FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

Report
on the practical task No. 5
“Transfer learning”

Performed by
Chernobrovkin T.V. (412642)
Academic group J4133c
Accepted by
Gladilin P.E.

St. Petersburg
2023

Goal

Learn and practice the transfer learning method.

Problem

Data:

https://drive.google.com/drive/folders/1nzVk4GOvKR6P87uPszUkKMPtaXV_wrZf?usp=sharing

Fill all the necessary gaps in **CNN_and_Transfer_Learning.ipynb** and fit neural networks for solving the binary classification task.

Part 1:

1. Build and fit CNN with 3 convolutional layers for binary classification
2. Evaluate accuracy on test data
3. Plot the graphs for Loss(number_of_epochs) and Accuracy(number_of_epochs)

Part 2

1. Build and fit Transfer Learning model using pre-trained **VGG16-model** weights from keras application.
2. Do the same with **one more** avaliable pre-trained deep learning model from keras application, for example Xception - https://keras.io/api/applications/.
2. Evaluate accuracy on test data for p.1 and p.2
3. Plot the graphs for Loss(number_of_epochs) and Accuracy(number_of_epochs)
4. Check the performance of your model with **the custom image of cat or dog** (so the model will tell which class this image belongs to). Develop the function for the inference of the best algorithm.

*****Part 3 (not necessary)**

Train the DL model with **the highest possible accuracy <u>on test data</u>.** You may experiment with any architecture and / or ensemble of models on your own choice (not limited to keras.io/api/applications).

The student, whose model will have the highest score, will get "5A" mark for the MLT Exam automatically.

Theory

Transfer learning is the application of knowledge extracted by a neural network when solving another problem to solving a problem.

Deep neural networks require large amounts of data for training to converge. Therefore, a situation often occurs when for the problem being solved there is not enough data to properly train all layers of the neural network. Transfer learning is used to solve this problem.

Most often, transfer learning looks like this: to a neural network trained for a specific task, several more hidden layers are added, which allow you to use the knowledge already acquired to solve a more specific problem. For example, knowledge gained from learning to recognize various objects can be applied to solving the task of recognizing food.

Materials and methods

In this task, all calculations were performed on the student's personal laptop in Google Colab. The work was performed in the Python programming language.

Results

Part 1

We created and installed a CNN with three convolutional layers for binary classification. The architecture of the resulting network is shown in Figure 1.

```
model = Sequential(
    [
        keras.Input(shape=input_shape),
        Conv2D(16, kernel_size=(3, 3), activation="relu"),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(32, kernel_size=(3, 3), activation="relu"),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(64, kernel_size=(3, 3), activation="relu"),
        MaxPooling2D(pool_size=(2, 2)),
        Flatten(),
        Dense(64, activation="relu"),
        Dropout(0.5),
        Dense(1, activation="sigmoid"),
    ]
)
```

Figure 1 - Architecture of the created neural network

The next task was to train the network with 10, 15 and 20 epochs and compare the obtained accuracies. Looking ahead, the task included a remark that if the accuracy of the test data after 15 epochs is less than 80%, something is wrong. To speed up training, the input data size was reduced

to 32×32 pixels. At 15 epochs we had an accuracy below 80%, so we had to retrain the model on images of higher measurement and wait patiently.

At 10 epochs of training, we obtained an accuracy of 78.87% (Figure 2). The graph of accuracy and loss vs. number of epochs is shown in Figure 3 and Figure 4, respectively.

```
  scores = model.evaluate_generator(test_generator, nb_test_samples // batch_size)
Accuracy on test data: 78.87%
```

