Міністерство освіти і науки України
Національний університет “Львівська політехніка”
Інститут комп’ютерних наук та інформаційних технологій

Кафедра САПР

Лабораторна робота №2
Частина 1

з дисципліни: “Застосування систем штучного інтелекту у технологічних рішеннях”

на тему:

“Мережа Хемінга та MAXNET. Класифікація несправностей промислового обладнання”

Варіант №3

Виконав:
ст. групи ПП-44
Верещак Б. О.
Прийняв:
доц. Левкович М.В.

Львів - 2025

Мета роботи

Набути поглиблених практичних навичок у проєктуванні, реалізації та аналізі нейронних мереж Хеммінга та MAXNET. Застосувати розроблену модель для класифікації біполярних векторів, що імітують сигнали несправностей промислового обладнання. Дослідити ключові властивості мережі, зокрема її здатність до розпізнавання зашумлених образів та динаміку збіжності конкурентного шару MAXNET.

Індивідуальне завдання

Виконати класифікацію біполярних векторів за допомогою мережі Хеммінга та MAXNET. Завдання полягає у створенні набору еталонних образів (класів), генерації тестових зашумлених даних та дослідженні ефективності й стійкості моделі.

1. **Сформувати набір даних**: визначити 3–5 еталонних образів (прототипів класів) у вигляді біполярних векторів {-1, 1}. Зробити опис, що символізує кожен клас (наприклад, тип несправності обладнання). Створити тестовий набір даних, додаючи до еталонів шум різного рівня (інвертуючи певну кількість бітів).
2. **Підготувати дані**: переконатися, що всі вектори (еталонні та тестові) мають однаковий формат та розмірність. Для цієї моделі масштабування чи синхронізація не потрібні.
3. **Реалізувати модель**: написати код, що імплементує двошарову архітектуру:
   - Шар Хеммінга (ініціалізація ваг та зсувів на основі еталонів).
   - Шар MAXNET (ітеративний процес конкуренції).
4. **Оцінити якість**: протестувати модель на зашумлених даних та розрахувати ключові метрики ефективності.
5. **Побудувати візуалізації**: створити графіки, що демонструють процес роботи мережі та її характеристики.
6. **Зробити аналіз результатів**: проаналізувати отримані дані, описати сильні та слабкі сторони моделі, зробити висновки та запропонувати можливі покращення

№	Завдання	Тип даних	Результат	Пропозиції до даних/еталонів
3	Діагностика друкованих плат (PCB)	Вектор, що кодує наявність дефектів у зонах плати (коротке замикання, обрив доріжки)	Тип дефекту: 'Плата справна', 'Замикання', 'Обрив', 'Відсутній компонент'	Кожен дефект - це "гарячий" біт у відповідній зоні вектора. 'Норма' - вектор з -1.

Хід роботи

1. Формування набору даних

Визначимо 4 еталонні образи (класи) для діагностики PCB. Вектор має розмірність N=4. Кожен біт відповідає певній зоні плати або типу дефекту.

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import random

# Diagnostics of printed circuit boards
# Defect type: 'Board is OK', 'Short circuit', 'Open circuit', 'Missing
component'

prototypes = np.array([
    [-1, -1, -1, -1], # Board is OK
    [1, -1, -1, -1], # Short circuit
    [-1, 1, 1, -1], # Open circuit
    [-1, -1, -1, 1] # Missing component
])

classes = np.array(['Board is OK', 'Short circuit', 'Open circuit', 'Missing
component'])

K, N = prototypes.shape
```

2. Підготовка даних

На цьому кроці нам потрібно підготувати всі дані для майбутнього використання з мережею Хеммінга, для цього генеруємо рандомну похибку в векторах, тобто міняємо певну кількість бітів на протилежну.

