Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем і технологій

# Лабораторна робота №5

## **Прикладні задачі машинного навчання**

“Проектування та навчання штучної нейронної мережі для задач класифікації”

Виконав
студент групи ІК – 33:
Вересоцький А. Ю.

Перевірив:
асистент кафедри ІСТ
Нестерук А.О.

Київ 2025

**Завдання:**

**Підготовка і завантаження даних.**

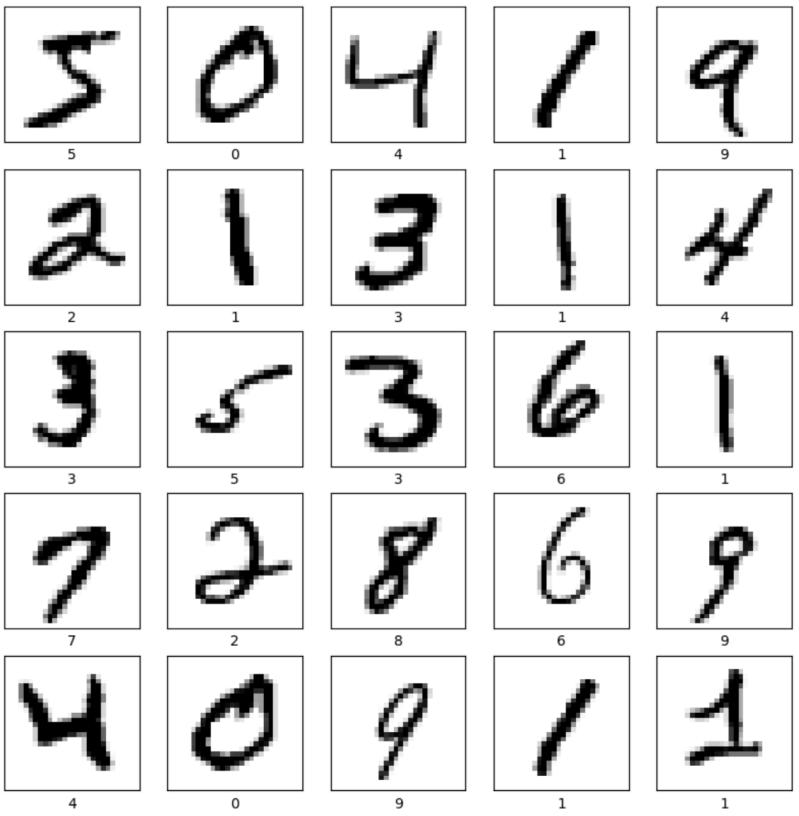
Ми будемо використовувати готовий набір даних чорно-білих зображень рукописних цифр із відомого набору даних MNIST.

Цей набір даних уже входить в Keras у вигляді чотирьох масивів і його можна завантажити наступним чином:

```
[2]: from keras.datasets import mnist
     (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Переглядаємо, перші 25 зображень та відповідні їм індекси:

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(train_labels[i])
plt.show()
```



Бачимо, що в результаті зображені перші 25 зображень цифр та відповідні їм індекси, що відповідають зображеним цифрам.

Далі ми перетворимо тривимірний масив в двовимірний масив (60000, 28x28) типу float32, а потім нормалізуємо його так, щоб отримати значення в інтервалі [0, 1]:

```
[4]: train_images = train_images.reshape((60000, 28 * 28))
     train_images = train_images.astype('float32') / 255
     test_images = test_images.reshape((10000, 28 * 28))
     test_images = test_images.astype('float32') / 255
```

Тепер підготуємо мітки, тобто завантажимо масиви із відгуками на кожне тестове зображення:

```
[5]: train_labels = to_categorical(train_labels)
     test_labels = to_categorical(test_labels)
```

**Архітектура моделі.** Ми будемо використовувати клас Sequential тому, що наша модель є простою послідовністю шарів – вихідного шару, який складається із 28x28=784 нейронів, тобто кожному пікселю зображення відповідає один нейрон. Інформація про вхідний шар міститься в описі прихованого шару, який створюється методом .add(). Ми будемо використовувати щільні, повнозв'язні (Dense) шари в яких кожен нейрон одного шару з'єднаний з кожним нейроном наступного шару. Прихований шар містить 512 нейронів, а вихідний шар містить 10 класифікаційних нейронів.

