuracy change
2	model2_32	1.654	0.805	1.654	0.805	30.7%
2	model2_64	1.736	0.833	1.736	0.833	3.5%
2	***model2_128***	***1.904***	***0.863***	***1.904***	***0.863***	***3.7%***
2	model2_256	1.107	0.889	1.107	0.889	3.0%

Table 2. Architecture 2 models result

So, the base for Architecture 3 would be the model - “model2_128”, due to the same factors on the previous step (accuracy, computation complexity). Althought, it did not show the best validation accuracy result, if differ from the “model2_256” for 3%. The further models were created and tested:

model3_1_05 - The best performed Model with respect to first and second architecture. One Dropout layer after all Convolution layers with rate - 0.5.

model3_1_01 - The best performed Model with respect to first and second architecture. One Dropout layer after all Convolution layers with rate - 0.1.

model3_grow_01 - The best performed Model with respect to first and second architecture. Dropout layers after all Convolution layers with rate increasing rate from 0.1 up to 0.5. Step - 0.1.

model3_all_05 - The best performed Model with respect to first and second architecture. Dropout layers after all Convolution layers with rate 0.5.

model3_all_01 - The best performed Model with respect to first and second architecture. Dropout layers after all Convolution layers with rate 0.1.

The result of architecture 3, provided in Table 3.

Archt. №	Model	Training loss	Training accuracy	Validation loss	Validation accuracy	Validation accuracy change
3	model3_1_01	1.729	0.852	1.729	0.852	-4.2%
3	model3_1_05	1.510	0.867	1.510	0.867	1.8%
3	model3_grow_01	1.056	0.870	1.056	0.870	0.3%
3	***model3_all_05***	***0.401***	***0.902***	***0.401***	***0.902***	***3.7%***
3	model3_all_01	1.641	0.863	1.641	0.863	-4.3%

Table 3. Architecture 3 models result

So, the base for Architecture 4 would be the model - “model3_all_05”,, due to the same factors on the previous step (accuracy, computation complexity), the further models were created and tested with Batch normalization - 64:

model4_last_b_normz - The best performed model among 3 Architectures, Batch normalization layer before last Dense layer.

model4_1_last_b_normz - The best performed model among 3 Architecture, Batch normalization layer before last Dense layer and after first Convolutional layer.

model4_2_last_b_normz - The best performed model among 3 Architecture, Batch normalization layer before last Dense layer and after first two Convolutional layers.

model4_3_last_b_normz - The best performed model among 3 Architecture, Batch normalization layer before last Dense layer and after first tree Convolutional layers.

The result of architecture 4, provided in Table 4.

Archt. №	Model	Training loss	Training accuracy	Validation loss	Validation accuracy	Validation accuracy change
4	***model4_last_b_normz***	***0.587***	***0.895***	***0.587***	***0.895***	***3.7%***
4	model4_1_last_b_normz	0.523	0.889	0.523	0.889	-0.7%
4	model4_2_last_b_normz	0.584	0.840	0.584	0.840	-5.5%
4	model4_3_last_b_normz	0.972	0.837	0.972	0.837	-0.4%

Table 4. Architecture 4 models result

The overall results over all models are provided below in Appendix 3. Also, the plots of loss and accuracy for training and validation sets provided in Appendix 1.

So far the best models regarding the validation accuracy are model3_all_05 and model4_last_b_normz. In Appendix 2 the structure of the these two best models are provided. In the next part, I would provide the 5-fold cross validation for these two models to obtain more clear and unbiased results.

4.4 Training with 5-fold cross validation

Ideally, all experiments should have been conducted this way, and on a larger grid of hyperparameters for each of the architectures. Previously, I didnt trained the model in such way, due to computational purposes and limitation of resources. The training loss remains the Binary Cross-Entropy while the validation loss is computed through the zero-one loss. The zero-one loss corresponds to the pary of misclassified observations. For optimization, I used 20 Epoches.

The next part of the work was conducted by training the two models:

1. Model3_all_05, as the model with best validation accuracy
2. Model4_last_b_normz, as the model with second best validation accuracy and as the model that includes all types of different layers.

The result of the models provided in Table 5, by averaging the losses and accuracy metrics. As it can be observed, the model4_last_b_normz model shows the best accuracy and lowest loss. The average results of model3_all_05 and model4_last_b_normz is down performed

the same models above because CV results show more unbiased results. The standart deviation for model3_all_05 is 0.104 and for model4_last_b_normz - 0.037, that means that among these two models the last one is more stable, regarding the results, that is also the advantage.

Model	Average CV Zero-one Loss	Average CV Accuracy
Model3_all_05	0.185918	0.814081
Model4_last_b_normz	0.166323	0.833676

Table 5. Results of 5-Fold Cross Validation

5 Conclusion

The aim of this work was to develop and try different network architectures in order to perform binary classification on images of chihuahuas and muffins.

The best model showed the accuracy level of - 83.4% (Model4_last_b_normz). This model includes 3 convolutional and maxpooling layers, 1 drop out layer with 0.5 rate and one batch normalization layer with size of the batch - 64. Overall, I believe that is great results.

The experimental results helped me to draw the further conclusions:

1. Adding blocks of layers not allways means increasing of model performance accuracy.
2. Regularization technique as Dropout is effective in reducing overfitting.
3. Hyperparameters tuning can improve the performance of the models, but it is not behave constantly during the increasing. Also, it is time and computation consuming, but can improve the accuracy of the model.
4. Cross validation decrease the accuracy of “one set” trained model and makes the result more stable and constant, ideally all models should be trained in the same way.
5. It is possible to obtaine high level of accuracy using only basing structure of Convolutional and MaxPooling layers.

