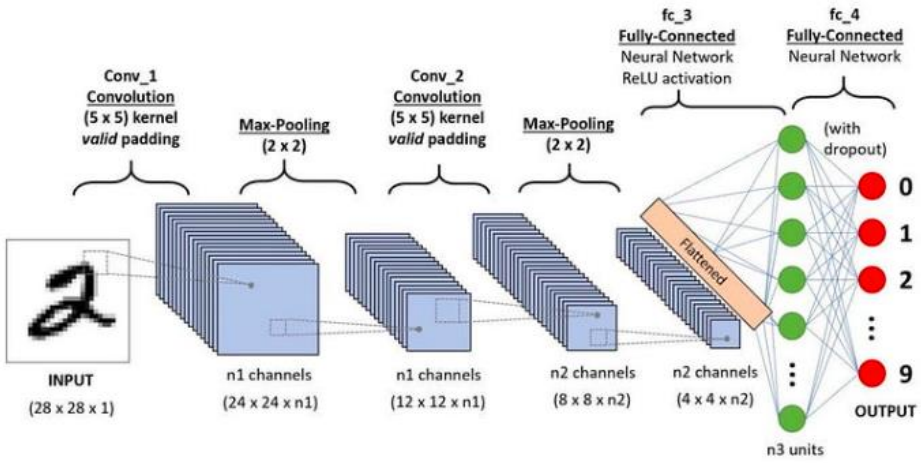
Daria Shcherbakova, 17487A
*

Experimental project: Neural Networks (binary classification of Muffins and Chihuahuas based on images)

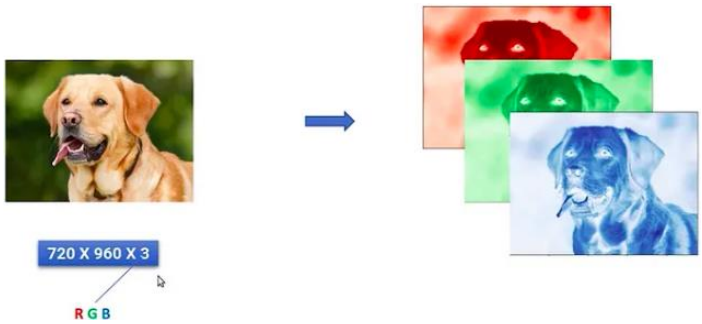
The project's main aim is to perform binary classification of muffins' and Chihuahuas' images. In order to achieve the goal of the work Convolutional Neural Networks (CNN) have to be applied due to the fact that CNNs are able to extract the features of an image, and convert them into lower dimensions without losing their characteristics. A deep learning framework Keras (from the Tensflow library) has been used.



pic 1. Architecture for a Convolutional Neural Network

Theoretical framework

I want to briefly describe how CNN for classification work from a methodological point of view. Before going to CNNs, it is important to note that some image transformations could be applied, for instance, RGB in output we get the 3 kernels channels with pixels corresponding to Red, Green or Blue colors. This step should be implemented in the data processing step.



pic 2. Example of RGB transformation

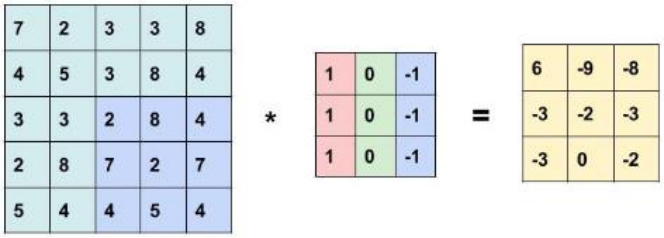
Basically, NN organizes their neurons into layers. To start with, image recognition (feature extraction) is used implementing neural network filters (kernels) on image to extract features from it (matrix with extracted features from the image). The dot product between the filter values and the image pixel values forms the *Convolution layer*. The implementation of a filter in CNN may lead to losing some patterns from input (like pixels on the edges/borders of the image). To cope with we can apply the paddling technique ("Zero" or "Same" paddling) to add zero-pixes on the edge. The problem is that this approach can train our NN for these patterns (understanding the edges of the image) and we can lose variability. Thus, we can claim paddling "Valid" without adding extra zero pixels around the input data (image), but adjusting the amount of padding to maintain the spatial dimensions of the input image in the output feature maps. Also, the NN should be "activated" by adding non-linearity to learn complex patterns and relationships in the data. There are some possible functions, but Rectified Linear Unit (ReLu) function is commonly used:

$$f(x)=max(0,x)$$

Or Sigmoid function (for classification):

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

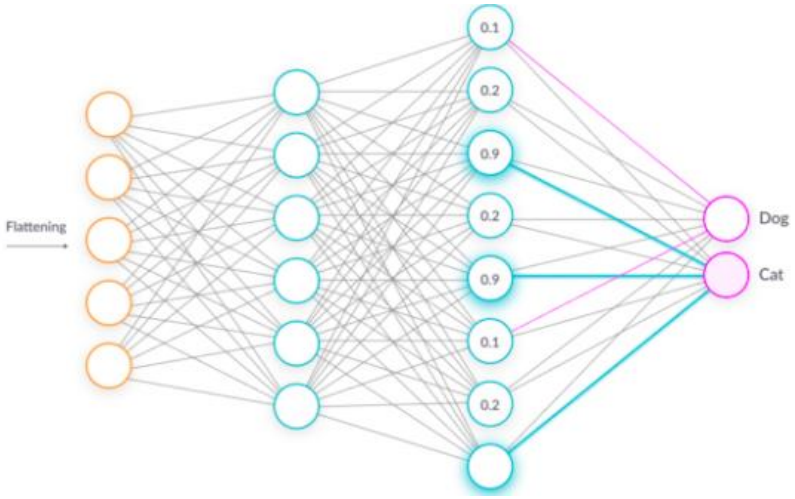
Then the image goes on the pooling layer in order to reduce the size of the filter layer optimizing the training time and escaping overfitting (max pooling or average pooling are commonly used). These layers can be transformed and applied several times, in other words, on the features extraction.



