Data Science and Economics
Universit‘a degli Studi di Milano

Neural Networks

(Convolutional Neural Networks)

Student: Vojimir Ranitovic 963780
Email: vojimir.ranitovic@studenti.unimi.it
Module: Statistical Methods for Machine Learning

Abstract

Convolutional Neural Networks (CNN) are usually used for image classification. CNN is a regular Neural Network (NN) with added layers before it. Those layers are introduced to reduce the number of parameters of regular NN and to reduce overfitting. The goal is to classify pictures of cats and dogs and in this project, we will see different models, starting from the simplest one and moving to the more complex models, with aim to reach high accuracy and low loss. After choosing the best model according to the accuracy and log loss, the model's hyperparameters are tuned, and then the model is validated using 5-Fold cross-validation. The final model is evaluated on the testing set, and also on 20 self-made pictures of cats and dogs.

Introduction

Neural networks are algorithms (a family of algorithms) created to imitate the human brain. They are based on interconnected neurons depending on the type of network, and there are many types of them, but generally, can be divided into three classes: Fully connected neural networks, Convolutional neural networks, and Recurrent neural networks. In this project, we will use the first two types. Our goal is to do binary classification with high accuracy, and as we will see it is not enough to use just fully connected neural networks. It needs extensions such as convolution layers to reach more satisfying results with image classification. Convolutional neural networks (CNN) are subclass of neural networks that have at least one convolution layer. They are specifically made to process pixel data and are used in processing and image recognition. We use them to obtain local information, for instance, from a neighbor pixel in an image, and also to reduce the complexity of the model in terms of the number of parameters. A fully connected layer treats all features as independent from one another. A convolutional layer treats features nearby in space as connected by clustering them into smaller groups. In essence, they are trying to introduce information (spatial positioning) into the layer that would not be present otherwise.

Dataset, preprocessing and environment

Dataset[1]that will be used contains 25,000 images, separated into two folders, one for cats and another for dogs images, perfectly balanced and containing 12,500 images each. Most of the images are in .jpg format, but some of them are corrupted or in a different format, and those images are deleted (1590 of them). After this, whole remaining dataset is divided into training, validation, and test set. For training, 70% of the images have been taken (8214 cats and 8169 dogs), for validation 20% (2506 cats and 2467 dogs) and

[1]Dataset can be downloaded at: https://unimibox.unimi.it/index.php/s/eNGYGSYmqynNMqF

the testing set is about 10% (1009 cats and 1039 dogs). The test set was used only at the end of the project, after finding the best model. As we can see, classes are mostly balanced, so we can rely on accuracy as the measure of this binary classification task. Regarding preprocessing part of images, before entering the model, images are resized to 150x150 resolution and pixels are divided by 255 so after that pixel range is between 0 and 1. Also, the image's colors were left so the images as inputs are (150x150,3), where 3 is the number of channels (red, green, and blue). Another additional preprocessing part is data augmentation in sense of rotating and zooming. I have not seen all pictures in dataset, but I assumed that some owners love to take pictures of their pets in very similar positions (and even background). So, our random split could get data leakage of training data into validation and test sets.

Data augmentation

Data augmentation is a technique to increase the diversity of the training set by applying some realistic transformation, such as rotation, zoom, different brightness, crop, or contrast. After experimenting a little bit I chose to randomly rotate and zoom some of the training pictures. Random brightness and contrast would also be helpful, but in my experience, it can considerably slow down the training process. My training pictures can be randomly rotated 20% to the left or right and 30% zoomed in or out. This is slowing down the training phase but not as much as mentioned changes in brightness and contrast. All this augmentation will be helpful to model to learn on more diverse data, and also to prevent possible overfitting and make better accuracy and lower loss on the test set. These preprocessing techniques will be inactive in the evaluation phase. Thus, augmentation will only take place while fitting the model. Below is one example of possible random zooms and rotates.