Figure 2 - Accuracy at 10 epochs

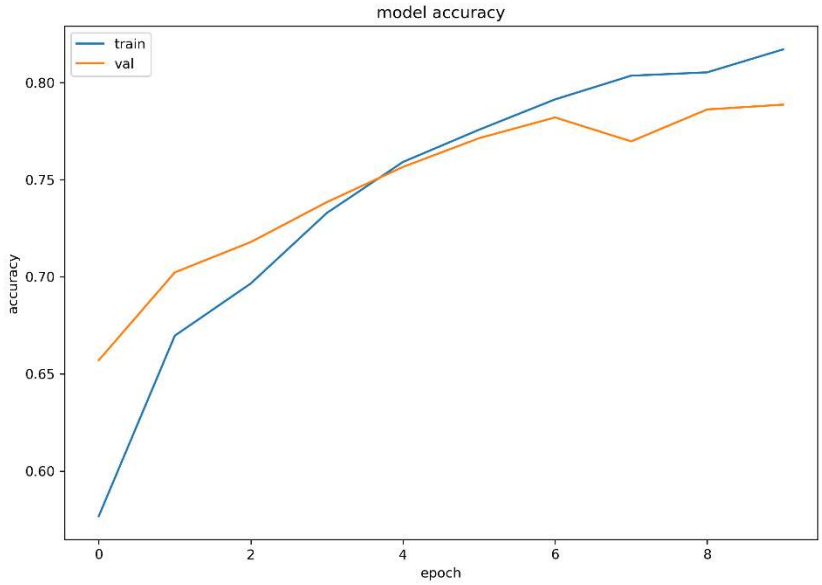

Figure 3 - Dependence of accuracy on the number of epochs

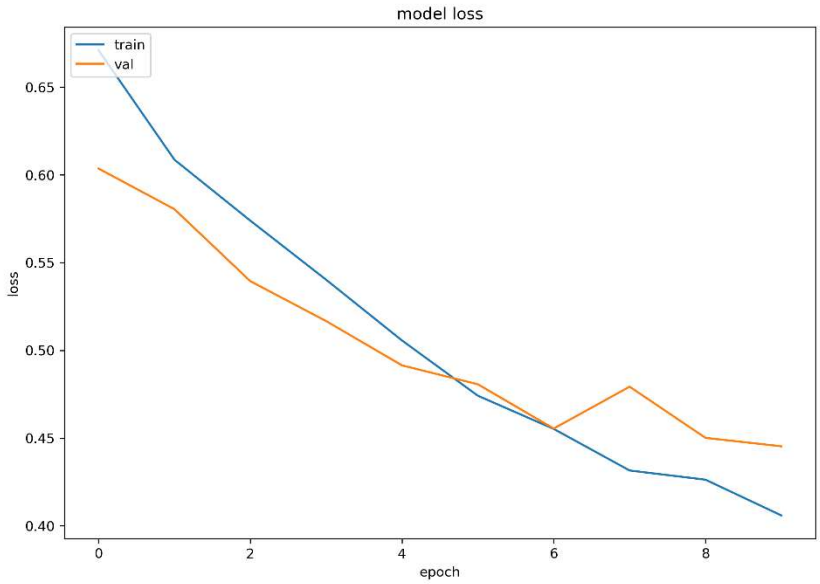

Figure 4 - Dependence of losses on the number of epochs

With 15 training epochs, we obtained an accuracy of 84.05% (Figure 5). The graph of accuracy and loss vs. number of epochs is shown in Figures 6 and 7, respectively.

```
  scores = model.evaluate_generator(test_generator, nb_test_samples // batch_size)
Accuracy on test data: 84.05%
```

