

МІНІСТЕРСТВО НАУКИ ТА ОСВІТИ УКРАЇНИ

Київський авіаційний інститут

Факультет комп'ютерних наук та технологій

Кафедра прикладної математики

Модульна робота №2

Тема: «Computer Vision Project»

З дисципліни «Використання мови Python для глибокого навчання»

Виконав студент групи

Б-122-23-1-ШІ:

Щербинський Д.Г.

Прийняв:

Зівакін В.Д.

Київ-2025

## Зміст

<b>1. Постановка задачі</b>	2
<b>2. Вступ</b>	4
<b>3. Теоретична частина</b>	6
3.1. Основи глибокого навчання	6
3.2 Згорткові нейронні мережі	7
3.3. Transfer Learning	9
3.4 Регуляризація	10
3.5. Аугментація	12
3.4. Автоенкодери	13
3.5. Метрики оцінювання	14
<b>4. Практична частина</b>	17
4.1. <u>Етап 1: Вибір та підготовка датасету</u>	17
4.2. <u>Етап 2: Baseline CNN Model</u>	17
4.3. <u>Етап 3: Transfer Learning</u>	26
4.4. <u>Етап 4: Data Augmentation та Regularization</u>	36
4.5. <u>Етап 5: Автоенкодер для датасету</u>	52
<b>5. Висновок</b>	57
<b>6. Використані джерела</b>	59
<b>7. Додатки</b>	60

## 1. Постановка задачі

### **Мета дослідження**

Розробити та порівняти різні підходи глибокого навчання для автоматичної класифікації діабетичної ретинопатії на основі зображень очного дна.

Дослідження спрямоване на створення ефективної системи діагностики, яка може допомогти у ранньому виявленні та класифікації стадій діабетичної ретинопатії.

### **Опис проблеми**

Діабетична ретинопатія є одним із найпоширеніших ускладнень цукрового діабету та провідною причиною втрати зору серед дорослого населення.

Своєчасна діагностика та класифікація стадії захворювання мають критичне значення для запобігання незворотній втраті зору. Традиційні методи діагностики вимагають експертної оцінки офтальмолога, що є часозатратним процесом та обмежує доступність скринінгу у віддалених регіонах.

### **Характеристики датасету**

- **Джерело:** Kaggle - Diabetic Retinopathy DL
- **Кількість класів:** 5 (різні стадії діабетичної ретинопатії)
  - Клас 0: Відсутність ретинопатії
  - Клас 1: Легка непроліферативна ретинопатія
  - Клас 2: Помірна непроліферативна ретинопатія
  - Клас 3: Важка непроліферативна ретинопатія
  - Клас 4: Проліферативна ретинопатія
- **Кількість зображень:** 2750
- **Дані:** RGB-зображення очного дна,  $256 \times 256$  пікселів

### **Основні завдання дослідження**

#### **1. Розробка базової CNN-моделі**

- Створення власної згорткової нейронної мережі з нуля
- Навчання моделі протягом мінімум 20 епох
- Реалізація механізму раннього зупинення (early stopping)
- Досягнення точності понад 60% на валідаційній вибірці
- Побудова та аналіз кривих навчання

#### **2. Імплементація Transfer Learning**

- Порівняння мінімум двох pretrained моделей (ResNet, VGG, EfficientNet)
- Дослідження двох підходів: feature extraction vs fine-tuning
- Експериментування з різними швидкостями навчання (learning rates)
- Застосування диференційованих learning rates для різних шарів мережі

### **3. Аугментація даних та регуляризація**

- Порівняння базової та розширеної аугментації даних
- Дослідження впливу різних методів регуляризації:
  - Dropout
  - L2-регуляризація
  - Label Smoothing
- Створення ансамблю моделей для покращення точності

### **4. Розробка автоенкодера**

Створення автоенкодера з практичними застосуваннями:

- Детекція аномалій у зображеннях датасету
- Візуалізація латентного простору для аналізу структури даних

### **5. Порівняльний аналіз**

Комплексне порівняння всіх розроблених підходів за критеріями:

- Точність на валідаційній вибірці
- Точність на тестовій вибірці
- Розмір моделі (кількість параметрів)
- Час навчання
- Побудова матриці помилок для найкращої моделі

### **Очікувані результати**

1. Функціональна система класифікації діабетичної ретинопатії з точністю понад 65%
2. Детальний порівняльний аналіз різних архітектур та підходів
3. Рекомендації щодо оптимального вибору моделі для практичного застосування
4. Візуалізація результатів та інтерпретація помилок моделі
5. Insights щодо структури даних через аналіз латентного простору автоенкодера

## 2. Вступ

### Актуальність дослідження

Діабетична ретинопатія (ДР) є серйозним ускладненням цукрового діабету, яке уражає кровоносні судини сітківки ока і залишається однією з провідних причин втрати зору серед працездатного населення у всьому світі. За даними Всесвітньої організації охорони здоров'я, приблизно 422 мільйони людей живуть з діабетом, і значна частина з них ризикує розвинути діабетичну ретинопатію. Без своєчасної діагностики та лікування, це захворювання може привести до незворотної сліпоти.

Традиційний процес діагностики діабетичної ретинопатії вимагає ретельного огляду зображень очного дна досвідченим офтальмологом, що є трудомістким, часозатратним та вимагає високої кваліфікації спеціаліста. В умовах зростаючої кількості пацієнтів з діабетом та обмеженої кількості спеціалістів, особливо у віддалених та недостатньо забезпечених регіонах, існує критична потреба в автоматизованих системах скринінгу.

### Роль штучного інтелекту в медичній діагностиці

Останні досягнення в галузі глибокого навчання та комп'ютерного зору відкрили нові можливості для автоматизації медичної діагностики. Згорткові нейронні мережі (CNN) продемонстрували видатні результати в аналізі медичних зображень, часом досягаючи точності, порівнянної з експертною оцінкою лікарів. Застосування методів глибокого навчання до діагностики діабетичної ретинопатії може значно прискорити процес скринінгу, зменшити навантаження на медичних працівників та забезпечити більш широкий доступ до якісної діагностики.

### Підходи до вирішення задачі

У цьому дослідженні розглядаються декілька сучасних підходів до класифікації медичних зображень:

**Базові CNN-моделі** дозволяють зрозуміти фундаментальні принципи роботи згорткових мереж та встановити benchmark для подальших експериментів. Створення власної архітектури з нуля надає повний контроль над процесом навчання та допомагає виявити специфічні характеристики датасету.

**Transfer Learning** використовує знання, здобуті моделями на великих загальних датасетах (таких як ImageNet), та адаптує їх до специфічної медичної задачі. Цей підхід особливо ефективний при обмежених обсягах даних, що є типовою ситуацією в медичній сфері, де створення великих анотованих датасетів вимагає значних ресурсів та експертизи.

