## **Державний університет інформаційно-комунікаційних технологій**

**Індивідуальна робота №4**

**з дисципліни: Технології штучного інтелекту**

**Тема: Дослідження та застосування згорткових нейронних мереж VGG-16 та VGG-19 в задачах комп'ютерного зору**

Виконав:  
студент ДУІКТ  
Тертишний В.Ю.

група: ШІДМ-51

м.Київ

**Мета:** Ознайомитися з архітектурою згорткових нейронних мереж VGG-16 і VGG-19. Дослідити особливості та переваги VGG-16 та VGG-19 у задачах комп'ютерного зору. Набути практичних навичок застосування попередньо натренованих моделей VGG-16 та VGG-19 для класифікації зображень. Провести порівняльний аналіз результатів роботи моделей VGG- 16 і VGG-19 на вибраних наборах зображень.

**Код №1:**

import tensorflow as tf

from tensorflow.keras.applications import VGG16

from tensorflow.keras import models, layers, optimizers

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.datasets import cifar10

import numpy as np

# Завантаження попередньо натренованої моделі VGG-16

base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

# Заморожування базової моделі

base\_model.trainable = False

# Створення власної моделі

model = models.Sequential([

base\_model,

layers.Flatten(),

layers.Dense(256, activation='relu'),

layers.Dropout(0.5),

layers.Dense(10, activation='softmax') # 10 класів для CIFAR-10

])

# Компіляція моделі

model.compile(optimizer=optimizers.Adam(learning\_rate=0.001),

loss='categorical\_crossentropy',

metrics=['accuracy'])

# Завантаження даних CIFAR-10

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

# Зміна розміру зображень до (224, 224)

x\_train\_resized = np.array([tf.image.resize(img, (224, 224)).numpy() for img in x\_train])

x\_test\_resized = np.array([tf.image.resize(img, (224, 224)).numpy() for img in x\_test])

# Нормалізація пікселів

x\_train\_resized = x\_train\_resized / 255.0

x\_test\_resized = x\_test\_resized / 255.0

# Підготовка міток

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

# Аугментація даних

datagen = ImageDataGenerator(

rotation\_range=20,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

horizontal\_flip=True

)

datagen.fit(x\_train\_resized)

# Навчання моделі

history = model.fit(datagen.flow(x\_train\_resized, y\_train, batch\_size=32),

epochs=10,

validation\_data=(x\_test\_resized, y\_test))

# Оцінка моделі

test\_loss, test\_acc = model.evaluate(x\_test\_resized, y\_test)

print(f"Точність на тестових даних: {test\_acc}")

# Візуалізація результатів навчання

import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.legend()

plt.show()

**Код №2:**

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.applications import VGG19

from tensorflow.keras.models import Model

from tensorflow.keras.preprocessing.image import load\_img, img\_to\_array

# Завантаження VGG-19

vgg = VGG19(weights="imagenet", include\_top=False)

# Вибір шарів для виділення стилю та змісту

content\_layer = 'block5\_conv2'

style\_layers = ['block1\_conv1', 'block2\_conv1', 'block3\_conv1', 'block4\_conv1', 'block5\_conv1']

# Функція для виділення активацій

def get\_model\_outputs(model, style\_layers, content\_layer):

outputs = [model.get\_layer(name).output for name in style\_layers + [content\_layer]]

return Model(inputs=model.input, outputs=outputs)

model = get\_model\_outputs(vgg, style\_layers, content\_layer)

model.trainable = False

# Завантаження зображень

def load\_and\_process\_img(path\_to\_img):

img = load\_img(path\_to\_img, target\_size=(224, 224))

img = img\_to\_array(img)

img = tf.keras.applications.vgg19.preprocess\_input(img)

return np.expand\_dims(img, axis=0)

content\_image = load\_and\_process\_img('ip600x800.jpg') # Вкажіть шлях до зображення змісту

style\_image = load\_and\_process\_img('2ip600x800.jpg') # Вкажіть шлях до зображення стилю

# Функції для розрахунку втрат

def gram\_matrix(input\_tensor):

channels = int(input\_tensor.shape[-1])

vectorized = tf.reshape(input\_tensor, [-1, channels])

return tf.matmul(tf.transpose(vectorized), vectorized)

def compute\_loss(model, content\_image, style\_image, generated\_image, style\_weight=1e-2, content\_weight=1e4):

outputs = model(generated\_image)

style\_outputs, content\_output = outputs[:len(style\_layers)], outputs[len(style\_layers):]

# Втрата стилю

style\_loss = 0

for style\_output, style\_target in zip(style\_outputs, model(style\_image)[:len(style\_layers)]):

style\_loss += tf.reduce\_mean((gram\_matrix(style\_output) - gram\_matrix(style\_target)) \*\* 2)

style\_loss /= len(style\_layers)

# Втрата змісту

content\_loss = tf.reduce\_mean((content\_output - model(content\_image)[-1]) \*\* 2)

# Загальна втрата

total\_loss = style\_weight \* style\_loss + content\_weight \* content\_loss

return total\_loss

# Оптимізація

generated\_image = tf.Variable(content\_image, dtype=tf.float32)

optimizer = tf.optimizers.Adam(learning\_rate=0.02)

@tf.function

def train\_step(image):

with tf.GradientTape() as tape:

loss = compute\_loss(model, content\_image, style\_image, image)

gradients = tape.gradient(loss, image)

optimizer.apply\_gradients([(gradients, image)])

image.assign(tf.clip\_by\_value(image, 0, 255))

# Тренування

epochs = 100

for epoch in range(epochs):

train\_step(generated\_image)

if epoch % 10 == 0:

print(f"Епоха {epoch}, Втрата: {compute\_loss(model, content\_image, style\_image, generated\_image).numpy()}")

# Візуалізація результату

plt.imshow(np.squeeze(generated\_image.numpy().astype('uint8')))

plt.axis('off')

plt.show()

**Робота коду №2:**  
(**>Скріни сюди<**)

**Висновки**

В процесі дослідження архітектур згорткових нейронних мереж VGG-16 і VGG-19 було розглянуто їхні ключові особливості та переваги при вирішенні задач комп'ютерного зору. Моделі VGG-16 і VGG-19 мають схожу структуру, з відмінністю в кількості шарів, де VGG-19 містить на один додатковий шар, що може впливати на здатність моделі до виявлення більш складних патернів у зображеннях. Оскільки ці мережі широко використовуються для класифікації зображень, вони демонструють високий рівень точності при застосуванні на різноманітних наборах зображень. Практичне застосування попередньо натренованих моделей показало, що для деяких типів зображень VGG-19 може показувати кращі результати, завдяки більшій глибині мережі, але й більша складність моделі може призвести до більшої потреби в обчислювальних ресурсах. Порівняльний аналіз результатів показав, що обидві моделі є потужними інструментами для класифікації зображень, і вибір між ними може залежати від вимог до точності та ресурсів, доступних для обробки даних.