Figure 5 - Accuracy at 15 epochs

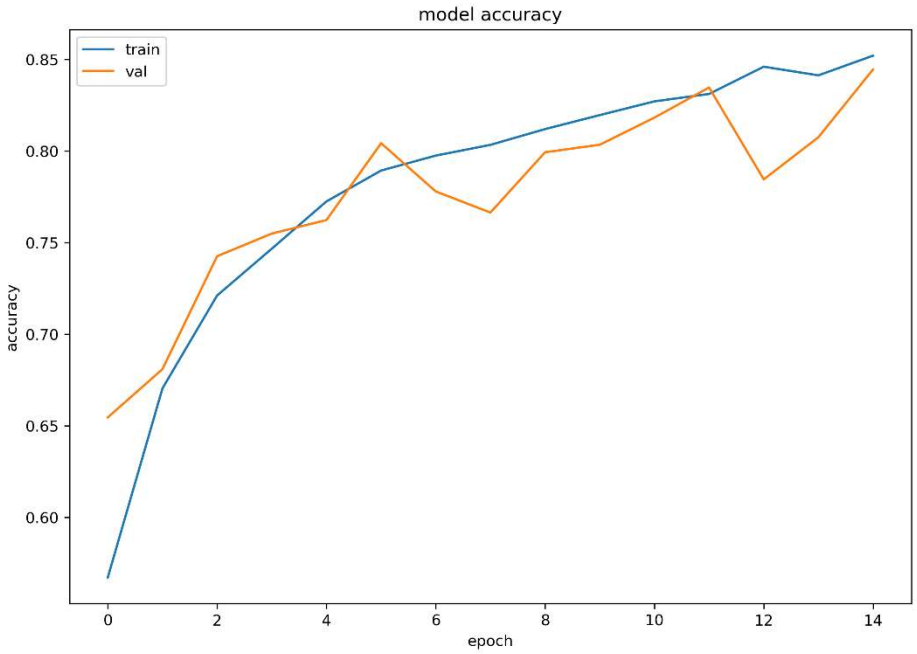

Figure 6 - Dependence of accuracy on the number of epochs

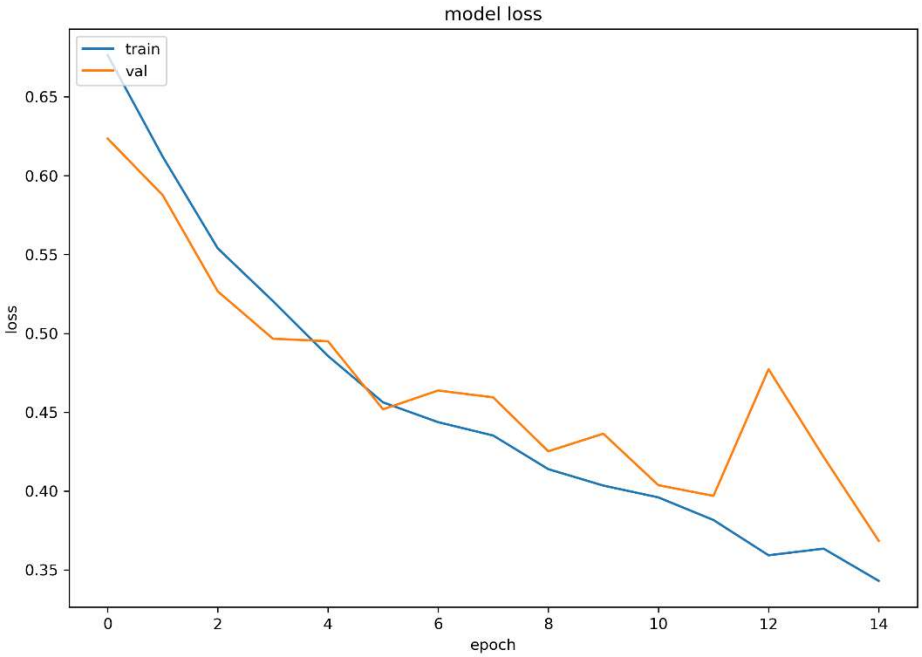

Figure 7 - Dependence of losses on the number of epochs

With 20 training epochs, we obtained an accuracy of 83.80% (Figure 8). The graph of accuracy and loss vs. number of epochs is shown in Figure 9 and Figure 10, respectively.

```
  scores = model.evaluate_generator(test_generator, nb_test_samples // batch_size)
Accuracy on test data: 83.80%
```

