# ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

# Комп'ютерна система обробки зображень з використанням машинного навчання

*Кваліфікаційна робота*

**перший (бакалаврський) рівень**

Виконав:
студент гр. КІУКІ-21-3
Томащук І.І

Керівник:
ас. каф. ЕОМ
Бондаренко М.Е.

## *Мета роботи та завдання*

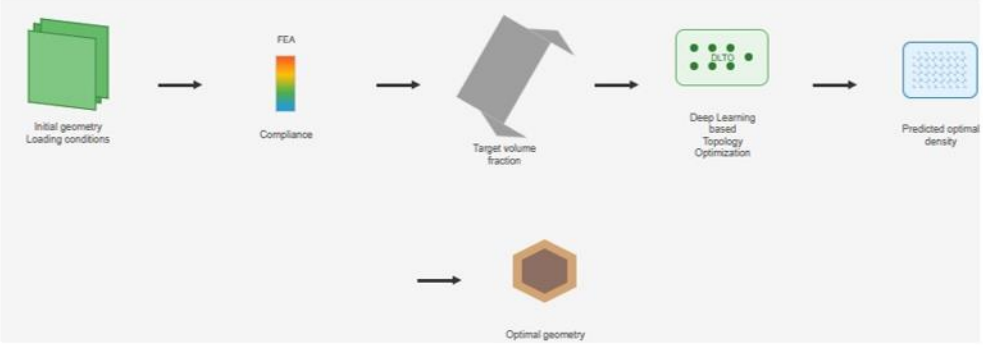


**Метою роботи** є розробка програмних засобів обробки зображень з використанням методів машинного навчання та генетичних алгоритмів.

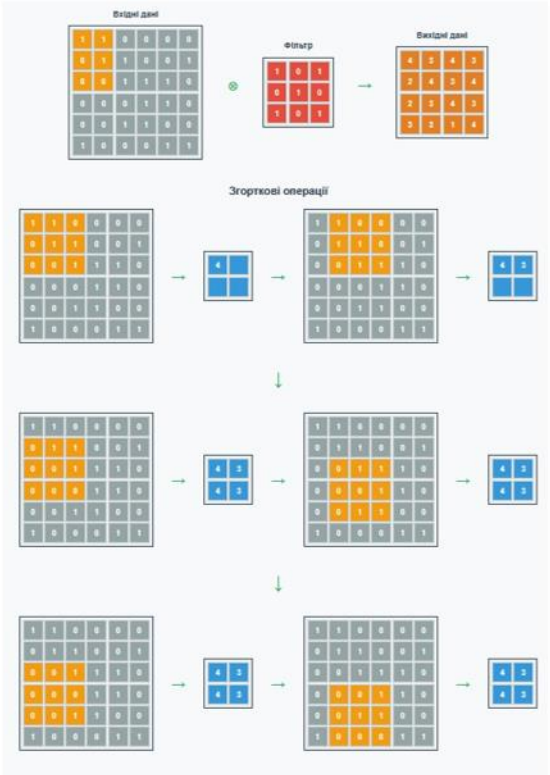
**Завдання:**

- аналіз алгоритмів навчання глибоких нейронних мереж;
- розробка алгоритму структурного синтезу згорткових нейронних мереж з використанням генетичних алгоритмів;
- вибір технологій та програмно-апаратних засобів для реалізації ПЗ для обробки зображень;
- розробка програмних засобів оптимізації топології на основі розглянутих алгоритмів.

