Лабораторная работа №5. Применение сверточных нейронных сетей (бинарная классификация)

1. Загрузите данные. Разделите исходный набор данных на обучающую, валидационную и контрольную выборки.
2. Реализуйте глубокую нейронную сеть с как минимум тремя сверточными слоями. Какое качество классификации получено?
3. Примените дополнение данных (data augmentation). Как это повлияло на качество классификатора?
4. Поэкспериментируйте с готовыми нейронными сетями (например, AlexNet, VGG16, Inception и т.п.), применив передаточное обучение. Как это повлияло на качество классификатора? Какой максимальный результат удалось получить на сайте Kaggle? Почему?

```
# Install Kaggle library
!pip install -q kaggle

import os
os.environ['KAGGLE_USERNAME'] = "awful1996" # username from the json file
os.environ['KAGGLE_KEY'] = "5e55f76a1cc7a4772bd7803cba8fb2c1" # key from the json file
#!kaggle datasets download -d iarunava/happy-house-dataset # api copied from kaggle
!kaggle competitions download -c dogs-vs-cats
```

```
Warning: Looks like you're using an outdated API Version, please consider updating (server 1.5.6 / client 1.5.4)
Downloading train.zip to /content
 98% 533M/543M [00:04<00:00, 164MB/s]
100% 543M/543M [00:04<00:00, 137MB/s]
Downloading sampleSubmission.csv to /content
  0% 0.00/86.8k [00:00<?, ?B/s]
100% 86.8k/86.8k [00:00<00:00, 91.6MB/s]
Downloading test1.zip to /content
 98% 265M/271M [00:01<00:00, 160MB/s]
100% 271M/271M [00:01<00:00, 161MB/s]
```

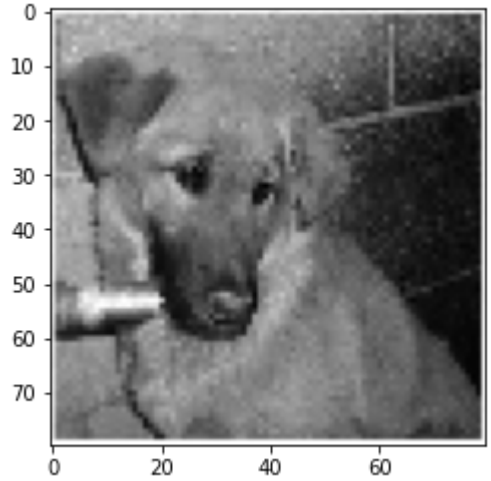
```
!unzip train.zip
!unzip test1.zip
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

```
import cv2

import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout, Activation, Conv2D, MaxPooling2D, BatchNormalization, GlobalA
import tensorflow_datasets as tfds

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

import os

main_dir = ""
train_dir = "train"
path = os.path.join(main_dir,train_dir)
print(path)
```

```
train
```

```
for p in os.listdir(path):
    category = p.split(".")[0]
    img_array = cv2.imread(os.path.join(path,p),cv2.IMREAD_GRAYSCALE)
    new_img_array = cv2.resize(img_array, dsize=(80, 80))
    plt.imshow(new_img_array,cmap="gray")
    break
```




```
X = []
y = []
convert = lambda category : int(category == 'dog')
def create_test_data(path):
    for p in os.listdir(path):
        category = p.split(".")[0]
        category = convert(category)
        img_array = cv2.imread(os.path.join(path,p),cv2.IMREAD_GRAYSCALE)
        new_img_array = cv2.resize(img_array, dsize=(80, 80))
        X.append(new_img_array)
        y.append(category)
create_test_data(path)
X = np.array(X).reshape(-1, 80,80,1)
y = np.array(y)

#Normalize data
X = X/255.0

from sklearn.model_selection import train_test_split
X, X_test, y, y_test = train_test_split(X, y, test_size=0.1)
X, X_val, y, y_val = train_test_split(X, y, test_size=0.1)
```

