ДОДАТОК А

Вихідний код програми

```
!pip install imutils
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from keras.preprocessing.image import ImageDataGenerator
from keras.applications import VGG19
from keras.layers import AveragePooling2D
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import Input
from keras.models import Model
from keras.optimizers import Adam
from keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from imutils import paths
import shutil
import random
```

```
import cv2
import argparse
import os

device_name = tf.test.gpu_device_name()
if "GPU" not in device_name:
    print("GPU device not found")
print('Found GPU at: {}'.format(device_name))

plt.rcParams['figure.figsize'] = [8.0, 8.0]
plt.rcParams['figure.dpi'] = 100

dataset_path = './dataset'

%%bash
rm -rf dataset
mkdir -p dataset/covid
mkdir -p dataset/normal
mkdir -p dataset/pneumonia
mkdir -p dataset/new

covid_19_path = "../input/covid-chest-xray"
pneumonia_dataset_path ='../input/chest-xray-pneumonia/chest_xray
csvPath = os.path.sep.join([covid_19_path, "metadata.csv"])
df = pd.read_csv(csvPath)

for (i, row) in df.iterrows():
        if row["finding"] != "COVID-19" or row["modality"] != "X-
ray":
            continue
```

```
        imagePath = os.path.sep.join([covid_19_path, "images",
row["filename"]])
        if not os.path.exists(imagePath):
            continue
        filename = row["filename"].split(os.path.sep)[-1]
        outputPath = os.path.sep.join([f"{dataset_path}/covid",
filename])
        shutil.copy2(imagePath, outputPath)

basePath = os.path.sep.join([pneumonia_dataset_path, "train",
"NORMAL"])
imagePaths = list(paths.list_images(basePath))
samples = 253

random.seed(42)
random.shuffle(imagePaths)
imagePaths = imagePaths[:samples]

for (i, imagePath) in enumerate(imagePaths):
        filename = imagePath.split(os.path.sep)[-1]
        outputPath = os.path.sep.join([f"{dataset_path}/normal",
filename])

shutil.copy2(imagePath, outputPath)



basePath = os.path.sep.join([pneumonia_dataset_path, "train",
"PNEUMONIA"])
imagePaths = list(paths.list_images(basePath))
samples = 253

random.seed(42)
```

```
random.shuffle(imagePaths)
imagePaths = imagePaths[:samples]

for (i, imagePath) in enumerate(imagePaths):
    filename = imagePath.split(os.path.sep)[-1]
    outputPath = os.path.sep.join([f"{dataset_path}/pneumonia", filename])

    shutil.copy2(imagePath, outputPath)

def ceildiv(a, b):
    return -(-a // b)

def plots_from_files(imspaths, figsize=(10,5), rows=1, titles=None, maintitle=None):
    """Plot the images in a grid"""
    f = plt.figure(figsize=figsize)
    if maintitle is not None: plt.suptitle(maintitle, fontsize=10)
    for i in range(len(imspaths)):
        sp = f.add_subplot(rows, ceildiv(len(imspaths), rows), i+1)
        sp.axis('Off')
        if titles is not None: sp.set_title(titles[i], fontsize=16)
        img = plt.imread(imspaths[i])
        plt.imshow(img)

normal_images = list(paths.list_images(f"{dataset_path}/normal"))
covid_images = list(paths.list_images(f"{dataset_path}/covid"))
```

```
pneumonia_images =
list(paths.list_images(f"{dataset_path}/pneumonia"))

print("The length of normal_images is: ", len(normal_images))
print("The length of covid_images is: ", len(covid_images))
print("The length of pneumonia_images is: ",
len(pneumonia_images))

INIT_LR = 1e-3
EPOCHS = 100
BS = 8

print("[INFO] loading images...")