**Аугментація даних та регуляризація** допомагають підвищити узагальнючу здатність моделей та запобігти перенавчанню. Медичні зображення часто характеризуються великою варіативністю через різні умови знімання, обладнання та індивідуальні особливості пацієнтів, що робить аугментацію критично важливою.

**Автоенкодери** надають унікальну можливість для аналізу структури даних, виявлення аномалій та розуміння латентних представлень зображень. Це може допомогти у виявленні нетипових випадків та поліпшенні якості датасету.

## Структура дослідження

Дослідження організовано послідовно, від простих до складних підходів:

1. Спочатку розробляється базова CNN-модель для встановлення benchmark та розуміння специфіки датасету
2. Далі досліджуються pretrained моделі з різними стратегіями transfer learning
3. Проводиться систематичне дослідження впливу аугментації та регуляризації
4. Створюється автоенкодер для глибшого аналізу структури даних
5. Всі підходи порівнюються за об'єктивними метриками

Такий підхід дозволяє не лише знайти найкращу модель для конкретної задачі, але й зрозуміти, які фактори та техніки найбільше впливають на якість класифікації діабетичної ретинопатії.

## Наукова та практична значущість

Результати цього дослідження мають як наукову, так і практичну цінність. З наукової точки зору, робота демонструє систематичне порівняння різних архітектур та методів глибокого навчання на реальній медичній задачі, що може бути корисним для дослідників у галузі медичного комп'ютерного зору.

З практичної сторони, розроблена система може стати основою для реальних діагностичних інструментів, які допоможуть лікарям у скринінгу великих груп пацієнтів, ранньому виявленні захворювання та пріоритизації випадків, що потребують термінової уваги. Особливо цінною така система може бути для телемедицини та мобільних діагностичних центрів у регіонах з обмеженим доступом до спеціалізованої офтальмологічної допомоги.

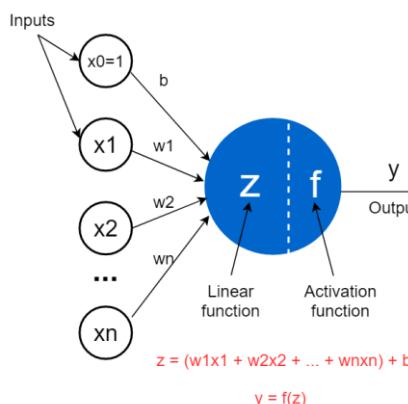
### 3. Теоретична частина

#### 3.1 Основи глибокого навчання

**Глибоке навчання** (Deep Learning) - це підгалузь машинного навчання, що використовує багатошарові нейронні мережі для автоматичного вилучення ознак з даних. На відміну від традиційних методів машинного навчання, де ознаки проектуються вручну, глибокі нейронні мережі самостійно навчаються ієрархічним представленням даних: від простих ознак на нижніх рівнях до складних абстракцій на вищих.

#### Штучний нейрон

Базовою одиницею нейронної мережі є штучний нейрон (перцептрон), який моделює роботу біологічного нейрона.



де:

- $x_1, x_2, \dots, x_n$  - вхідні сигнали
- $w_1, w_2, \dots, w_n$  - ваги (навчальні параметри)
- $b$  - зміщення (bias)
- $f$  - функція активації
- $y$  - вихідний сигнал

Функція активації вносить нелінійність у модель, що дозволяє мережі моделювати складні залежності. Найпоширеніші функції активації:

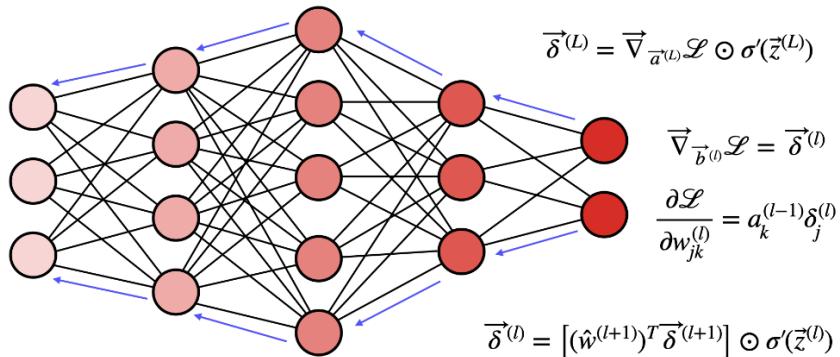
- **ReLU** (Rectified Linear Unit):  $f(x) = \max(0, x)$  - найпопулярніша у глибоких мережах
- **Sigmoid**:  $f(x) = 1/(1 + e^{-x})$  - використовується для бінарної класифікації
- **Softmax**: використовується на виході для багатокласової класифікації
- **Tanh**:  $f(x) = (e^x - e^{-x})/(e^x + e^{-x})$  - центрована навколо нуля

#### Процес навчання

Навчання нейронної мережі відбувається ітеративно через алгоритм **зворотного поширення помилки** (backpropagation):

1. **Forward pass**: дані подаються на вхід мережі, проходять через всі шари, і генерується передбачення
2. **Обчислення функції втрат**: порівнюється передбачення з реальними мітками

3. **Backward pass:** градієнти помилки обчислюються та поширюються назад через мережу, за допомогою chain rule (правило знаходження похідної складеної функції)



4. **Оновлення ваг:** параметри мережі коригуються у напрямку, що зменшує

$$w_0 = w_0 - \eta \frac{1}{L} \sum_{i=1}^L (w \cdot x_i - y_i),$$

$$w_j = w_j - \eta \frac{1}{L} \sum_{i=1}^L (w \cdot x_i - y_i) \cdot x_{ij}.$$

Найпоширеніші функції втрат для класифікації:

- **Cross-Entropy Loss** - для багатокласової класифікації
- **Binary Cross-Entropy** - для бінарної класифікації

Для оптимізації використовуються різні алгоритми:

- **SGD (Stochastic Gradient Descent)** - базовий метод
- **Adam** - адаптивний метод, що зберігає окремі швидкості навчання для кожного параметра
- **RMSprop, AdaGrad** - інші адаптивні методи

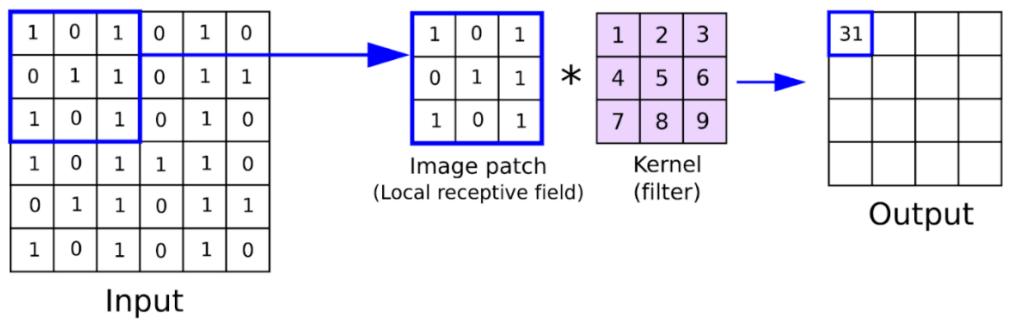
### 3.2. Згорткові нейронні мережі (CNN)

**Згорткові нейронні мережі** є спеціалізованим типом нейронних мереж, розробленим спеціально для обробки даних з сіткоподібною топологією, таких як зображення. CNN революціонізували комп'ютерний зір, досягнувши результатів, що перевершують людські, у багатьох задачах розпізнавання образів.