```
np.random.seed(3)

def add_noise(vector, noise_level):
    noisy_vector = np.copy(vector)
    indices_to_flip = np.random.choice(N, noise_level, replace=False)
    noisy_vector[indices_to_flip] *= -1
    return noisy_vector

test_set = []
true_labels = []
sample_noise_levels = []
noise_levels = [0, 1, 2, 3]
N_per_level = 100

for k in range(K):
    for noise in noise_levels:
        for _ in range(N_per_level):
            noisy_vector = add_noise(prototypes[k], noise)
            test_set.append(noisy_vector)
            true_labels.append(k)
            sample_noise_levels.append(noise)

test_set = np.array(test_set)
```

```
true_labels = np.array(true_labels)
test_set = np.array(test_set)
true_labels = np.array(true_labels)
sample_noise_levels = np.array(sample_noise_levels)

print(f"Кількість прототипів (класів) K = {K}")
print(f"Розмірність векторів N = {N}")
print(f"Еталонні образи (прототипи):\n{prototypes}")
print(f"Загальний розмір тестового набору: {test_set.shape[0]} векторів.")
print(f"Розподіл зразків за рівнями шуму: {len(noise_levels) * N_per_level *
K} (K * {len(noise_levels)} * {N_per_level})")
```

В результаті ми отримали такі характеристики підготовлених данних:

```
Кількість прототипів (класів) K = 4
Розмірність векторів N = 4
Еталонні образи (прототипи):
[[-1 -1 -1 -1]
 [ 1 -1 -1 -1]
 [-1  1  1 -1]
 [-1 -1 -1  1]]
Загальний розмір тестового набору: 1600 векторів.
Розподіл зразків за рівнями шуму: 1600 (K * 4 * 100)
```

3. Реалізація моделі

Для реалізації моделі був створений відповідний клас Хеммінга, який реалізує всі потрібні нам функції від ініціалізації до прогнозу. Він містить два основних компоненти: шар Хемінга для обчислення схожості вхідних векторів з прототипами та конкурентний шар MAXNET для визначення переможця.
Метод fit() виконує ініціалізацію ваг мережі на основі еталонних образів.
Конкурентний шар MAXNET реалізує процес взаємного гальмування нейронів.
Метод predict() виконує класифікацію вхідних векторів.

```
class HammingNetwork:
  def __init__(self, epsilon=0.1, max_iter=100):
    self.epsilon = epsilon
    self.max_iter = max_iter
    self.prototypes_ = None # Нормалізовані прототипи
    self.classes_ = None
    self.K_ = 0
    # Сховище для візуалізації (словник, як і раніше)
    self.maxnet_history_ = {}

  def fit(self, prototypes, classes):
    self.prototypes_ = prototypes
    self.classes_ = classes
    self.K_, self.N_ = prototypes.shape
    self.W_ = self.prototypes_
    self.b_ = np.full(self.K_, self.N_)

  def _maxnet(self, scores):
    """Ітеративний алгоритм MAXNET."""
    activations = np.copy(scores)
```

```
    history = [np.copy(activations)]
    for _ in range(self.max_iter):
      if np.sum(activations > 0) <= 1:
        break

      prev_activations = np.copy(activations)
      total_activation = np.sum(prev_activations)
      for j in range(self.K_):
        if prev_activations[j] > 0:
          inhibition = total_activation - prev_activations[j]
          activations[j] = prev_activations[j] - self.epsilon * inhibition
          if activations[j] < 0:
            activations[j] = 0
      history.append(np.copy(activations))

    if np.sum(activations > 0) == 1:
        winner_idx = np.argmax(activations)
        return winner_idx, history
    else:
        return -1, history

  def predict(self, X):
    """Класифікує вхідні вектори"""
    X_arr = np.array(X)
    if X_arr.ndim == 1:
        X_arr = X_arr.reshape(1, -1)
    num_samples = X_arr.shape[0]
    predictions = []
    self.maxnet_history_.clear()
    scores_matrix = np.dot(X_arr, self.W_.T) + self.b_
    for i in range(num_samples):
      sample_scores = scores_matrix[i]
      winner_idx, history = self._maxnet(sample_scores)
      self.maxnet_history_[i] = history
      if winner_idx != -1:
          predictions.append(self.classes_[winner_idx])
      else:
          predictions.append("Unknown")
    return np.array(predictions)

model = HammingNetwork()
```

4. Оцінка якості

Оцінка якості представляє собою оцінювання збудованої мережі на згенерованому сеті, який ми наповнили шумом.