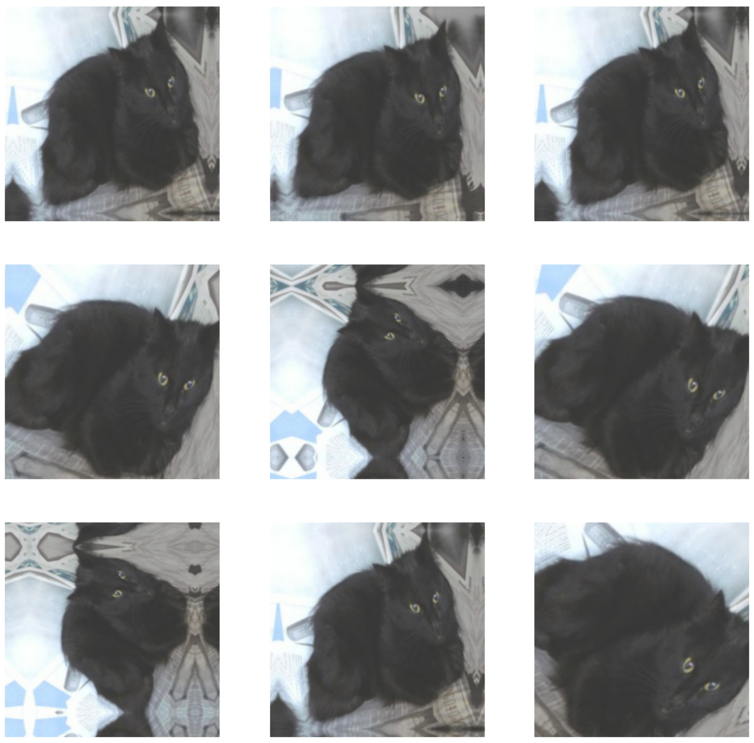


Figure 1: A possible example of zoom and rotation randomly applied.

Google Colab environment

Training neural network models are usually a slow process, especially CNN and working with images or videos (graphic processing). Since this is our case, I wanted to make code efficient but also to use most of the hardware. CNN models can be trained faster by running all operations at the same time instead of one after the other. It can be achieved by using a GPU (Graphics Processing Unit) to train our model. A GPU is a specialized processor with dedicated memory used for extensive graphical and mathematical computations which frees up CPU cycles for other jobs. My laptop does not have a dedicated GPU, so I used Google Colab, a cloud service. Google provides us with some free GPU processing power (GPU models can be changed with every new session[2]), and 12GB of RAM. With this setup, my model could be trained quicker (17s per epoch), with a cached train and validation set. Together with GPU, caching ability helps with processing speed. It was able to store training and validation sets of 150x150 resolution images, anything above that resolution will not be storable in 12GB RAM.

Building blocks of CNN

Convolution Neural Network is a sequence of layers, and every layer transforms one volume of activations to another through a differentiable function. Three main types of

[2]This change can lead to different results every time we run model, but setting tensorflow seed and all local seeds should make this difference negligible.

layers will be used in the building architecture of my model. Those layers are the Convolutional Layer, Pooling Layer, and Fully Connected Layer (regular Neural Network). These layers are used together to form a Convolution Neural Network architecture. One CNN architecture can have more than one convolutional and pooling layers. There exist numerous different well-known architectures like LeNet, AlexNet, VGG-16, and VGG-19. At first, I was trying to replicate some of these architectures, but it was too complex and slow for our task and resources, so I decided to make a few models on my own, from the simplest to the more complex one.
Convolutional Layer is the first one after importing an image of size (height, width, n), where n is the number of channels (if it is RGB then it is equal to 3, or equal to 1 if it is a grayscale image). Each convolutional layer has one or more **filters** (or kernels). Filters are matrices that are usually 3x3, 5x5, or 7x7 in size. There could also be different sizes, but these one works well and are mostly used. The filter goes through the matrix, calculates Dot Product, and fills a new matrix that is called **Feature Map**. On the Feature Map matrix is then applied **activation function**. There are many activation functions, but the most used are sigmoid, tanh, ReLU (Rectified Linear Unit), Leaky ReLU, etc. In our models, only sigmoid and ReLU will be used. The sigmoid function will be used only for the output layer (it gives us the final probability), and ReLU is used for convolutional layers and all hidden layers in connected layers. Below we can see graphs of these functions.