## Ключові компоненти CNN

### 1. Згортковий шар (Convolutional Layer)

Згортковий шар застосовує набір навчальних фільтрів (ядер) до вхідного зображення. Кожен фільтр "ковзає" по зображеню та обчислює скалярний добуток між своїми вагами та локальною областю зображення.



$$\text{Вихід}[i,j] = \sum \sum (\text{Вхід}[i+m, j+n] \times \text{Фільтр}[m,n]) + \text{bias}$$

Основні переваги:

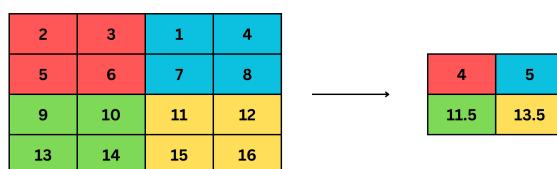
- **Локальна зв'язність:** кожен нейрон з'єднаний лише з локальною областю
- **Спільне використання параметрів:** той самий фільтр застосовується до всього зображення
- **Інваріантність до зсувів:** виявляє ознаки незалежно від їх позиції

### 2. Pooling шар

Pooling зменшує просторову розмірність feature maps, занижуючи обчислювальну складність та контролюючи перенавчання:

- **Max Pooling:** бере максимальне значення з кожної області
- **Average Pooling:** обчислює середнє значення

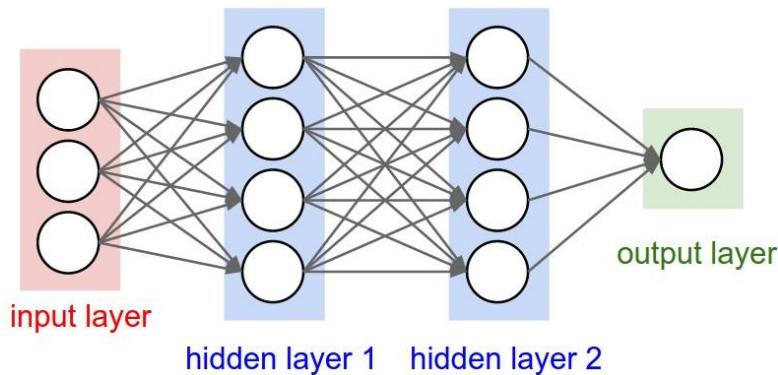
### Average Pooling



*Take average of all values in the window*

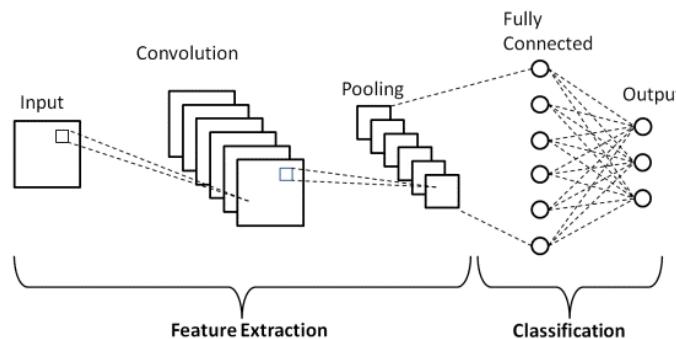
### 3. Fully Connected шар

Після декількох згорткових та pooling шарів, feature maps розгортаються у вектор та подаються на повнозв'язні шари для фінальної класифікації.



### Типова архітектура CNN

Вхід  $(256 \times 256 \times 3)$   $\rightarrow$  [Conv + ReLU + MaxPool]  $\times$  n разів  $\rightarrow$  Flatten  $\rightarrow$   
 $\rightarrow$  [Fully Connected + ReLU]  $\times$  m разів  $\rightarrow$  Output (Softmax для класифікації)



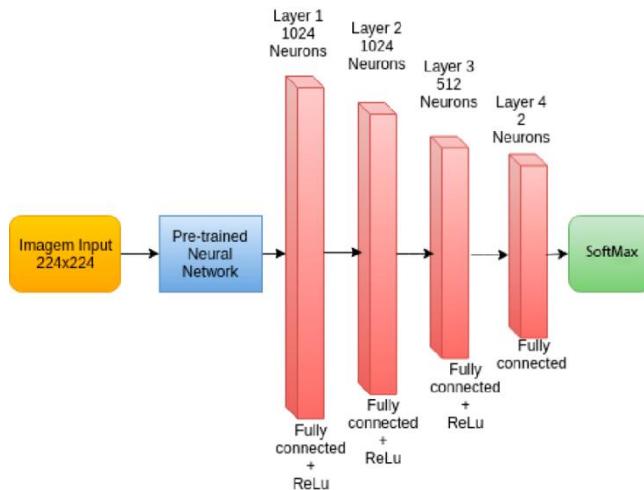
Перші шари виявляють прості ознаки (краї, кути, текстури), а глибші шари комбінують їх у складніші паттерни (форми, об'єкти).

### 3.3. Transfer Learning

**Transfer Learning** - це техніка, при якій модель, навчена на одній задачі, адаптується для вирішення іншої, пов'язаної задачі. Це особливо ефективно в медичній візуалізації, де великих аnotatedованих датасетів рідкісні.

#### Чому Transfer Learning працює?

Нижні шари CNN вивчають загальні ознаки (краї, текстури, кольори), які є корисними для широкого спектру задач комп'ютерного зору. Вищі шари вивчають більш специфічні ознаки для конкретної задачі.



