**Міністерство освіти і науки України**

**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

# Звіт

Лабораторна робота № 6 з дисципліни

«Штучний інтелект в задачах обробки зображень»

**«Реалізація архітектури AlexNet CNN за допомогою TensorFlow і Keras»**

**Виконав:**

*ІП-01 Черпак А. В.*

(шифр, прізвище, ім'я, по батькові)

**Перевірив:**

*Нікітін В. А.*

(прізвище, ім'я, по батькові)

Київ 2022

# Мета:

# Отримати навички реалізації архітектури AlexNet CNN з використанням бібліотек TensorFlow та Keras.

# Завдання

1. Реалізувати засобами TensorFlow та Keras AlexNet;
2. Отримати оцінку точності навченої мережі.

# Хід роботи

1. Методи для завантаження та обробка даних:

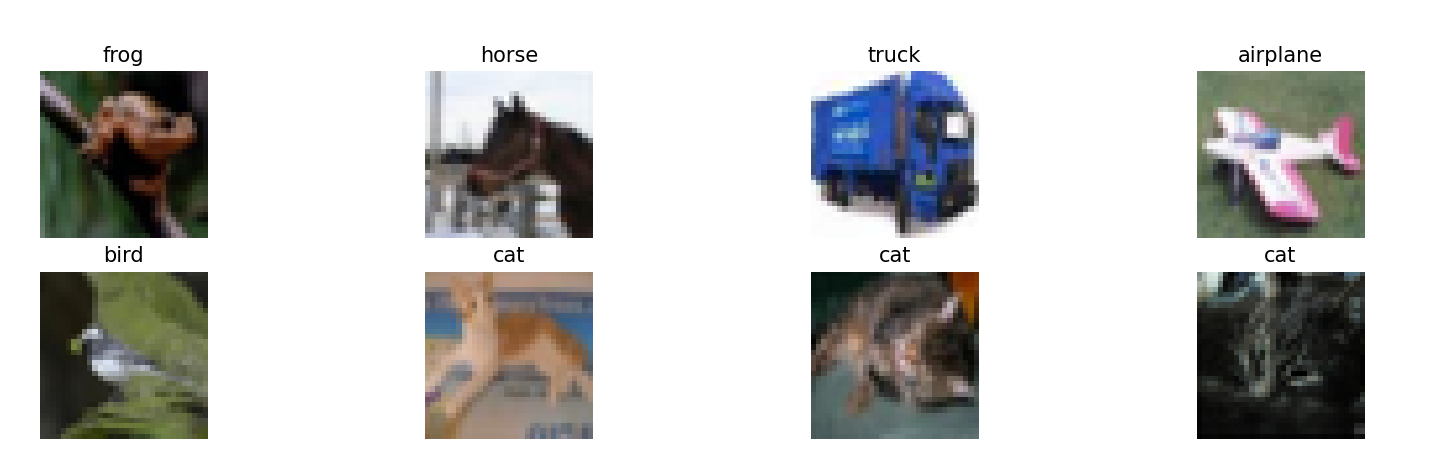
import tensorflow as tf  
from tensorflow import keras  
import matplotlib.pyplot as plt  
from tensorflow.python.data.ops.dataset\_ops import DatasetV1  
from numpy import ndarray  
  
  
def get\_data() -> tuple[DatasetV1, DatasetV1, DatasetV1]:  
 (train\_images, train\_labels), (test\_images, test\_labels) = keras.datasets.cifar10.load\_data()  
  
 validation\_images, validation\_labels = train\_images[:5000], train\_labels[:5000]  
 train\_images, train\_labels = train\_images[5000:], train\_labels[5000:]  
  
 train\_ds = tf.data.Dataset.from\_tensor\_slices((train\_images, train\_labels))  
 test\_ds = tf.data.Dataset.from\_tensor\_slices((test\_images, test\_labels))  
 validation\_ds = tf.data.Dataset.from\_tensor\_slices((validation\_images, validation\_labels))  
 return train\_ds, test\_ds, validation\_ds  
  
  
def visualize\_data(train\_ds: DatasetV1, classes: list[str]) -> None:  
 plt.figure(figsize=(20, 20))  
 for i, (image, label) in enumerate(train\_ds.take(10)):  
 ax = plt.subplot(5, 5, i + 1)  
 plt.imshow(image)  
 plt.title(classes[label.numpy()[0]])  
 plt.axis('off')  
  