```
datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True, validation_split=0.1)
datagen.fit(X)
```

Задание 2. Реализуйте глубокую нейронную сеть с как минимум тремя сверточными слоями. Какое качество классификации получено?

Качество 53 процента

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=X.shape[1:]))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(2, activation='softmax'))
```

```
model.compile(optimizer="adam",
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])


model.fit(
    X,y,
    epochs=100,
    batch_size=32,
    validation_split=0.1,
    callbacks=[
      tf.keras.callbacks.EarlyStopping(
        patience=10,
        restore_best_weights=True
      )
    ]
)
```

```
Epoch 1/100
570/570 [==============================] - 6s 10ms/step - loss: 0.6837 - accuracy: 0.5525 - val_loss: 0.6195 - val_acc
Epoch 2/100
570/570 [==============================] - 5s 10ms/step - loss: 0.5979 - accuracy: 0.6765 - val_loss: 0.5795 - val_acc
Epoch 3/100
570/570 [==============================] - 5s 9ms/step - loss: 0.5155 - accuracy: 0.7451 - val_loss: 0.5229 - val_accu
Epoch 4/100
570/570 [==============================] - 5s 9ms/step - loss: 0.4658 - accuracy: 0.7771 - val_loss: 0.4455 - val_accu
Epoch 5/100
570/570 [==============================] - 5s 10ms/step - loss: 0.4206 - accuracy: 0.8052 - val_loss: 0.4145 - val_acc
Epoch 6/100
570/570 [==============================] - 6s 10ms/step - loss: 0.3904 - accuracy: 0.8187 - val_loss: 0.3973 - val_acc
Epoch 7/100
570/570 [==============================] - 5s 10ms/step - loss: 0.3616 - accuracy: 0.8388 - val_loss: 0.3840 - val_acc
Epoch 8/100
570/570 [==============================] - 5s 9ms/step - loss: 0.3331 - accuracy: 0.8523 - val_loss: 0.3673 - val_accu
Epoch 9/100
570/570 [==============================] - 5s 9ms/step - loss: 0.2986 - accuracy: 0.8707 - val_loss: 0.3670 - val_accu
Epoch 10/100
570/570 [==============================] - 5s 9ms/step - loss: 0.2679 - accuracy: 0.8841 - val_loss: 0.3872 - val_accu
Epoch 11/100
570/570 [==============================] - 5s 10ms/step - loss: 0.2444 - accuracy: 0.8964 - val_loss: 0.3633 - val_acc
Epoch 12/100
570/570 [==============================] - 5s 9ms/step - loss: 0.2153 - accuracy: 0.9139 - val_loss: 0.3619 - val_accu
Epoch 13/100
570/570 [==============================] - 5s 9ms/step - loss: 0.1909 - accuracy: 0.9219 - val_loss: 0.3809 - val_accu
Epoch 14/100
570/570 [==============================] - 5s 9ms/step - loss: 0.1795 - accuracy: 0.9277 - val_loss: 0.4134 - val_accu
Epoch 15/100
570/570 [==============================] - 5s 10ms/step - loss: 0.1696 - accuracy: 0.9335 - val_loss: 0.3882 - val_acc
Epoch 16/100
570/570 [==============================] - 5s 9ms/step - loss: 0.1509 - accuracy: 0.9411 - val_loss: 0.3977 - val_accu
Epoch 17/100
570/570 [==============================] - 5s 9ms/step - loss: 0.1365 - accuracy: 0.9469 - val_loss: 0.4182 - val_accu
Epoch 18/100
570/570 [==============================] - 5s 9ms/step - loss: 0.1282 - accuracy: 0.9489 - val_loss: 0.4475 - val_accu
Epoch 19/100
570/570 [==============================] - 5s 9ms/step - loss: 0.1110 - accuracy: 0.9581 - val_loss: 0.5010 - val_accu
Epoch 20/100
570/570 [==============================] - 5s 9ms/step - loss: 0.1141 - accuracy: 0.9568 - val_loss: 0.4391 - val_accu
Epoch 21/100
570/570 [==============================] - 5s 9ms/step - loss: 0.1014 - accuracy: 0.9610 - val_loss: 0.4863 - val_accu
```

```
Epoch 22/100
570/570 [==============================] - 5s 9ms/step - loss: 0.0935 - accuracy: 0.9652 - val_loss: 0.5002 - val_accu
<tensorflow.python.keras.callbacks.History at 0x7f7dc014a898>
```

```
_, acc = model.evaluate(X_test, y_test)
print(f'Accuracy = {acc:.5f}')
```

```
79/79 [==============================] - 0s 4ms/step - loss: 0.3768 - accuracy: 0.8364
Accuracy = 0.83640
```