## Два основні підходи

### 1. Feature Extraction

- Заморожуємо ваги pretrained моделі
- Видаляємо фінальні класифікаційні шари
- Додаємо нові шари для нашої задачі
- Навчаємо тільки нові шари

[Pretrained Backbone - FROZEN]



[Custom Classifier - TRAINABLE]

Переваги: швидше навчання, менше ризик перенавчання

### 2. Fine-Tuning

- Ініціалізуємо модель pretrained вагами
- "Розморожуємо" частину або всі шари
- Навчаємо з малою швидкістю навчання

[Pretrained Backbone - TRAINABLE з низьким LR]



[Custom Classifier - TRAINABLE з високим LR]

Переваги: потенційно вища точність, адаптація до специфіки датасету

## Популярні pretrained моделі

### ResNet (Residual Networks)

- Використовує residual connections (skip connections)

- Вирішує проблему зникаючого градієнта
- Варіанти: ResNet-50, ResNet-101, ResNet-152

## VGG (Visual Geometry Group)

- Проста архітектура з однорідними блоками
- Використовує малі  $3 \times 3$  фільтри
- VGG-16, VGG-19 — класичні моделі

## EfficientNet

- Збалансоване масштабування глибини, ширини та роздільної здатності
- Висока точність при меншій кількості параметрів
- EfficientNet-B0 до B7

## MobileNet

- Оптимізована для мобільних пристройів
- Використовує depthwise separable convolutions
- Малий розмір та швидка інференція

## 3.4. Регуляризація та запобігання перенавчанню

**Перенавчання** (overfitting) виникає, коли модель надто добре вивчає тренувальні дані, включаючи шум та випадкові флюктуації, що погіршує її продуктивність на нових даних.

### Методи регуляризації

**1. Dropout** Випадково "вимикає" частину нейронів під час навчання з заданою ймовірністю  $p$  (типово 0.5):

```
output = input * random_mask(p=0.5) # Під час навчання
output = input * (1 - p) # Під час тестування
```

Це змушує мережу не покладатися на окремі нейрони та вивчати більш робастні ознаки.

### 2. L2 Regularization (Weight Decay)

Додає штраф до функції втрат за великі ваги:

$$\text{Loss\_total} = \text{Loss\_classification} + \lambda \times \sum(w_i^2)$$

Це стимулює модель використовувати менші ваги, що покращує узагальнення.

### 3. Batch Normalization

Нормалізує активації між шарами:

$$x_{\text{norm}} = (x - \mu) / \sqrt(\sigma^2 + \epsilon)$$

Це прискорює навчання та діє як регуляризатор.

### 4. Early Stopping

Зупиняє навчання, коли продуктивність на валідаційній вибірці перестає покращуватися протягом певної кількості епох (patience).

### 5. Label Smoothing

М'яко розподіляє ймовірність між класами замість hard labels:

- # Замість [0, 0, 1, 0, 0]
- # Використовує [0.025, 0.025, 0.9, 0.025, 0.025]

Це запобігає надмірній впевненості моделі.

### 3.5. Аугментація даних

**Аугментація даних** - це техніка штучного розширення тренувального датасету шляхом створення модифікованих версій існуючих зображень. Це особливо важливо при обмежених обсягах даних.

#### Геометричні трансформації

- **Обертання** (rotation): поворот зображення на випадковий кут
- **Відображення** (flipping): горизонтальне/вертикальне відображення
- **Масштабування** (scaling): збільшення/зменшення
- **Зсув** (translation): переміщення зображення
- **Обрізання** (cropping): випадкове вирізання частини зображення

#### Колірні трансформації

- **Зміна яскравості** (brightness adjustment)
- **Зміна контрасту** (contrast adjustment)
- **Зміна насиченості** (saturation adjustment)
- **Зміна відтінку** (hue adjustment)

#### Спеціалізовані методи

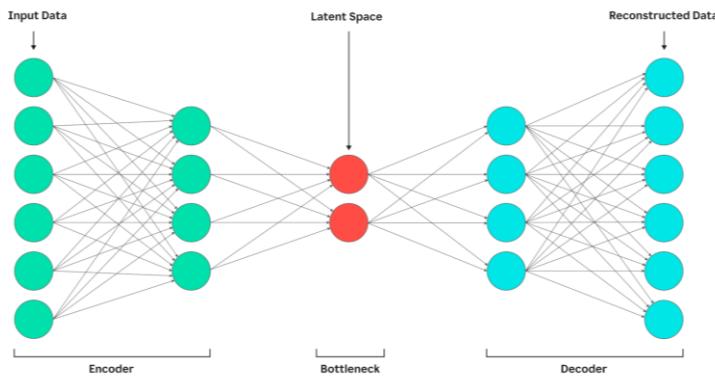
- **Cutout**: випадкове маскування прямокутних областей
- **Mixup**: створення зважених комбінацій зображень та їх міток
- **CutMix**: комбінування частин різних зображень

Для медичних зображень важливо використовувати лише клінічно прийнятні аугментації, які не змінюють діагностичні ознаки.

### 3.6. Автоенкодери

**Автоенкодер** - це нейронна мережа, що навчається стискати (кодувати) вхідні дані у компактне латентне представлення та відновлювати (декодувати) їх назад.

#### Архітектура



**Encoder** поступово зменшує розмірність:

$256 \times 256 \rightarrow 128 \times 128 \rightarrow 64 \times 64 \rightarrow 32 \times 32 \rightarrow$  латентний вектор

**Decoder** відновлює зображення у зворотному порядку:

латентний вектор  $\rightarrow 32 \times 32 \rightarrow 64 \times 64 \rightarrow 128 \times 128 \rightarrow 256 \times 256$

#### Функція втрат

Типова функція втрат вимірює різницю між вхідним зображенням та його реконструкцією:

$\text{Loss} = \text{MSE}(\text{вхід}, \text{реконструкція}) = \sum (x_i - \hat{x}_i)^2$  (або Binary Cross-Entropy для нормалізованих зображень)

#### Застосування

##### 1. Детекція аномалій

Автоенкодер навчається на "нормальних" зображеннях. Аномальні зображення матимуть високу помилку реконструкції, оскільки модель не бачила схожих паттернів під час навчання.

##### 2. Зменшення розмірності

Латентний простір містить стиснуте представлення даних, яке можна візуалізувати за допомогою t-SNE або PCA для аналізу структури датасету.

##### 3. Деноїзінг (видалення шуму)

Автоенкодер навчається відновлювати чисті зображення з зашумлених, вивчаючи важливі структурні ознаки.

##### 4. Variational Autoencoder (VAE)

VAE моделює латентний простір як розподіл ймовірностей, що дозволяє генерувати нові реалістичні зображення через семплювання з латентного простору.