## *Згорткові нейронні мережі* 3

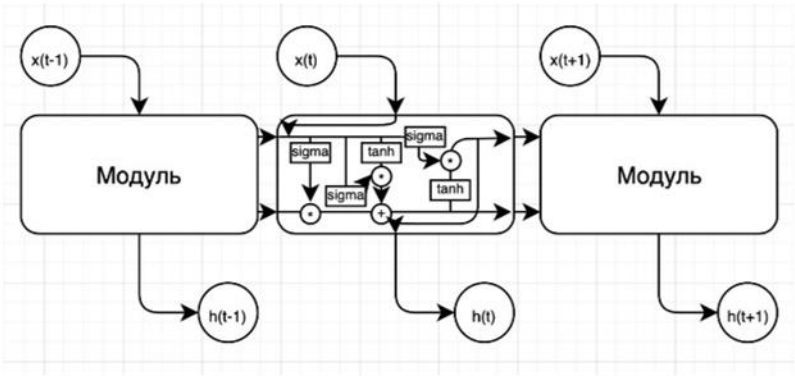


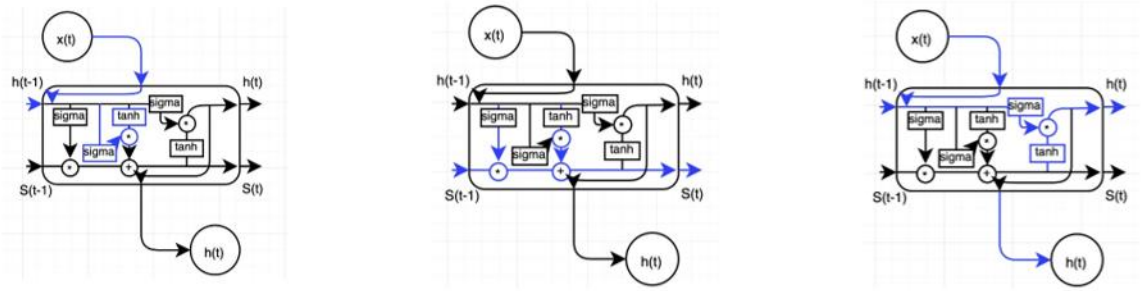

## *Згорткові нейронні мережі* 4



5

# *Приклад мережі LSTM*



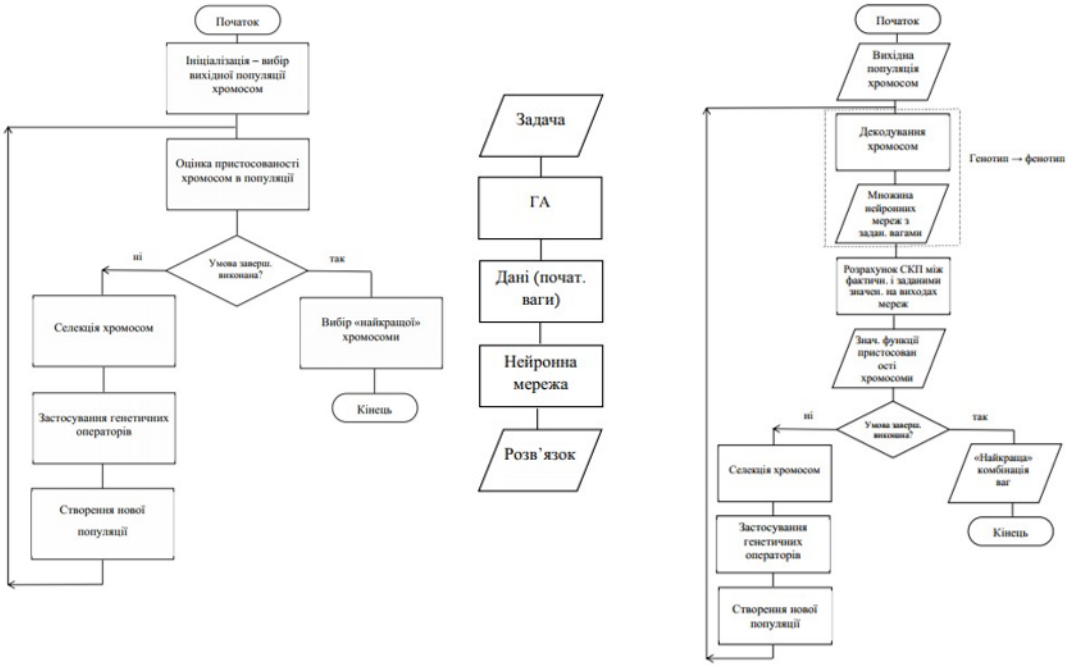




6

# *Блок-схема методу SIMP*

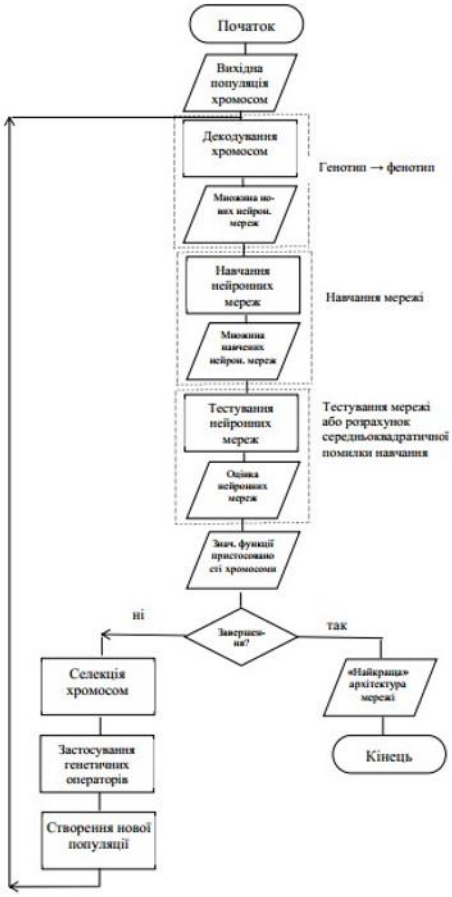
## *Оптимізація структури згорткової нейронної мережі*

7



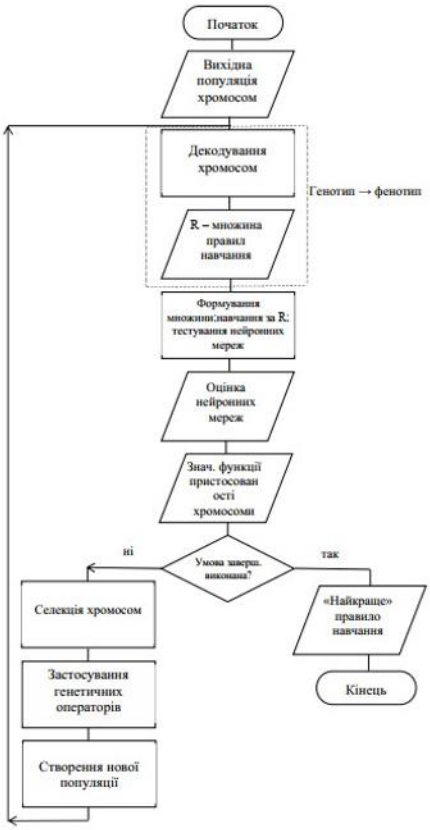

## ***Блок-схема генетичного алгоритму ( пошук архітектури)***

8



9

## *Пошук найкращого правила навчання*




10

## *Схема роботи алгоритму*

## Вибір програмних засобів

11

```
# 🧠 Встановлення залежностей
!pip install deap tensorflow numpy matplotlib
```

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt
from deap import base, creator, tools, algorithms
import random
```

```
# 📦 Завантаження та підготовка даних (CIFAR-10)
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