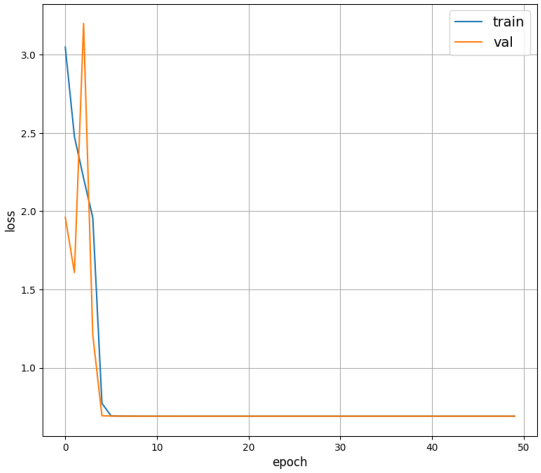
To further improvement of the predictions some steps should be performed. For example, originally pictures, scaled down to 150x150 size, and in greyscale. A larger size for the images and using the colored images could theoretically improved the results significantly and find certain patterns.

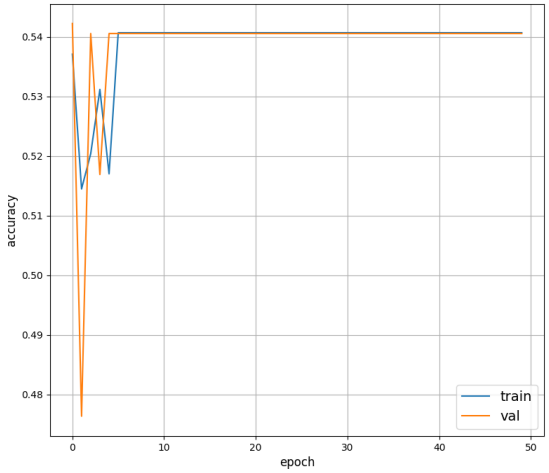
Important to mention, that data augmentation, that increase the number of dataset by rotating, flipping, cropping and adjusting the brightness of existing images could also be used and increased the accuracy of the model. Additionally, “prepare” the model to the new - out of dataset images, because it is only 6000 images in the dataset.

There are numerous variation of architectures that can be tried constantly adding different variations of layers, hyperparametrs can be tunned with lower step. Tme consumption and complexity with respect of performance improvements should be considered in choosing model.

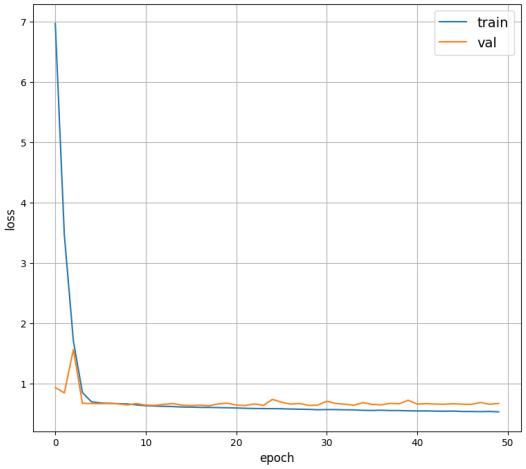
Appendix 1. Plots of loss and accuracy for training and validation sets.

