Лабораторна робота № 12. Побудова та оптимізація нейронних мереж для класифікації зображень

Мета: набуття практичних навичок у побудові та оптимізації згорткових нейронних мереж (Convolutional Neural Networks, CNNs) для задач класифікації зображень. Здобувачі ознайомляться з основними концепціями CNN, навчаться реалізовувати архітектури нейронних мереж у середовищі Google Colab з використанням бібліотеки Keras (або TensorFlow), а також досліджуватимуть вплив різних гіперпараметрів та методів оптимізації на продуктивність моделі. Окрім цього, метою є розвиток навичок командної роботи та розподілу завдань між членами команди.

Теоретичні відомості

Згорткові нейронні мережі (CNN) є особливим класом глибоких нейронних мереж, які продемонстрували виняткову ефективність у задачах обробки зображень, включаючи класифікацію. Основними компонентами CNN є:

- **Згорткові шари (Convolutional Layers)** – шари, які застосовують до вхідного зображення набір фільтрів (ядер згортки), кожен з яких виділяє певну ознаку, наприклад, краї, кути або текстури. Результатом роботи згорткового шару є карта ознак (feature map), яка відображає активації відповідного фільтра на різних ділянках зображення. Важливими гіперпараметрами згорткового шару є розмір ядра (kernel size), кількість фільтрів (number of filters), крок згортки (stride) та доповнення (padding).
- **Шари пулінгу (Pooling Layers)** – шари зменшують просторову розмірність карт ознак, тим самим зменшуючи обчислювальну складність і ризик перенавчання. Найбільш поширеними типами пулінгу є max-pooling та average-pooling. Max-pooling вибирає максимальне значення з кожного підвікна, а average-pooling обчислює середнє значення. Важливими гіперпараметрами шару пулінгу є розмір підвікна (pool size) та крок (stride).
- **Шари повнозв’язного шару (Fully Connected Layers).** Після декількох згорткових та пулінгових шарів карта ознак перетворюється на одновимірний вектор, який подається на вхід повнозв'язного шару. Повнозв’язні шари аналогічні шарам у звичайних нейронних мережах і виконують класифікацію на основі виділених ознак.

- **Функція активації (Activation Function).** Кожен нейрон у мережі має функцію активації, яка вносить нелінійність у модель, дозволяючи їй вивчати складні залежності між даними. Популярними функціями активації є ReLU (Rectified Linear Unit), Sigmoid та Tanh. ReLU є особливо ефективною для згорткових шарів, оскільки вона вирішує проблему зникаючого градієнта, що часто виникає при навчанні глибоких мереж.
- **Функція втрат (Loss Function).** Функція втрат вимірює розбіжність між передбаченнями моделі та істинними мітками. Для задач класифікації зазвичай використовують категоріальну перехресну ентропію (categorical cross-entropy).
- **Оптимізатор (Optimizer).** Оптимізатор відповідає за оновлення ваг мережі під час навчання з метою мінімізації функції втрат. Популярними оптимізаторами є стохастичний градієнтний спуск (Stochastic Gradient Descent, SGD), Adam та RMSprop.

Оптимізація нейронних мереж:

Для досягнення високої точності класифікації необхідно оптимізувати архітектуру та гіперпараметри нейронної мережі:

- **Підбір архітектури.** Вибір кількості та типів шарів, розміру ядер згортки, кількості фільтрів, розміру пулінгу та інших параметрів.
- **Регуляризація.** Застосування технік регуляризації, таких як L1 та L2 регуляризація, dropout та батч-нормалізація, для запобігання перенавчанню.
- **Аугментація даних.** Штучне розширення навчального набору даних шляхом застосування різних перетворень до зображень, таких як повороти, зсуви, масштабування та відображення, що допомагає підвищити узагальнюючу здатність моделі.
- **Підбір швидкості навчання.** Швидкість навчання (learning rate) є важливим гіперпараметром, який визначає, наскільки сильно оновлюються ваги мережі на кожному кроці оптимізації. Занадто велика швидкість навчання може призвести до нестабільності процесу навчання, а занадто мала – до повільної збіжності. Часто використовують техніки адаптивного підбору швидкості навчання, такі як Adam та RMSprop.
- **Рання зупинка (Early Stopping).** Моніторинг продуктивності моделі на валідаційному наборі даних та припинення навчання, коли продуктивність перестає покращуватися.