imagePaths = list(paths.list_images(dataset_path))
data = []
labels = []
# loop over the image paths
for imagePath in imagePaths:
    # extract the class label from the filename
    label = imagePath.split(os.path.sep)[-2]
    # load the image, swap color channels, and resize it to be a
fixed
    # 224x224 pixels while ignoring aspect ratio
    image = cv2.imread(imagePath)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, (224, 224))
    # update the data and labels lists, respectively
    data.append(image)
    labels.append(label)
```

```
# convert the data and labels to NumPy arrays while scaling the
pixel
# intensities to the range [0, 1]
data = np.array(data) / 255.0
labels = np.array(labels)
print("[INFO] Images successfully loaded")

lb_encoder = LabelEncoder()
labels = lb_encoder.fit_transform(labels)
labels = to_categorical(labels)


# Split the data into training and testing using the 80% of
training and 20% to testing
X_train, X_test, y_train, y_test = train_test_split(data, labels,
test_size = 0.2, stratify = labels,random_state = 42)


# Set the image augmentation of the training data
trainAug = ImageDataGenerator(rotation_range= 15,
fill_mode='nearest')


base_model = VGG19(weights = 'imagenet', include_top=False,
input_tensor=Input(shape=(224,224,3)))

headmodel = base_model.output

headmodel = AveragePooling2D(pool_size =(2, 2))(headmodel)

headmodel = Flatten(name ='Flatten')(headmodel)
```

```
headmodel = Dense(64, activation = 'relu')(headmodel)

headmodel = Dropout(0.5)(headmodel)

headmodel = Dense(3, activation = 'softmax')(headmodel)

model = Model(inputs = base_model.input, outputs = headmodel)

for layers in base_model.layers:

layers.trainable = False

opt = Adam(lr = INIT_LR, decay = INIT_LR/EPOCHS)

model.compile(loss = 'binary_crossentropy', optimizer = opt, metrics
= ['accuracy'])

print("The length of X_train is: ", len(X_train))

print("The length of y_train is: ", len(y_train))

print("The length of X_test is: ", len(X_test))

print("The length of y_test is: ", len(y_test))

with tf.device('/gpu:0'):

print("Training the model with gpu . . .")training =
model.fit_generator(trainAug.flow(X_train, y_train, batch_size =
BS),steps_per_epoch=len(X_train) // BS, validation_data=(X_test,
y_test), validation_steps=len(X_test) // BS, epochs=100)

N = EPOCHS

plt.style.use('ggplot')
```

```
plt.figure()

plt.plot(np.arange(0, N), training.history["loss"], label="train_loss")

plt.plot(np.arange(0, N), training.history["val_loss"], label="val_loss")

plt.plot(np.arange(0, N), training.history["accuracy"], label="train_acc")

plt.plot(np.arange(0, N), training.history["val_accuracy"], label="val_acc")

plt.title("Training Loss and Accuracy for Classification between COVID-19/Pneumonia/Normal")

plt.xlabel("Epoch #")

plt.ylabel("Loss/Accuracy")

plt.legend(loc="upper right", bbox_to_anchor=(1.25, 1))

plt.savefig("plot.png")

print("[INFO] evaluating network...")

predIdxs = model.predict(X_test, batch_size=BS)

# for each image in the testing set we need to find the index of the

# label with corresponding largest predicted probability

predIdxs = np.argmax(predIdxs, axis=1)
```

```
# show a nicely formatted classification report

print(classification_report(y_test.argmax(axis=1),          predIdxs,
target_names=lb_encoder.classes_))



df_cm=          pd.DataFrame(cm,          columns=lb_encoder.classes_,
index=lb_encoder.classes_)

df_cm.index.name = 'Actual'

df_cm.columns.name = 'Predicted'

plt.figure(figsize = (10,7))

sns.heatmap(df_cm/np.sum(df_cm),    fmt='.2%',    annot    =    True,
annot_kws={'size':16})

plt.show()

rows = 10

columns = 9

fig = plt.figure(figsize=(20, 20))

for m in range(1, 88):

    if str(lb_encoder.inverse_transform(predIdxs)[m-1]) == "covid":

        text = "COVID"

        color = (0, 255, 0)
```

```
    elif     str(lb_encoder.inverse_transform(predIdxs)[m-1])     ==
"normal":

        text = "Normal"

        color = (255, 0, 0)

    elif     str(lb_encoder.inverse_transform(predIdxs)[m-1])     ==
"pneumonia":

        text = "Pneumonia"

        color = (0, 0, 255)



    if  str(lb_encoder.inverse_transform(y_test.argmax(axis=1))[m-1])
== "covid":

        text2 = "COVID"

        color2 = (0, 255, 0)

    elif   str(lb_encoder.inverse_transform(y_test.argmax(axis=1))[m-
1]) == "normal":

        text2 = "Normal"

        color2 = (255, 0, 0)

    elif   str(lb_encoder.inverse_transform(y_test.argmax(axis=1))[m-
1]) == "pneumonia":

        text2 = "Pneumonia"
```

```
        color2 = (0, 0, 255)

    img = X_test[m-1].copy()

    # Window name in which image is displayed

    window_name = text

      # font

    font = cv2.FONT_HERSHEY_SIMPLEX

    org = (50, 50)

    # fontScale

    fontScale = 1

    # Line thickness of 2 px

    thickness = 2

    img = cv2.putText(img, text, org, font,

                      fontScale, color, thickness, cv2.LINE_AA)

    fig.add_subplot(rows, columns, m)

    plt.imshow(img)

    plt.title("Actual: " + text2)

    plt.axis('off')

plt.show()
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	77 с.
	Інші документи	
2.	Презентаційні матеріали	26 плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Горішній Д.О			Дослідження штучних нейронних мереж у задачі діагностики респіраторних захворювань за допомогою аналізу рентгенівських знімків	Шифр групи	Код напр./спец.
Перевірив		Терзіян Я.В.				СШІм-19-2	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Філатов В.О.					