МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ЗВІТ

з лабораторної роботи №5

з навчальної дисципліни «Проектування та реалізація програмних систем з нейронними мережами»

Виконав:
студент групи ІП-15
Мєшков Андрій Ігорович

Перевірив:
Шимкович В.М.

Київ 2024

ЛАБОРАТОРНА РОБОТА №5

Тема: Згорткові нейронні мережі типу Inception.

Завдання – Написати програму що реалізує згорткову нейронну мережу Inception V3 для розпізнавання об’єктів на зображеннях. Створити власний дата сет з папки на диску, навчити нейронну мережу на цьому датасеті розпізнавати породу Вашої улюбленої собаки чи кота. Навчену нейронну мережу зберегти на комп’ютер написати програму, що відкриває та аналізує зображення.

```
import os
import random
import numpy as np
import cv2
import tensorflow as tf
from imutils import paths
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import RMSprop
from keras.applications.inception_v3 import InceptionV3
import tensorflow.keras.layers as layers
from keras.models import Model
from matplotlib import pyplot as plt

# Define constants
epochs = 30
data_path = "../src/cats-breads"

# Load and preprocess data
data = []
labels = []
image_paths = sorted(list(paths.list_images(data_path)))
random.seed(34)
random.shuffle(image_paths)

for image_path in image_paths:
    image = cv2.imread(image_path)
    if image is None:
        continue
    image = cv2.resize(image, (128, 128))
    data.append(image)
    label = image_path.split(os.path.sep)[-2]
    labels.append(label)

data = np.array(data, dtype="float32") / 255.0
labels = np.array(labels)

(trainX, testX, trainY, testY) = train_test_split(data, labels,
test_size=0.25, random_state=34)

# Visualize sample images
plt.figure(figsize=(12, 12))
for i in range(9):
    plt.subplot(3, 3, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(trainX[i], cmap=plt.cm.binary)
```

```
    plt.xlabel(f'{trainY[i]}')
plt.show()
```

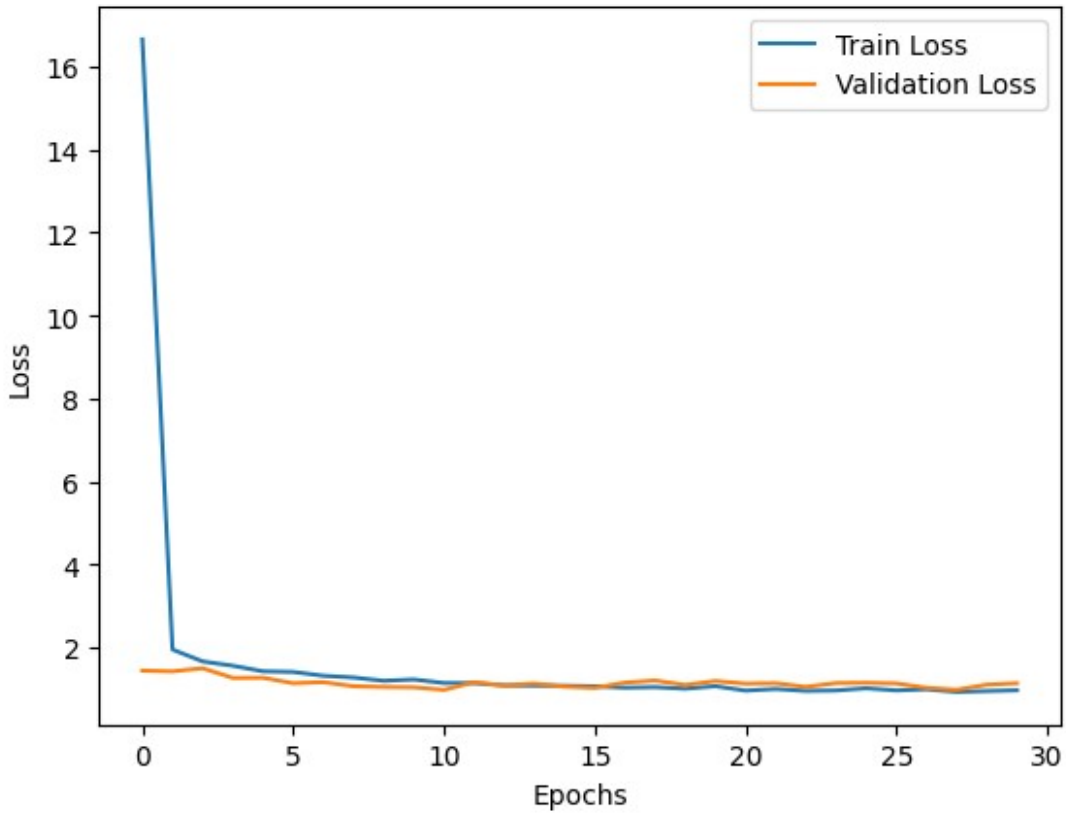

```
# One-hot encode labels
enc = OneHotEncoder()
trainY = enc.fit_transform(trainY.reshape(-1, 1)).toarray()
testY = enc.transform(testY.reshape(-1, 1)).toarray()

# Data augmentation
train_datagen = ImageDataGenerator(rotation_range=30,
width_shift_range=0.1, height_shift_range=0.1,
```