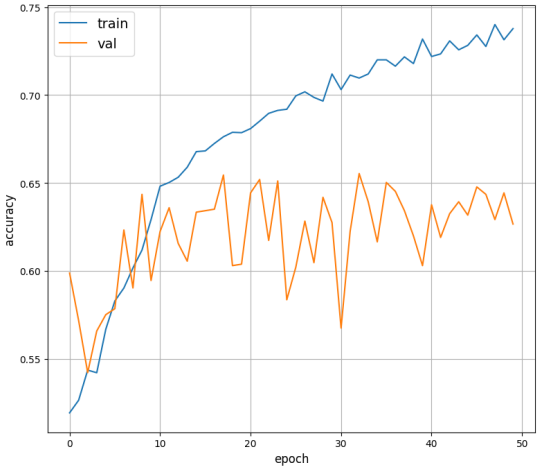
model1_32



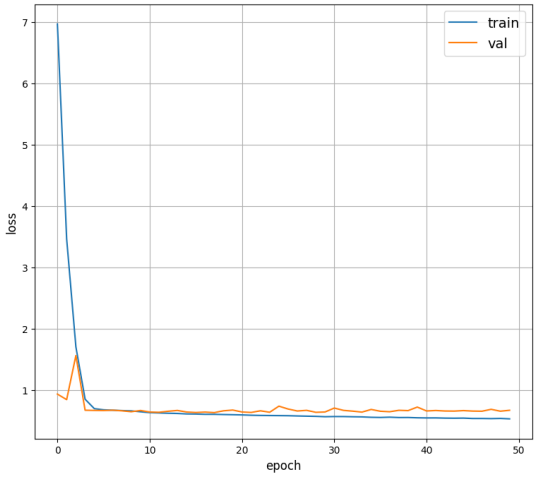


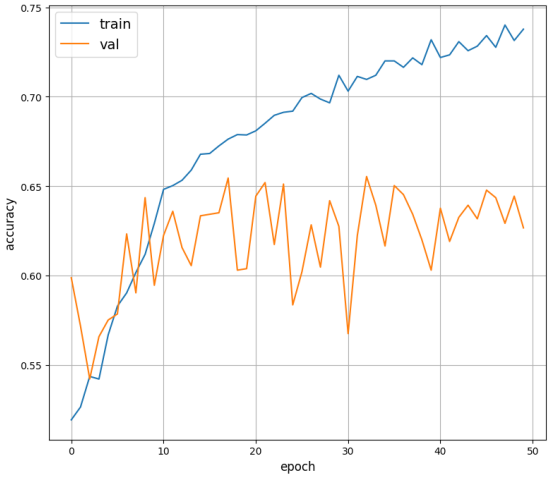
model1_64



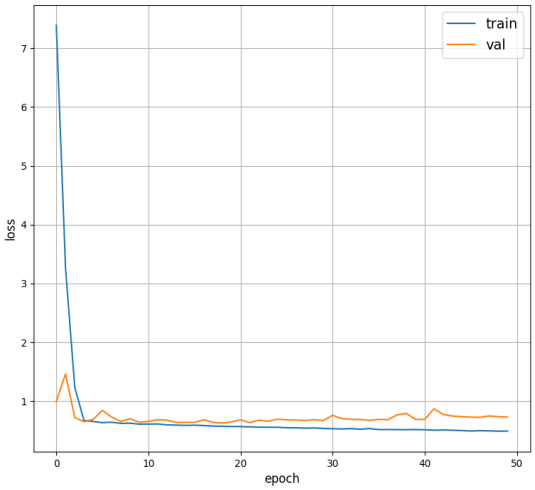


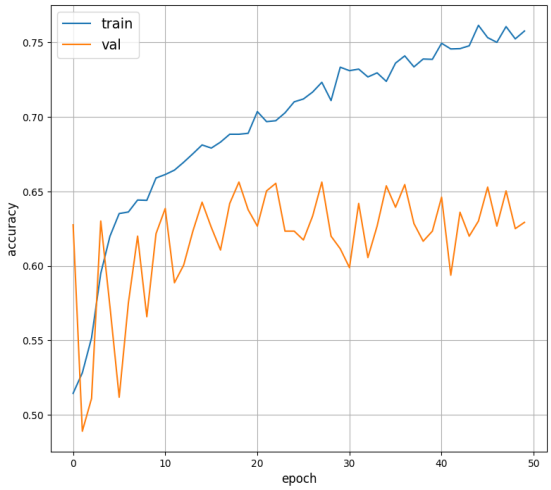
model1_128



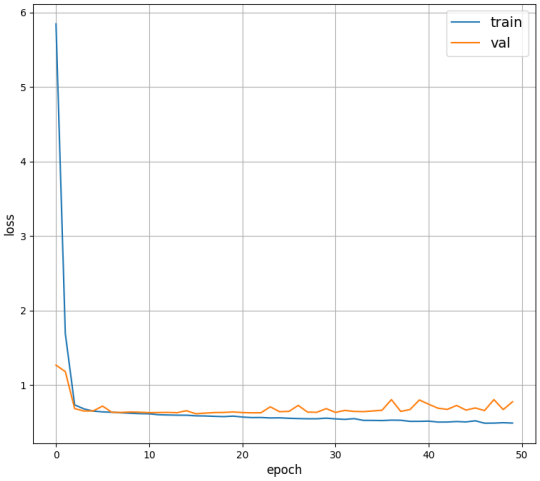


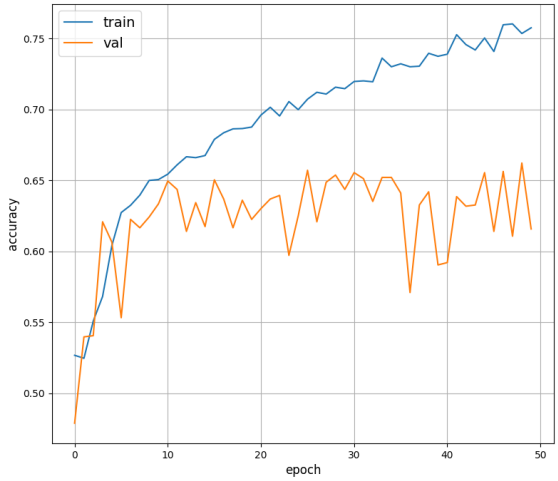
model1_256



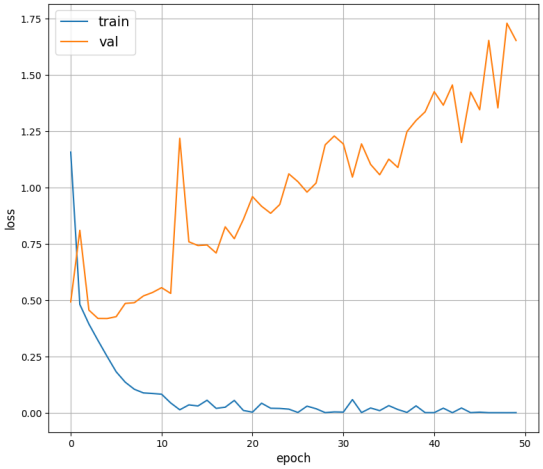


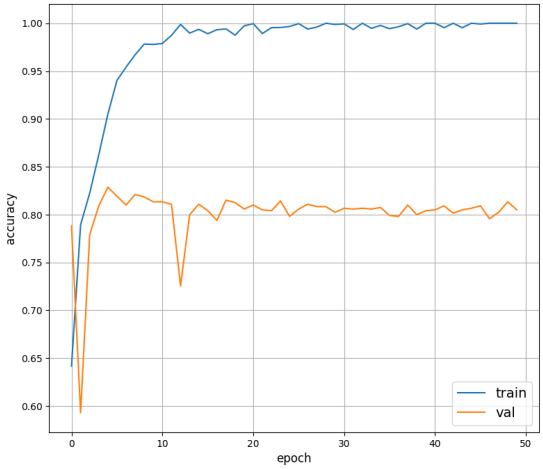
model1_512



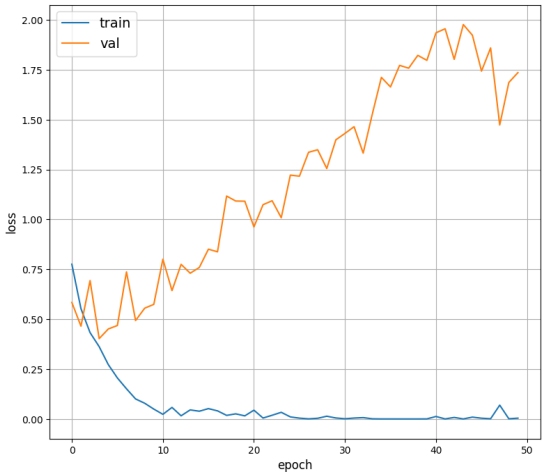


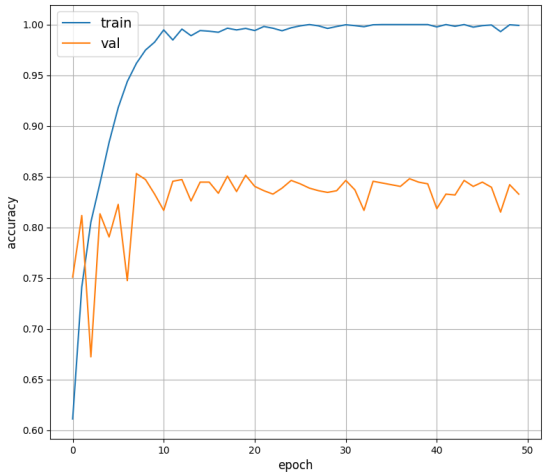
model2_32