Figure 8 - Accuracy at 20 epochs

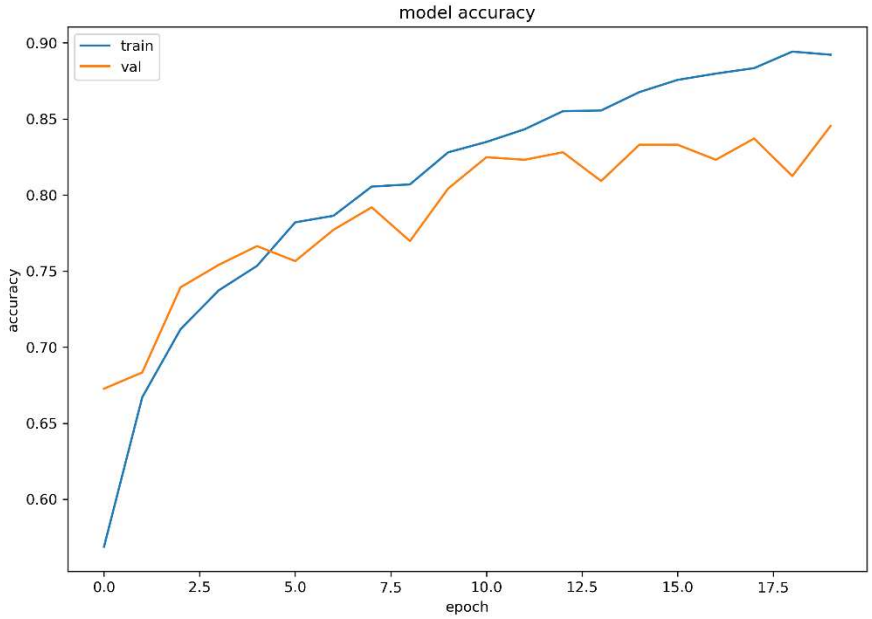

Figure 9 - Dependence of accuracy on the number of epochs

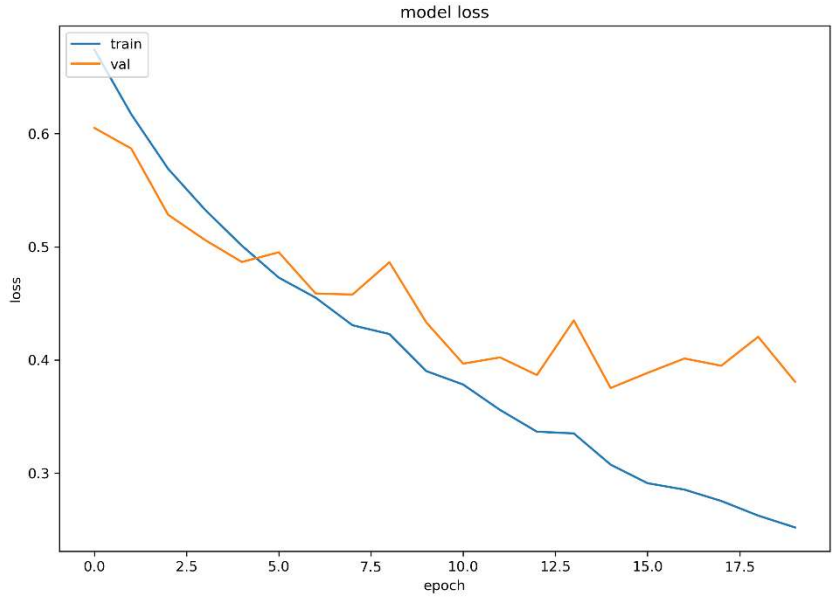

Figure 10 - Dependence of losses on the number of epochs

Part 2

For the second part of the assignment, we loaded the VGG16 architecture with the ready weights obtained by training on the 'imagenet' set (Figure 11). Then we added additional layers to the model according to the task (Figure 12)

```
3 base_model = VGG16(weights='imagenet',
4                    include_top=False,      #
5                    input_shape=(32, 32, 3))
6 base_model.trainable = False
7                                            #
8
9 base_model.summary()
```

Figure 11 - Loading the pre-trained network

```
1 # add layers to VGG16:
2 inputs = keras.Input(shape=(32, 32, 3))
3 x = base_model(inputs, training=False)
4 x = Flatten()(x)                          # + flattening
5 x = Dense(256, activation="relu")(x)      # + Dense fullyconnected layer with 256 neurons + ReLu
6 x = Dropout(0.5)(x)                       # + Dropout
7 outputs = Dense(1, activation="sigmoid")(x) # + Dense layer with 1 neuron + sigmoid
8 model = keras.Model(inputs, outputs)
9
10 model.summary()
```