pic 3. Filtering work

After the feature extraction step, we go to the image classification part. At this stage, we are working with fully connected layers. In order to get dense layers, the linear units having a common set of inputs have to be collected together. Firstly, we have to flatten the input to receive the vector from our tensor's elements. After this, we can apply a fully connected layer to extracted the vector. The number of fully connected layers can vary, but at the last fully connected layer we should ask to assign probabilities on that particular "pre-output" number that exists on the input image by activation functions. We can also apply dropout (randomly

setting some of the output neurons to zero) to make the process of training faster and escape the overfitting of the model.



pic 4. A fully connected layer in a CNN

Since the CNN building process is finished, the model can be trained and optimized. At this step, we focus on the loss function to minimize it. In CNNs Cross-Entropy is used. To optimize the model, we need to find the best weights leading to the lowest loss, in CNNs Adam optimizer is commonly used (it is a modification of gradient descent). After finding optimal architecture of CNN, the hyper parameters can be tuned (layers' parameters, learning rate, number of epochs).

Results

To achieve the goal of the project we have set some objectives:

- Transform images from JPG to RGB pixel values and scale pixels;
- Build at least 3 network architectures and train hyper parameters;
- Perform 5-fold cross-validation to compute risk estimates.

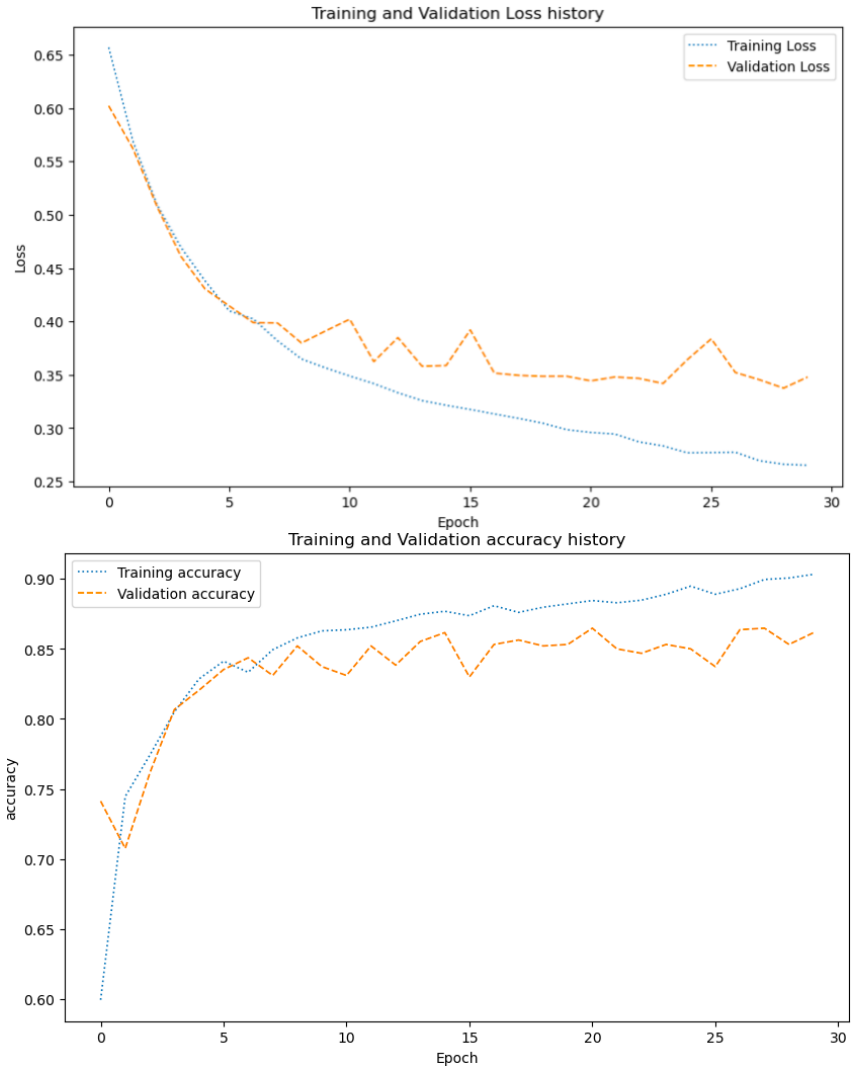
In order to find the best architecture among performed, the training set has been divided into training (80%) and validation (20%) samples to control models' performance, and under- and overfitting. Moreover, 5-folds cross-validation has been performed on 5 folds based on initial training set. The same approach has been implemented for hyper parameters tuning. After model's tuning and 5-folds cross-validation the "best model" has been fitted and evaluated on the whole initial training sample and predicted on test sample.