Приклад виконання роботи

Завдання

Побудувати та навчити згорткову нейронну мережу для класифікації зображень з набору даних CIFAR-10. Набір CIFAR-10 містить 60000 кольорових зображень розміром 32x32 пікселів, розділених на 10 класів (літак, автомобіль, птах, кіт, олень, собака, жаба, кінь, корабель, вантажівка). 50000 зображень використовуються для навчання, а 10000 – для тестування.

1. Завантажити та попередньо обробити дані CIFAR-10.
2. Визначити архітектуру згорткової нейронної мережі.
3. Скомпілювати модель, вибравши функцію втрат, оптимізатор та метрики.
4. Навчити модель на навчальному наборі даних, використовуючи аугментацію даних.
5. Оцінити продуктивність моделі на тестовому наборі даних.
6. Провести експерименти з різними гіперпараметрами та методами оптимізації та проаналізувати результати.

Контрольне запитання

Які фактори впливають на вибір архітектури згорткової нейронної мережі та як вони впливають на її продуктивність?
Відповідь: _____

Хід роботи

1. Завантаження та попередня обробка даних.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator

(x_train, y_train), (x_test, y_test) = cifar10.load_data()

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)

print(f"Розмір навчального набору даних: {x_train.shape}")
print(f"Розмір тестового набору даних: {x_test.shape}")
```

У цьому коді ми спочатку імпортуємо необхідні бібліотеки, такі як TensorFlow, Keras та ImageDataGenerator. Потім ми завантажуємо набір даних CIFAR-10, використовуючи функцію `cifar10.load_data()`. Після цього ми нормалізуємо значення пікселів зображень, ділячи їх на 255.0, щоб вони знаходилися в діапазоні від 0 до 1. Нарешті, ми перетворюємо мітки класів у формат one-hot encoding, використовуючи функцію `to_categorical()`.

2. Визначення архітектури згорткової нейронної мережі.

```
model = keras.Sequential(
    [
        layers.Conv2D(32, (3, 3), activation='relu', padding='same',
input_shape=(32, 32, 3)),
        layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
```

```
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(10, activation='softmax')
    ]
)

model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9,248
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	18,496
conv2d_3 (Conv2D)	(None, 16, 16, 64)	36,928
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_4 (Conv2D)	(None, 8, 8, 128)	73,856
conv2d_5 (Conv2D)	(None, 8, 8, 128)	147,584
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 128)	262,272
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

Total params: 550,570 (2.10 MB)
Trainable params: 550,570 (2.10 MB)
Non-trainable params: 0 (0.00 B)

Рисунок 1.1. Результат виконання завдання №2

Тут ми визначаємо архітектуру згорткової нейронної мережі, використовуючи клас `keras.Sequential`. Модель складається з декількох згорткових шарів, шарів пулінгу, шару вирівнювання, повнозв'язного шару та вихідного шару. Згорткові шари використовують функцію активації ReLU та доповнення 'same', щоб зберегти просторову розмірність. Шари пулінгу використовують max-pooling з розміром вікна 2x2. Шар вирівнювання перетворює вихід з останнього шару пулінгу на одновимірний вектор.

Повнозв'язний шар має 128 нейронів та функцію активації ReLU. Для запобігання перенавчанню додано шар Dropout з коефіцієнтом 0.5. Вихідний шар має 10 нейронів (по одному для кожного класу) та функцію активації softmax. Функція `model.summary()` виводить детальну інформацію про архітектуру моделі.

3. Компіляція моделі.

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

На цьому етапі ми компілюємо модель, вказуючи оптимізатор, функцію втрат та метрики. Ми використовуємо оптимізатор Adam, функцію втрат категоріальної перехресної ентропії та метрику точності (accuracy).