Figure 12 - Adding additional layers

With 10 training epochs, we obtained an accuracy of 72.70% (Figure 13). The graph of accuracy and loss vs. number of epochs is shown in Figures 14 and 15, respectively.

```
1 scores = model.evaluate_generator(test_generator, nb_test_samples // batch_size)
2 print("Accuracy on test data: %.2f%%" % (scores[1]*100))
```

```
<ipython-input-58-dca356712a64>:1: UserWarning: `Model.evaluate_generator` is deprecat
  scores = model.evaluate_generator(test_generator, nb_test_samples // batch_size)
Accuracy on test data: 72.70%
```

Figure 13 - Accuracy at 10 epochs

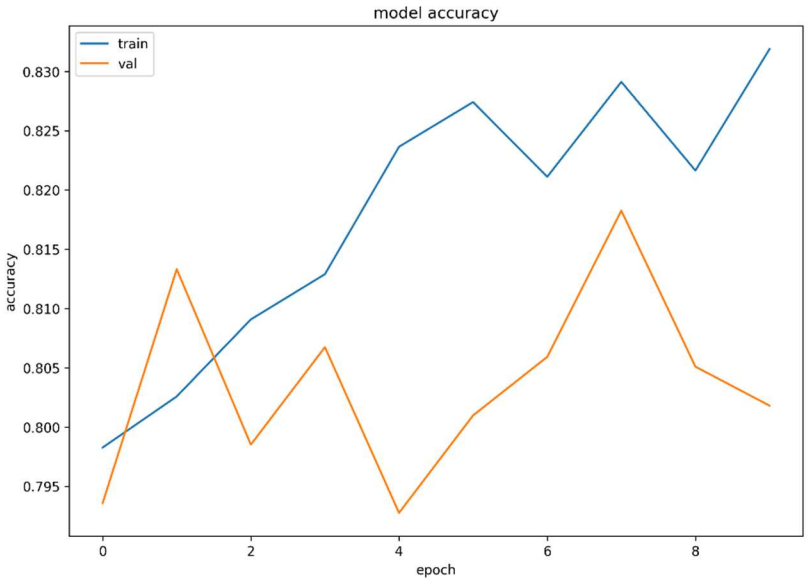

Figure 14 - Dependence of accuracy on the number of epochs

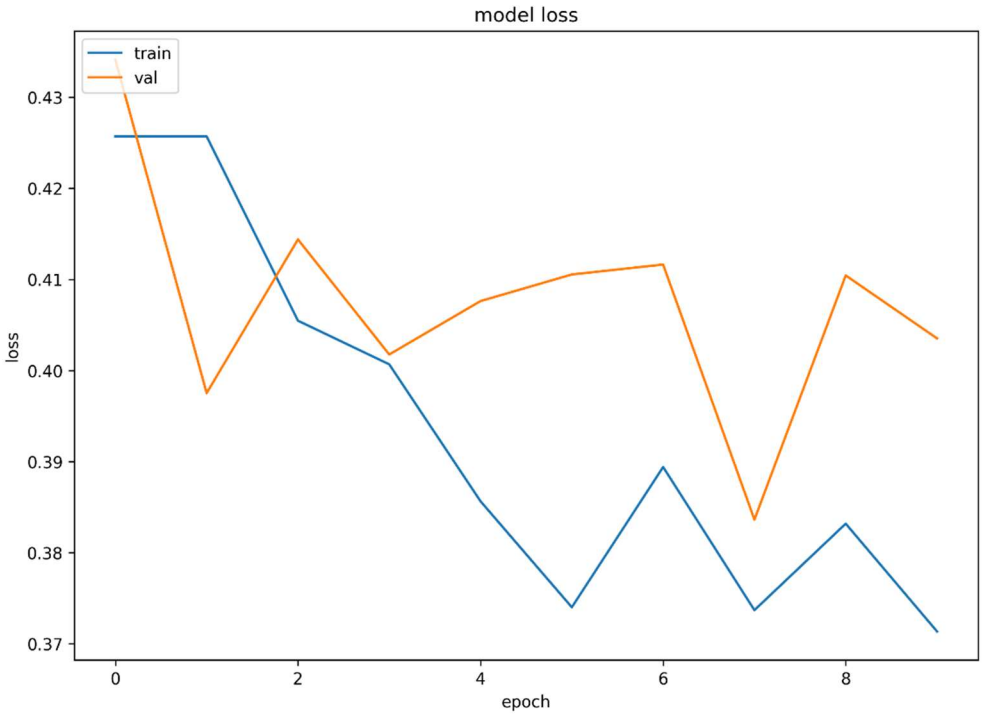

Figure 15 - Dependence of losses on the number of epochs

We then trained the model based on the 'MobileNetV2' architecture (Figure 16). Then we added additional layers to the model according to the task (Figure 17).

```
base_model = MobileNetV2(weights='imagenet',
                         include_top=False,      #
                         input_shape=(64, 64, 3))
base_model.trainable = False                     #
                                                 #

base_model.summary()
```

Figure 16 - Loading a pre-trained network

```
# add layers to MobileNetV2:
inputs = keras.Input(shape=(64, 64, 3))
x = base_model(inputs, training=False)
x = Flatten()(x)                            # + flattening
x = Dense(256, activation="relu")(x)        # + Dense fullyconnected layer with 256 neurons + ReLu
x = Dropout(0.5)(x)                         # + Dropout
outputs = Dense(1, activation="sigmoid")(x) # + Dense layer with 1 neuron + sigmoid
model = keras.Model(inputs, outputs)

model.summary()
```

Figure 17 - Adding additional layers

With 10 training epochs, we obtained an accuracy of 84.29% (Figure 18). The graph of accuracy and loss vs. number of epochs is shown in Figures 19 and 20, respectively.

```
scores = model.evaluate_generator(test_generator, nb_test_samples // batch_size)
print("Accuracy on test data: %.2f%%" % (scores[1]*100))
```

```
<ipython-input-51-dca356712a64>:1: UserWarning: `Model.evaluate_generator` is deprec
  scores = model.evaluate_generator(test_generator, nb_test_samples // batch_size)
Accuracy on test data: 84.29%
```