### Особливості для медичних даних

- В медичній візуалізації автоенкодери можуть:
- Виявляти рідкісні патології як аномалії
- Знаходити схожі випадки через латентний простір
- Очищати зображення від артефактів
- Аналізувати прогресію захворювання

### 3.7. Метрики оцінювання

Щоб оцінити якість класифікаційної моделі, використовують кілька ключових метрик. Вони базуються на так званій **матриці помилок** (*confusion matrix*), яка має такі поняття:

#### Пояснення термінів (TP, TN, FP, FN)

Позначення	Назва	Що означає
<b>TP (True Positive)</b>	Істинно позитивні	Модель правильно передбачила клас “позитивний”, і в реальноті цей об’єкт дійсно позитивний.
<b>TN (True Negative)</b>	Істинно негативні	Модель правильно передбачила клас “негативний”, і в реальноті об’єкт дійсно негативний.
<b>FP (False Positive)</b>	Хибно позитивні	Модель передбачила “позитивний”, але в реальноті об’єкт <b>негативний</b> . (помилка I роду)
<b>FN (False Negative)</b>	Хибно негативні	Модель передбачила “негативний”, але в реальноті об’єкт <b>позитивний</b> . (помилка II роду)

#### Accuracy

$$\text{Accuracy} = (\text{TP} + \text{TN}) / (\text{TP} + \text{TN} + \text{FP} + \text{FN})$$

Показує, яку частку всіх передбачень модель зробила правильно.  
Добра, коли класи збалансовані. Невдала метрика для задачі з дисбалансом даних.

#### Precision (Точність)

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$$

Показує, який відсоток передбачених позитивів є справді позитивами.

Питання, на яке відповідає:

"Якщо модель каже «позитивний», наскільки її можна довіряти?"

## Recall (Повнота, Sensitivity)

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$$

Показує, яку частину реальних позитивів модель змогла знайти.

Питання, на яке відповідає:

"Скільки з усіх реальних позитивних випадків модель виявила?"

## F1-Score

$$\text{F1} = 2 \times (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})$$

Гармонійне середнє precision та recall.

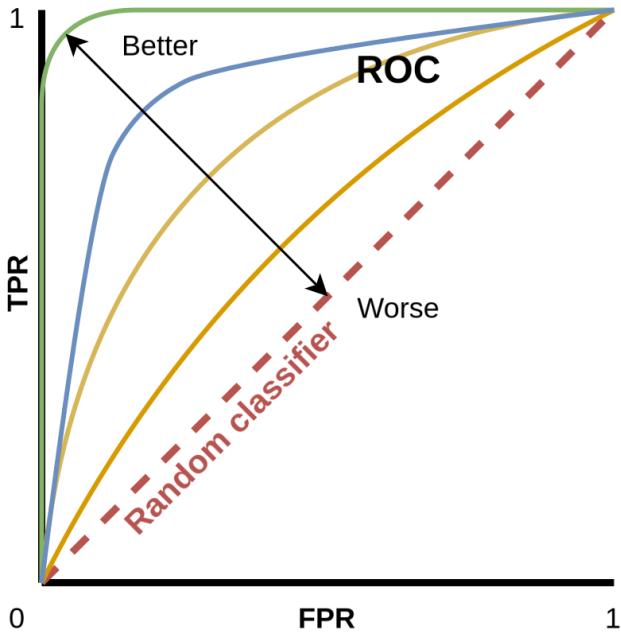
Корисна метрика, коли важливий баланс між FP і FN.

## Confusion Matrix (Матриця помилок)

Показує, як модель класифікує об'єкти:

		Predicted Values	
		Positive	Negative
Actual Values	Positive	TP	FN
	Negative	FP	TN

**ROC-AUC** (для бінарної/багатокласової класифікації) Площа під ROC-кривою (Area Under Curve), що відображає компроміс між **true positive rate** та **false positive rate**.



Для медичних задач особливо важливі **Recall** (щоб не пропустити хворих) та аналіз **Confusion Matrix** (щоб зрозуміти типи помилок).

## 4. Практична частина

### 4.1. Етап 1: Підготовка датасету

#### 4.1.1. Завантаження та первинний аналіз даних

Для дослідження було обрано датасет **Diabetic Retinopathy Detection** з платформи Kaggle, який містить зображення очного дна для діагностики діабетичної ретинопатії.

**Характеристики датасету:**

- Загальна кількість зображень: 2750
- Розмір зображень:  $256 \times 256$  пікселів
- Формат: RGB (3 канали)
- Кількість класів: 5

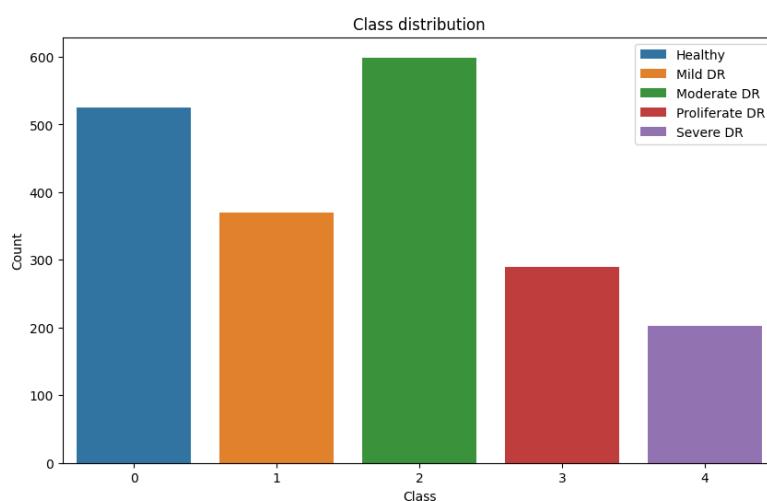
**Структура директорій:**

```
data/
    ├── No_DR/          # Без ретинопатії
    ├── Mild_DR /      # Легка ретинопатія
    ├── Moderate_DR /  # Помірна ретинопатія
    ├── Severe_DR /    # Важка ретинопатія
    └── Proliferative_DR / # Проліферативна ретинопатія
```

#### 4.1.2. Аналіз збалансованості класів

Для оцінки розподілу класів у датасеті було проведено аналіз кількості зображень у кожній категорії:

**Розподіл класів:**

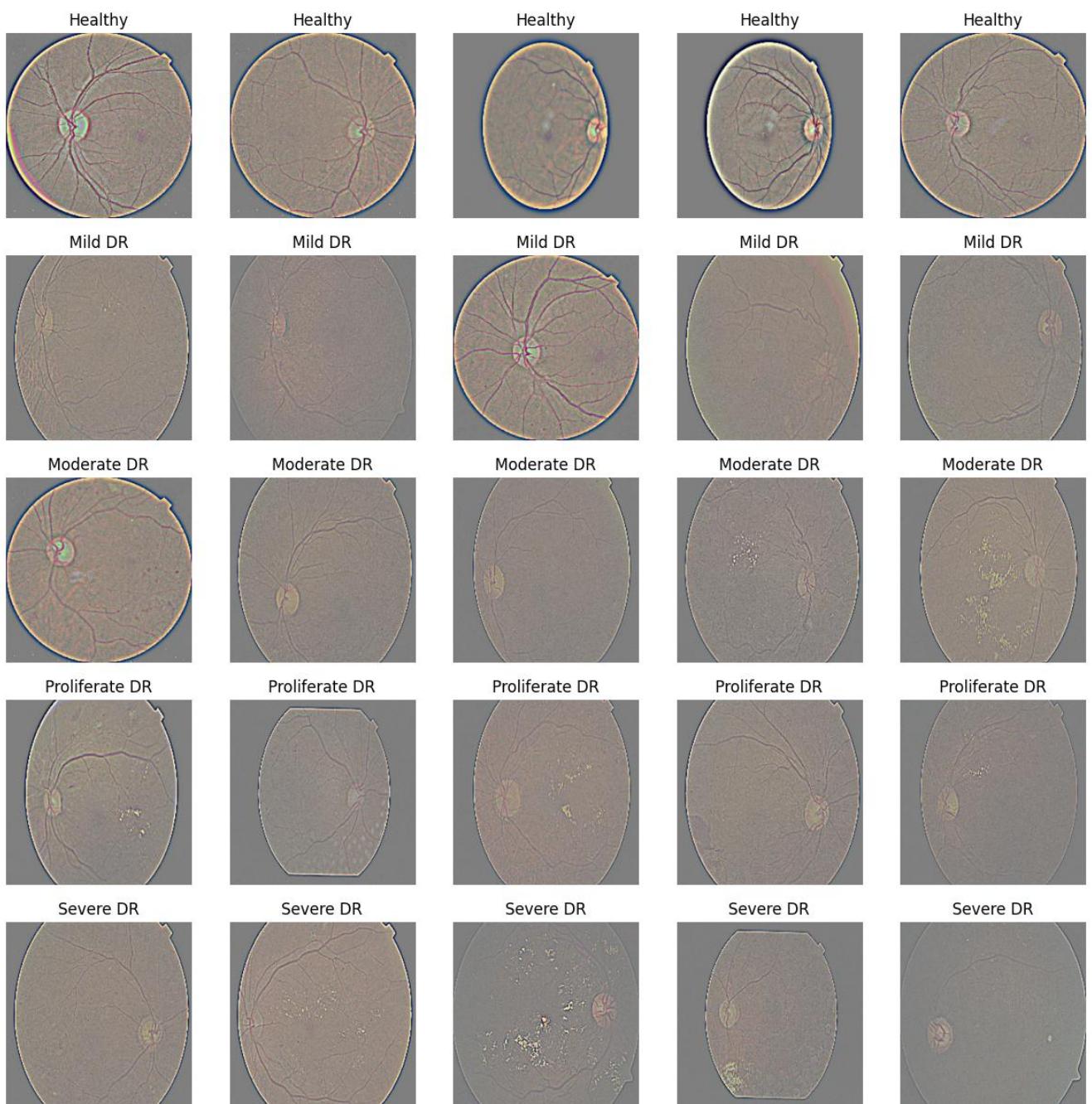


Попри те, що класів не надто багато, розподіл не є ідеально збалансованим: деякі категорії представлені значно частіше (наприклад, Moderate DR), тоді як Severe DR має помітно менше зразків. Така нерівномірність усе ще може впливати на навчання моделі й потребувати додаткових технік на кшталт зважування loss-функції або oversampling для міноритарних класів.

#### 4.1.3. Візуалізація зразків датасету

Для кращого розуміння характеристик кожного класу було візуалізовано по 5 репрезентативних зображень зожної категорії:

##### Приклади зображень по класах:



## Спостереження з візуального аналізу:

- Зображення мають природну варіативність в освітленні та контрасті
- Деякі зображення містять артефакти (відблиски, нерівномірне освітлення)
- Клас 0 має найбільш виразні відмінності – відсутність плям, характерних для ретинотапії, та чіткі кровоносні сосуди.

### 4.1.4. Попередня обробка

#### Нормалізація:

Для стабільного навчання нейронної мережі необхідно нормалізувати вхідні дані. Спочатку зображення конвертуються у тензори зі значеннями в діапазоні [0, 1], після чого застосовується стандартизація з використанням середніх значень та стандартних відхилень, обчислених на тренувальній вибірці.

#### Розділення на вибірки:

Датасет було розділено з використанням стратифікації для збереження пропорцій класів:

- **Тренувальна вибірка:** 70%
- **Валідаційна вибірка:** 15%
- **Тестова вибірка:** 15%

Для забезпечення відтворюваності результатів використано фіксоване значення random\_seed = 17.

#### Базові трансформації:

Для всіх вибірок застосовуються базові трансформації:

```
base_transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[means], std=[stds])
])
```

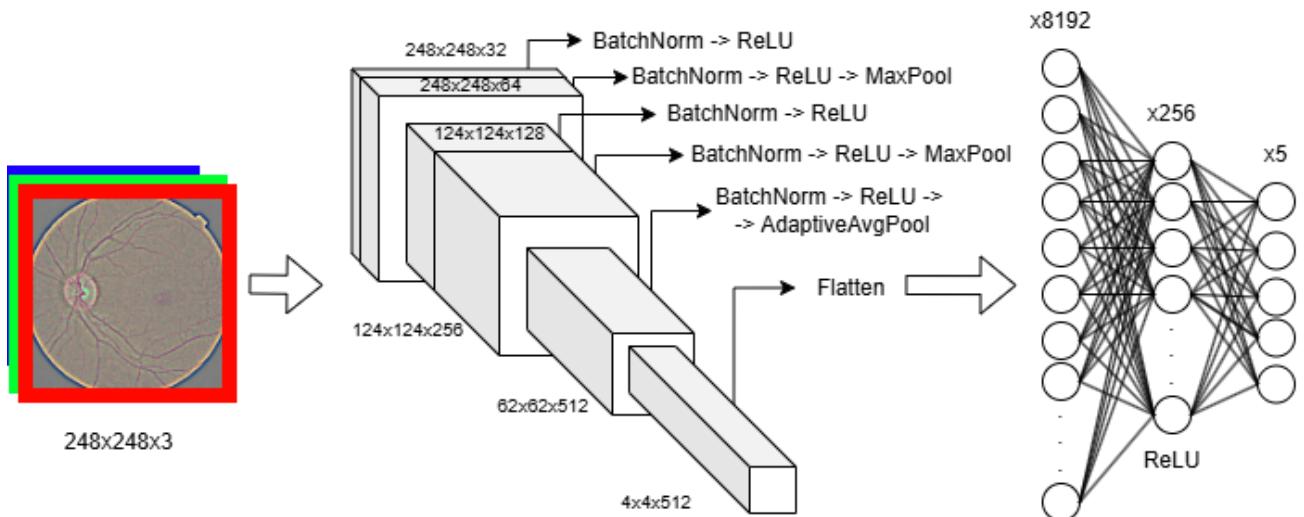
На етапі baseline моделі аугментація **не використовувалась** для встановлення чистого benchmark.