The first architecture

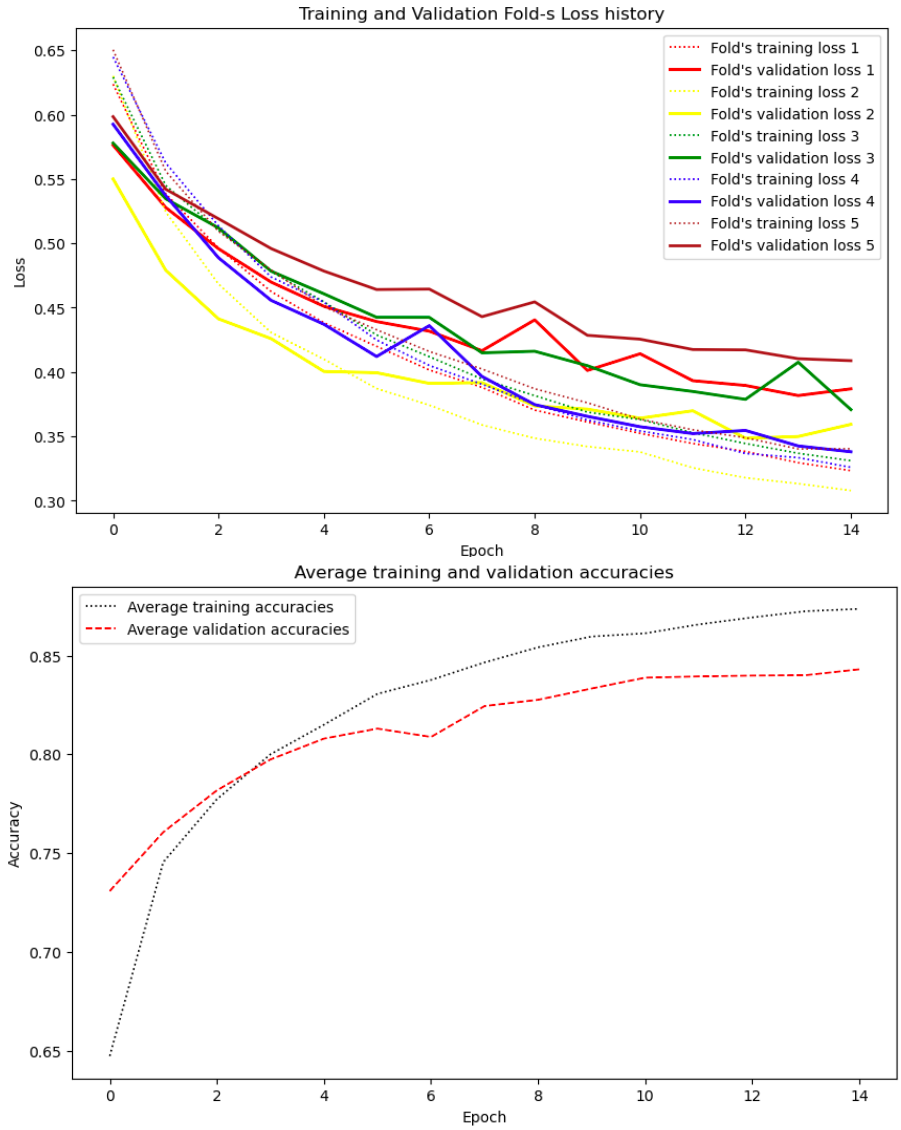
To start with, a simple CNN model has been realized, including feature extraction and classification steps. The first layer is Convolutional one with 32 filters with 3x3 kernels and with ReLU activation function for non-linearity. Then Max-pooling layer with a pool sized 2x2 is drawn to reduce dimensions without losing important information on features (getting the

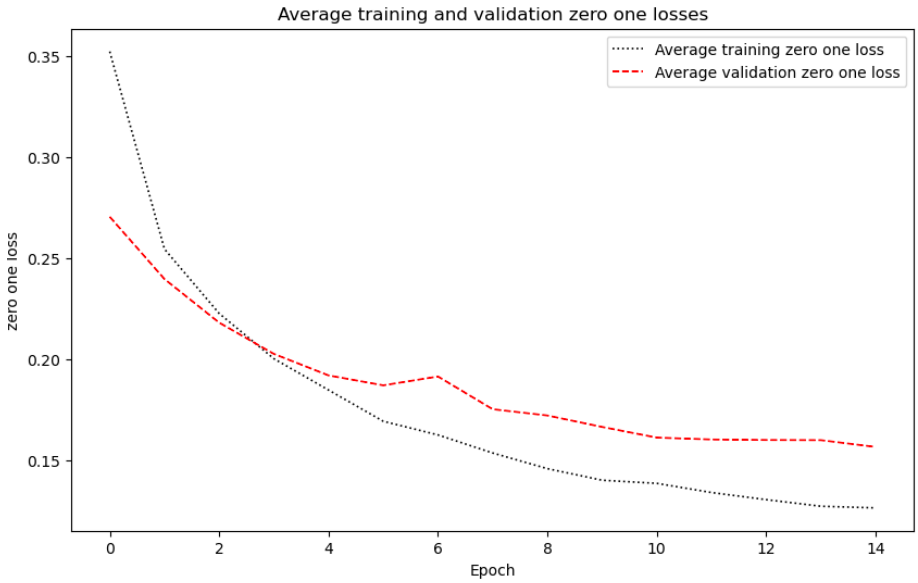
maximum from 2x2 dimension). For classification, Flatten layer is assigned to get a single vector from the feature extraction step and the Dense (fully connected layer) with the Sigmoid activation function getting 1 neuron from the vector and transforming into 0 or 1 (binary classification). Compilation of the model includes Adam optimizer, for loss function, we use binary cross-entropy due to binary classification. In other words, there is the simple CNN with one convolutional layer, one padding layer and one fully connected layer for binary image classification.

During the first trial of training in the Adam optimizer the learning rate has been set up by default and the training loss is equal to 0.1453 and accuracy is equal to 0.9599 which leads to “fast” learning and may be a cause of overfitting on test data. Thus, learning rate has been decrease to 0.0001, and the number of epochs for learning has been increased to 15. After this tuning, training loss is equal to 0.2585 and accuracy is equal to 0.9052, while validation loss is equal to 0.3479 and accuracy is equal to 0.8617. In this case, the model has learned training data well, but cannot performed the same way on “new unknown data”.



However, to check this, 5-folds cross-validation with zero-one loss has been performed. Zero-One loss by itself means 1 – accuracy, to calculate which the custom function has been created this way: we need to assess the average of absolute difference between true y labels and predicted y labels (which are binary). According to this step, Zero-One loss decreases with some fluctuation for each fold, but on validation data this model performs worse. Therefore, this model has learned training data well and showed good performance on training sample, but it cannot perform well on "new unknown" sample, causing possible overfitting on test data.