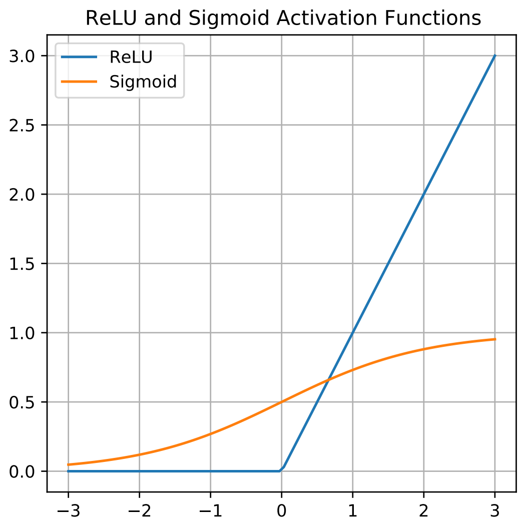


Figure 2: Two activation functions that will be used in models.

After applying the ReLU function on Feature Matrix there will be left only zeros and positive numbers. **Pooling layer** is then applied to this matrix. Pooling layer is the new matrix that takes the maximum[3] or average values of the previous Feature Matrix

[3]All coming models will use only pooling layers that take maximum values.

(similar to kernels). The pooling matrix is usually 2x2 or 3x3 in size but could be also a higher dimension. After this, we can apply another convolutional layer and do everything from the start, but now we are doing it on the output of the previous pooling layer. After finishing with these layers, we approach an ordinary neural network with or without hidden layers. Below is the simplification of one possible architecture and different layers.

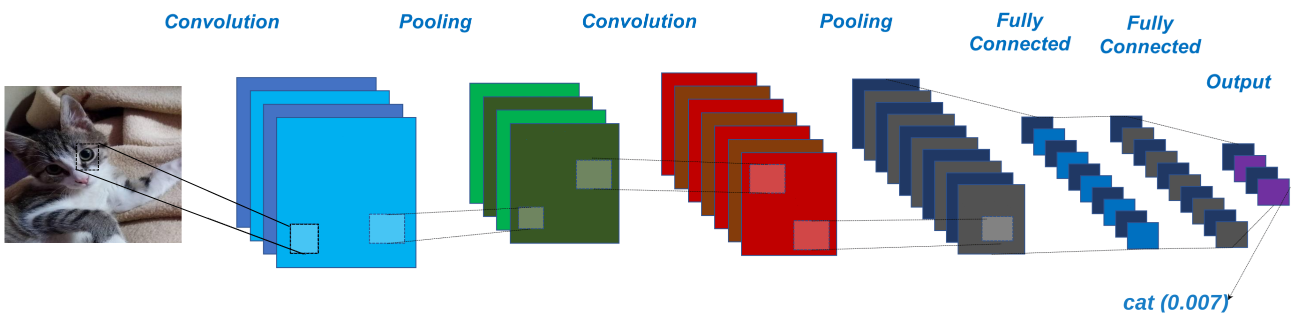


Figure 3: CNN road from the input image to class prediction.

Creating the models

In this part, the goal is to seek the most suitable model for our binary classification. At the start, the dataset was split into training, validation, and test. A training set will be used for fitting the model and validation one will be to evaluate it in each epoch. For each model, 20 epochs were used, and the batch size of 128. Pictures are imported into the model in (150,150,3) dimensions, where 150x150 is the size of the picture. Adam is used as an optimizer and the learning rate is set to 0.0006. Also, early stopping build-in function is used to prevent overfitting and shorten training time, so training would stop if there is no improvement in validation loss in 3 consecutive epochs. All these parameters were fixed, only the complexity of the model's architecture was changed. Accuracy is used as metric and binary cross-entropy as loss function (also named log loss). Belowis shown the equation of the log loss function.

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

In the dataset, a cat is represented as 0 and a dog as 1. The sigmoid function from the last layer in every model will predict with probability from 0 to 1. If the prediction probability is lower than 0.5, the predicted class is a cat, otherwise, it is a dog. So, if the true label is 0, and if a class is predicted with a probability higher than 0, log loss will increase. The maximum increase in loss is when the true label is 0 and we predicted class with probability 1 (completely wrong prediction with the highest confidence), or vice versa. In the following parts, I will try four models with the settings mentioned above.

Only regular Neural Networks (model-0)

This model consists of the layer that flattens the image and 8 dense layers whereas the last one is the output layer with only one neuron. The number of neurons in each layer is different and is [64,128,256,512,256,128,64,1], it is mirrored neural network. Each dense layer uses the ReLU activation function except the last one which uses sigmoid one.