## 4.2. Етап 2: Baseline CNN Model

### 4.2.1. Архітектура власної CNN

Для створення baseline моделі було розроблено власну архітектуру згорткової нейронної мережі 'BaseLineCNN', яка складається з трьох основних компонентів: блоку вилучення ознак, глобального pooling та класифікатора.

**Архітектура моделі** (*код можна подивитися в Додатку 1*):



**Ключові архітектурні рішення:**

1. **Парна структура згорткових шарів:** Використання двох послідовних згорток перед pooling дозволяє моделі вивчати більш складні ознаки на кожному рівні абстракції
2. **Поступове збільшення глибини:**  $32 \rightarrow 64 \rightarrow 128 \rightarrow 256 \rightarrow 512$  фільтрів, що є класичним підходом у CNN архітектурах
3. **Batch Normalization:** Застосовується після кожного згорткового шару для:
  - Стабілізації процесу навчання
  - Зменшення internal covariate shift
  - Прискорення збіжності
  - Регуляризації моделі
4. **AdaptiveAvgPool2d:** Замість послідовних MaxPool шарів після кожного блоку, використано глобальний adaptive pooling, що:
  - Зменшує кількість параметрів
  - Робить архітектуру більш гнучкою до різних розмірів вхідних зображень
  - Зберігає просторову інформацію довше
5. **Компактний класифікатор:** Лише два повнозв'язні шари ( $8192 \rightarrow 256 \rightarrow 5$ ) для ефективної класифікації
6. **Відсутність Dropout:** На baseline етапі dropout було вимкнено для оцінки чистої ємності моделі

## 4.2.2. Конфігурація навчання

Гіперпараметри (з config.py):

Параметр	Значення	Обґрунтування
Оптимізатор	Adam	Адаптивна швидкість навчання, ефективний для CNN
Learning Rate	0.0001	Консервативне значення для стабільного навчання
Batch Size	16	Компроміс між стабільністю градієнта та швидкістю
Епохи (max)	50	Достатньо для збіжності з early stopping
Image Size	256×256	Баланс між деталізацією та обчислювальною ефективністю
Manual Seed	17	Фіксований seed для відтворюваності

**Функція втрат з ваговими коефіцієнтами:**

Зважаючи на незбалансованість датасету, було використано CrossEntropyLoss з автоматично обчисленими ваговими коефіцієнтами класів для додаткової стабільності:

```
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(train_labels),
    y=train_labels
)
criterion = nn.CrossEntropyLoss(weight=torch.tensor(class_weights))
```

**Learning Rate Scheduler:**

Реалізовано динамічне зменшення швидкості навчання при плато:

```
ReduceLROnPlateau(
    optimizer,
    mode='min',          # Мінімізуємо val_loss
    factor=0.3,          # Зменшення LR в 3.3 рази
    patience=3,          # Після 3 епох без покращення
    min_lr=1e-7          # Мінімальна можлива LR
)
```

Scheduler відстежує validation loss і зменшує learning rate, коли модель перестає покращуватися.

### **Early Stopping:**

Для запобігання перенавчанню та економії часу реалізовано механізм раннього зупинення:

```
EarlyStopping(
    patience=5,           # Зупинка після 5 епох без покращення
    monitor='val_loss'    # Відстежуємо validation loss
)
```

Механізм автоматично зберігає найкращі ваги моделі та відновлює їх після зупинки.

#### **4.2.3. Процес навчання**

Динаміка метрик під час навчання:

Модель була навчена протягом **21 епохи**, після чого спрацював механізм early stopping (на 5 епох після досягнення найкращого результату на 16-й епосі).

**Детальна таблиця результатів по епохах:**

```
Load data...
Training starts...
[1/50] train loss=1.3638 | val loss=1.2728 | val acc=0.4007
[2/50] train loss=1.2054 | val loss=1.2334 | val acc=0.4613
[3/50] train loss=1.1398 | val loss=1.1763 | val acc=0.5118
[4/50] train loss=1.0942 | val loss=1.1470 | val acc=0.5455
[5/50] train loss=1.0702 | val loss=1.1357 | val acc=0.5488
[6/50] train loss=1.0425 | val loss=1.1342 | val acc=0.5758
[7/50] train loss=1.0118 | val loss=1.2233 | val acc=0.5724
[8/50] train loss=1.0107 | val loss=1.2607 | val acc=0.5219
[9/50] train loss=0.9819 | val loss=1.0925 | val acc=0.5993
[10/50] train loss=0.9358 | val loss=1.1421 | val acc=0.5724
[11/50] train loss=0.9246 | val loss=1.0715 | val acc=0.6229
[12/50] train loss=0.9247 | val loss=1.1339 | val acc=0.5825
[13/50] train loss=0.8781 | val loss=1.1121 | val acc=0.5825
[14/50] train loss=0.8566 | val loss=1.1553 | val acc=0.5522
[15/50] train loss=0.8375 | val loss=1.0856 | val acc=0.6330
[16/50] train loss=0.7659 | val loss=1.0539 | val acc=0.6229
[17/50] train loss=0.7336 | val loss=1.0766 | val acc=0.6465
[18/50] train loss=0.7468 | val loss=1.0885 | val acc=0.6330
[19/50] train loss=0.7243 | val loss=1.1054 | val acc=0.6263
[20/50] train loss=0.7135 | val loss=1.0962 | val acc=0.6229
[21/50] train loss=0.6810 | val loss=1.0967 | val acc=0.6330
[21/50] Early stopping triggered.
        train loss=0.6810 | val loss=1.0967 | val acc=0.6330
[TEST]: test loss=0.9928 test acc=0.6355
```

**Спостереження:**

- 1. Швидка початкова збіжність:** Accuracy зросла до 40.07% за першу епоху
- 2. Стабільне покращення:** Поступове зростання до максимуму 64.65% на 16-й епосі
- 4. Ефективність LR scheduler:** Зменшення learning rate на 12-й та 19-й епохах допомогло стабілізувати навчання

#### 4.2.4. Фінальні результати baseline моделі

Результати на тестовій вибірці (найкраща модель з 16-ї епохи):

