Лабораторна робота №5
"Оптимізація нейронної мережі"

Роботу виконав:
Климентьєв Максим
3-го курсу
групи ФІ-21

Зміст

1 Опис параметрів оптимізації та мета оптимізації

2 Опис експерименту, обраного алгоритму та отримані результати

2.1 Обраний алгоритм

```python
def DE(pop_size, iterations, function, limits, ints):
    dim = len(limits)
    limits = np.array(limits)
    x_low = limits[:, 0]
    x_high = limits[:, 1]

    population = np.random.uniform(x_low, x_high, (pop_size, dim))

    if ints:
        for it in ints:
            population[:, it] = np.round(population[:, it])

    max_f = -float('inf')
    best_f = float('inf')
    best_pop = np.zeros(dim)

    # function calls = pop_size + iterations * pop_size
    fitness = np.array([function(X) for X in population])
    for iteration in tqdm(
        range(iterations),
        desc="Processing",
        unit="step",
        bar_format="{l_bar}{bar:40}{r_bar}",
        colour='cyan',
        total=iterations
    ):
        for i in range(pop_size):
            F = np.random.uniform(1e-6, 2)
            P = np.random.uniform(1e-6, 1)
            r = np.random.uniform(1e-6, 1, dim)
            x1, x2, x3 = np.random.choice(population.shape[0], size=3, replace=False
    )
            while np.all(population[x1] == population[i]) or np.all(population[x2]
    == population[i]) or np.all(population[x3] == population[i]):
                x1, x2, x3 = np.random.choice(population.shape[0], size=3, replace=
    False)
            mutant_vector = population[x1] + F * (population[x2] - population[x3])
            mutant_vector[r < P] = population[i][r < P]
            if ints:
                for it in ints:
                    mutant_vector[it] = np.round(mutant_vector[it])
            mutant_vector = np.clip(mutant_vector, x_low, x_high)
            mutant_fitness = function(mutant_vector)
            if fitness[i] > mutant_fitness:
                fitness[i] = mutant_fitness
                population[i] = mutant_vector.copy()
```

```
        el_min = np.argmin(fitness)
        if best_f > fitness[el_min]:
            best_f = fitness[el_min]
            best_pop = population[el_min].copy()
        el_max = np.max(fitness)
        if max_f < el_max:
            max_f = el_max

    return best_f, best_pop

```

Лістинг 1: DE

2.2 Фітнес функція

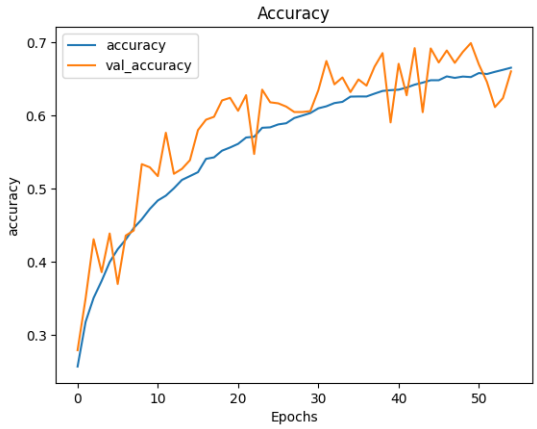
```
def func(X):
    X = X.astype(int)
    amount_of_layer_1 = X[0]
    amount_of_layer_2 = X[1]
    amount_of_layer_3 = X[2]
    amount_of_one_dence = X[3]
    amount_of_one_filters = X[4]

    model_layers = [
        layers.Input(shape=(32, 32, 3)),
        data_augmentation
    ]

    for _ in range(amount_of_layer_1):
        model_layers.append(layers.Conv2D(int(amount_of_one_filters), (3, 3),
    activation='relu', padding='same'))
        model_layers.append(layers.BatchNormalization())
    model_layers.append(layers.MaxPooling2D(2, 2))

    for _ in range(amount_of_layer_2):
        model_layers.append(layers.Conv2D(int(amount_of_one_filters * 2), (3, 3),
    activation='relu', padding='same'))
        model_layers.append(layers.BatchNormalization())
    model_layers.append(layers.MaxPooling2D(2, 2))
    model_layers.append(layers.Dropout(0.3))

    for _ in range(amount_of_layer_3):
        model_layers.append(layers.Conv2D(int(amount_of_one_filters * 4), (3, 3),
    activation='relu', padding='same'))
        model_layers.append(layers.BatchNormalization())
    model_layers.append(layers.MaxPooling2D(4, 4))
    model_layers.append(layers.Dropout(0.4))

    model_layers.append(layers.Flatten())
    for _ in range(amount_of_one_dence):
        model_layers.append(layers.Dense(int(amount_of_one_filters * 8), activation=
    'relu'))
    model_layers.append(layers.Dropout(0.5))
    model_layers.append(layers.Dense(10, activation='softmax'))
```