 plt.show()  
  
  
def process\_images(image: ndarray, label: str) -> tuple[ndarray, str]:  
 # Normalize images to have a mean of 0 and standard deviation of 1  
 image = tf.image.per\_image\_standardization(image)  
 # Resize images from 32x32 to 277x277  
 image = tf.image.resize(image, (227, 227))  
 return image, label  
  
  
def get\_ds\_size(ds: DatasetV1) -> int:  
 return tf.data.experimental.cardinality(ds).numpy()  
  
  
def process\_ds(ds: DatasetV1) -> DatasetV1:  
 return ds.map(process\_images).shuffle(buffer\_size=get\_ds\_size(ds)).batch(batch\_size=32, drop\_remainder=True)

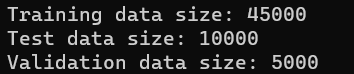
1. Створення та компіляція моделі:

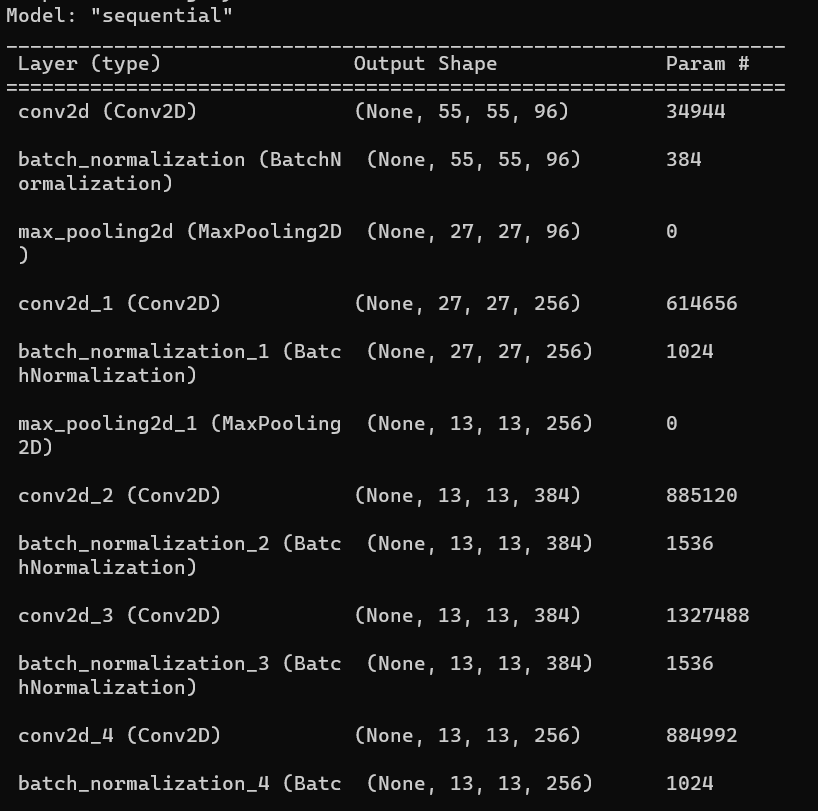
import tensorflow as tf  
from keras.layers import Conv2D, BatchNormalization, MaxPool2D, Flatten, Dense, Dropout  
from keras.models import Sequential  
  
  
def AlexNet(class\_number) -> Sequential:  
 model = Sequential([  
 Conv2D(filters=96, kernel\_size=(11, 11), strides=(4, 4),  
 activation='relu', input\_shape=(227, 227, 3)),  
 BatchNormalization(),  
 MaxPool2D(pool\_size=(3, 3), strides=(2, 2)),  
 Conv2D(filters=256, kernel\_size=(5, 5), strides=(1, 1),  
 activation='relu', padding='same'),  
 BatchNormalization(),  
 MaxPool2D(pool\_size=(3, 3), strides=(2, 2)),  
 Conv2D(filters=384, kernel\_size=(3, 3), strides=(1, 1),  
 activation='relu', padding='same'),  
 BatchNormalization(),  
 Conv2D(filters=384, kernel\_size=(3, 3), strides=(1, 1),  
 activation='relu', padding='same'),  
 BatchNormalization(),  
 Conv2D(filters=256, kernel\_size=(3, 3), strides=(1, 1),  
 activation='relu', padding='same'),  
 BatchNormalization(),  
 MaxPool2D(pool\_size=(3, 3), strides=(2, 2)),  
 Flatten(),  
 Dense(4096, activation='relu'),  
 Dropout(0.5),  
 Dense(4096, activation='relu'),  
 Dropout(0.5),  
 Dense(class\_number, activation='softmax')  
 ])  
  