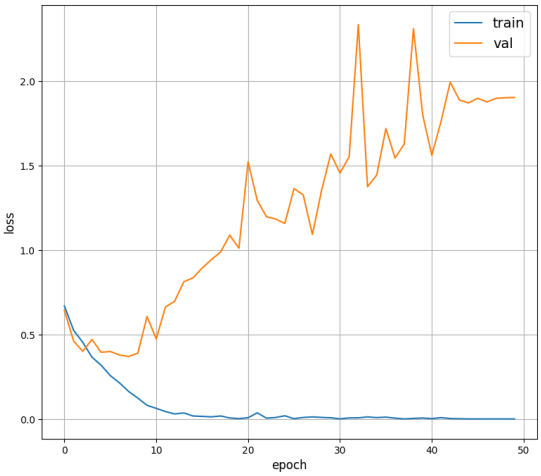


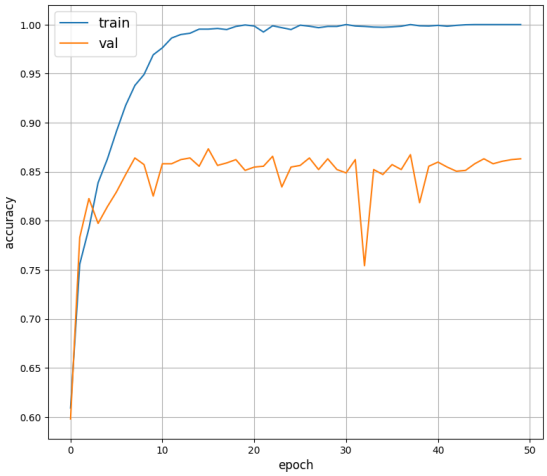
model2_64



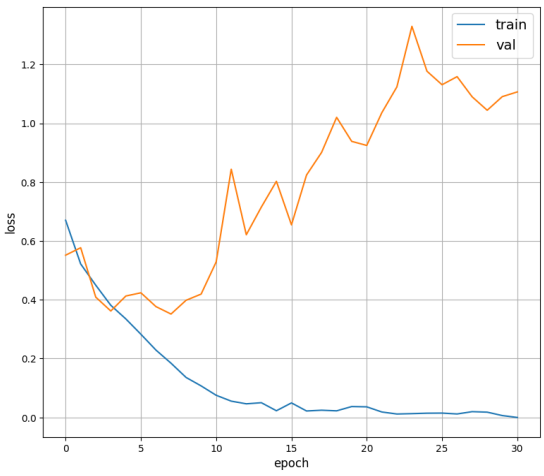


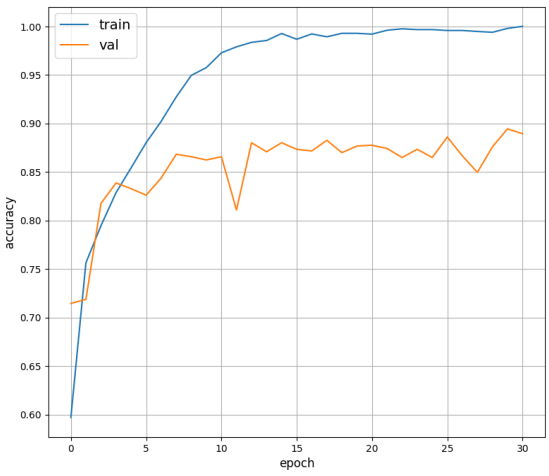
model2_128



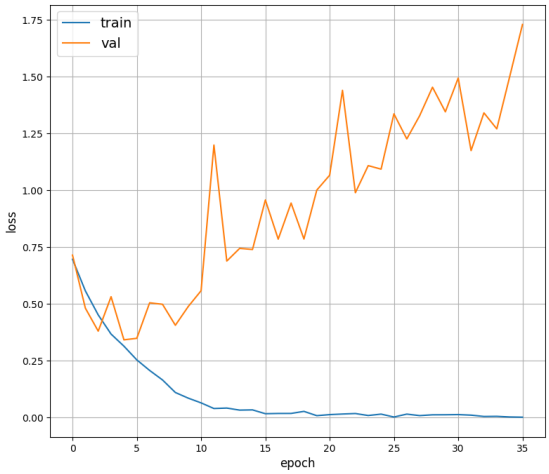


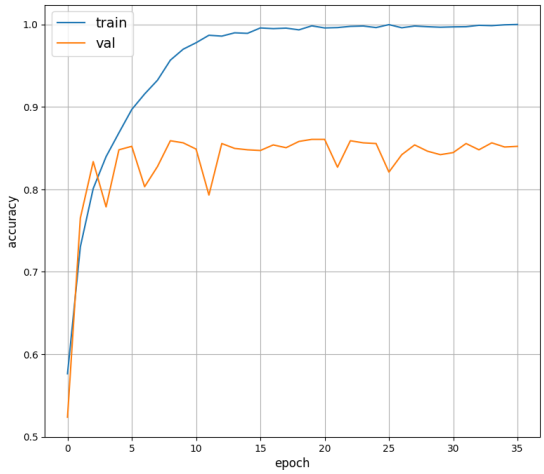
model2_256



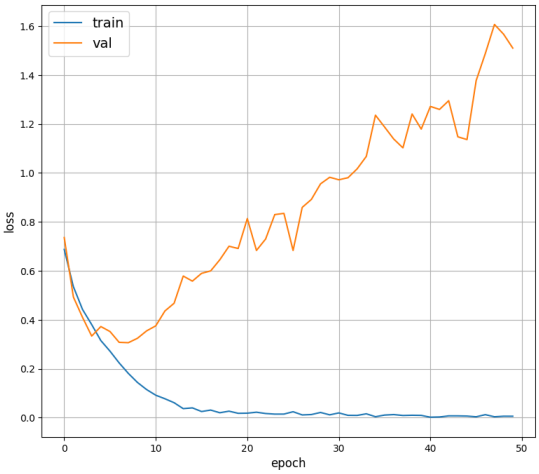


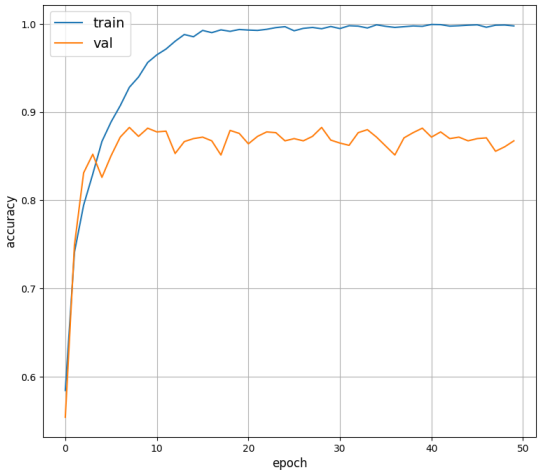
model3_1_01



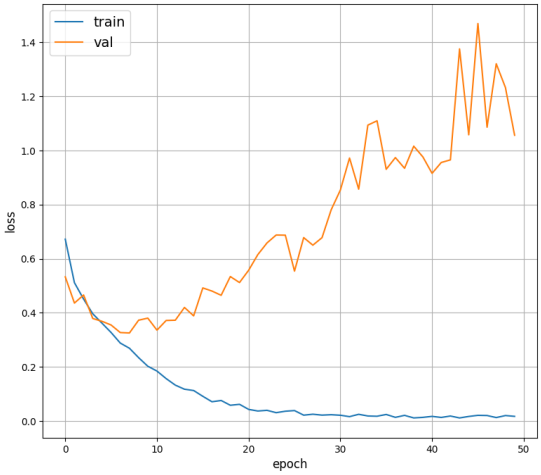


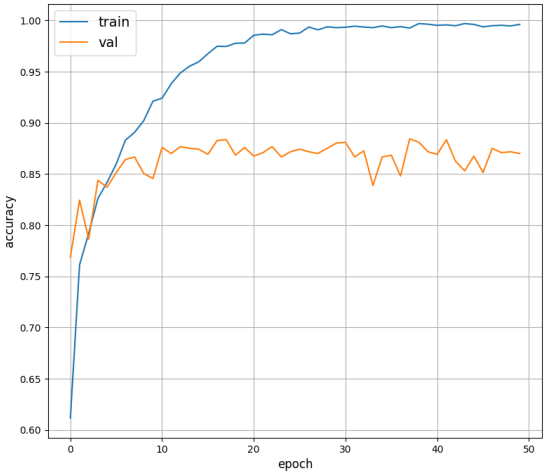
model3_1_05



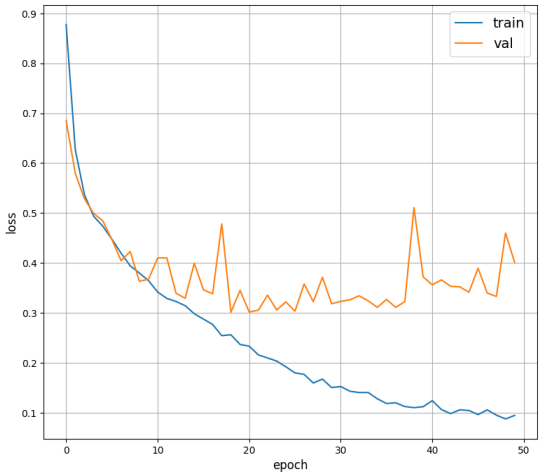