Задание 3. Примените дополнение данных (data augmentation). Как это повлияло на качество классификатора?

Качество модели понизилось до 59 процентов

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=X.shape[1:]))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(2, activation='softmax'))

model.compile(optimizer="adam",
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
model.fit_generator(
    datagen.flow(X, y, batch_size=32),
    steps_per_epoch=len(X) / 32,
    epochs=100,
    callbacks=[
      tf.keras.callbacks.EarlyStopping(
        patience=2,
        restore_best_weights=True,
        monitor='accuracy'
      )
    ])
```

```
WARNING:tensorflow:From <ipython-input-13-584c199b5e6f>:10: Model.fit_generator (from tensorflow.python.keras.engine.t
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/100
633/632 [==============================] - 14s 22ms/step - loss: 0.6941 - accuracy: 0.5403
Epoch 2/100
633/632 [==============================] - 14s 22ms/step - loss: 0.6341 - accuracy: 0.6421
Epoch 3/100
633/632 [==============================] - 14s 22ms/step - loss: 0.5796 - accuracy: 0.6940
Epoch 4/100
633/632 [==============================] - 14s 22ms/step - loss: 0.5407 - accuracy: 0.7278
Epoch 5/100
633/632 [==============================] - 14s 22ms/step - loss: 0.5067 - accuracy: 0.7519
Epoch 6/100
633/632 [==============================] - 14s 22ms/step - loss: 0.4779 - accuracy: 0.7702
Epoch 7/100
633/632 [==============================] - 14s 22ms/step - loss: 0.4584 - accuracy: 0.7824
Epoch 8/100
633/632 [==============================] - 14s 22ms/step - loss: 0.4381 - accuracy: 0.7960
Epoch 9/100
633/632 [==============================] - 14s 22ms/step - loss: 0.4286 - accuracy: 0.7993
Epoch 10/100
633/632 [==============================] - 14s 22ms/step - loss: 0.4208 - accuracy: 0.8041
Epoch 11/100
633/632 [==============================] - 14s 23ms/step - loss: 0.4061 - accuracy: 0.8132
Epoch 12/100
633/632 [==============================] - 14s 22ms/step - loss: 0.3921 - accuracy: 0.8212
Epoch 13/100
633/632 [==============================] - 14s 23ms/step - loss: 0.3855 - accuracy: 0.8269
Epoch 14/100
633/632 [==============================] - 14s 22ms/step - loss: 0.3787 - accuracy: 0.8298
Epoch 15/100
633/632 [==============================] - 14s 22ms/step - loss: 0.3755 - accuracy: 0.8277
Epoch 16/100
633/632 [==============================] - 14s 23ms/step - loss: 0.3688 - accuracy: 0.8335
Epoch 17/100
633/632 [==============================] - 14s 22ms/step - loss: 0.3677 - accuracy: 0.8353
Epoch 18/100
633/632 [==============================] - 14s 22ms/step - loss: 0.3602 - accuracy: 0.8378
Epoch 19/100
633/632 [==============================] - 14s 23ms/step - loss: 0.3531 - accuracy: 0.8429
Epoch 20/100
```