За функції активації нейронів ми будемо використовувати функції relu у прихованому шарі і softmax в останньому шарі.

```
[6]: network = models.Sequential()

     network.add(layers.Dense(512, activation='relu'))
     network.add(layers.Dense(10, activation='softmax'))
```

**Компіляція мережі.**

Після того, як модель визначена, її потрібно підготовити до навчання і компілювати, тобто привести її до вигляду, сумісного із базовою бібліотекою TensorFlow. Для цього налаштуємо ще три додаткові параметри: оптимізатор, функція втрат, метрики.

Після визначення цих параметрів процес компіляції виконується методом .compile():

```
[7]: network.compile(optimizer = 'rmsprop',
                     loss = 'categorical_crossentropy',
                     metrics = ['accuracy'])
```

**Навчання моделі.**

Для навчання викликаємо метод fit, який намагається адаптувати модель під навчальні дані.

```
[8]: network.fit(train_images, train_labels, epochs = 5, batch_size = 128)
Epoch 1/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8717 - loss: 0.4386
Epoch 2/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9657 - loss: 0.1164
Epoch 3/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9784 - loss: 0.0709
Epoch 4/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9849 - loss: 0.0513
Epoch 5/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9894 - loss: 0.0348
[8]: <keras.src.callbacks.history.History at 0x155b0f4a0>
```

**Перевірка моделі.**

Після того, як модель навчена, її потрібно перевірити на контрольному наборі даних, які ще не пред'являлися моделі. Перевіримо як модель розпізнає контрольний набір:

```
[9]: test_loss, test_acc = network.evaluate(test_images, test_labels)
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 600us/step - accuracy: 0.9785 - loss: 0.0713
```

**Збереження моделі.**

Команда model.save(filepath) зберігає модель Keras в одному файлі HDF5.

```
[10]: network.save('my_model.h5')
```

При потребі можна зберегти лише архітектуру моделі, наприклад у .json форматі:

```
[11]: json_string = network.to_json()
```

Або можна зберегти лише значення вагових коефіцієнтів:

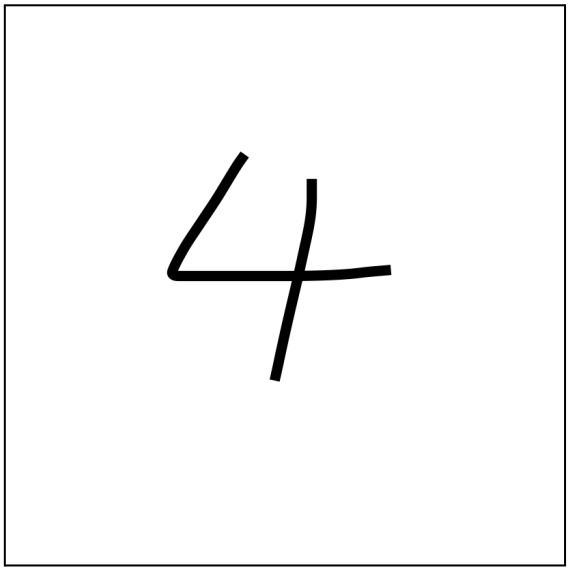
```
[12]: network.save_weights('my_model.weights.h5')
```

**Робота із моделлю.**

Завантажуємо попередньо збережену модель:

```
[13]: model = load_model('my_model.h5')
```

Для класифікації зображення із тестового файлу (наприклад my_test.png) із цифрою зберігаємо його в каталозі зі виконуваним файлом:



Завантажуємо його в OpenCV як сіре зображення і отримуємо данні у форматі ndarray. Потім цей масив переформатовується до того розміру на якому відбувалося тренування нашої моделі:

```
[16]: tst2 = 255 - cv2.imread('my_test.png', 0)
      tst2 = cv2.resize(tst2, (28, 28))
      tst2 = tst2.reshape((1, 28*28))
      tst2 = tst2.astype('float32') / 255
```

Підготовлені данні передаємо функції model.predict(tst). На виході отримаємо список списків із одним елементом – списком довжини 10, у якому на і-ій позиції знаходяться ймовірність того, що на вхідному зображенні є число і. Нам потрібна позиція з максимальною ймовірністю.

```
[17]: pred=list(model.predict(tst2)[0])
      print(pred.index(max(pred)))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
4
```

Бачимо, що модель коректно визначила зображену цифру.