4. Навчання моделі з аугментацією даних.

```
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
)

datagen.fit(x_train)

batch_size = 64
epochs = 50

history = model.fit(datagen.flow(x_train, y_train, batch_size=batch_size),
                    epochs=epochs,
                    validation_data=(x_test, y_test),
                    steps_per_epoch=x_train.shape[0] // batch_size)
```

```
Epoch 46/50
781/781 ━━━━━━━━━━━━━━━━━━━━ 1s 830us/step - accuracy: 0.7031 - loss: 0.7214 - val_accuracy: 0.8022 - val_loss: 0.6127
Epoch 47/50
781/781 ━━━━━━━━━━━━━━━━━━━━ 38s 49ms/step - accuracy: 0.8096 - loss: 0.5654 - val_accuracy: 0.8122 - val_loss: 0.5911
Epoch 48/50
781/781 ━━━━━━━━━━━━━━━━━━━━ 1s 829us/step - accuracy: 0.8594 - loss: 0.4058 - val_accuracy: 0.8149 - val_loss: 0.5850
Epoch 49/50
781/781 ━━━━━━━━━━━━━━━━━━━━ 43s 52ms/step - accuracy: 0.8150 - loss: 0.5569 - val_accuracy: 0.7964 - val_loss: 0.6476
Epoch 50/50
781/781 ━━━━━━━━━━━━━━━━━━━━ 1s 807us/step - accuracy: 0.7812 - loss: 0.5613 - val_accuracy: 0.7956 - val_loss: 0.6507
```

Рисунок 1.3. Результат виконання завдання №3

Тут ми використовуємо клас `ImageDataGenerator` для аугментації даних. Ми застосовуємо випадкові повороти, зсуви та горизонтальні відображення до зображень. Потім ми навчаємо модель, використовуючи метод `model.fit()`, передаючи йому генератор даних `datagen.flow()`, кількість епох, валідаційний набір даних та кількість кроків на епоху. Збільшена кількість епох до 50 дозволяє моделі краще вивчити залежності в даних.

5. Оцінка продуктивності моделі.

```
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print(f"Test loss: {loss:.4f}")
print(f"Test accuracy: {accuracy:.4f}")
```

```
Test loss: 0.6507
Test accuracy: 0.7956
```

Рисунок 1.4. Результат виконання завдання №4

Ми оцінюємо продуктивність навченої моделі на тестовому наборі даних, використовуючи метод `model.evaluate()`. Результати виводяться на екран.

6. Експерименти з гіперпараметрами та аналіз результатів.

На цьому етапі є можливість поекспериментувати з різними гіперпараметрами моделі, такими як кількість згорткових шарів, кількість фільтрів, розмір ядра, швидкість навчання, тип оптимізатора тощо. Також можна спробувати різні методи регуляризації, такі як L1/L2 регуляризація та батч-нормалізація.

Наприклад, змінимо оптимізатор на RMSprop та зменшимо швидкість навчання:

```
model_rmsprop = keras.Sequential(
    [
        layers.Conv2D(32, (3, 3), activation='relu', padding='same',
input_shape=(32, 32, 3)),
        layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
```

```
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(10, activation='softmax')
    ]
)

model_rmsprop.compile(optimizer=keras.optimizers.RMSprop(learning_rate=0.0005), # Змінений оптимізатор
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history_rmsprop = model_rmsprop.fit(datagen.flow(x_train, y_train, batch_size=batch_size),
                    epochs=epochs,
                    validation_data=(x_test, y_test),
                    steps_per_epoch=x_train.shape[0] // batch_size)

loss_rmsprop, accuracy_rmsprop = model_rmsprop.evaluate(x_test, y_test, verbose=0)
print(f"Test loss (RMSprop): {loss_rmsprop:.4f}")
print(f"Test accuracy (RMSprop): {accuracy_rmsprop:.4f}")
```

```
Epoch 49/50
781/781 ━━━━━━━━━━━━━━━━━━━━ 37s 47ms/step - accuracy: 0.8000 - loss: 0.6356 - val_accuracy: 0.7982 - val_loss: 0.6543
Epoch 50/50
781/781 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8750 - loss: 0.4678 - val_accuracy: 0.8059 - val_loss: 0.6477
Test loss (RMSprop): 0.6477
Test accuracy (RMSprop): 0.8059
```

Рисунок 1.5. Результат виконання завдання №5

В данному прикладі, ми визначили нову модель `model_rmsprop` з ідентичною архітектурою попередній моделі, але змінили оптимізатор на RMSprop. Також ми встановили швидкість навчання на рівні 0.0005, що є меншим за замовчування значення для Adam. Зменшення швидкості навчання може допомогти покращити стабільність процесу навчання та потенційно досягти кращої точності.