```
633/632 [==============================] - 14s 22ms/step - loss: 0.3506 - accuracy: 0.8427
Epoch 21/100
633/632 [==============================] - 14s 22ms/step - loss: 0.3525 - accuracy: 0.8439
Epoch 22/100
633/632 [==============================] - 14s 22ms/step - loss: 0.3473 - accuracy: 0.8466
Epoch 23/100
633/632 [==============================] - 14s 23ms/step - loss: 0.3424 - accuracy: 0.8477
Epoch 24/100
633/632 [==============================] - 14s 23ms/step - loss: 0.3446 - accuracy: 0.8487
Epoch 25/100
633/632 [==============================] - 14s 23ms/step - loss: 0.3407 - accuracy: 0.8515
Epoch 26/100
633/632 [==============================] - 14s 22ms/step - loss: 0.3398 - accuracy: 0.8495
Epoch 27/100
633/632 [==============================] - 14s 23ms/step - loss: 0.3306 - accuracy: 0.8524
Epoch 28/100
633/632 [==============================] - 14s 23ms/step - loss: 0.3333 - accuracy: 0.8540
Epoch 29/100
633/632 [==============================] - 14s 22ms/step - loss: 0.3292 - accuracy: 0.8550
Epoch 30/100
633/632 [==============================] - 14s 23ms/step - loss: 0.3246 - accuracy: 0.8555
Epoch 31/100
633/632 [==============================] - 14s 22ms/step - loss: 0.3233 - accuracy: 0.8577
Epoch 32/100
633/632 [==============================] - 14s 23ms/step - loss: 0.3229 - accuracy: 0.8595
Epoch 33/100
633/632 [==============================] - 14s 22ms/step - loss: 0.3152 - accuracy: 0.8647
Epoch 34/100
633/632 [==============================] - 14s 22ms/step - loss: 0.3140 - accuracy: 0.8616
Epoch 35/100
633/632 [==============================] - 14s 22ms/step - loss: 0.3182 - accuracy: 0.8625
<tensorflow.python.keras.callbacks.History at 0x7f7dd7c5fcc0>
```

```
_, acc = model.evaluate(X_test, y_test)
print(f'Accuracy = {acc:.5f}')
```

```
79/79 [==============================] - 0s 4ms/step - loss: 0.6999 - accuracy: 0.5980
Accuracy = 0.59800
```

Задание 4. Поэкспериментируйте с готовыми нейронными сетями (например, AlexNet, VGG16, Inception и т.п.), применив передаточное обучение. Как это повлияло на качество классификатора? Какой максимальный результат удалось получить на сайте Kaggle? Почему?

Удалось получить результат 91 процент, так как MobileNet была обучена на данных ImageNet

```
from keras.applications import MobileNet
from keras.layers import GlobalAveragePooling2D, Dense
from keras.models import Model

base_model=MobileNet(weights='imagenet',include_top=False)

x=base_model.output
x=GlobalAveragePooling2D()(x)
x=Dense(1024,activation='relu')(x)
x=Dense(1024,activation='relu')(x)
x=Dense(512,activation='relu')(x)
preds=Dense(2,activation='softmax')(x)
model=Model(inputs=base_model.input,outputs=preds)
model.summary()
```