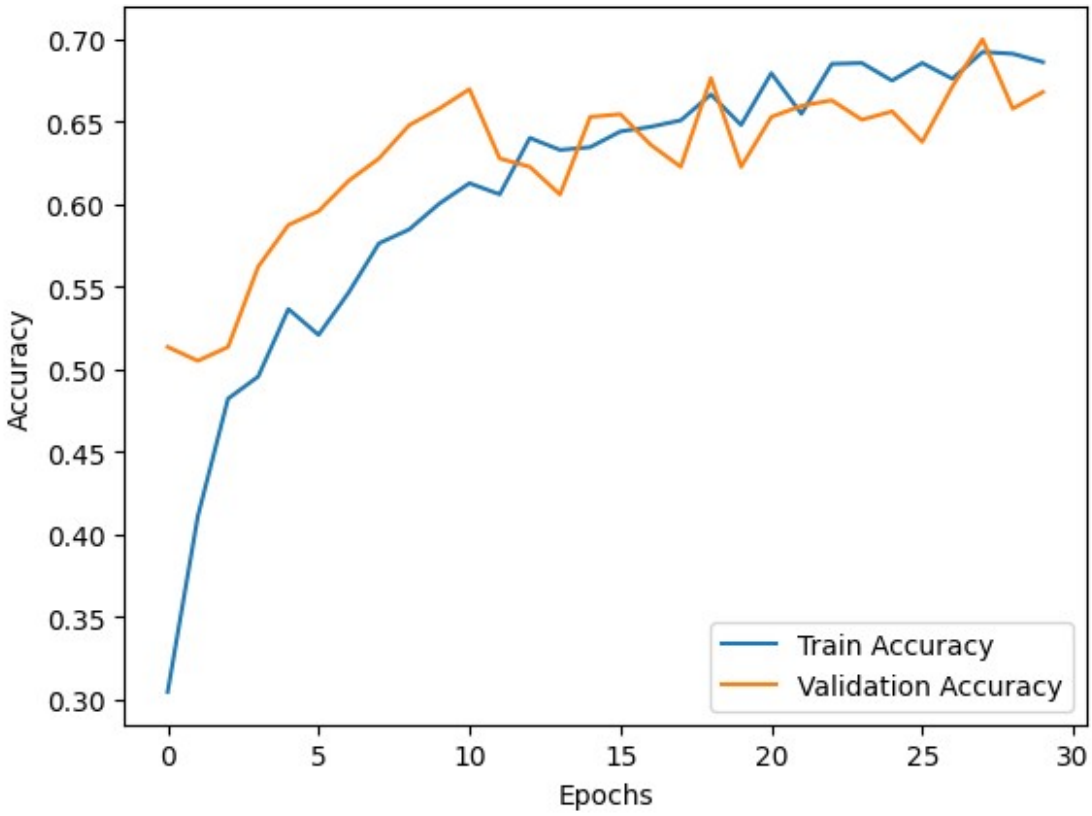
```
                                    shear_range=0.2, zoom_range=0.2,
horizontal_flip=True, fill_mode="nearest")


pretrained_model = InceptionV3(input_shape=trainX[0].shape,
include_top=False, weights='imagenet')

for layer in pretrained_model.layers:
    layer.trainable = False

x = layers.Flatten()(pretrained_model.output)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dropout(0.2)(x)
x = layers.Dense(trainY.shape[1], activation='sigmoid')(x)

model = Model(inputs=pretrained_model.input, outputs=x)
model.compile(optimizer=RMSprop(learning_rate=0.001),
loss="categorical_crossentropy", metrics=["accuracy"])

history = model.fit(train_datagen.flow(trainX, trainY),
validation_data=(testX, testY), epochs=epochs)
```

```
Epoch 1/30

/Users/andrey/Documents/DOcument_study/Year 3.2/ПЗПтаРНС/Lab
5/src/.venv/lib/python3.9/site-packages/keras/src/trainers/data_adapte
rs/py_dataset_adapter.py:120: UserWarning: Your `PyDataset` class
should call `super().__init__(**kwargs)` in its constructor.
`**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will
be ignored.
  self._warn_if_super_not_called()

56/56 ━━━━━━━━━━━━━━━━━━━━ 37s 522ms/step - accuracy: 0.2319 - loss:
36.7523 - val_accuracy: 0.5134 - val_loss: 1.4415
Epoch 2/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 36s 619ms/step - accuracy: 0.4259 - loss:
1.8397 - val_accuracy: 0.5050 - val_loss: 1.4245
Epoch 3/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 35s 588ms/step - accuracy: 0.4741 - loss:
1.6539 - val_accuracy: 0.5134 - val_loss: 1.4994
Epoch 4/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 45s 793ms/step - accuracy: 0.4815 - loss:
1.6058 - val_accuracy: 0.5621 - val_loss: 1.2663
Epoch 5/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 46s 794ms/step - accuracy: 0.5262 - loss:
1.3948 - val_accuracy: 0.5872 - val_loss: 1.2705
Epoch 6/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 46s 789ms/step - accuracy: 0.5297 - loss:
1.3862 - val_accuracy: 0.5956 - val_loss: 1.1383
Epoch 7/30
```

```
56/56 ━━━━━━━━━━━━━━━━━━━━ 53s 912ms/step - accuracy: 0.5415 - loss:
1.3094 - val_accuracy: 0.6141 - val_loss: 1.1644
Epoch 8/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 34s 572ms/step - accuracy: 0.5619 - loss:
1.3078 - val_accuracy: 0.6275 - val_loss: 1.0658
Epoch 9/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 30s 522ms/step - accuracy: 0.5701 - loss:
1.2088 - val_accuracy: 0.6477 - val_loss: 1.0486
Epoch 10/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 31s 531ms/step - accuracy: 0.6130 - loss:
1.1739 - val_accuracy: 0.6577 - val_loss: 1.0413
Epoch 11/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 32s 557ms/step - accuracy: 0.5887 - loss:
1.1909 - val_accuracy: 0.6695 - val_loss: 0.9784
Epoch 12/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 31s 530ms/step - accuracy: 0.6024 - loss:
1.1075 - val_accuracy: 0.6275 - val_loss: 1.1610
Epoch 13/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 32s 549ms/step - accuracy: 0.6367 - loss:
1.0715 - val_accuracy: 0.6225 - val_loss: 1.0831
Epoch 14/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 31s 536ms/step - accuracy: 0.6379 - loss:
1.0411 - val_accuracy: 0.6057 - val_loss: 1.1315
Epoch 15/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 32s 546ms/step - accuracy: 0.6140 - loss:
1.1455 - val_accuracy: 0.6527 - val_loss: 1.0554
Epoch 16/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 30s 522ms/step - accuracy: 0.6372 - loss:
1.0817 - val_accuracy: 0.6544 - val_loss: 1.0268
Epoch 17/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 31s 532ms/step - accuracy: 0.6561 - loss:
1.0016 - val_accuracy: 0.6359 - val_loss: 1.1534
Epoch 18/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 32s 546ms/step - accuracy: 0.6417 - loss:
1.0652 - val_accuracy: 0.6225 - val_loss: 1.2027
Epoch 19/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 30s 515ms/step - accuracy: 0.6674 - loss:
0.9661 - val_accuracy: 0.6762 - val_loss: 1.0963
Epoch 20/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 31s 532ms/step - accuracy: 0.6570 - loss:
1.0385 - val_accuracy: 0.6225 - val_loss: 1.1918
Epoch 21/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 31s 532ms/step - accuracy: 0.6904 - loss:
0.9476 - val_accuracy: 0.6527 - val_loss: 1.1286
Epoch 22/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 31s 534ms/step - accuracy: 0.6466 - loss:
1.0414 - val_accuracy: 0.6594 - val_loss: 1.1380
Epoch 23/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 31s 530ms/step - accuracy: 0.6864 - loss:
```

```
0.9439 - val_accuracy: 0.6628 - val_loss: 1.0505
Epoch 24/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 30s 514ms/step - accuracy: 0.6854 - loss:
0.9491 - val_accuracy: 0.6510 - val_loss: 1.1412
Epoch 25/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 31s 531ms/step - accuracy: 0.6595 - loss:
1.0145 - val_accuracy: 0.6560 - val_loss: 1.1477
Epoch 26/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 32s 541ms/step - accuracy: 0.6812 - loss:
0.9331 - val_accuracy: 0.6376 - val_loss: 1.1326
Epoch 27/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 30s 511ms/step - accuracy: 0.6508 - loss:
1.0446 - val_accuracy: 0.6711 - val_loss: 1.0264
Epoch 28/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 30s 515ms/step - accuracy: 0.7040 - loss:
0.8947 - val_accuracy: 0.6997 - val_loss: 0.9806
Epoch 29/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 31s 534ms/step - accuracy: 0.7061 - loss:
0.9146 - val_accuracy: 0.6577 - val_loss: 1.1032
Epoch 30/30
56/56 ━━━━━━━━━━━━━━━━━━━━ 31s 539ms/step - accuracy: 0.6663 - loss:
1.0093 - val_accuracy: 0.6678 - val_loss: 1.1369

#Visualizing loss and accuracy
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```


Train Loss
Validation Loss
Loss
Epochs
16
14
12
10
8
6
4
2
0
5
10
15
20
25
30


Train Accuracy
Validation Accuracy
Accuracy
Epochs
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0
5
10
15
20
25
30

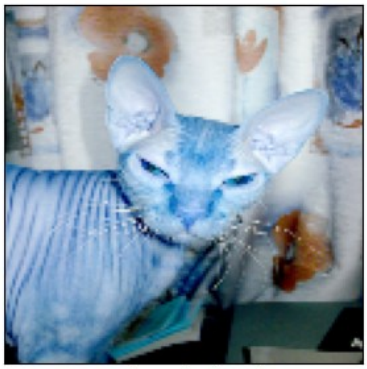
```
# Evaluate the model
test_loss, test_acc = model.evaluate(testX, testY)
print(f'Test accuracy: {test_acc*100}%')
print(f'Test loss: {test_loss}')
```

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 6s 309ms/step - accuracy: 0.6641 - loss:
1.1299
Test accuracy: 66.77852272987366%
Test loss: 1.1369472742080688
```