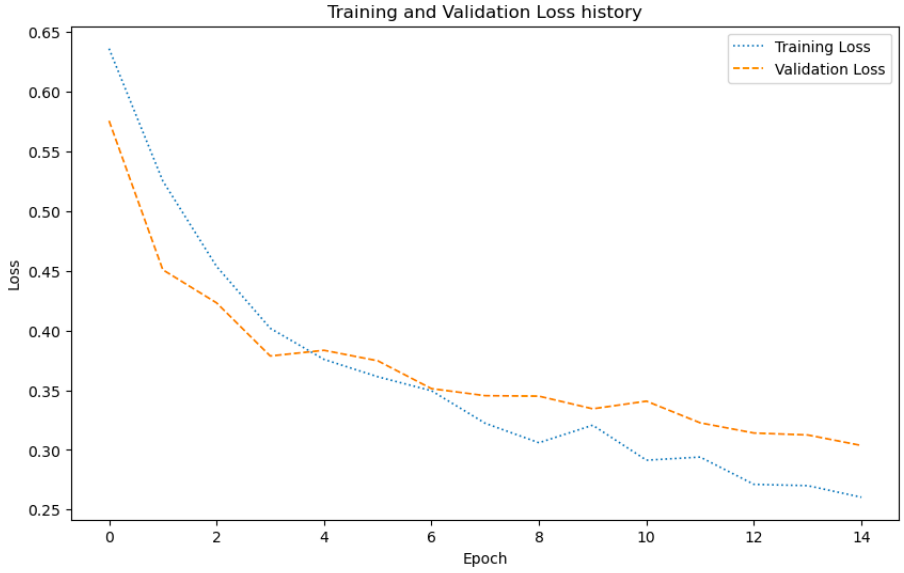


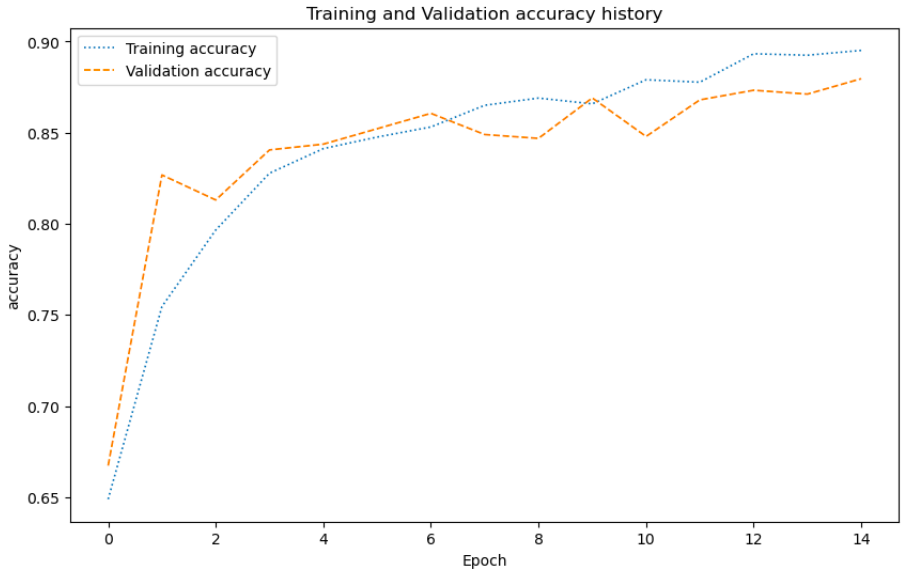


The second architecture

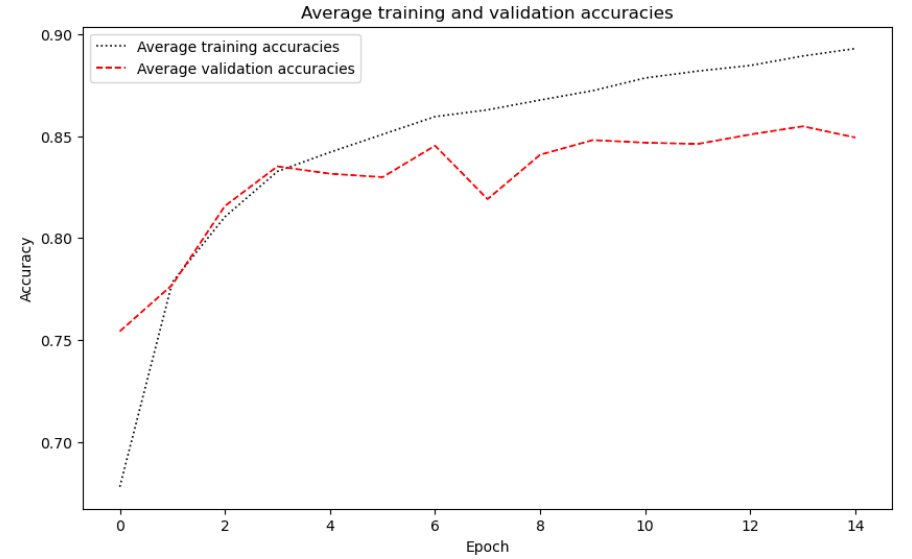
The change of architecture should upgrade the model itself and help to avoid possible overfitting. For this purpose, extra layers (Convolutional layer with fewer filters, another Max-pooling layer and the second Dense layer) are added in the first model. Once again, the model is compiled with Adam optimizer and binary cross-entropy loss function.

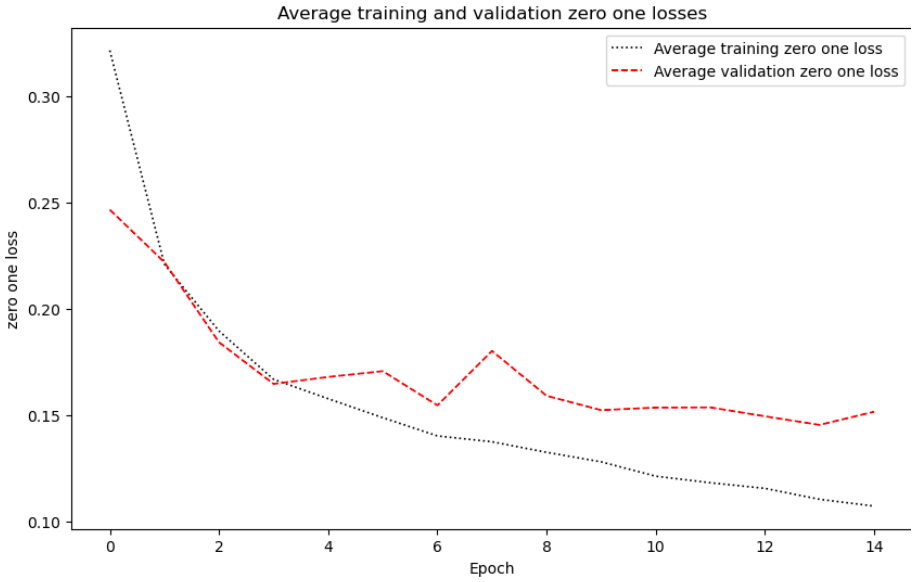
Assessing the upgraded model, training loss remains approximately the same (0.24) but validation loss slightly decreases (0.3037), while accuracies in both cases do not change (0.9062 and 0.8796 respectively). In other words, the updated architecture is still not the best option for the project's goal and may badly perform on test data.