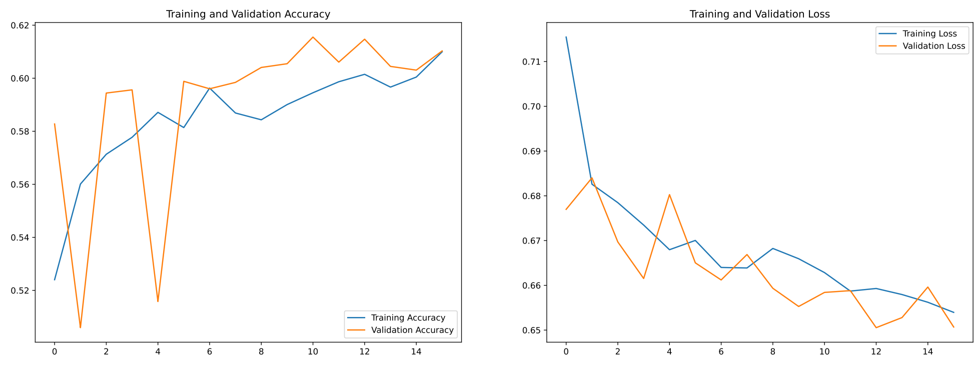


Figure 4: Accuracy and loss functions - Only Neural Network

Train	Accuracy: 0.6174	Loss: 0.6432
Validation	Accuracy: 0.6102	Loss: 0.6506

Table 1: Accuracy and loss of model-0.

One Convolutional layer + NN (model-1)

In this model, one convolutional layer is added, with 32 filters of 3x3 size. Pooling layer with the matrix size of 2x2 is added after. Also, in the neural network part only flatten and two dense layers were used, the first layer has 256 neurons and the output still has one neuron. Important thing is that from now on, in the neural network part **dropout**[4] layer will be used. The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting.

[4]Dropout and data augmentantion are the reasons of this weird graphs where validation loss is lower than training one.

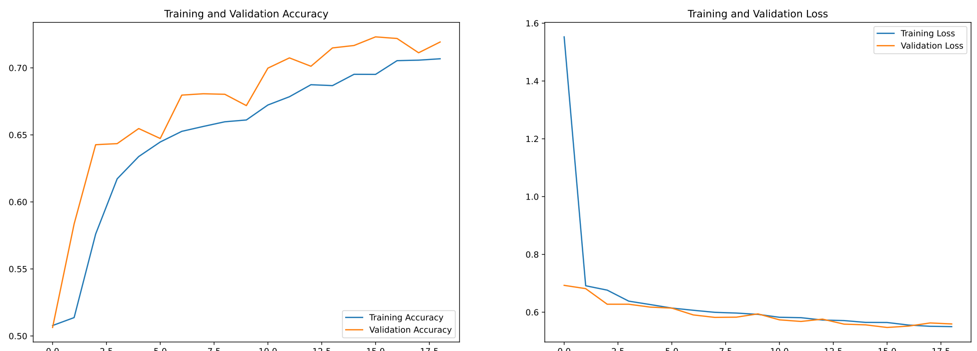


Figure 5: Accuracy and loss functions - One Conv. layer + NN

Train	Accuracy: 0.7320	Loss: 0.5408
Validation	Accuracy: 0.7192	Loss: 0.5593

Table 2: Accuracy and loss of model-1.

As we can see, this model is better than using only NN, but still, we have to improve it by adding more layers.

Two Convolutional layers + NN (model-2)

Here, we added a new convolutional layer with 64 filters of 3x3 size, and also an additional dense layer in the NN part with 256 neurons and a dropout set to 10%. After each convolutional layer, one pooling layer with 2x2 size matrix is added.

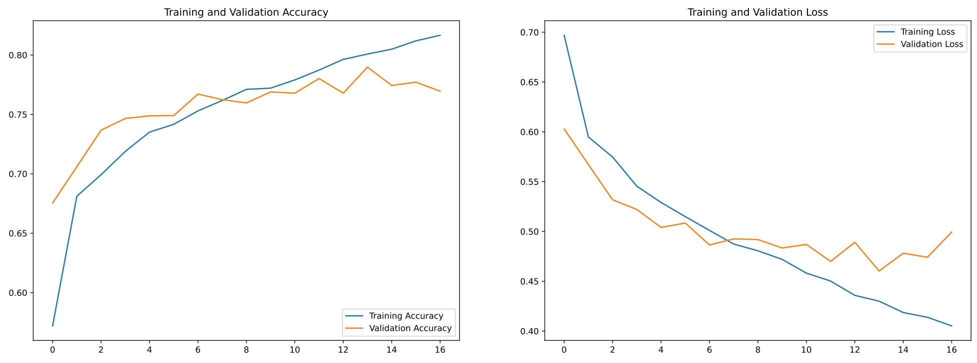


Figure 6: Accuracy and loss functions - Two Conv. layers + NN

Train	Accuracy: 0.8166	Loss: 0.4052
Validation	Accuracy: 0.7696	Loss: 0.4995

Table 3: Accuracy and Loss of model-2.