7. Аналіз результатів.

Після проведення експериментів з різними гіперпараметрами та методами оптимізації необхідно проаналізувати отримані результати. Слід звернути увагу на те, як змінилися показники точності та втрат на тестовому наборі даних. Також корисно побудувати графіки залежності точності та втрат від кількості епох навчання для різних конфігурацій

моделі, що дозволить візуалізувати процес навчання та виявити можливі проблеми, такі як перенавчання або недонавчання.

Наприклад, порівняємо графіки навчання для моделі з оптимізатором Adam та моделі з оптимізатором RMSprop та зменшеною швидкістю навчання.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Adam: Accuracy vs. Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Adam: Loss vs. Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history_rmsprop.history['accuracy'], label='Train Accuracy')
plt.plot(history_rmsprop.history['val_accuracy'], label='Validation
Accuracy')
plt.title('RMSprop: Accuracy vs. Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history_rmsprop.history['loss'], label='Train Loss')
plt.plot(history_rmsprop.history['val_loss'], label='Validation Loss')
plt.title('RMSprop: Loss vs. Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

```
plt.legend()

plt.tight_layout()
plt.show()
```

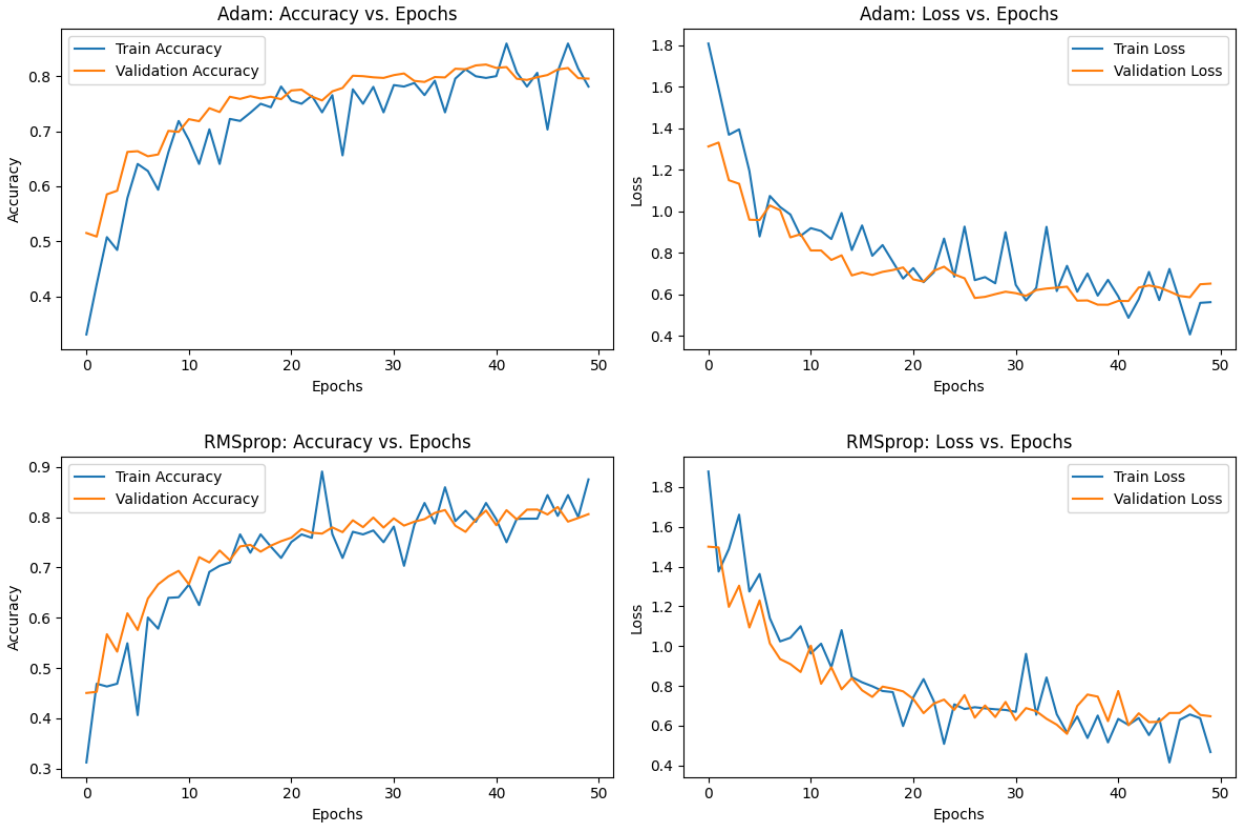

Рисунок 1.6. Результат виконання завдання №6

Цей код генерує графіки, які відображають зміну точності (accuracy) та функції втрат (loss) протягом епох навчання для обох моделей (з оптимізаторами Adam та RMSprop) і дозволяє візуально порівняти їхню продуктивність та стабільність процесу навчання.

Порівнюючи графіки та числові значення точності та втрат, Ви повинні зробити висновки щодо впливу різних гіперпараметрів та методів оптимізації на продуктивність моделі. Наприклад, Ви можете виявити, що модель з оптимізатором RMSprop та зменшеною швидкістю навчання досягає кращої точності на тестовому наборі даних та демонструє більш стабільний процес навчання порівняно з моделлю з оптимізатором Adam.

Важливі аспекти для обговорення:

- **Вплив кількості згорткових шарів та фільтрів.** Як змінюється продуктивність моделі при збільшенні або зменшенні кількості згорткових шарів та фільтрів? Чи

спостерігається перенавчання або недонавчання?

- **Вплив розміру ядра згортки.** Як впливає розмір ядра згортки на здатність моделі виділяти різні ознаки?
- **Вплив швидкості навчання.** Як впливає швидкість навчання на швидкість збіжності та стабільність процесу навчання?
- **Вплив методів регуляризації.** Як впливають методи регуляризації (dropout, L1/L2 регуляризація, батч-нормалізація) на здатність моделі до узагальнення?
- **Вплив аугментації даних.** Як впливає аугментація даних на продуктивність моделі та її здатність до узагальнення?
- **Вибір оптимізатора.** Який оптимізатор (Adam, RMSprop, SGD тощо) є найбільш ефективним для даної задачі та архітектури?

Контрольне запитання

Які фактори впливають на вибір архітектури згорткової нейронної мережі та як вони впливають на її продуктивність?
Відповідь: _____

Висновки: …

Завдання для виконання лабораторної роботи

Кожній команді (2-3 здобувачі) необхідно розробити та навчити згорткову нейронну мережу для класифікації зображень з набору даних CIFAR-100. Набір CIFAR-100 схожий на CIFAR-10, але містить 100 класів замість 10. Кожен клас містить 600 зображень. 500 зображень використовуються для навчання, а 100 – для тестування. Класи згруповані в 20 суперкласів. Кожне зображення має мітку «fine» (клас, до якого воно належить) та мітку «coarse» (суперклас, до якого воно належить).

Розподіл завдань між членами команди (приклад):

- **Здобувач 1.** Відповідає за завантаження та попередню обробку даних CIFAR-100, включаючи нормалізацію та аугментацію даних. Досліджує різні методи аугментації та їх вплив на продуктивність.
- **Здобувач 2.** Відповідає за розробку архітектури згорткової нейронної мережі. Експериментує з різною кількістю згорткових та повнозв'язних шарів, кількістю фільтрів, розмірами ядер та функціями активації.
- **Здобувач 3.** Відповідає за навчання та оптимізацію моделі. Експериментує з різними оптимізаторами, швидкостями навчання та методами регуляризації. Використовує техніку ранньої зупинки.