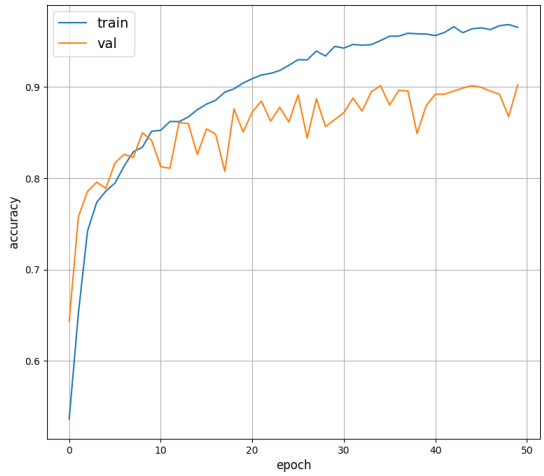
model3_grow_01



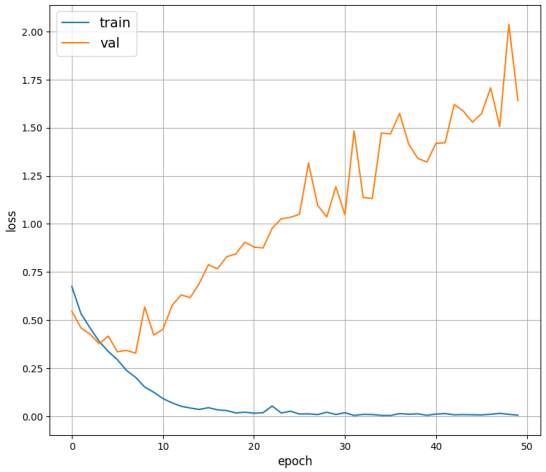


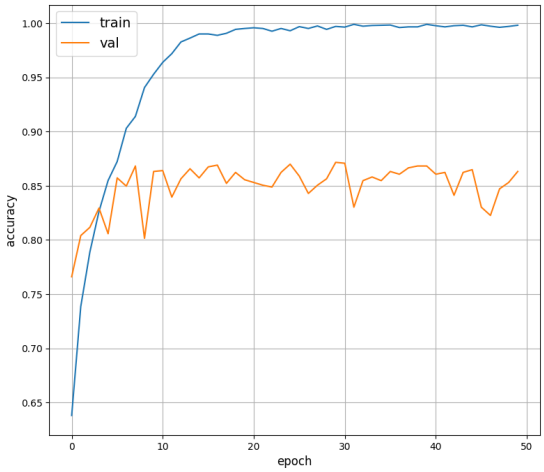
model3_all_05



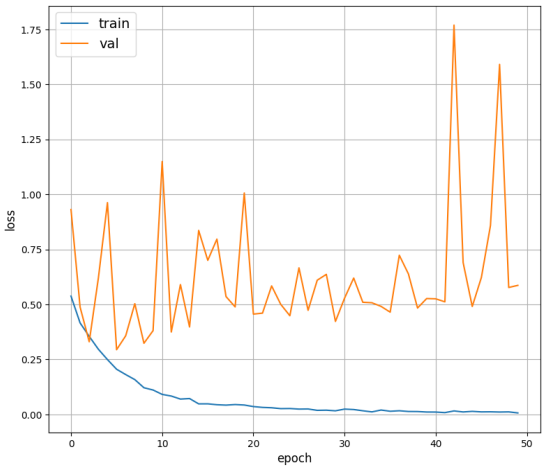


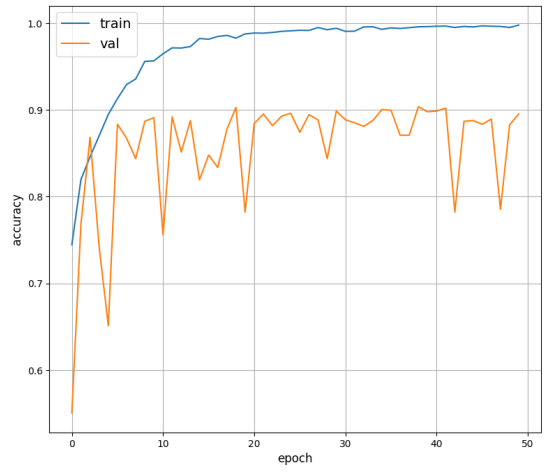
model3_all_01



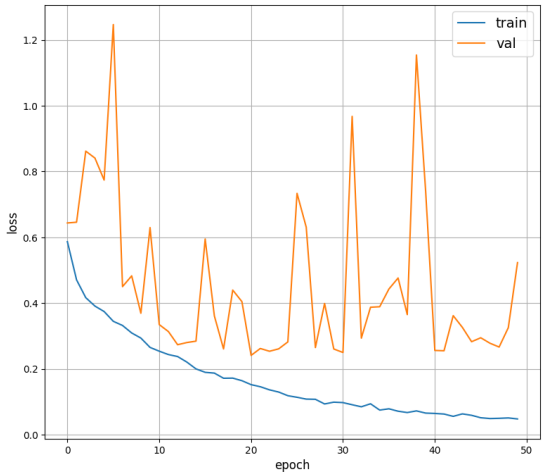


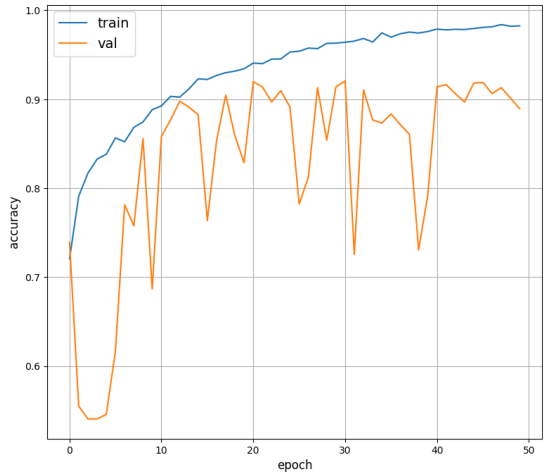
model4_last_b_normz



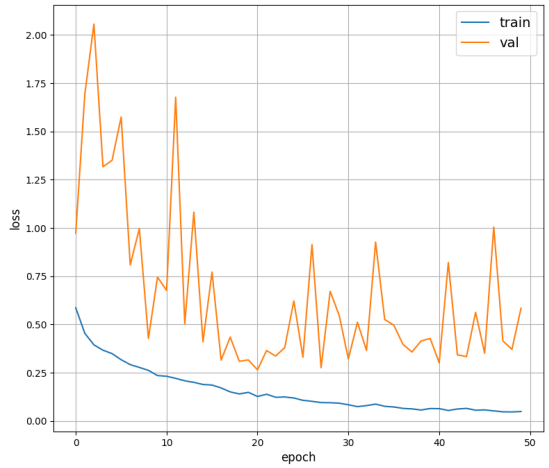


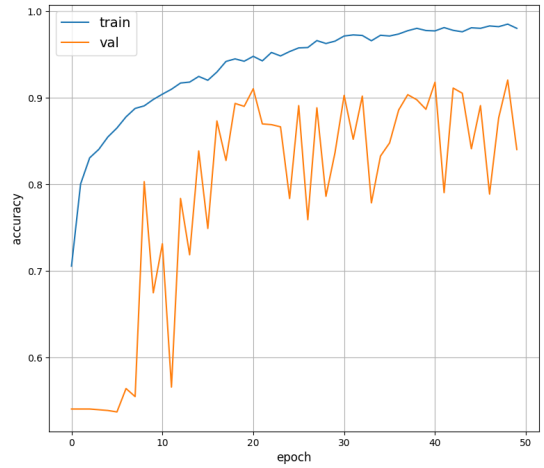
model4_1_last_b_normz



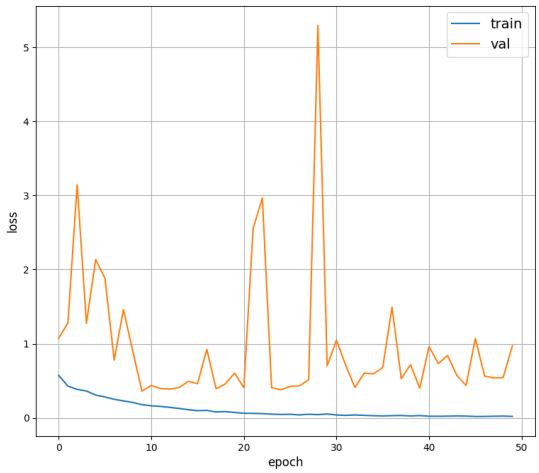


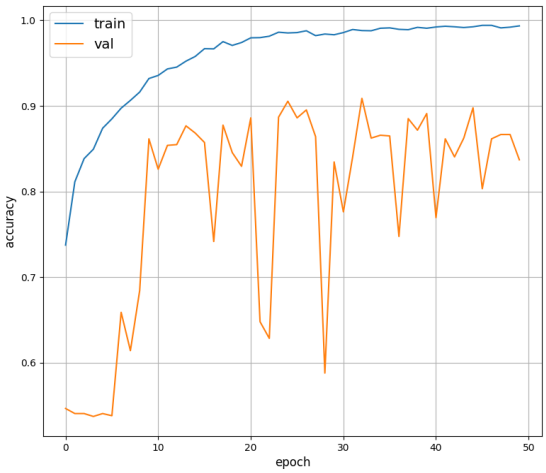