Also, according to 5-folds cross-validation, there is still a significant gap between model's performance on training and validation samples. Due to the fact that there are some Zero-One losses' fluctuations form the 3^{d} to 15th epochs on validation sample. Moreover, on average there is significant difference between training and validation Zero-One loss. Also, on average training accuracy is close to perfect one, while average validation accuracy cannot catch up with that rate. Therefore, provided architecture is still could be improved to achieve the project's aim, since the model has learned training data and performs worse on validation data, which may lead to overfitting on test sample.

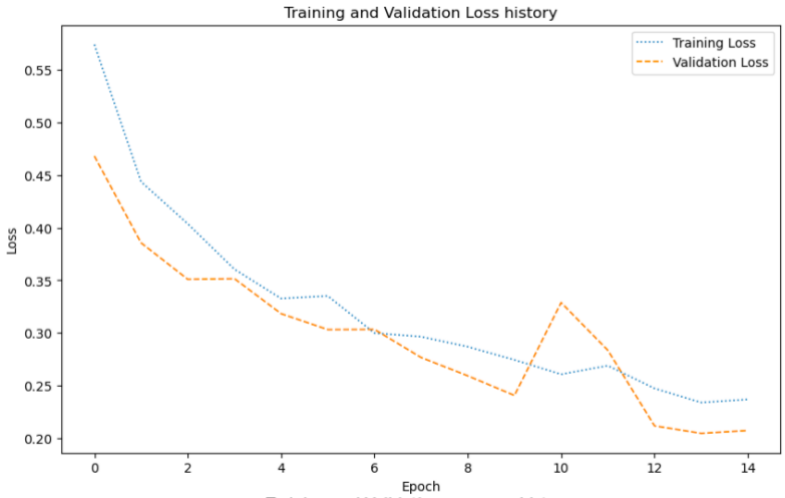




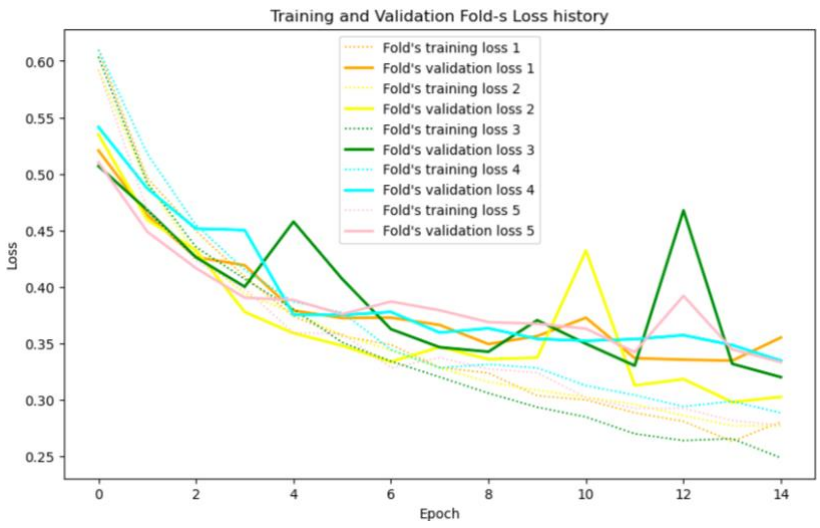
The third architecture

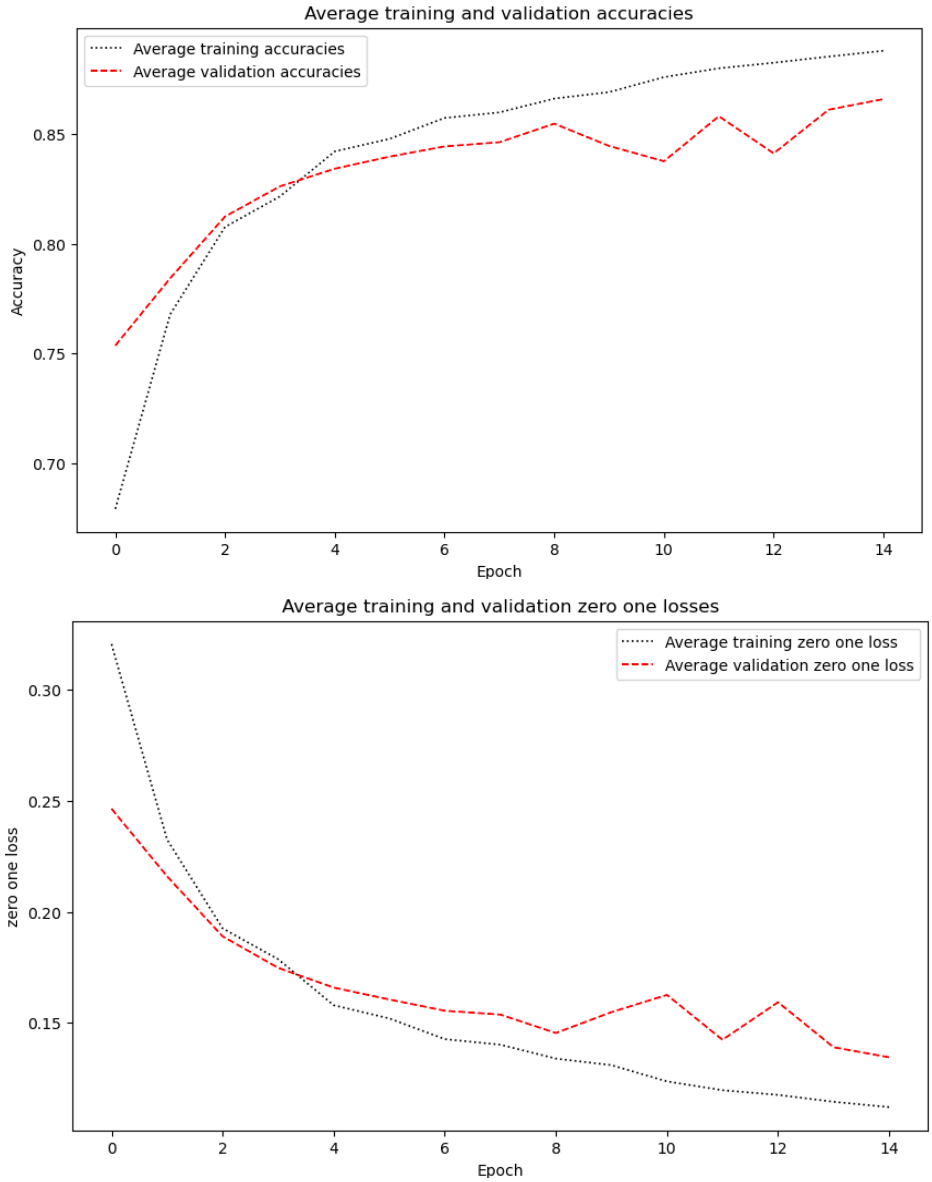
The next architecture is the same as from the pic.1. to cope with possibility of overfitting. It involves 2 Convolutional layers followed by 2 Max-pooling layers respectively, after Flatten layer a Dropout layer (with 50% probability of output neurons setting to zero) is incorporated to control overfitting, and, final Dense layer to assign probability of the input with Sigmoid activation function. The model is compiled and optimized as previous ones.