```
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import seaborn as sns

model.fit(prototypes, classes)
```

```
y_pred = model.predict(test_set)

true_class_names = classes[true_labels]

all_labels = list(classes)

print("Загальний звіт по класифікації (для всіх рівнів шуму):")
print(classification_report(
    true_class_names,
    y_pred,
    labels=all_labels,
    target_names=all_labels,
    zero_division=0
))

accuracies = []
print("Точність (Accuracy) в залежності від рівня шуму:")
for noise in noise_levels:
    indices = np.where(sample_noise_levels == noise)[0]

    acc = accuracy_score(true_class_names[indices], y_pred[indices])
    accuracies.append(acc)
    print(f"  Рівень шуму {noise} (інвертовано бітів): {acc * 100:.2f}%")
```

У результаті експериментів отримано звіт про класифікацію для всіх рівнів шуму. Метрика precision показує, наскільки модель впевнено робить правильні передбачення (частка правильних серед усіх передбачених позитивних). Recall відображає, яку частку реальних позитивних прикладів модель змогла знайти. F1-score — це середнє гармонійне між precision і recall, яке показує баланс між точністю та повнотою. Середні результати (precision ≈ 0.50, recall ≈ 0.35, f1-score ≈ 0.40) свідчать про помірну ефективність класифікації в умовах шуму.
Аналіз точності залежно від рівня шуму показав: без шуму модель класифікує ідеально (100%), але зі збільшенням кількості інвертованих бітів точність різко падає — до 37.5%, 3.25% і зрештою 0%.

```
Загальний звіт по класифікації (для всіх рівнів шуму):
                   precision    recall  f1-score   support

      Board is OK       0.65      0.25      0.36       400
    Short circuit       0.44      0.39      0.42       400
     Open circuit       0.44      0.39      0.42       400
Missing component       0.45      0.37      0.41       400

        micro avg       0.47      0.35      0.40      1600
        macro avg       0.50      0.35      0.40      1600
     weighted avg       0.50      0.35      0.40      1600