 return model  
  
  
def compile\_model(model: Sequential):  
 model.compile(  
 loss='sparse\_categorical\_crossentropy',  
 optimizer=tf.optimizers.SGD(lr=0.001),  
 metrics=['accuracy']  
 )  
 model.summary()  
 return model

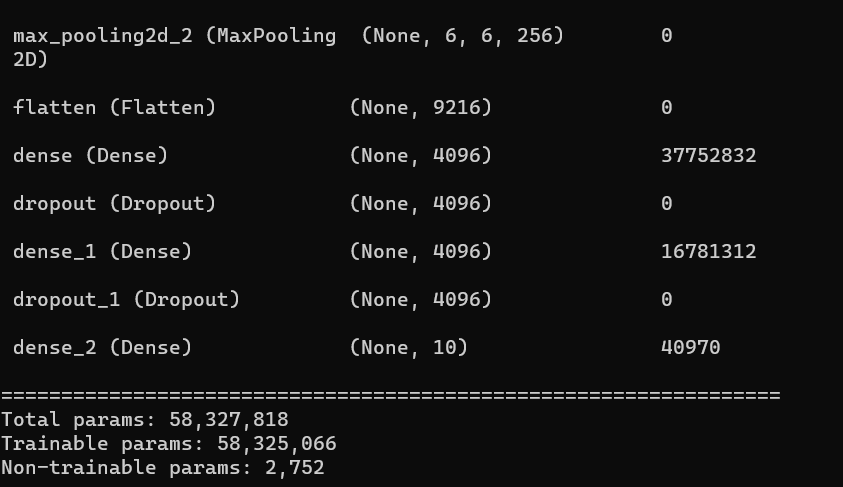
1. Тренування та оцінювання моделі:

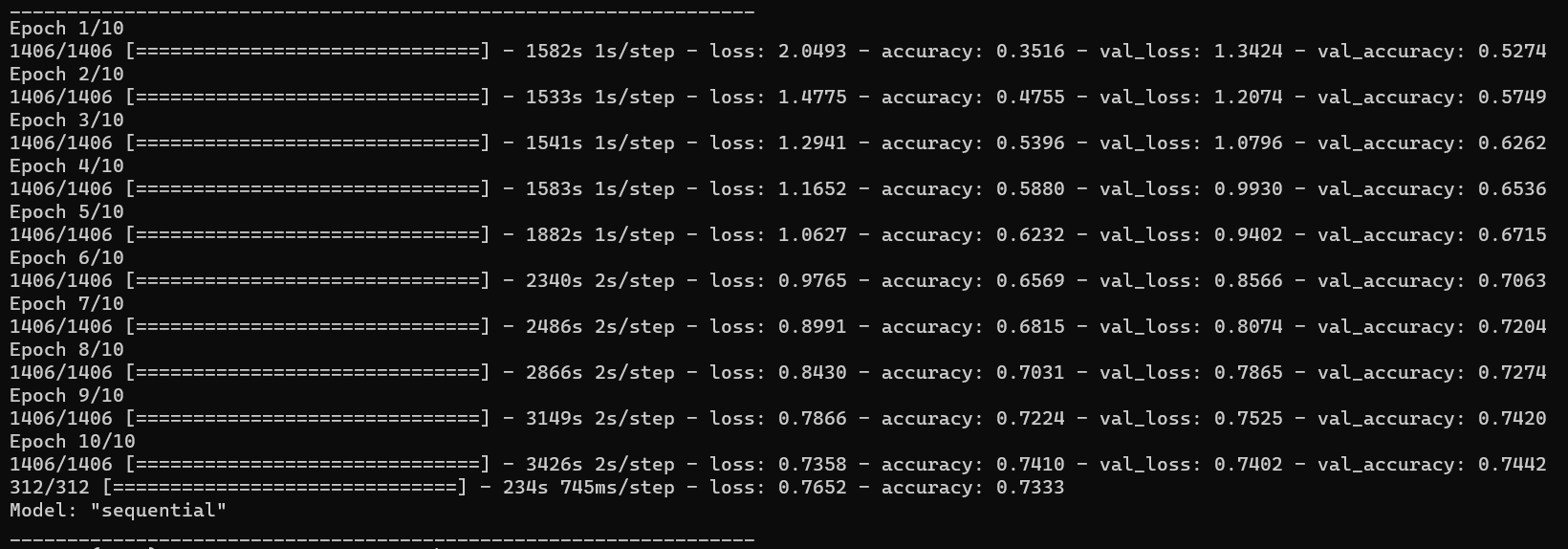
from get\_data import get\_data, process\_ds, get\_ds\_size, visualize\_data  
from nn\_model import AlexNet, compile\_model  
  