Model 2 is an improvement, but it starts to overfit after the 6th epoch, so we have to take care of it with adding more convolutional layers and/or increasing dropout.

Five Convolutional layers + NN (model-3)

In this model we have 5 convolutional layers, where they have 32,64,64,128,128 filters respectively, with the same size of 3x3 each. Pooling layer after each of convolutonal layers is added. Also, there is one more dense layer added in NN part, together with dropout layer of 10%, so now we have 3 dense layers of 256 neurons each.

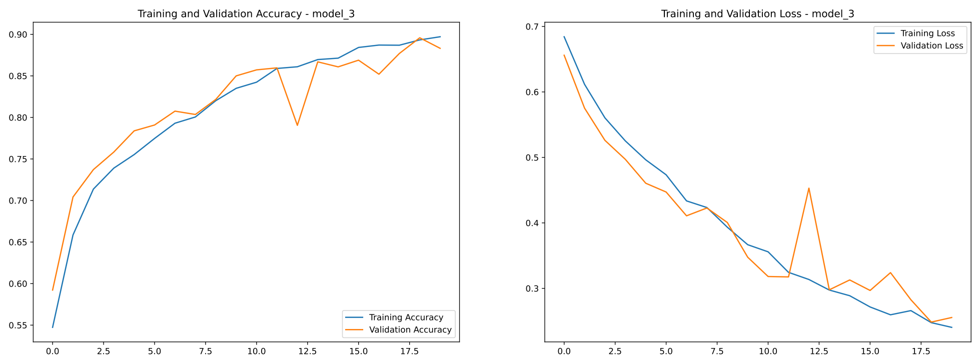


Figure 7: Accuracy and loss functions - Five Conv. layers + NN

Train	Accuracy: 0.9081	Loss: 0.2125
Validation	Accuracy: 0.8831	Loss: 0.2554

Table 4: Accuracy and loss of model-3.

Model 3 is the best one, so we will use it for the next steps such as hyperparameter tuning and cross-validation, and after all, it will be evaluated on the test set (set that we have not used from the start).

Hyperparameters tuning and cross-validation

Many parameters can be tuned in CNN, such as batch size, number of epochs, different optimizers, learning rate, number of nodes in the NN part and number of kernels in convolutional part, size of kernels, size of pooling filters, number of convolutional layers and so on. The default value of batch size is 32, and after experimenting with it, I realize that the best size would be 128 (that can be cached in RAM with an image size of 150x150). After changing the batch size from default, the learning rate has to be tuned, and there is a rule of thumb that higher batch sizes need higher learning rates. Starting from the Adam optimizer and learning rate of 0.0001, I gradually increased it where it is found that values from 0.0006 up to 0.001 work well. The model-3 architecture was not changed, and the next parameter that I wanted to tune is the number of neurons in the NN part, and also the dropout layers percentage parameter. In the end, the parameters to tune were the number of neurons in the NN part [128, 256, 512] neurons respectively, dropout rate [10, 20, 30] percent, and learning rate [0.0006, 0.0008, 0.001]. So, there were 27 models to train and evaluate. A train set was used for fitting each model, and a validation set was used to evaluate it. The number of the epoch was 20, with early stopping regularization after 3 epochs if there was no improvement in validation loss. After evaluating each of them, the model with 128 neurons in the NN part, a dropout of 10%, and a learning rate of 0.0008 was the best model with the lowest validation loss.

Model-3 after tuning	Accuracy: 0.9002	Loss: 0.2285

Table 5: Accuracy and loss of model-3 after tuning.