With this architecture, the training loss decreases compared to 0.2091, while accuracy increases to 0.9208 as well as the validation loss – 0.3021, while accuracy remains the same (0.8796). That is why, this architecture has been upgraded (tuned) with Early Stopping method to stop when the performance on training and validation samples get worse to save optimal model's weights. After this step, the significant improvement on validation sample has been achieved (validation loss decreases to 0.2073, accuracy on validation sample increases to 0.9219).



However, to be more precisely about model performance the 5-folds cross-validation has been implemented. Once again there are some fluctuations during learning on validation sample. Considering average Zero-One loss, there is the lowest one for validation sample among performed architectures. This model is able to perform well on “new unknown” data. Nevertheless, this architecture can still be improved. That is why, the last model has been chosen for tuning.



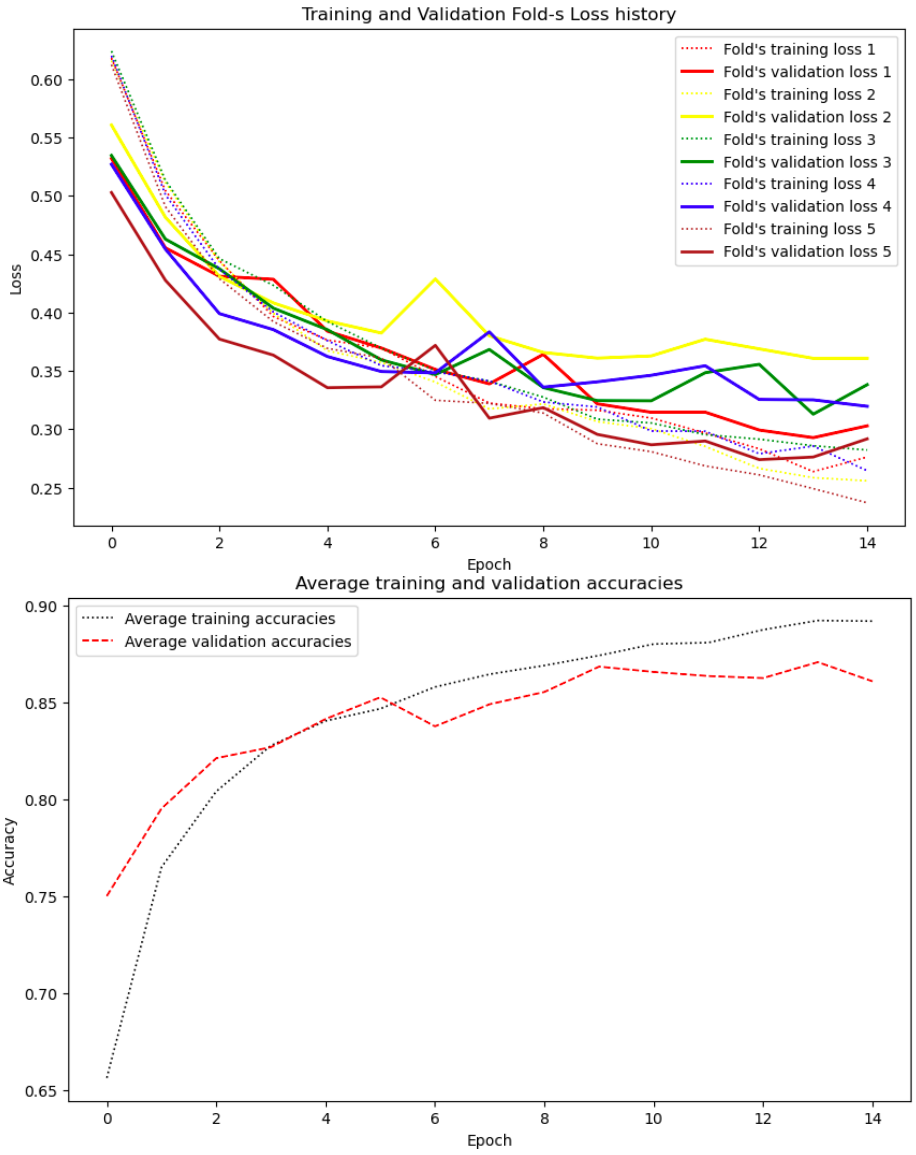


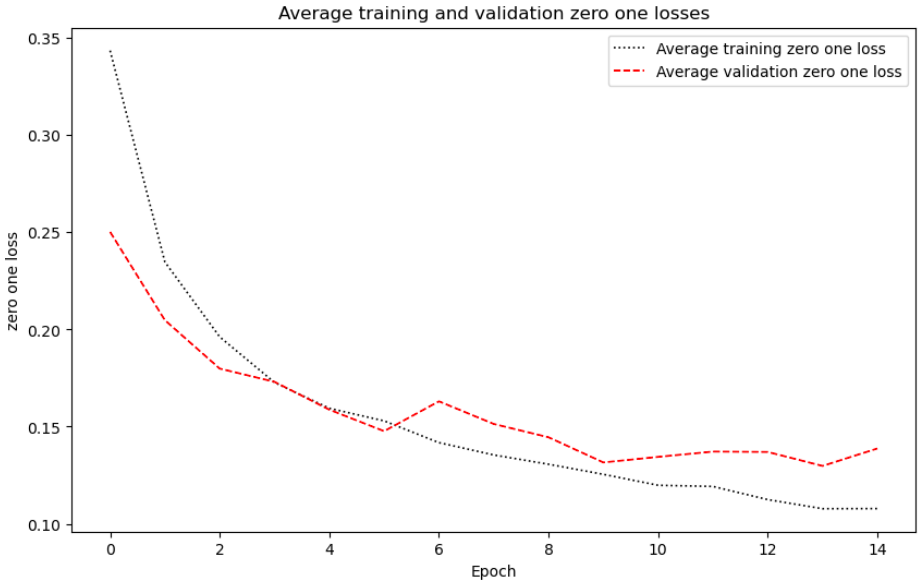
The 1st model's tuning

For the tuning of model's hyper parameters, the main focus is on layers' parameters. Two tuning has been performed in this project. The 1st one is applied with keras tunner, where we are looking for such layers' parameters as number of filters for Convolutional layer, Dense units, and Dropout rate. On the 14th trial the best hyper model has been found out such that the filter for the 1st Convolutional layer is equal to 128, the filter for the 2nd Convolutional layer is equal to 64, the number of neurons for the 1st Dense layer is equal to 128, and Dropout is equal to 0.5. Basically, the algorithm suggests using maximum values for hyper parameters. This tuned model has been fitted and evaluated on the whole initial training sample. Training metrics are equal to these: binary cross-entropy loss = 0.2274; accuracy = 0.9051; precision = 0.8671; recall = 0.9738; zero-one loss = 0.0949.

The 5-folds cross-validation

For this updated architecture, the 5-folds cross-validation has been performed. According to this step, there are some fluctuations during learning process. However, the model can perform well on “new unknown” data, on average showing small loss (few errors of misclassification) and big accuracy. Consequently, we can expect to see good performance on test data without overfitting, since CNN may adapt to new data.





Prediction of tuned model on test sample

Evaluating model on test data, good results has been achieved. Considering metrics of hyper model, we may state that this model predicts the class correctly for 87.33% of data. We have approximately 85% of data predicted as the 1 class which are true positive (correctly positive predictions) and 91% of 0 class, and it identifies 93% (for class 1) and 80% (for class 0) of the actual positive samples were correctly predicted, and these metrics are balanced well. However, 12.97% of data has been classified in a wrong way. Basically, with all performed optimizations we have avoided overfitting, since the model well performs on "unknown" test data.