```
/usr/local/lib/python3.6/dist-packages/keras_applications/mobilenet.py:207: UserWarning: `input_shape` is undefined o
  warnings.warn('`input_shape` is undefined or non-square, '
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         (None, None, None, 3)     0
_________________________________________________________________
conv1_pad (ZeroPadding2D)    (None, None, None, 3)     0
_________________________________________________________________
conv1 (Conv2D)               (None, None, None, 32)    864
_________________________________________________________________
conv1_bn (BatchNormalization (None, None, None, 32)    128
_________________________________________________________________
conv1_relu (ReLU)            (None, None, None, 32)    0
_________________________________________________________________
conv_dw_1 (DepthwiseConv2D)  (None, None, None, 32)    288
_________________________________________________________________
conv_dw_1_bn (BatchNormaliza (None, None, None, 32)    128
_________________________________________________________________
conv_dw_1_relu (ReLU)        (None, None, None, 32)    0
_________________________________________________________________
conv_pw_1 (Conv2D)           (None, None, None, 64)    2048
_________________________________________________________________
conv_pw_1_bn (BatchNormaliza (None, None, None, 64)    256
_________________________________________________________________
conv_pw_1_relu (ReLU)        (None, None, None, 64)    0
_________________________________________________________________
conv_pad_2 (ZeroPadding2D)   (None, None, None, 64)    0
_________________________________________________________________
conv_dw_2 (DepthwiseConv2D)  (None, None, None, 64)    576
_________________________________________________________________
conv_dw_2_bn (BatchNormaliza (None, None, None, 64)    256
_________________________________________________________________
conv_dw_2_relu (ReLU)        (None, None, None, 64)    0
_________________________________________________________________
conv_pw_2 (Conv2D)           (None, None, None, 128)   8192
_________________________________________________________________
conv_pw_2_bn (BatchNormaliza (None, None, None, 128)   512
_________________________________________________________________
conv_pw_2_relu (ReLU)        (None, None, None, 128)   0
_________________________________________________________________
```

conv_dw_3 (DepthwiseConv2D)	(None, None, None, 128)	1152
conv_dw_3_bn (BatchNormaliza	(None, None, None, 128)	512
conv_dw_3_relu (ReLU)	(None, None, None, 128)	0
conv_pw_3 (Conv2D)	(None, None, None, 128)	16384
conv_pw_3_bn (BatchNormaliza	(None, None, None, 128)	512
conv_pw_3_relu (ReLU)	(None, None, None, 128)	0
conv_pad_4 (ZeroPadding2D)	(None, None, None, 128)	0
conv_dw_4 (DepthwiseConv2D)	(None, None, None, 128)	1152
conv_dw_4_bn (BatchNormaliza	(None, None, None, 128)	512
conv_dw_4_relu (ReLU)	(None, None, None, 128)	0
conv_pw_4 (Conv2D)	(None, None, None, 256)	32768
conv_pw_4_bn (BatchNormaliza	(None, None, None, 256)	1024
conv_pw_4_relu (ReLU)	(None, None, None, 256)	0
conv_dw_5 (DepthwiseConv2D)	(None, None, None, 256)	2304
conv_dw_5_bn (BatchNormaliza	(None, None, None, 256)	1024
conv_dw_5_relu (ReLU)	(None, None, None, 256)	0
conv_pw_5 (Conv2D)	(None, None, None, 256)	65536
conv_pw_5_bn (BatchNormaliza	(None, None, None, 256)	1024
conv_pw_5_relu (ReLU)	(None, None, None, 256)	0
conv_pad_6 (ZeroPadding2D)	(None, None, None, 256)	0
conv_dw_6 (DepthwiseConv2D)	(None, None, None, 256)	2304