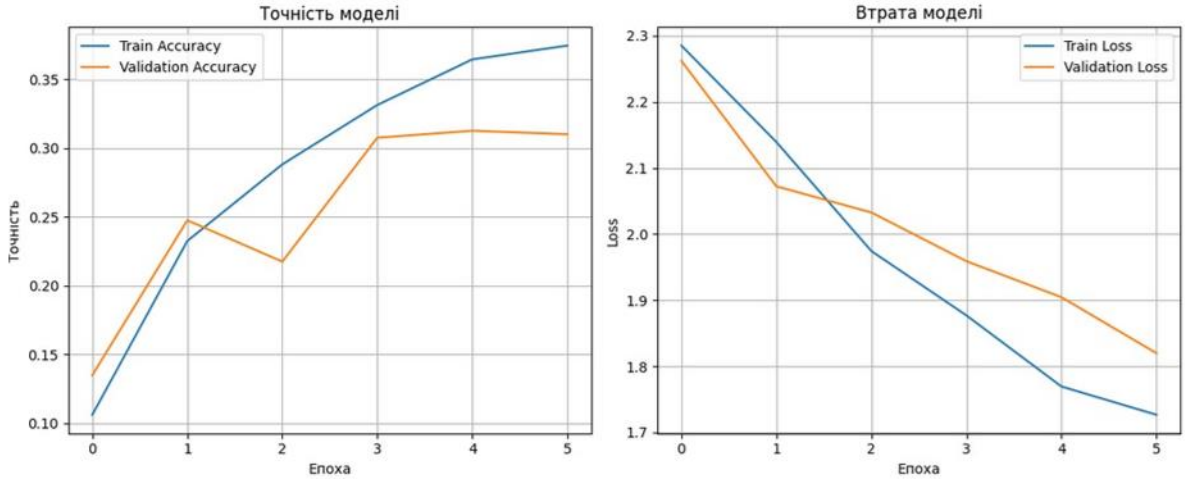
```
# 🔧 Побудова CNN за хромосомою
def build_model(genes):
    model = models.Sequential()
    model.add(layers.Input(shape=(32, 32, 3)))
    for num_filters in genes:
        model.add(layers.Conv2D(num_filters, (3, 3), activation='relu', padding='same'))
        model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(10, activation='softmax'))
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

```
# 🎯 Оцінка хромосоми
def evaluate(individual):
    model = build_model(individual)
    history = model.fit(x_train[:2000], y_train[:2000], epochs=3, batch_size=64, verbose=0)
    _, acc = model.evaluate(x_test[:1000], y_test[:1000], verbose=0)
    return (acc,)
```

```
# 🧬 Налаштування генетичного алгоритму
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)
toolbox = base.Toolbox()
toolbox.register("attr_filter", random.choice, [16, 32, 64, 128])
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_filter, n=3)
```

12

## Результати



13

# *Висновки*

В ході виконання кваліфікаційної робооти розроблено програмні засоби обробки зображень з використанням методів машинного навчання, зокрема глибокого навчання, та генетичних алгоритмів. Проведено аналіз алгоритмів навчання глибоких нейронних мереж, запропоновано алгоритм структурного синтезу згорткових нейронних мереж з використанням генетичних алгоритмів; обрано технології та програмні засоби для реалізації програмного забезпечення.

# ДОДАТОК Б

## Програмний код

```
def create_hybrid_cnn_transformer(input_shape=(224, 224, 3),
num_classes=7):
    """Гібридна архітектура, що поєднує CNN та Transformer"""

    inputs = tf.keras.Input(shape=input_shape)

    # CNN backbone для вивчення низькорівневих ознак
    x = tf.keras.applications.EfficientNetB4(
        include_top=False, weights='imagenet'
    )(inputs)

    # Перетворення в sequence для Transformer
    batch_size = tf.shape(x)[0]
    height, width, channels = x.shape[1], x.shape[2], x.shape[3]

    # Patch extraction
    patches = tf.reshape(x, [batch_size, height * width,
channels])

    # Positional embedding
    positions = tf.range(height * width)
    pos_embedding = tf.keras.layers.Embedding(height * width,
channels)(positions)
    patches = patches + pos_embedding

    # Multi-head self-attention layers
    for _ in range(6):
        # Self-attention
        attention_output = tf.keras.layers.MultiHeadAttention(
            num_heads=8, key_dim=channels // 8
        )(patches, patches)

        # Add & Norm
        patches = tf.keras.layers.Add()([patches,
attention_output])
        patches = tf.keras.layers.LayerNormalization()(patches)

        # Feed-forward network
        ffn_output = tf.keras.layers.Dense(channels * 4,
activation='gelu')(patches)
        ffn_output = tf.keras.layers.Dense(channels)(ffn_output)

        # Add & Norm
        patches = tf.keras.layers.Add()([patches, ffn_output])
        patches = tf.keras.layers.LayerNormalization()(patches)
```