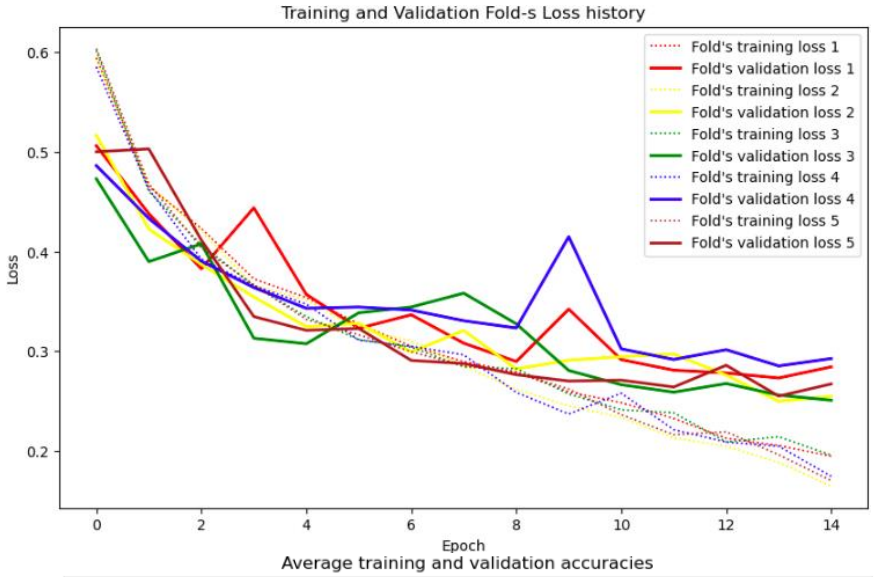
The 2nd model's tuning

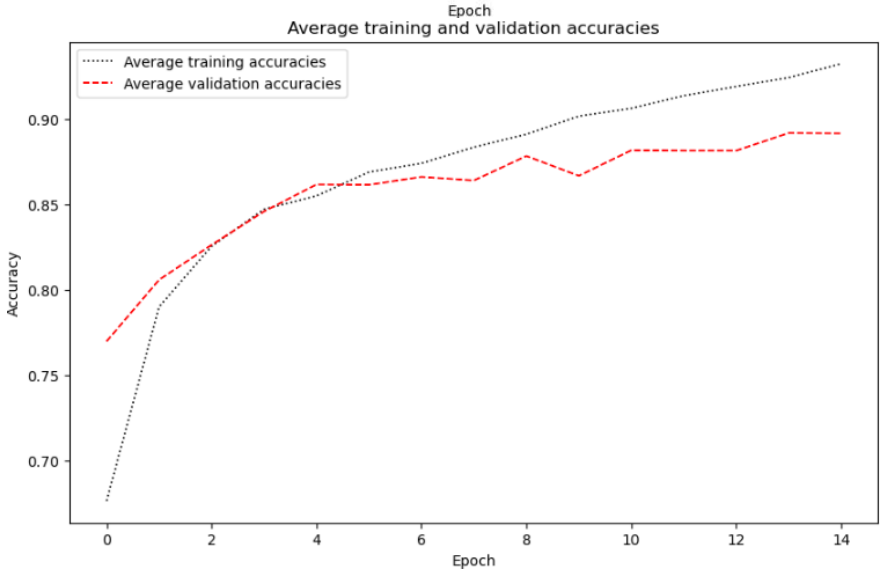
Despite good results on test data, other tuning has been applied with optuna (automated search for optimal hyper parameters). Basically, with this framework we can tune hyper parameters, such as the number of layers and the number of hidden nodes in each layer, activation function. For this step the training data has been divided on training subsample (75%) and validation subsample (25%) to control learning process. The best hyper model has been achieved on 19th trial with 112 filters and 3 kernel's sizes for the 1st Convolutional layer and ReLU activation function, with 128 filters and 5 kernel's sizes for the 2nd Convolutional layer and ReLU activation function, with 80 neurons for the 1st Dense layer and Dropout rate is equal to 0.4395.

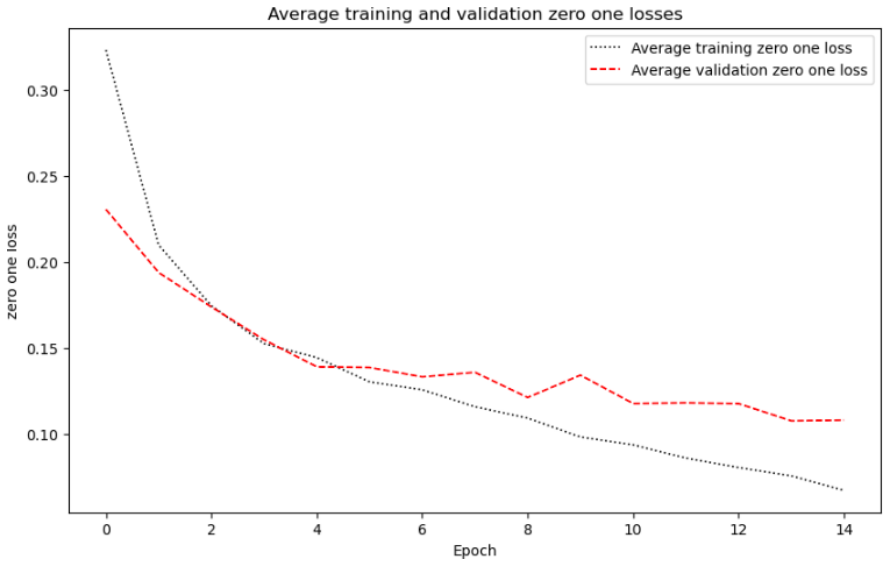
This tuned model has been fitted and evaluated on the whole initial training sample. Training metrics are equal to these: binary cross-entropy loss = 0.1498; accuracy = 0.9465; precision = 0.94; recall = 0.9624; zero-one loss = 0.0535.

The 5-folds cross-validation

For this tuned architecture, the 5-folds cross-validation has been performed. According to this step, there are also some fluctuations of Zero-One loss during learning, but, in general, it goes down despite a small growth on 3 last epochs. On average, Zero-One loss on validation data at the end of learning is less than 0.15. Also, on average, accuracy at the end of learning is higher than 85%. As a result, we can state that tuning of hyper parameters has helped in improving it by itself. In other words, the performance of the model could be assessed as good one, since CNN can adapt to "new unknown data" with some fluctuations. Therefore, this architecture can be predicted on test sample without expectations of overfitting.







Final prediction of tuned model on test sample