```
conv_dw_6_bn (BatchNormaliza (None, None, None, 256)   1024
_________________________________________________________________
conv_dw_6_relu (ReLU)        (None, None, None, 256)   0
_________________________________________________________________
conv_pw_6 (Conv2D)           (None, None, None, 512)   131072
_________________________________________________________________
conv_pw_6_bn (BatchNormaliza (None, None, None, 512)   2048
_________________________________________________________________
conv_pw_6_relu (ReLU)        (None, None, None, 512)   0
_________________________________________________________________
conv_dw_7 (DepthwiseConv2D)  (None, None, None, 512)   4608
_________________________________________________________________
conv_dw_7_bn (BatchNormaliza (None, None, None, 512)   2048
_________________________________________________________________
conv_dw_7_relu (ReLU)        (None, None, None, 512)   0
_________________________________________________________________
conv_pw_7 (Conv2D)           (None, None, None, 512)   262144
_________________________________________________________________
conv_pw_7_bn (BatchNormaliza (None, None, None, 512)   2048
_________________________________________________________________
conv_pw_7_relu (ReLU)        (None, None, None, 512)   0
_________________________________________________________________
conv_dw_8 (DepthwiseConv2D)  (None, None, None, 512)   4608
_________________________________________________________________
conv_dw_8_bn (BatchNormaliza (None, None, None, 512)   2048
_________________________________________________________________
conv_dw_8_relu (ReLU)        (None, None, None, 512)   0
_________________________________________________________________
conv_pw_8 (Conv2D)           (None, None, None, 512)   262144
_________________________________________________________________
conv_pw_8_bn (BatchNormaliza (None, None, None, 512)   2048
_________________________________________________________________
conv_pw_8_relu (ReLU)        (None, None, None, 512)   0
_________________________________________________________________
conv_dw_9 (DepthwiseConv2D)  (None, None, None, 512)   4608
_________________________________________________________________
conv_dw_9_bn (BatchNormaliza (None, None, None, 512)   2048
_________________________________________________________________
conv_dw_9_relu (ReLU)        (None, None, None, 512)   0
_________________________________________________________________
conv_pw_9 (Conv2D)           (None, None, None, 512)   262144
_________________________________________________________________
conv_pw_9_bn (BatchNormaliza (None, None, None, 512)   2048
```

```
conv_pw_9_bn (BatchNormaliza (None, None, None, 512)   2048
_________________________________________________________________
conv_pw_9_relu (ReLU)        (None, None, None, 512)   0
_________________________________________________________________
conv_dw_10 (DepthwiseConv2D) (None, None, None, 512)   4608
_________________________________________________________________
conv_dw_10_bn (BatchNormaliz (None, None, None, 512)   2048
_________________________________________________________________
conv_dw_10_relu (ReLU)       (None, None, None, 512)   0
_________________________________________________________________
conv_pw_10 (Conv2D)          (None, None, None, 512)   262144
_________________________________________________________________
conv_pw_10_bn (BatchNormaliz (None, None, None, 512)   2048
_________________________________________________________________
conv_pw_10_relu (ReLU)       (None, None, None, 512)   0
_________________________________________________________________
conv_dw_11 (DepthwiseConv2D) (None, None, None, 512)   4608
_________________________________________________________________
conv_dw_11_bn (BatchNormaliz (None, None, None, 512)   2048
_________________________________________________________________
conv_dw_11_relu (ReLU)       (None, None, None, 512)   0
_________________________________________________________________
conv_pw_11 (Conv2D)          (None, None, None, 512)   262144
_________________________________________________________________
conv_pw_11_bn (BatchNormaliz (None, None, None, 512)   2048
_________________________________________________________________
conv_pw_11_relu (ReLU)       (None, None, None, 512)   0
_________________________________________________________________
conv_pad_12 (ZeroPadding2D)  (None, None, None, 512)   0
_________________________________________________________________
conv_dw_12 (DepthwiseConv2D) (None, None, None, 512)   4608
_________________________________________________________________
conv_dw_12_bn (BatchNormaliz (None, None, None, 512)   2048
_________________________________________________________________
conv_dw_12_relu (ReLU)       (None, None, None, 512)   0
_________________________________________________________________
conv_pw_12 (Conv2D)          (None, None, None, 1024)  524288
_________________________________________________________________
conv_pw_12_bn (BatchNormaliz (None, None, None, 1024)  4096
_________________________________________________________________
conv_pw_12_relu (ReLU)       (None, None, None, 1024)  0
_________________________________________________________________
conv_dw_13 (DepthwiseConv2D) (None, None, None, 1024)  9216
```