**Проектування та розробка нейронної мережі на основі таких наборів даних імплементованих в Keras:**

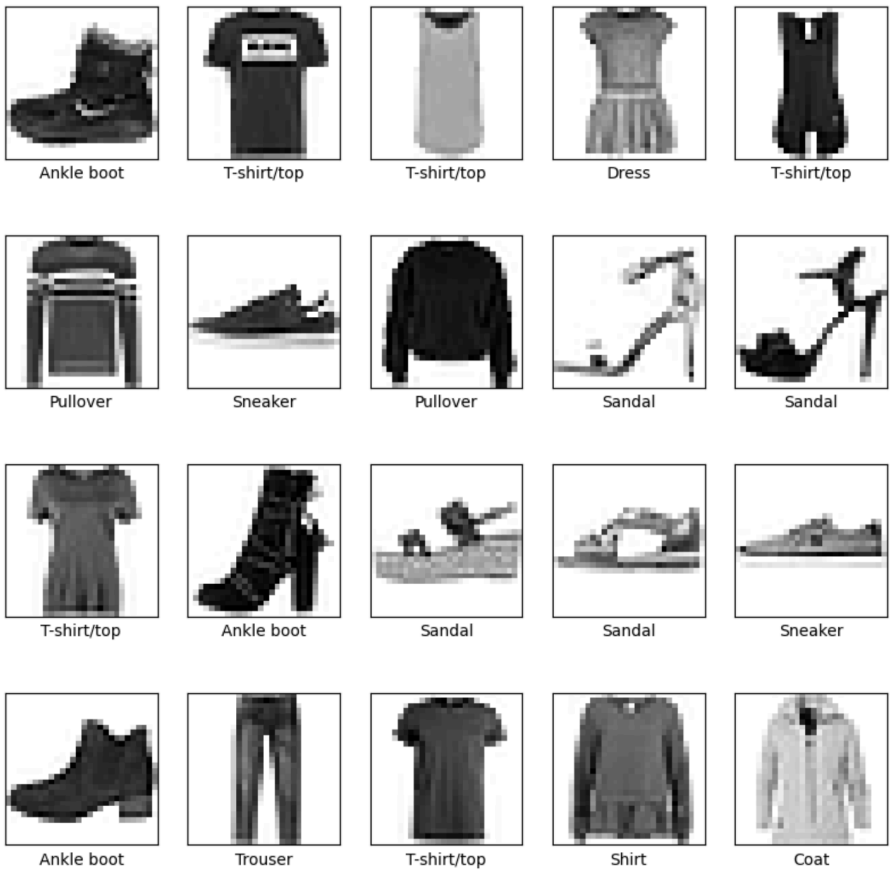
## 1) FMNIST

Завантажуємо набір даних та виводимо перші 20 зображень:

```
[18]: from keras.datasets import fashion_mnist
      (training_images, training_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

```
[19]: fmnist_classes = ['T-shirt/top','Trouser','Pullover','Dress','Coat','Sandal','Shirt','Sneaker','Bag','Ankle boot']

      plt.figure(figsize=(10,10))
      for i in range(20):
          plt.subplot(4,5,i+1)
          plt.xticks([])
          plt.yticks([])
          plt.grid(False)
          plt.imshow(training_images[i], cmap=plt.cm.binary)
          plt.xlabel(fmnist_classes[training_labels[i]])
      plt.show()
```



Нормалізуємо масив так, щоб отримати значення в інтервалі [0, 1]:

```
[20]: training_images = training_images / 255.0
      test_images = test_images / 255.0
```

Використаємо клас Sequential. Прихований шар містить 128 нейронів, а вихідний шар містить 10 класифікаційних нейронів.

За функції активації нейронів ми будемо використовувати функції relu у прихованому шарі і softmax в останньому шарі.