Завдання по пунктам:

1. Завантаження та попередня обробка даних (**Здобувач 1**):

1.1. Завантажити набір даних CIFAR-100.

1.2. Нормалізувати дані, поділивши значення пікселів на 255.0.

1.3. Перетворити мітки класів у формат one-hot encoding.

1.4. Розділити дані на навчальний, валідаційний та тестовий набори (наприклад, 80%, 10%, 10%).

1.5. Реалізувати аугментацію даних, використовуючи `ImageDataGenerator`. Дослідити різні параметри аугментації, такі як повороти, зсуви, масштабування, відображення, зміна яскравості та контрастності.

2. Розробка архітектури нейронної мережі (**Здобувач 2**):

2.1. Створити базову архітектуру згорткової нейронної мережі, використовуючи шари

`Conv2D`, `MaxPooling2D`, `Flatten`, `Dense` та `Dropout`.

2.2. Експериментувати з різною кількістю згорткових шарів (наприклад, від 2 до 5).

2.3. Експериментувати з різною кількістю фільтрів у кожному згортковому шарі (наприклад, від 32 до 128).

2.4. Експериментувати з різними розмірами ядер згортки (наприклад, 3x3, 5x5).

2.5. Експериментувати з різними функціями активації (ReLU, Leaky ReLU, ELU).

2.6. Використати шар `Dropout` після одного або декількох повнозв'язних шарів для запобігання перенавчанню.

2.7. Дослідити можливість використання батч-нормалізації.

2.8. Спробувати різні конфігурації повнозв'язних шарів (кількість шарів та кількість нейронів).

3. Навчання та оптимізація моделі (**Здобувач 3**):

3.1. Скомпілювати модель, вибравши оптимізатор (Adam, RMSprop, SGD), функцію втрат (categorical cross-entropy) та метрики (accuracy).

3.2. Навчити модель на навчальному наборі даних, використовуючи аугментацію даних та валідаційний набір.

3.3. Експериментувати з різними швидкостями навчання (наприклад, 0.001, 0.0001).

3.4. Використовувати техніку ранньої зупинки, щоб запобігти перенавчанню.

3.5. Дослідити вплив L1/L2 регуляризації на продуктивність моделі.

3.6. Зберегти найкращу модель на основі точності на валідаційному наборі.

4. Оцінка та аналіз результатів (**Вся команда**):

4.1. Оцінити продуктивність найкращої моделі на тестовому наборі даних.

4.2. Побудувати матрицю невідповідностей (confusion matrix) для аналізу помилок класифікації.

4.3. Побудувати графіки залежності точності та втрат від кількості епох навчання для різних конфігурацій моделі.

4.4. Проаналізувати результати та зробити висновки щодо впливу різних гіперпараметрів та методів оптимізації на продуктивність моделі.

4.5. Обговорити, які класи або суперкласи розпізнаються найкраще, а які найгірше, та чому.

4.6. Запропонувати можливі шляхи покращення продуктивності моделі.

Варіанти для виконання завдання

За бажанням Ви маєте можливість експерементувати будь-яким іншим чином (що не вказаний в межах завдання), задля досягнення кращих результатів.