After hyperparameters are set, we have to investigate if our model is truly well performing on the validation set, or if it is just by random chance of splitting. To understand this, cross-validation is used, specifically 5-fold cross-validation[5]. Firstly, the train and validation set were merged and then split into 5 equal folds. In each of the 5 iterations, one different fold was used for evaluation and the rest four folds were used for training. Below we can see Log loss, Zero-One loss, and Accuracy in each of 5 folds, average and standard deviations.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	**Average**	**St.dev**
0-1 loss	0.1342	0.1145	0.1588	0.1288	0.1096	0.1292	0.0193
Log-loss	0.3578	0.2686	0.3521	0.2981	0.2613	0.3076	0.0454
Accuracy	0.8657	0.8854	0.8411	0.8711	0.8903	0.8707	0.0193

Table 6: Accuracy, log-loss and zero-one loss for each of 5 folds.

As we can notice, there are no big differences across the folds, so we are more confident that our model will have a lower loss and high accuracy in the testing set.

[5]The best way would be to use nested cross-validation but that would increase the number of trained models to 135.

Final model and results

After hyperparameters tuning and cross-validation, model-3 is trained again, this time until the 18th epoch, since after that epoch model starts to overfit (this epoch is found with help of an early stopping function). After this, the testing set will be used for the first time and evaluated accuracy and loss will be displayed together with validation and training one.

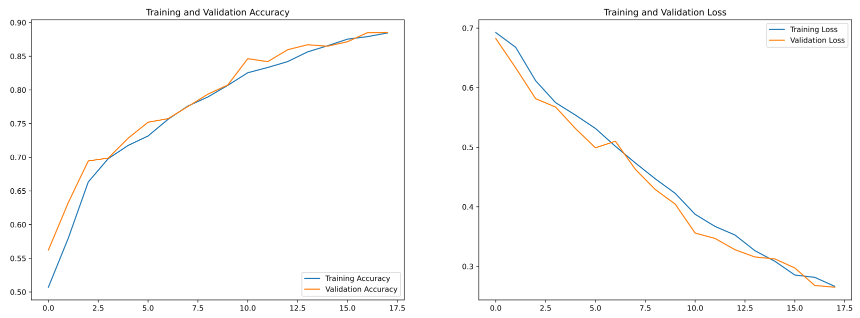


Figure 8: Accuracy and loss functions - model-3 and image size:150x150

Train	Accuracy: 0.9047	Loss: 0.2265
Validation	Accuracy: 0.8851	Loss: 0.2650
Test	Accuracy: 0.8784	Loss: 0.2857