```
[21]: fm = models.Sequential([
          layers.Flatten(input_shape=(28, 28)),
          layers.Dense(128, activation='relu'),
          layers.Dense(10, activation='softmax')
      ])
```

Виконуємо процес компіляції методом .compile():

```
[22]: fm.compile(optimizer='adam',
                 loss='sparse_categorical_crossentropy',
                 metrics=['accuracy'])
```

Для навчання викликаємо метод fit:

```
[23]: fm.fit(training_images, training_labels, epochs=10)
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.7770 - loss: 0.6324
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8587 - loss: 0.3896
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8779 - loss: 0.3361
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8855 - loss: 0.3146
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8913 - loss: 0.2954
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8968 - loss: 0.2792
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9015 - loss: 0.2687
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9047 - loss: 0.2559
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9090 - loss: 0.2432
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9137 - loss: 0.2358
[23]: <keras.src.callbacks.history.History at 0x16343c2c0>
```

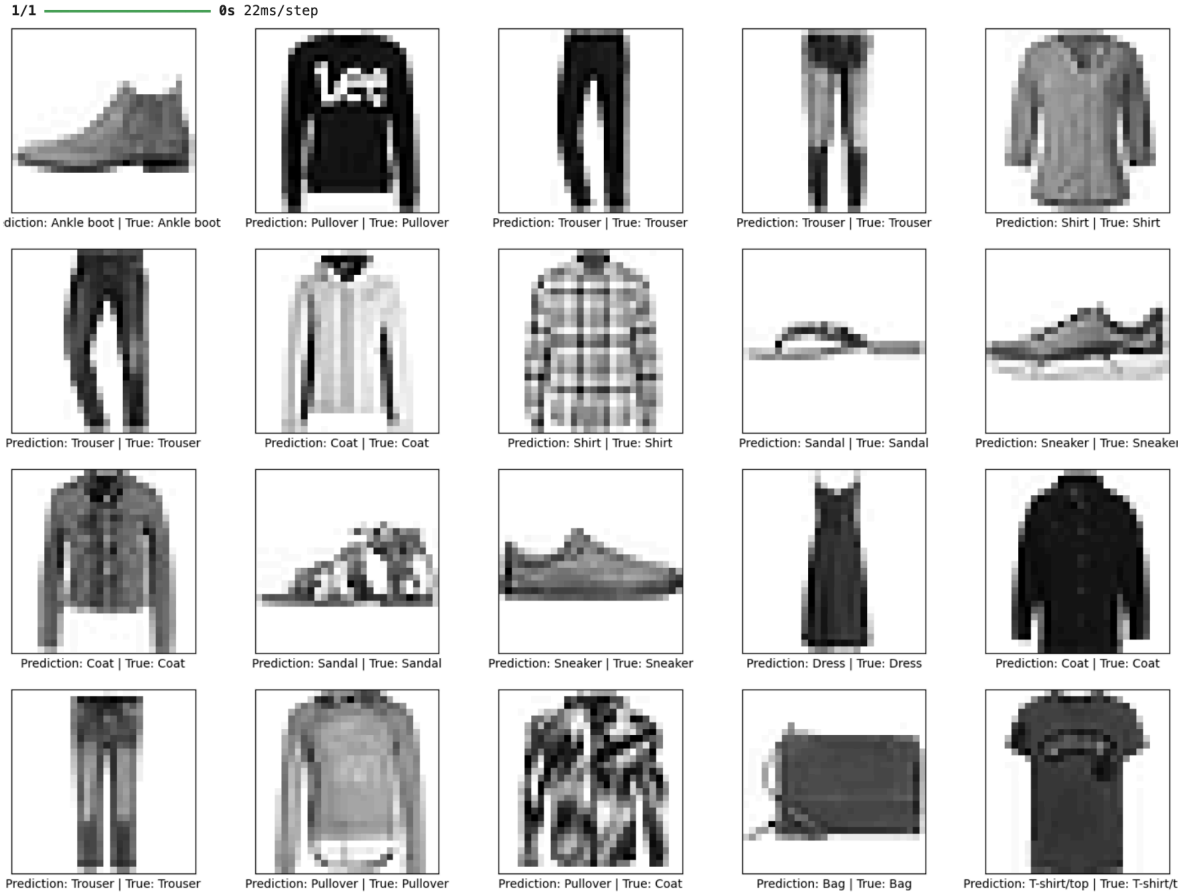
Перевіримо як модель розпізнає контрольний набір:

```
[24]: test_loss, test_acc = fm.evaluate(test_images, test_labels)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 493us/step - accuracy: 0.8854 - loss: 0.3334
```

Візуалізуємо роботу нашої моделі на тестових даних:

```
[25]: predictions = fm.predict(test_images[:20])
      plt.figure(figsize=(18,13))
      for i in range(20):
          plt.subplot(4,5,i+1)
          plt.xticks([])
          plt.yticks([])
          plt.grid(False)
          plt.imshow(test_images[i], cmap=plt.cm.binary)
          plt.xlabel(f"Prediction: {fmnist_classes[np.argmax(predictions[i])]} | True: {fmnist_classes[test_labels[i]]}")
      plt.show()
```




Перевіримо також як наша модель буде справлятися з зображенням одягу з інтернету. Для цього завантажимо фотографію штанів (jeans.png):

[26]:
```
tst = 255 - cv2.imread('jeans.png', 0)
tst = cv2.resize(tst, (28, 28))
tst = np.expand_dims(tst, axis=0)
tst = tst.astype('float32') / 255
```

[27]:
```
pred=list(fm.predict(tst)[0])
print(fmnist_classes[pred.index(max(pred))])
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
Trouser
```

Як можемо побачити, модель працює коректно.

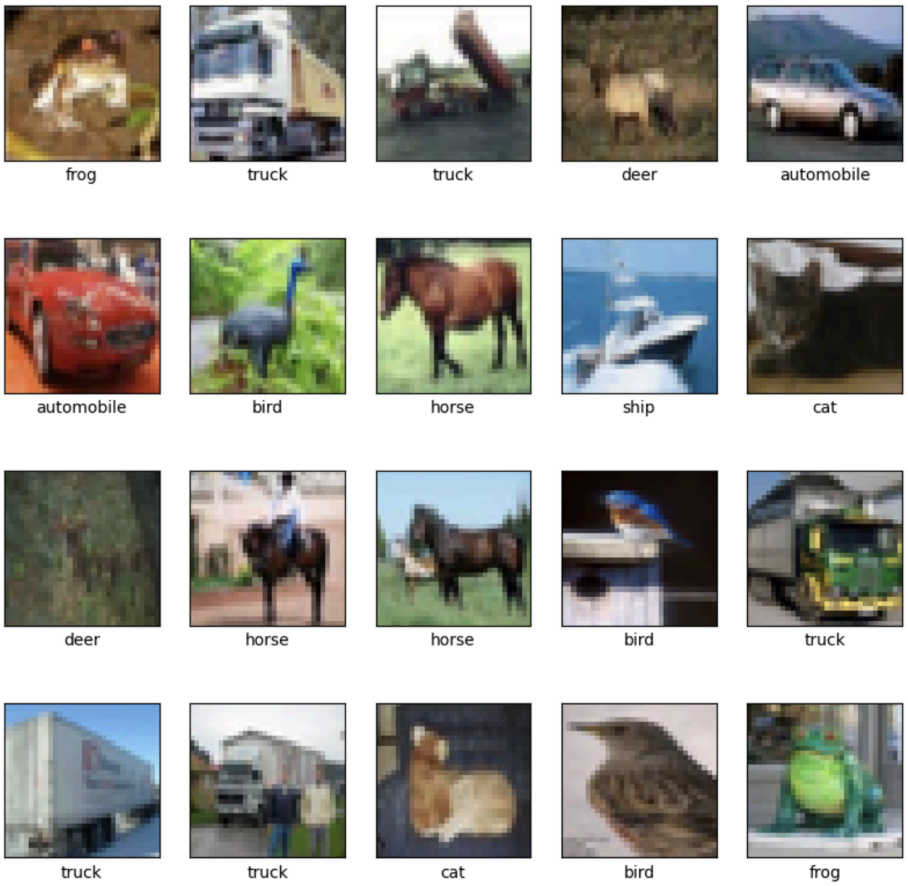
## 2) Cifar10

Завантажуємо набір даних та виводимо перші 20 зображень:

[28]:
```
from keras.datasets import cifar10
(training_images, training_labels), (test_images, test_labels) = cifar10.load_data()
```

[29]:
```
cifar_classes = ['airplane','automobile','bird','cat','deer','dog','frog','horse','ship','truck']

plt.figure(figsize=(10,10))
for i in range(20):
    plt.subplot(4,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(training_images[i])
    plt.xlabel(cifar_classes[training_labels[i][0]])
plt.show()
```



Нормалізуємо масив так, щоб отримати значення в інтервалі [0, 1]:

```
[30]: training_images, test_images = training_images / 255.0, test_images / 255.0
```

**Архітектура моделі.**

1. Згортковий шар – "дивиться" на маленькі шматочки зображення — віконце 3x3 пікселі — і шукає фрагменти, наприклад краї, текстури чи контури.
   Тут створюється 32 фільтри — значить з одного зображення мережа робить 32 нові карти ознак (feature maps). input_shape=(32, 32, 3) — це розмір зображення: 32x32 пікселі та 3 канали (RGB).
2. Шар підвибірки (Pooling) – Зменшує розмір зображення вдвічі, залишаючи найсильніші ознаки.
3. Ще один згортковий шар. Тепер уже 64 фільтри. Цей шар ще глибше аналізує зображення, вже не просто краї, а складніші шаблони (наприклад, форма автомобіля чи крила птаха).
4. Знову підвибірка, знову зменшує зображення, виділяючи важливі риси.
5. Ще один згортковий шар з 64 фільтрами. Тут мережа вчиться бачити вже абстрактні особливості об'єктів на зображенні.
6. **layers.Flatten()** – перетворює 2D-карти ознак у плоский вектор для подачі в класичний шар (Dense).
7. Щільний шар (Dense) з 64 нейронами – обробляє вектор і навчається комбінаціям ознак для класифікації.
8. Фінальний шар – 10 нейронів — бо в CIFAR-10 10 класів (машина, кіт, птах, корабель тощо).

```
[31]: cif = models.Sequential([
            layers.Conv2D(32, (3,3), activation='relu', input_shape=(32, 32, 3)),
            layers.MaxPooling2D((2,2)),
            layers.Conv2D(64, (3,3), activation='relu'),
            layers.MaxPooling2D((2,2)),
            layers.Conv2D(64, (3,3), activation='relu'),
            layers.Flatten(),
            layers.Dense(64, activation='relu'),
            layers.Dense(10, activation='softmax')
      ])
      cif.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Для навчання викликаємо метод fit:

```
[32]: cif.fit(training_images, training_labels, epochs=10, batch_size=128)
```