```
conv_dw_13_bn (BatchNormaliz (None, None, None, 1024)  4096
_________________________________________________________________
conv_dw_13_relu (ReLU)       (None, None, None, 1024)  0
_________________________________________________________________
conv_pw_13 (Conv2D)          (None, None, None, 1024)  1048576
_________________________________________________________________
conv_pw_13_bn (BatchNormaliz (None, None, None, 1024)  4096
_________________________________________________________________
conv_pw_13_relu (ReLU)       (None, None, None, 1024)  0
_________________________________________________________________
global_average_pooling2d_3 ( (None, 1024)              0
_________________________________________________________________
dense_9 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dense_10 (Dense)             (None, 1024)              1049600
_________________________________________________________________
dense_11 (Dense)             (None, 512)               524800
_________________________________________________________________
dense_12 (Dense)             (None, 2)                 1026
=================================================================
Total params: 5,853,890
Trainable params: 5,832,002
Non-trainable params: 21,888
_________________________________________________________________
```

```
import numpy as np
def transform(dataset):
  newDataset = list()
  for x in dataset:
    x = np.repeat(x, 3, 2)
    newDataset.append(x)
  return np.array(newDataset)

newTrainX = transform(X)

print(newTrainX.shape)
```

```
(20250, 80, 80, 3)
```

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model.fit(newTrainX, y, validation_split=0.1, epochs=100, callbacks=[
      tf.keras.callbacks.EarlyStopping(
        patience=2,
        restore_best_weights=True,
        monitor='accuracy'
      )
    ]
)
```

```
Train on 18225 samples, validate on 2025 samples
Epoch 1/100
18225/18225 [==============================] - 40s 2ms/step - loss: 0.3213 - accuracy: 0.8714 - val_loss: 0.4389 - va
Epoch 2/100
18225/18225 [==============================] - 35s 2ms/step - loss: 0.2064 - accuracy: 0.9198 - val_loss: 0.1841 - va
Epoch 3/100
18225/18225 [==============================] - 35s 2ms/step - loss: 0.1565 - accuracy: 0.9402 - val_loss: 0.2012 - va
Epoch 4/100
18225/18225 [==============================] - 34s 2ms/step - loss: 0.1232 - accuracy: 0.9514 - val_loss: 0.1932 - va
Epoch 5/100
18225/18225 [==============================] - 34s 2ms/step - loss: 0.1017 - accuracy: 0.9624 - val_loss: 0.2306 - va
Epoch 6/100
18225/18225 [==============================] - 34s 2ms/step - loss: 0.0878 - accuracy: 0.9671 - val_loss: 0.3484 - va
Epoch 7/100
18225/18225 [==============================] - 34s 2ms/step - loss: 0.0775 - accuracy: 0.9722 - val_loss: 0.2390 - va
Epoch 8/100
18225/18225 [==============================] - 34s 2ms/step - loss: 0.0532 - accuracy: 0.9807 - val_loss: 0.2921 - va
Epoch 9/100
18225/18225 [==============================] - 34s 2ms/step - loss: 0.0510 - accuracy: 0.9811 - val_loss: 0.2676 - va
Epoch 10/100
18225/18225 [==============================] - 34s 2ms/step - loss: 0.0597 - accuracy: 0.9801 - val_loss: 0.3326 - va
Epoch 11/100
18225/18225 [==============================] - 34s 2ms/step - loss: 0.0462 - accuracy: 0.9849 - val_loss: 0.4403 - va
Epoch 12/100
18225/18225 [==============================] - 34s 2ms/step - loss: 0.0419 - accuracy: 0.9856 - val_loss: 0.5484 - va
Epoch 13/100
18225/18225 [==============================] - 34s 2ms/step - loss: 0.0428 - accuracy: 0.9856 - val_loss: 0.3226 - va
Epoch 14/100
18225/18225 [==============================] - 34s 2ms/step - loss: 0.0497 - accuracy: 0.9829 - val_loss: 0.4706 - va
<keras.callbacks.callbacks.History at 0x7f7da94ca470>
```

```
newTestX = transform(X_test)
_, acc = model.evaluate(newTestX, y_test)
print(f'Accuracy = {acc:.5f}')
```

```
2500/2500 [==============================] - 1s 410us/step
Accuracy = 0.91080
```