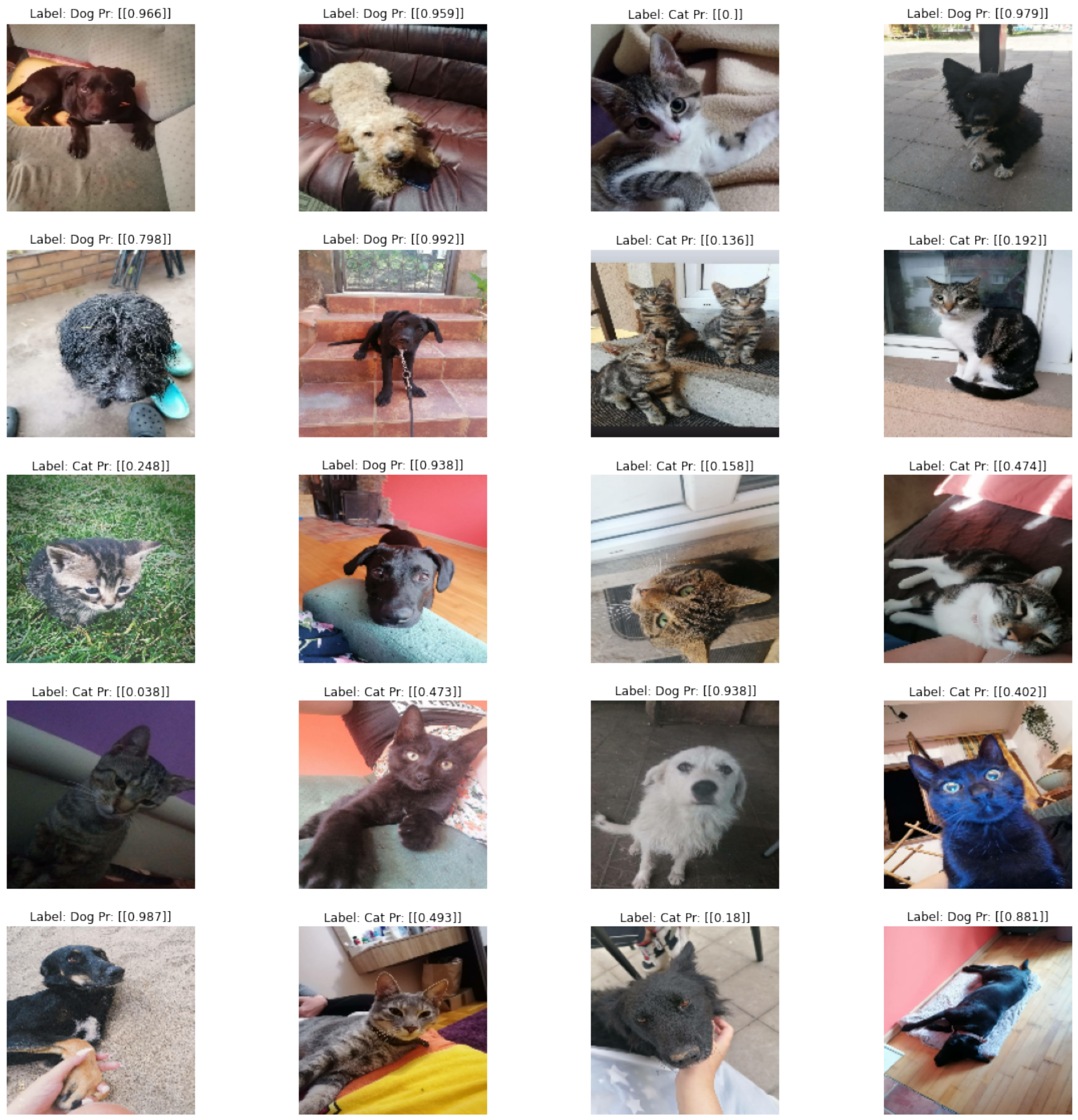
Table 7: Accuracy and loss of model-3.

Accuracy and loss on testing set are not the best like what can be achieved with some pre-trained models like VGG-19 for example, but it is still quite good. To be sure that my model is adequate, I tested it on the pictures that are not in the dataset at all. I took 20 personal pictures of cats and dogs, 10 pictures of cats, and 10 of dogs. Below is the result.

Test on my pictures	Accuracy: 0.9499	Loss: 0.2821

Table 8: Accuracy and loss of model-3 after testing it on personal images.

Results are quite good, accuracy is high, but that is probably because the set is small, just 20 pictures. Below we can see each picture and their predictions.



As it is shown, the model made only one mistake. It is a dog at position (5,3). The model predicted that it is a cat with a probability of 0.18, which is quite confident. Mistakes like this can increase log loss a lot. Also, the model predicted the remaining pictures correctly but it is not so confident with some of them, like the cat in position (5,2), the probability is 0.493 which is near to 0.5[6]. The perfect case with no log loss increase is the cat at position (1,3), where the prediction is 0, and the model is correctly confident that that is a cat.

We can also try to increase accuracy without changing anything within the model. To do that we can train everything from the start with images of higher resolution, for example

[6]If prediction is from 0.5 to 1, prediction is a dog

256x256.

Test 256x256	Accuracy: 0.9130	Loss: 0.2186
Test 150x150	Accuracy: 0.8784	Loss: 0.2857

Table 9: Accuracy and loss of model-3 and image size:256x256 vs 150x150

Increased resolution raised accuracy by 0.039 and reduced the loss by 0.067 evaluated on the test set. Indeed, this is improvement, but train and validation set could not be stored in RAM anymore, so time per each epoch increased 3.5 times (also increased resolution contributes to this higher training time per epoch).

Conclusion

In this project, we have seen that simple CNN model can obtain good results. The difficulty with convolutional neural networks is that training could be very slow, so before starting we have to choose which parameters we want to tune, it would be nearly impossible to tune it all. The increased resolution also adds more complexity and increases training time, so it would be better to make a more complex model at first and try to increase testing accuracy, then to increase resolution and use a simpler model. In the future, I would try to increase model complexity, but also add more options in the data augmentation layer, like random brightness or random contrast. That setup will require more computational power but also could possibly increase accuracy and reduce loss, all avoiding overfitting.

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.