CLASS\_NAMES = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 train\_ds, test\_ds, validation\_ds = get\_data()  
 visualize\_data(train\_ds, CLASS\_NAMES)  
  
 train\_ds\_size = get\_ds\_size(train\_ds)  
 test\_ds\_size = get\_ds\_size(test\_ds)  
 validation\_ds\_size = get\_ds\_size(validation\_ds)  
  
 print("Training data size:", train\_ds\_size)  
 print("Test data size:", test\_ds\_size)  
 print("Validation data size:", validation\_ds\_size)  
  
 train\_ds = process\_ds(train\_ds)  
 test\_ds = process\_ds(test\_ds)  
 validation\_ds = process\_ds(validation\_ds)  
  
 mdl = AlexNet(len(CLASS\_NAMES))  
 model = compile\_model(mdl)  
  
 model.fit(train\_ds, epochs=10, validation\_data=validation\_ds, validation\_freq=1)  
 model.evaluate(test\_ds)  
 model.summary()  
  
 model.save('alexNet.h5')











Результати оцінювання:



# Контрольні запитання

1. AlexNet – це згорткова нейронна мережа для розпізнавання зображень, що була розроблена ще у 2012 році, але й досі задає стандарти у технологіях комп’ютерного зору.
2. Дана мережа складається з 5 згорткових і 3 повнозв’язних шарів. Між повнозв’язними шарами також розташовані Dropout, що дозволяють боротися з проблемою перенавчання, хоч і збільшують час навчання моделі. Мережа містить 58,327,818 параметрів (у моїй реалізації, в оригінальній, як зазначається, було 62,3м), і застосовує більше мільярда обчислень при прямому проході.
3. AlexNet вважається віхою загорткових нейронних мереж. Технології, що використовуються в ньому, досі задають стандарт для подібних мереж. Також дана модель дозволяє пряме введення зображення, а шари згортки можуть виділити краї зображень. Шляхом додавання більшої кількості загорткових шарів, можна досягти розпізнавання візуальних шаблонів більшої складності.

Втім, дана модель менш глибока та недостатньо продуктивна порівняно з пізнішими моделями. Крім того, для ініціювання вагових коефіцієнтів використовується нормальний розподіл, що не дозволяє ефективно вирішувати проблему зникнення градієнта.

1. Під час компіляції мережі відбувається налаштування таких характеристик, як алгоритм оптимізації, функція втрат та швидкість навчання.
2. Під час навчання нейронної мережі ми передаємо у неї певні вхідні дані та очікуваний результат. Нейромережа спершу намагається передбачити результат для введених даних, а потім порівнює його з очікуваним результатом і коригує свої вагові коефіцієнти за допомогою методу зворотнього поширення помилки. Звісно, чим більшу кількість унікальних зображень було розглянуто мережею у процесі навчання, і чим більшою була кількість епох навчання – тим більш точною виявиться отримана мережа. Втім, варто також контролювати точність мережі на тестових даних, які не були безпосередньо розглянуті мережею у процесі навчання. Якщо точність з часом починає падати – ми, ймовірно, зіткнулися з таким явищем, як перенавчання, коли нейромережа «звикла» до таких вхідних даних і інші вже не сприймає. В такому випадку варто припинити навчання на цьому датасеті і спробувати потренувати мережу на нових даних.