Figure 18 - Accuracy at 10 epochs

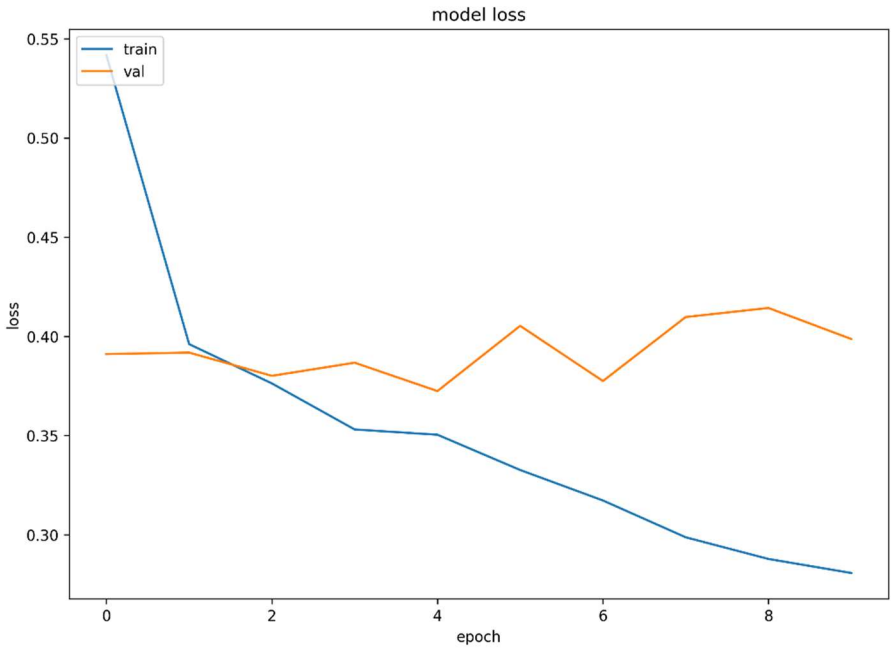

Figure 19 - Dependence of accuracy on the number of epochs

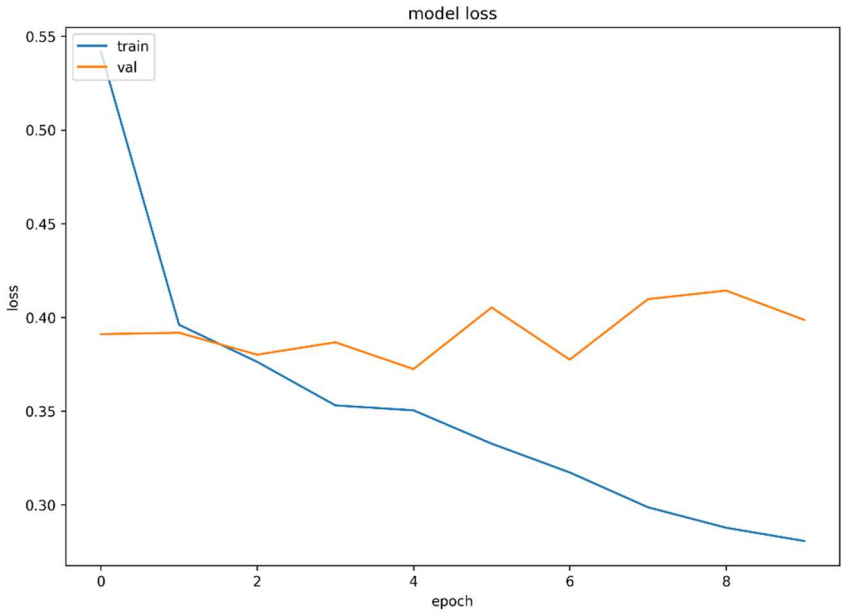

Figure 20 - Dependence of losses on the number of epochs

A custom dataset with 80 images of cats and dogs was then generated to validate our models. An example of an image from the dataset is shown in Figure 21.

Figure 21 - Example of an image from a dataset belonging to the "cat" class

Since the architecture from Part 1 has the highest accuracy at 15 epochs, we will check the classification accuracy on our set at this number of epochs. The obtained accuracy was 84.38%, for VGG16 architecture 77.50% and for MobileNetV2 architecture 83.88%.

Таблица 1 – Точность на X_test и нашем наборе данных

	Our architecture	VGG16	MobileNetV2
X_test	84.05	72.70	84.29
Our test data	84.38	77.50	83.88

Finally, a function was compiled to select the model with the best accuracy (Figure 22).

```
f_lst = [model, vgg16, mobilenetv2]
epochs = [10, 15, 20]
acc = []
```

```
def best_f(f_lst):
  for i in range(len(f_lst)):
    for j in (epochs):
      mdl = f_lst[i](j)
      scores = mdl.evaluate_generator(test_generator, nb_test_samples // batch_size)

      acc.append([str(f_lst[i]), scores, j]) # [name, accuracy, epoch]
  return acc
```

Figure 22 - Model selection function with better accuracy

Conclusion

In this assignment, the transfer learning method was studied. A CNN with three convolutional layers for binary classification was created and adapted. Accuracy on test data at 10, 15 and 20 epochs was evaluated and Loss(number_of_epochs) and Accuracy(number_of_epochs) were plotted. The model had the highest accuracy of 84.05% at 10 epochs of training.

A transfer learning model was created and fitted using the pre-trained weights of the VGG16 and MobileNetV2 model from the keras application. These architectures were trained for 10 epochs and their accuracy was 72.70% and 84.29% respectively.

A custom set of cat or dog images was compiled to test the performance of the model. The composed set contains 80 images of both classes. When trained on the custom set, it was found that the accuracy of all the models except MobileNetV2 was found to be higher.

A function for inferring the best algorithm was also developed.

Appendix

GitHub link:

https://github.com/LesostepnoyGnom/homework_ML/blob/main/task5/Task_5_Chernobrovkin_J4133c_ipynb_.ipynb