№	Кількість	Кількість фільтрів	Розмір ядра	Функція активації	Оптимізатор	Швидкість навчання	Регуляризація	Аугментація
1	3	32, 64, 128	3x3	ReLU	Adam	0.001	Dropout (0.3)	Повороти, зсуви, горизонтальне відображення
2	4	16, 32, 64, 128	3x3	Leaky ReLU	RMSprop	0.0005	L2 (0.001)	Повороти, зсуви, масштабування
3	2	64, 128	5x5	ELU	SGD	0.01	Dropout (0.5), Batch Normalization	Повороти, зсуви, зміна яскравості
4	3	32, 64, 128	3x3	ReLU	Adam	0.0001	Dropout (0.4)	Зсуви, горизонтальне та вертикальне відображення
5	5	16, 32, 64, 64, 128	3x3	ReLU	Adam	0.001	L1 (0.0005)	Повороти, масштабування, зміна контрастності
6	2	32, 64	5x5	Leaky ReLU	RMSprop	0.0005	Batch Normalization	Горизонтальне відображення, зміна яскравості та контрастності
7	4	32, 32, 64, 64	3x3	ELU	SGD	0.01	Dropout (0.25)	Повороти, зсуви, масштабування, горизонтальне відображення
8	3	64, 128, 256	3x3	ReLU	Adam	0.0001	L2 (0.0001)	Зсуви, масштабування, зміна яскравості
9	4	16, 32, 64, 128	5x5	Leaky ReLU	RMSprop	0.001	Dropout (0.5)	Повороти, зсуви, горизонтальне та вертикальне відображення
10	2	64, 128	3x3	ReLU	Adam	0.005	L1 (0.001)	Масштабування, зміна яскравості та контрастності
11	3	32, 64, 128	3x3	ELU	SGD	0.001	Batch Normalization	Повороти, зсуви, горизонтальне відображення
12	5	8, 16, 32, 64, 128	3x3	ReLU	Adam	0.0001	Dropout (0.4)	Зсуви, масштабування, зміна контрастності
13	2	32, 64	5x5	Leaky ReLU	RMSprop	0.0005	L2 (0.0005)	Горизонтальне відображення, зміна яскравості та контрастності
14	4	32, 32, 64, 128	3x3	ReLU	Adam	0.001	Dropout (0.3)	Повороти, зсуви, масштабування, горизонтальне відображення
15	3	64, 128, 128	5x5	ELU	RMSprop	0.0001	L1 (0.0001)	Зсуви, масштабування, зміна яскравості

16	4	16, 32, 64, 128	3x3	ReLU	SGD	0.01	Dropout (0.5)	Повороти, зсуви, горизонтальне та вертикальне відображення
17	2	64, 128	3x3	Leaky ReLU	Adam	0.005	Batch Normalization	Масштабування, зміна яскравості та контрастності
18	3	32, 64, 128	5x5	ReLU	RMSprop	0.001	Dropout (0.2)	Повороти, горизонтальне відображення
19	5	8, 16, 32, 64, 128	3x3	ELU	Adam	0.0001	L2 (0.001)	Зсуви, зміна контрастності
20	2	32, 64	3x3	ReLU	SGD	0.01	Dropout (0.4)	Горизонтальне та вертикальне відображення
21	4	32, 32, 64, 64	3x3	Leaky ReLU	RMSprop	0.0005	L1 (0.0005)	Повороти, зсуви, масштабування
22	3	64, 128, 256	5x5	ReLU	Adam	0.001	Batch Normalization	Зсуви, масштабування, зміна яскравості
23	4	16, 32, 64, 128	3x3	ELU	Adam	0.0001	Dropout (0.5)	Повороти, зсуви, горизонтальне та вертикальне відображення
24	2	64, 128	3x3	ReLU	SGD	0.001	L2 (0.0001)	Масштабування, зміна яскравості та контрастності
25	3	32, 64, 128	5x5	Leaky ReLU	RMSprop	0.001	Dropout (0.3)	Повороти, зсуви, горизонтальне відображення
26	5	8, 16, 32, 64, 128	3x3	ReLU	Adam	0.0005	L1 (0.001)	Зсуви, масштабування, зміна контрастності
27	2	32, 64	3x3	ELU	RMSprop	0.0001	Batch Normalization	Горизонтальне відображення, зміна яскравості та контрастності
28	4	32, 32, 64, 128	3x3	ReLU	SGD	0.01	Dropout (0.4)	Повороти, зсуви, масштабування, горизонтальне відображення
29	3	64, 128, 128	3x3	Leaky ReLU	Adam	0.001	L2 (0.0005)	Зсуви, масштабування, зміна яскравості
30	4	16, 32, 64, 128	5x5	ReLU	RMSprop	0.0005	Dropout (0.5)	Повороти, зсуви, горизонтальне та вертикальне відображення

Welcome to Google Colab

https://colab.research.google.com/drive/1fhpBkHnnvPuF6U9ZJSTM7y1H6IKbT-Tu?usp=sharing