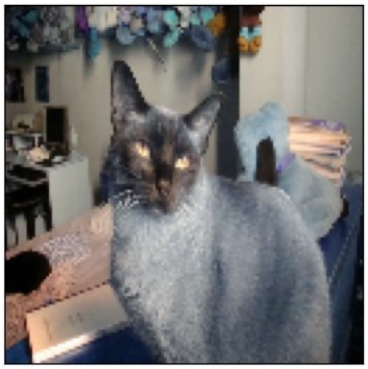
```
# Make predictions
predictions = model.predict(testX)
breeds = ["Abyssinian", "Bengal", "Birman", "Bombay", "British
Shorthair", "Egyptian Mau", "Maine Coon", "Persian", "Ragdoll",
"Russian_Blue", "Siamese", "Sphynx"]
```

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 13s 419ms/step
```

```
# Visualize predictions
plt.figure(figsize=(12, 12))
indices = random.sample(range(len(testX)), 9)
for i, x in enumerate(indices):
    plt.subplot(3, 3, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(testX[x], cmap=plt.cm.binary)
    pred_breed = np.argmax(predictions[x]);
    real_breed = np.argmax(testY[x]);
    color = 'green' if pred_breed == real_breed else 'red'
    plt.xlabel(f" {breeds[real_breed]} ({breeds[pred_breed]})",
color=color)
plt.show()
```

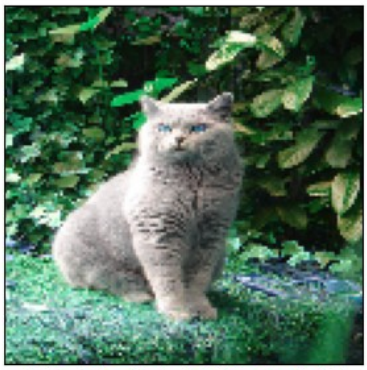


Sphynx (Sphynx)



Siamese (Siamese)

Sphynx (Sphynx)



British Shorthair (British Shorthair)

Siamese (Siamese)

Maine Coon (Bengal)

Russian_Blue (Russian_Blue)

Maine Coon (Bengal)

Ragdoll (Ragdoll)

Висновок:

Під час виконання даної лабораторної роботи було досліджено структуру та принцип роботи згорткових нейронних мереж, а саме було написано програму яка реалізує нейронну мережу Inception V3 для розпізнавання зображень з датасету https://www.kaggle.com/datasets/imbikramsaha/cat-breeds. Мережу було успішно навчено розпізнавати зображеня також її було протестовано та перевірено на тестових даних.