model4_2_last_b_normz



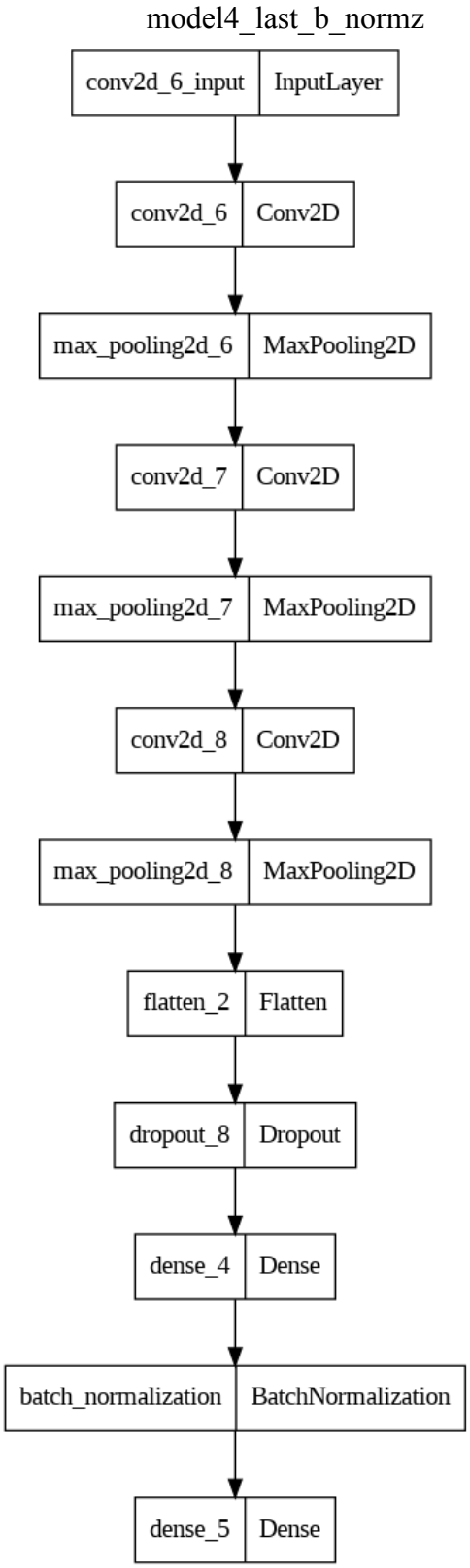


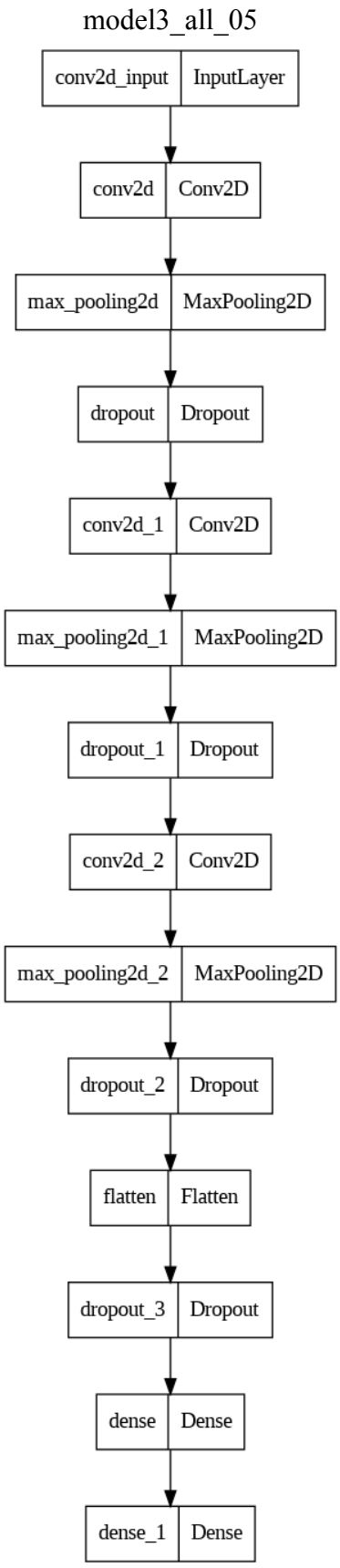
model4_3_last_b_normz





Appendix 2. The structure of the best models




model3_all_05
conv2d_input
InputLayer
conv2d
Conv2D
max_pooling2d
MaxPooling2D
dropout
Dropout
conv2d_1
Conv2D
max_pooling2d_1
MaxPooling2D
dropout_1
Dropout
conv2d_2
Conv2D
max_pooling2d_2
MaxPooling2D
dropout_2
Dropout
flatten
Flatten
dropout_3
Dropout
dense
Dense
dense_1
Dense

Appendix 3. The comparison table of all models