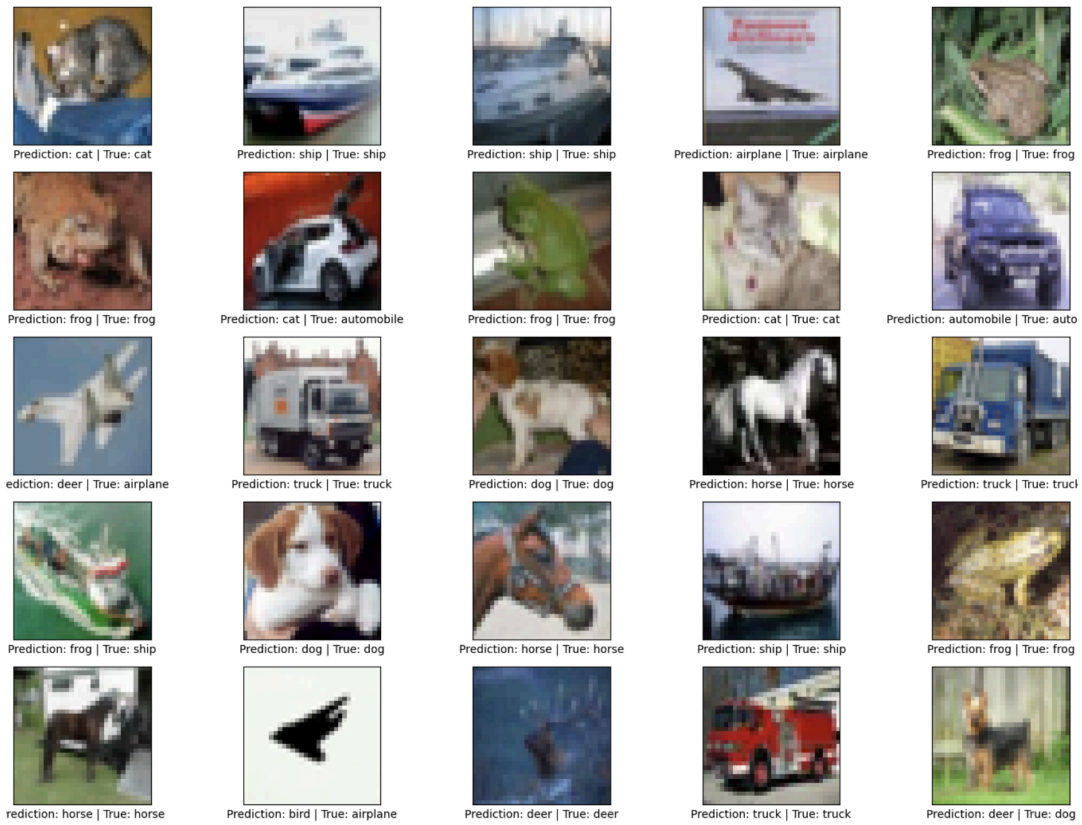
```
Epoch 1/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 23ms/step - accuracy: 0.3095 - loss: 1.8703
Epoch 2/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 23ms/step - accuracy: 0.5187 - loss: 1.3352
Epoch 3/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 23ms/step - accuracy: 0.5869 - loss: 1.1648
Epoch 4/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 24ms/step - accuracy: 0.6332 - loss: 1.0483
Epoch 5/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 9s 24ms/step - accuracy: 0.6593 - loss: 0.9808
Epoch 6/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 10s 24ms/step - accuracy: 0.6834 - loss: 0.9022
Epoch 7/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 10s 24ms/step - accuracy: 0.7053 - loss: 0.8532
Epoch 8/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 10s 24ms/step - accuracy: 0.7200 - loss: 0.8078
Epoch 9/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 10s 24ms/step - accuracy: 0.7357 - loss: 0.7682
Epoch 10/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 10s 24ms/step - accuracy: 0.7433 - loss: 0.7401
[32]: <keras.src.callbacks.history.History at 0x1686289b0>
```

Перевіримо як модель розпізнає контрольний набір:

```
[33]: test_loss, test_acc = cif.evaluate(test_images, test_labels)
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.7128 - loss: 0.8466
```

Візуалізуємо роботу нашої моделі на тестових даних:

```
[34]: predictions = cif.predict(test_images[:25])
plt.figure(figsize=(18,13))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(test_images[i])
    plt.xlabel(f"Prediction: {cifar_classes[np.argmax(predictions[i])]} | True: {cifar_classes[test_labels[i][0]]}")
plt.show()
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
```



Перевіримо також як наша модель буде справлятися з зображенням з інтернету. Для цього завантажимо фотографію літака (plane.png):

```
[36]: img = cv2.imread('plane.png')
img_resized = cv2.resize(img, (32, 32))
img_normalized = img_resized.astype('float32') / 255.0
input_image = np.expand_dims(img_normalized, axis=0)
predictions = cif.predict(input_image)
predicted_class = np.argmax(predictions)
print(f"Модель думає, що це: {cifar_classes[predicted_class]}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
Модель думає, що це: airplane
```

Як можемо побачити, модель працює коректно.

**Висновки:** У ході виконання лабораторної роботи було спроектовано, реалізовано та протестовано штучні нейронні мережі для задач класифікації з використанням наборів даних MNIST, Fashion-MNIST та CIFAR-10.

В результаті роботи я закріпив знання щодо побудови архітектури нейронних мереж у середовищі Keras з використанням класу Sequential. Також навчився готувати та нормалізувати дані для навчання, використовувати різні шари — Dense для простих задач та Conv2D для обробки зображень.

Особливо цікавим був експеримент із застосуванням моделі на зображеннях з інтернету, що дозволило оцінити здатність мережі до узагальнення на нових даних. Моделі продемонстрували задовільну якість класифікації, що підтверджує ефективність архітектури та обраних параметрів.