Точність (Accuracy) в залежності від рівня шуму:
  Рівень шуму 0 (інвертовано бітів): 100.00%
  Рівень шуму 1 (інвертовано бітів): 37.50%
  Рівень шуму 2 (інвертовано бітів): 3.25%
  Рівень шуму 3 (інвертовано бітів): 0.00%
```

5. Побудування візуалізацій

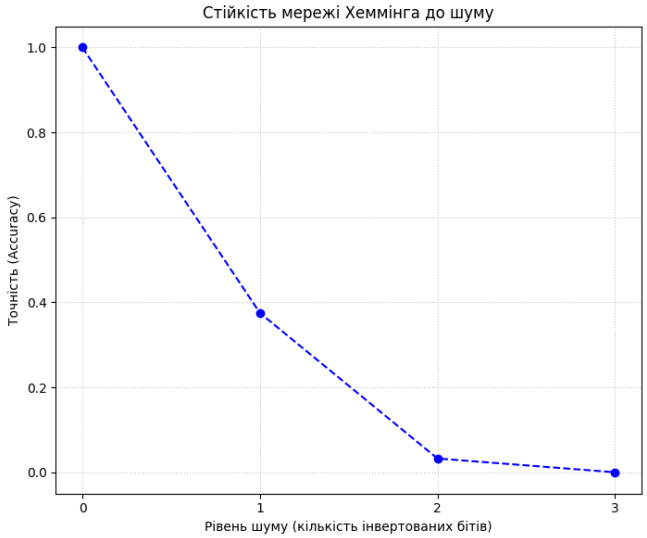
Для аналізу результатів навчання та стійкості мережі було побудовано кілька графіків і теплових карт:

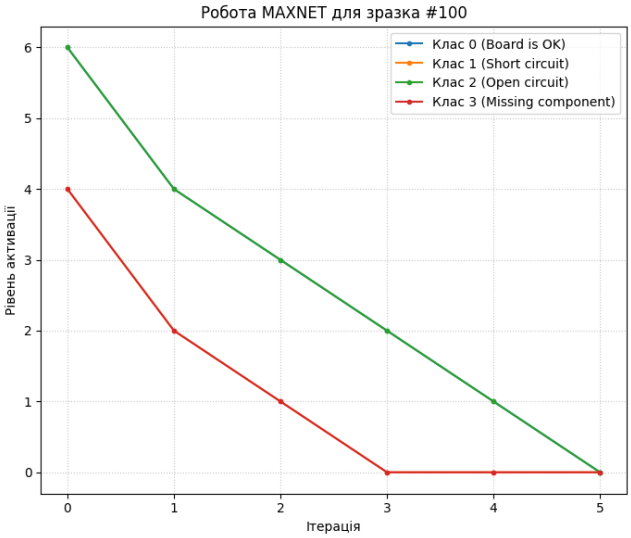
- Графік стійкості мережі Хеммінга до шуму (ліворуч) показує, як точність класифікації знижується зі збільшенням рівня шуму. Це дозволяє оцінити здатність моделі працювати при спотворених вхідних даних.
- Графік роботи MAXNET (праворуч) демонструє процес зниження активацій усіх нейронів, крім переможця, що візуалізує принцип конкурентного відбору у мережі.
- Матриця плутанини показує, які класи модель найчастіше плутає між собою. Найбільші значення на головній діагоналі свідчать про правильні класифікації, тоді як позадіагональні — про помилки.
- Графік точності по класах залежно від рівня шуму дозволяє побачити, які типи дефектів найбільш чутливі до шуму. Зокрема, класи «Short circuit» та «Open circuit» зберігають відносно вищу точність при малому шумі, тоді як «Board is OK» деградує швидше.

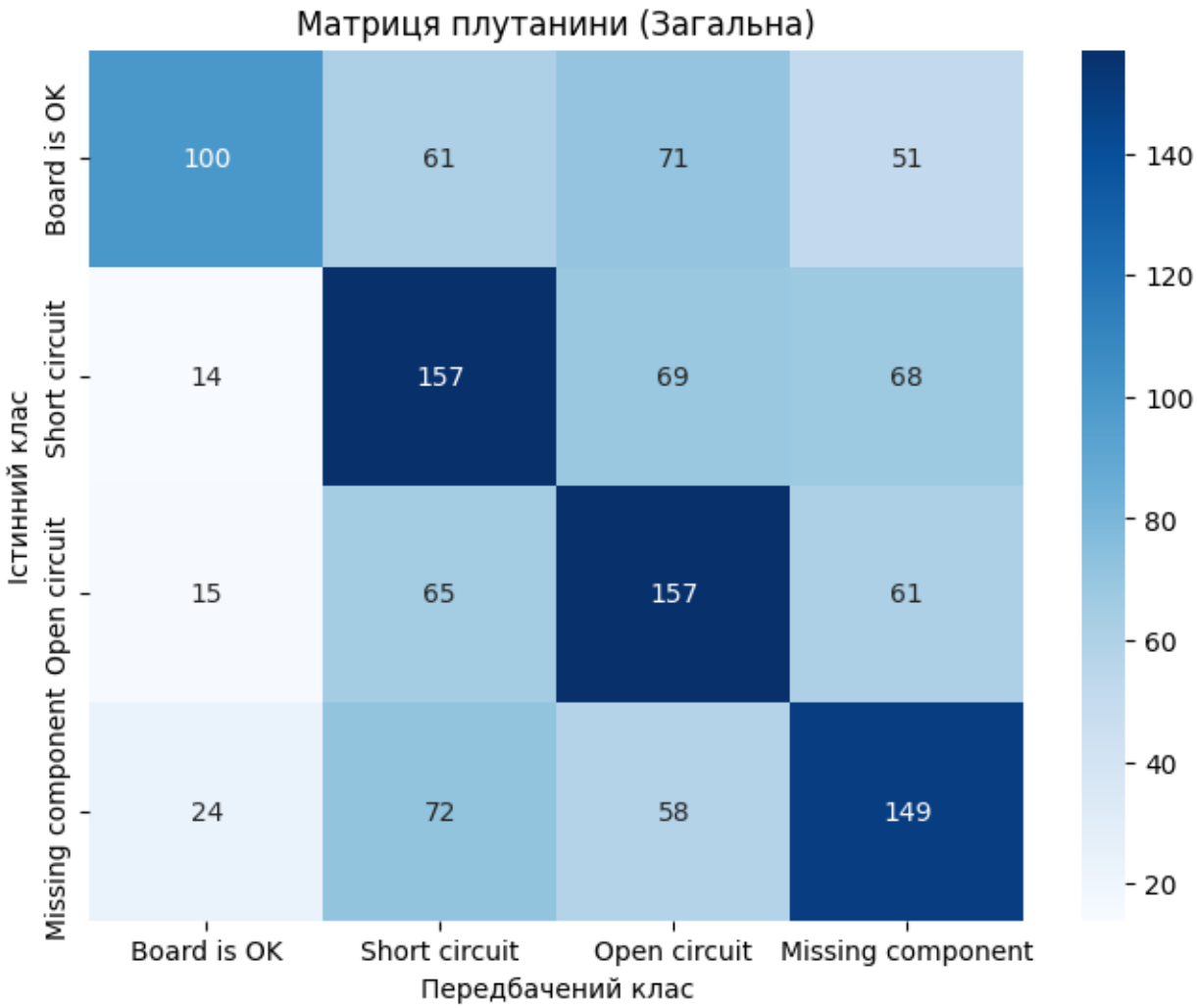
```
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.plot(noise_levels, accuracies, marker='o', linestyle='--', color='b')
plt.title('Стійкість мережі Хеммінга до шуму')
plt.xlabel('Рівень шуму (кількість інвертованих бітів)')
plt.ylabel('Точність (Accuracy)')
plt.xticks(noise_levels)
plt.ylim(-0.05, 1.05)
plt.grid(True, linestyle=':', alpha=0.7)