Archt. №	Model	Training loss	Training accuracy	Validation loss	Validation accuracy	Validation accuracy change
1	model1_32	0.690	0.541	0.690	0.541	-
1	model1_64	0.732	0.529	0.732	0.529	-2.2%
1	model1_128	0.666	0.627	0.666	0.627	18.5%
1	model1_256	0.723	0.629	0.723	0.629	0.4%
1	model1_512	0.774	0.616	0.774	0.616	-2.1%
2	model2_32	1.654	0.805	1.654	0.805	30.7%
2	model2_64	1.736	0.833	1.736	0.833	3.5%
2	model2_128	1.904	0.863	1.904	0.863	3.7%
2	model2_256	1.107	0.889	1.107	0.889	3.0%
3	model3_1_01	1.729	0.852	1.729	0.852	-4.2%
3	model3_1_05	1.510	0.867	1.510	0.867	1.8%
3	model3_grow_01	1.056	0.870	1.056	0.870	0.3%
3	*model3_all_05*	*0.401*	*0.902*	*0.401*	*0.902*	*3.7%*
3	model3_all_01	1.641	0.863	1.641	0.863	-4.3%
4	*model4_last_b_normz*	0.587	0.895	0.587	0.895	3.7%
4	model4_1_last_b_normz	0.523	0.889	0.523	0.889	-0.7%
4	model4_2_last_b_normz	0.584	0.840	0.584	0.840	-5.5%
4	model4_3_last_b_normz	0.972	0.837	0.972	0.837	-0.4%

Bibliography

1. Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, “An Introduction to Statistical Learning with Applications in R”, second edition, August 4, 2021.