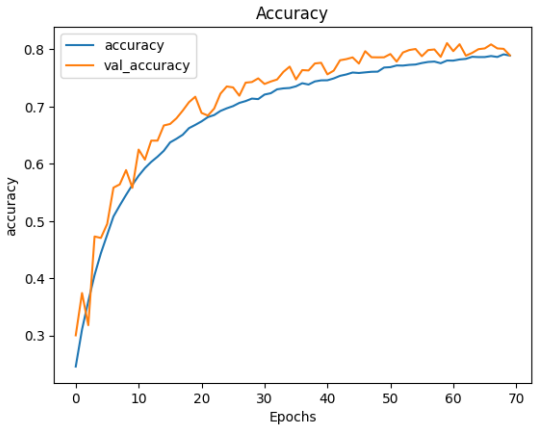
```
36
37      model = tf.keras.models.Sequential(model_layers)
38      summary_conv_and_last_dense(model)
39      model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
40                    loss=tf.keras.losses.CategoricalCrossentropy(),
41                    metrics=['accuracy', tf.keras.metrics.TopKCategoricalAccuracy(k=2,
    name="Top2")])
42      history = model.fit(train, batch_size=batch_size, epochs=100, validation_data=(
    val), callbacks=callbacks)
43      index = np.argmax(history.history['val_loss'])
44      evaluation = model.evaluate(test)
45      loss, acc, top2  = evaluation
46      return (1.0 - acc) + loss
47
48  func_limits = [[1, 4], [0, 4], [0, 4], [1, 4], [16, 64]]
49
```

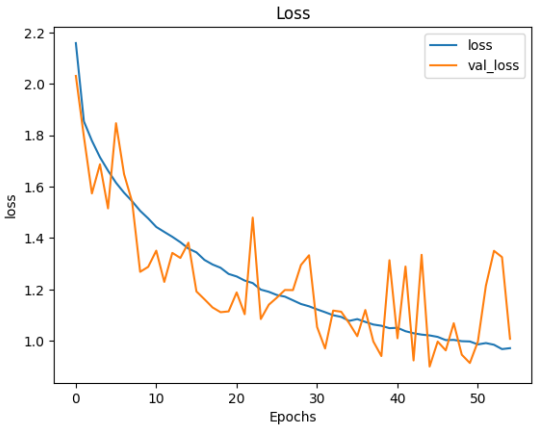
Лістинг 2: Fitness Func

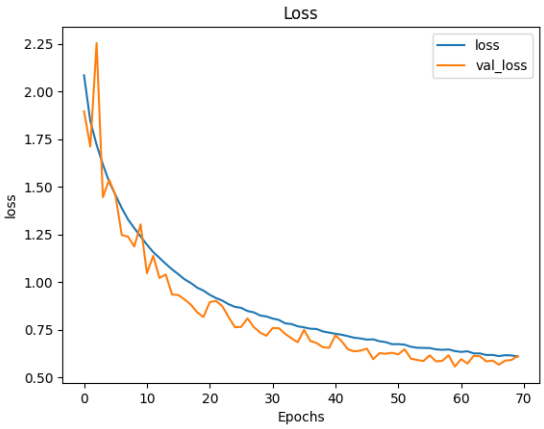
2.3 Порівняння результатів (до оптимізації параметрів нейронних мереж та після)