vis_idx = -1
for i in range(len(y_pred)):
    if sample_noise_levels[i] == 1 and true_class_names[i] != y_pred[i]:
        vis_idx = i
        break

if vis_idx == -1:
    print("Не знайдено неправильно класифікованого зразка з шумом 1, беру
перший зразок з шумом 1.")
    vis_idx = np.where(sample_noise_levels == 1)[0][0]

plt.subplot(1, 2, 2)
history = np.array(model.maxnet_history_[vis_idx])
initial_scores = history[0]
true_class = true_class_names[vis_idx]
pred_class = y_pred[vis_idx]
input_vector = test_set[vis_idx]

for k in range(K):
    plt.plot(history[:, k], marker='.', label=f'Клас {k} ({classes[k]})')
plt.title(f'Робота MAXNET для зразка #{vis_idx}')
plt.xlabel('Ітерація')
plt.ylabel('Рівень активації')
plt.legend()
plt.grid(True, linestyle=':', alpha=0.7)
plt.tight_layout()
```

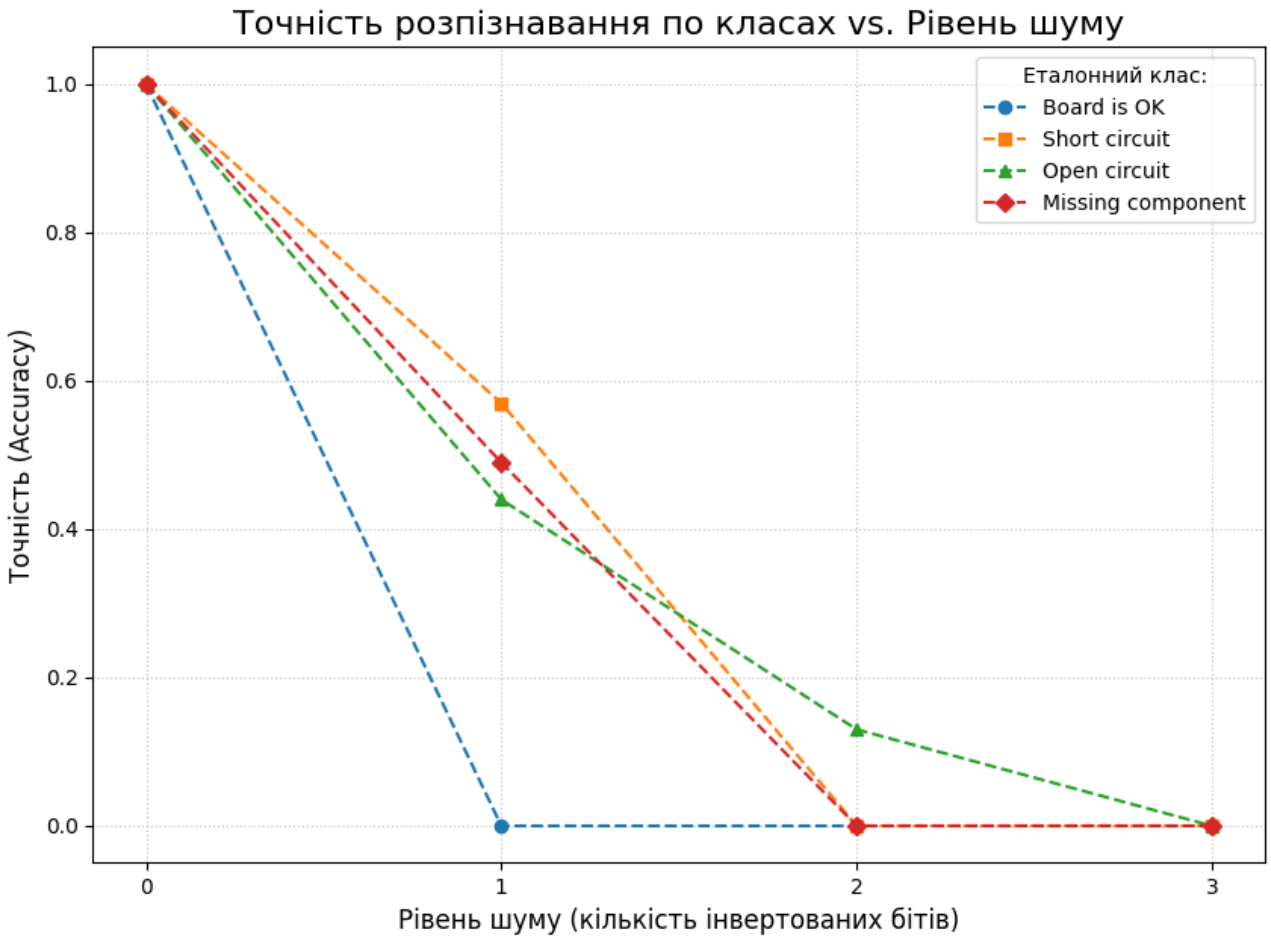
```
plt.show()