```
    # Global representation
    global_repr =
tf.keras.layers.GlobalAveragePooling1D()(patches)

    # Classification head
    outputs = tf.keras.layers.Dense(num_classes,
activation='softmax')(global_repr)

    return tf.keras.Model(inputs, outputs)
def advanced_optimization_experiments():
    """Експерименти з різними техніками оптимізації"""

    # Експерименти з функціями втрат
    loss_functions = {
        'categorical_crossentropy':
tf.keras.losses.CategoricalCrossentropy(),
        'focal_loss_2_0.25': focal_loss(gamma=2.0, alpha=0.25),
        'focal_loss_1_0.5': focal_loss(gamma=1.0, alpha=0.5),
        'class_balanced':
class_balanced_loss(samples_per_class),
        'label_smoothing':
tf.keras.losses.CategoricalCrossentropy(label_smoothing=0.1)
    }

    # Експерименти з оптимізаторами
    optimizers = {
        'adam_1e-4': tf.keras.optimizers.Adam(1e-4),
        'adam_1e-3': tf.keras.optimizers.Adam(1e-3),
        'adamw_1e-4': tfa.optimizers.AdamW(1e-4,
weight_decay=1e-5),
        'sgd_momentum': tf.keras.optimizers.SGD(1e-2,
momentum=0.9),
        'rmsprop': tf.keras.optimizers.RMSprop(1e-4)
    }

    # Експерименти з регуляризацією
    regularization_configs = {
        'baseline': {'dropout': 0.2, 'l2': 1e-4},
        'heavy_dropout': {'dropout': 0.5, 'l2': 1e-4},
        'heavy_l2': {'dropout': 0.2, 'l2': 1e-3},
        'light_reg': {'dropout': 0.1, 'l2': 1e-5},
        'no_reg': {'dropout': 0.0, 'l2': 0.0}
    }

    best_config = None
    best_score = 0
    all_results = {}

    for loss_name, loss_fn in loss_functions.items():
        for opt_name, optimizer in optimizers.items():
            for reg_name, reg_config in
regularization_configs.items():
```

```
                experiment_name =
f"{loss_name}_{opt_name}_{reg_name}"
                print(f"\n=== {experiment_name} ===")

                try:
                    # Створення моделі з поточною конфігурацією
                    model = create_medical_efficientnet(
                        dropout_rate=reg_config['dropout'],
                        l2_reg=reg_config['l2']
                    )

                    # Компіляція
                    model.compile(
                        optimizer=optimizer,
                        loss=loss_fn,
                        metrics=['accuracy']
                    )

                    # Навчання з раннім зупинком
                    history = model.fit(
                        train_generator,
                        epochs=30,  # Менше епох для швидкості
                        validation_data=val_generator,
                        callbacks=[

tf.keras.callbacks.EarlyStopping(patience=8),

tf.keras.callbacks.ReduceLROnPlateau(patience=4)
                        ],
                        verbose=0
                    )

                    # Оцінка
                    val_accuracy =
max(history.history['val_accuracy'])
                    test_accuracy =
model.evaluate(test_generator, verbose=0)[1]

                    all_results[experiment_name] = {
                        'val_accuracy': val_accuracy,
                        'test_accuracy': test_accuracy,
                        'config': {
                            'loss': loss_name,
                            'optimizer': opt_name,
                            'regularization': reg_name
                        }
                    }

                    # Оновлення кращої конфігурації
                    if val_accuracy > best_score:
                        best_score = val_accuracy
                        best_config = experiment_name
```

```
                    print(f"Val Accuracy: {val_accuracy:.4f},
Test Accuracy: {test_accuracy:.4f}")

                except Exception as e:
                    print(f"Помилка в експерименті
{experiment_name}: {str(e)}")
                    continue

    print(f"\n=== КРАЩИЙ РЕЗУЛЬТАТ ===")
    print(f"Конфігурація: {best_config}")
    print(f"Валідаційна точність: {best_score:.4f}")
    print(f"Тестова точність:
{all_results[best_config]['test_accuracy']:.4f}")

    return all_results, best_config
```