Тип	Час	Точність	Втрати
До	30	70	90
Після	34	80	58


Accuracy
accuracy
val_accuracy
accuracy
Epochs


Accuracy
accuracy
val_accuracy
accuracy
Epochs


Loss
loss
val_loss
loss
Epochs

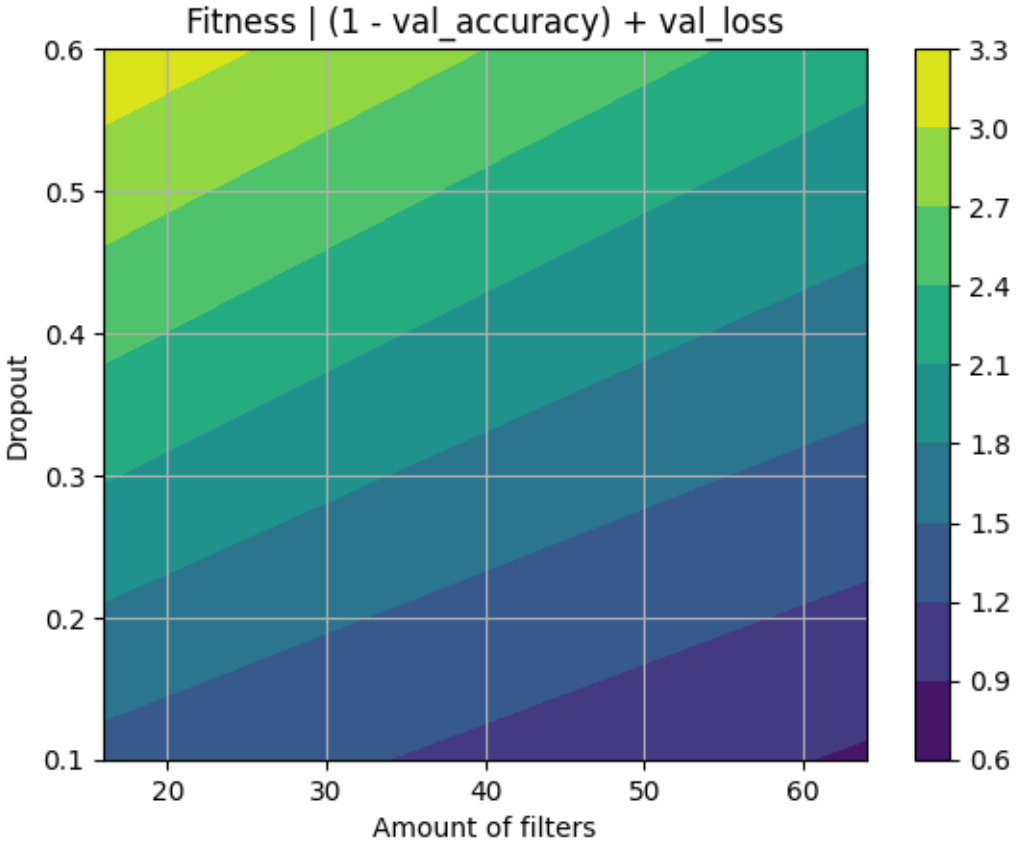

Loss
loss
val_loss
loss
Epochs

2.4 Час виконання (пошук оптимального рішення та інше)

Тип	Час (хв)
Пошук оптимального рішення	1500-2000
Побудова ландшафту	2000-4000
Інше	30

3 Висновки

Модель стає кращою як по збільшенню точності, так і по зменшенням втрат Також чим менше дропаут, та чим більше фільтрів тим точніше


Fitness | (1 - val_accuracy) + val_loss
Dropout
0.6
0.5
0.4
0.3
0.2
0.1
20
30
40
50
60
Amount of filters
3.3
3.0
2.7
2.4
2.1
1.8
1.5
1.2
0.9
0.6