plt.figure(figsize=(8, 6))
cm = confusion_matrix(true_class_names, y_pred, labels=classes)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=classes, yticklabels=classes)
plt.title('Матриця плутанини (Загальна)')
plt.xlabel('Передбачений клас')
plt.ylabel('Істинний клас')
plt.show()
class_to_idx = {name: i for i, name in enumerate(classes)}
class_to_idx['Unknown'] = -1

y_pred_indices = np.array([class_to_idx[name] for name in y_pred])

class_accuracies_vs_noise = []
for k in range(K): # k = 0, 1, 2, 3
    accs_for_this_class = []

    for noise in noise_levels: # noise = 0, 1, 2, 3
        true_indices = np.where((true_labels == k) & (sample_noise_levels ==
noise))[0]

        if len(true_indices) == 0:
            accs_for_this_class.append(0.0)
            continue

        true_subset = true_labels[true_indices]
        pred_subset = y_pred_indices[true_indices]

        acc = accuracy_score(true_subset, pred_subset)
        accs_for_this_class.append(acc)

    class_accuracies_vs_noise.append(accs_for_this_class)

plt.figure(figsize=(10, 7))
markers = ['o', 's', '^', 'D']

for k in range(K):
    plt.plot(noise_levels, class_accuracies_vs_noise[k],
             marker=markers[k],
             linestyle='--',
             label=classes[k])

plt.title('Точність розпізнавання по класах vs. Рівень шуму', fontsize=16)
plt.xlabel('Рівень шуму (кількість інвертованих бітів)', fontsize=12)
plt.ylabel('Точність (Accuracy)', fontsize=12)
plt.xticks(noise_levels)
plt.ylim(-0.05, 1.05)
plt.legend(title="Еталонний клас:", loc='upper right')
plt.grid(True, linestyle=':', alpha=0.7)
plt.show()
```

У результаті маємо, що модель добре працює при чистих даних, але зі збільшенням рівня шуму точність різко зменшується. Основною проблемою є нормальний клас у якого всі біти однакові, тому при зміні любого біта ми не зможемо вгадати нормальний клас.









Висновки

У ході виконання лабораторної роботи було розроблено та досліджено модель класифікації на основі мережі Хеммінга з конкурентним шаром MAXNET. Було сформовано набір еталонних образів, що імітують типові стани друкованих плат, та згенеровано тестові дані з різними рівнями шуму для оцінки стійкості системи.

Отримані результати показали, що модель гарантовано класифікує дані без шуму (100%), однак її точність суттєво знижується при зростанні рівня інверсій бітів: до 37.5% при одному спотворенні, 3.25% при двох і 0% при трьох. Аналіз матриці плутанини підтвердив, що основні помилки виникають через близькість або симетричність деяких векторів, особливо класу «Board is OK», який є найчутливішим до шуму.

Таким чином, мережа Хеммінга добре підходить для класифікації чітких або слабо зашумлених сигналів, однак потребує удосконалення для роботи з сильними спотвореннями. Для підвищення точності в майбутньому можна застосувати попереднє фільтрування, регуляризацію або модифікації архітектури, що враховують кореляцію між елементами вектора.