According to metrics report, we can conclude that model has been improved well. After this tuning, the model can correctly predict 92% of data, making just 8% of misclassification. Moreover, approximately 92% of data predicted as class 0 are true positives and 91% of the samples predicted as class 1 are true positives. Also, 90% of the actual positive samples (class 0) are correctly predicted and 93% of the actual positive samples (class 1) are correctly predicted, and these metrics are balanced well. In other words, our model has a good capability to distinguish and classify imagines of muffins and Chihuahuas. Basically, with all performed optimizations we have avoided overfitting, since the model well performs on "unknown" test data.

Discuss

According to considered results we may claim that CNNs on the used dataset should involves some approaches like regularization in order to control overfitting. The project has been started with the simplest CNN model complicating architecture on each step of creating. However, adding new layers on feature extraction and classification steps by itself could not help to prevent overfitting, while Early Stopping and learning rate regularization might be clues for the issue.

Nevertheless, hyper parameters of optimal CNN architecture is still supposed to be tuned for better performance on data and it is an essential step in CNN building to achieve the goal of the project. According to all steps, the model does not show overfitting. But still there is some options for improvement in architecture in order to increase accuracy of the model and reduce misclassification. Moreover, the CNN can be improved from time demanding point of view to make learning process more efficient due to resources costs.

References

Géron, A. (2022). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow. " O'Reilly Media, Inc.".

Nikolenko S., Arkhangelskaya E., Kadurin A. (2018) Deep Learning. Immersion in the world of neural networks. - SPb: Peter, 2018. - 480 c.- ISBN 978-5-496-02536-2. (in Russian).

Binary Classification. Kaggle (learn).URL: https://www.kaggle.com/code/ryanholbrook/binary-classification

Ferdinand, N. (2020). A Simple Guide to Convolutional Neural Networks: Convolutional Neural Networks demystified without any of the fancy tech jargon! Towards Data Science. URL: https://towardsdatascience.com/a-simple-guide-to-convolutional-neural-networks-751789e7bd88

Rajpal G. (2020). A Comprehensive Guide to Convolution Neural Network. Medium. URL: https://medium.com/swlh/a-comprehensive-guide-to-convolution-neural-network-86f931e55679