

**Лабораторна робота №5**  
**"Оптимізація нейронної мережі"**

**Роботу виконав:**  
Климентьев Максим  
3-го курсу  
групи ФІ-21

## Зміст

<b>1</b>	<b>Опис параметрів оптимізації та мета оптимізації</b>	<b>3</b>
<b>2</b>	<b>Опис експерименту, обраного алгоритму та отримані результати</b>	<b>3</b>
2.1	Обраний алгоритм . . . . .	3
2.2	Фітнес функція . . . . .	4
2.3	Порівняння результатів (до оптимізації параметрів нейронних мереж та після) .	5
2.4	Час виконання (пошук оптимального рішення та інше) . . . . .	7
<b>3</b>	<b>Висновки</b>	<b>7</b>

- 1 Опис параметрів оптимізації та мета оптимізації
- 2 Опис експерименту, обраного алгоритму та отримані результати

## 2.1 Обраний алгоритм

```
1 def DE(pop_size, iterations, function, limits, ints):
2     dim = len(limits)
3     limits = np.array(limits)
4     x_low = limits[:, 0]
5     x_high = limits[:, 1]
6
7     population = np.random.uniform(x_low, x_high, (pop_size, dim))
8
9     if ints:
10         for it in ints:
11             population[:, it] = np.round(population[:, it])
12
13     max_f = -float('inf')
14     best_f = float('inf')
15     best_pop = np.zeros(dim)
16
17     # function calls = pop_size + iterations * pop_size
18     fitness = np.array([function(X) for X in population])
19     for iteration in tqdm(
20         range(iterations),
21         desc="Processing",
22         unit="step",
23         bar_format="{l_bar}{bar:40}{r_bar}",
24         colour='cyan',
25         total=iterations
26     ):
27         for i in range(pop_size):
28             F = np.random.uniform(1e-6, 2)
29             P = np.random.uniform(1e-6, 1)
30             r = np.random.uniform(1e-6, 1, dim)
31             x1, x2, x3 = np.random.choice(population.shape[0], size=3, replace=False)
32
33             while np.all(population[x1] == population[i]) or np.all(population[x2]
34 == population[i]) or np.all(population[x3] == population[i]):
35                 x1, x2, x3 = np.random.choice(population.shape[0], size=3, replace=
36 False)
37
38             mutant_vector = population[x1] + F * (population[x2] - population[x3])
39             mutant_vector[r < P] = population[i][r < P]
40             if ints:
41                 for it in ints:
42                     mutant_vector[it] = np.round(mutant_vector[it])
43             mutant_vector = np.clip(mutant_vector, x_low, x_high)
44             mutant_fitness = function(mutant_vector)
45             if fitness[i] > mutant_fitness:
46                 fitness[i] = mutant_fitness
47                 population[i] = mutant_vector.copy()
```

```

44     el_min = np.argmin(fitness)
45     if best_f > fitness[el_min]:
46         best_f = fitness[el_min]
47         best_pop = population[el_min].copy()
48     el_max = np.max(fitness)
49     if max_f < el_max:
50         max_f = el_max
51
52     return best_f, best_pop
53

```

Лістинг 1: DE

## 2.2 Фітнес функція

```

1  def func(X):
2      X = X.astype(int)
3      amount_of_layer_1 = X[0]
4      amount_of_layer_2 = X[1]
5      amount_of_layer_3 = X[2]
6      amount_of_one_dence = X[3]
7      amount_of_one_filters = X[4]
8
9      model_layers = [
10         layers.Input(shape=(32, 32, 3)),
11         data_augmentation
12     ]
13
14     for _ in range(amount_of_layer_1):
15         model_layers.append(layers.Conv2D(int(amount_of_one_filters), (3, 3),
16             activation='relu', padding='same'))
17         model_layers.append(layers.BatchNormalization())
18         model_layers.append(layers.MaxPooling2D(2, 2))
19
20     for _ in range(amount_of_layer_2):
21         model_layers.append(layers.Conv2D(int(amount_of_one_filters * 2), (3, 3),
22             activation='relu', padding='same'))
23         model_layers.append(layers.BatchNormalization())
24         model_layers.append(layers.MaxPooling2D(2, 2))
25         model_layers.append(layers.Dropout(0.3))
26
27     for _ in range(amount_of_layer_3):
28         model_layers.append(layers.Conv2D(int(amount_of_one_filters * 4), (3, 3),
29             activation='relu', padding='same'))
30         model_layers.append(layers.BatchNormalization())
31         model_layers.append(layers.MaxPooling2D(4, 4))
32         model_layers.append(layers.Dropout(0.4))
33
34     model_layers.append(layers.Flatten())
35     for _ in range(amount_of_one_dence):
36         model_layers.append(layers.Dense(int(amount_of_one_filters * 8), activation=
37             'relu'))
38     model_layers.append(layers.Dropout(0.5))
39     model_layers.append(layers.Dense(10, activation='softmax'))

```

```

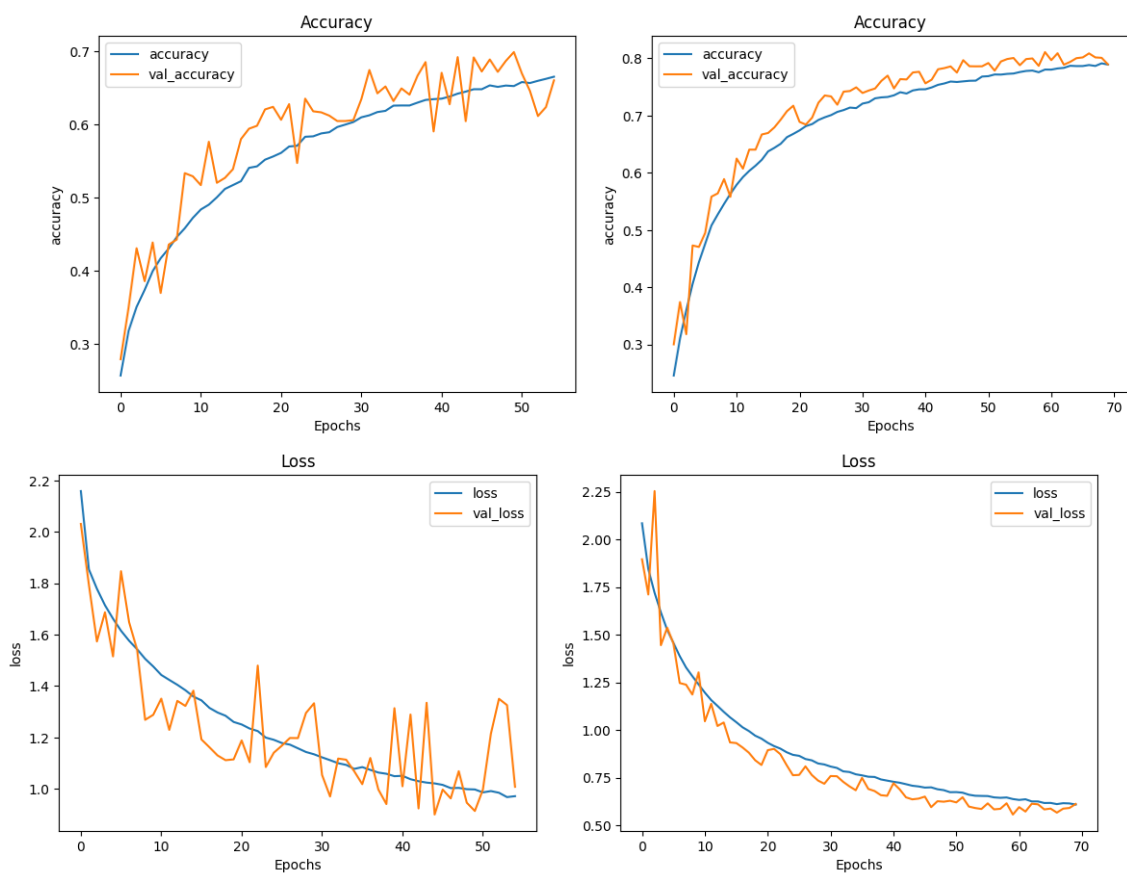
36
37     model = tf.keras.models.Sequential(model_layers)
38     summary_conv_and_last_dense(model)
39     model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
40                  loss=tf.keras.losses.CategoricalCrossentropy(),
41                  metrics=['accuracy', tf.keras.metrics.TopKCategoricalAccuracy(k=2,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

```

Лістинг 2: Fitness Func

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

Тип	Час	Точність	Втрати
До	30	70	0.90
Після	34	80	0.58



## 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 зменшенню втрат Також чим менше дропаут, та чим більше фільтрів тим точніше