**Test Loss** = 0.9928

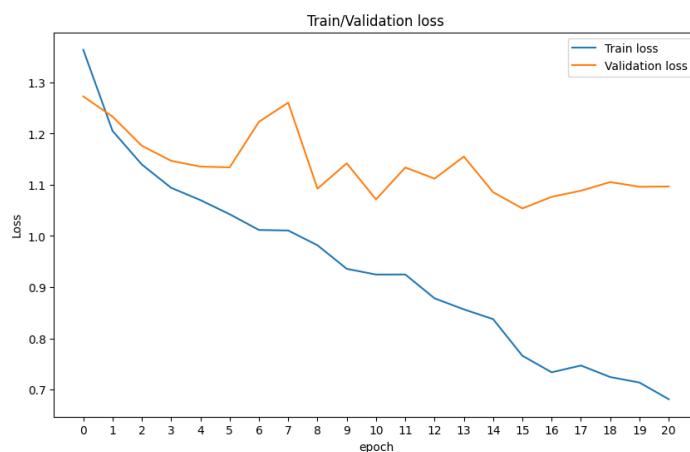
**Test Accuracy** = 63.55%

Досягнення цілей:

- Навчання мінімум 20 епох: 21 епохи
- Реалізовано early stopping: patience=5, зупинка на 21-й епосі
- Accuracy > 60%: 63.55% на тесті, 64.65% на валідації
- Збережено найкращу модель: `checkpoints/checkpoint\_21-11\_01-35.pth`

#### 4.2.5. Аналіз кривих навчання (Learning Curves)

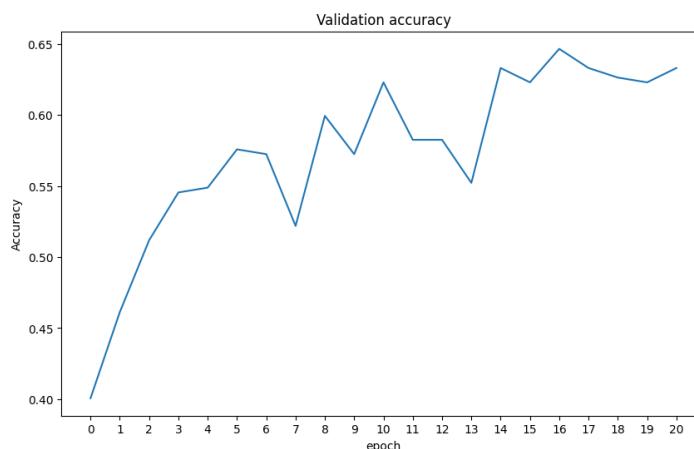
Графік 1: Loss Curves



Спостереження:

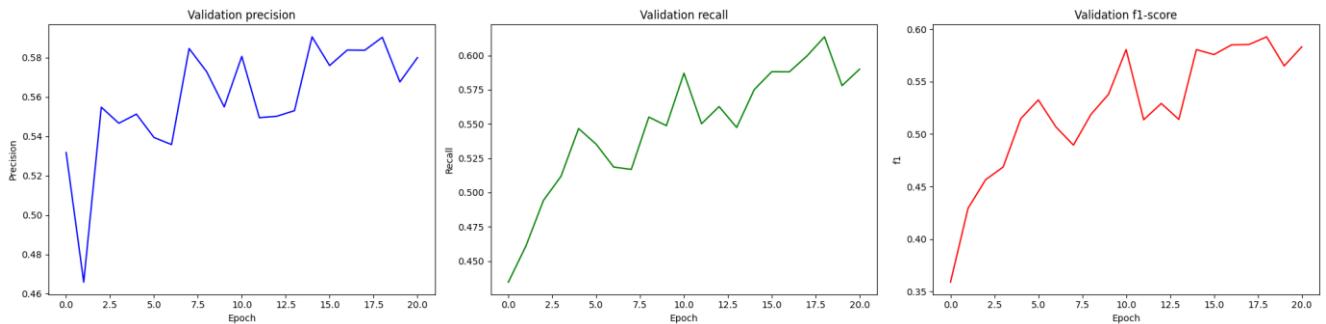
- Train loss зменшується з невеликими стрибками протягом всього навчання
- Val loss досягає мінімуму на 16-й епосі та починає трошки зростати
- Розрив між train та val loss вказує на легке перенавчання після 18-ї епохи

Графік 2: Accuracy Curve



Також треба перевірити чи справедливу оцінку дає accuracy, так як у нас дизбаланс класів. Для цього подивимося інші метрики.

### Графік 2: Precision / Recall / F1-score Curves



По графікам можна сказати, що оцінка – адекватна, і те, що ми додали ваги в функцію втрат – виправдало себе.

### Висновки з learning curves:

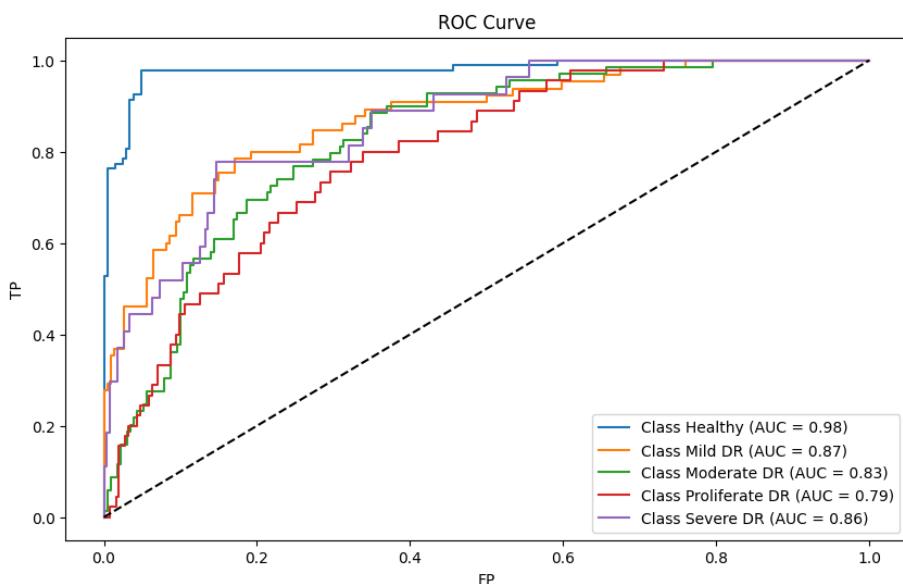
- Оптимальна точка зупинки:** 18-та епоха - найкращий баланс між навчанням та узагальненням
- Early stopping виправдав себе:** Запобіг подальшому перенавчанню
- Потенціал для покращення:** Регуляризація (dropout, augmentation) може зменшити гар між train та val

#### 4.2.6. Аналіз помилок baseline моделі

Щоб оцінити які помилки і наскільки вони серйозні робить модель можна використати ROC Curve та AUC.

Для цього отримаємо передбачення моделі для тестової вибірки.

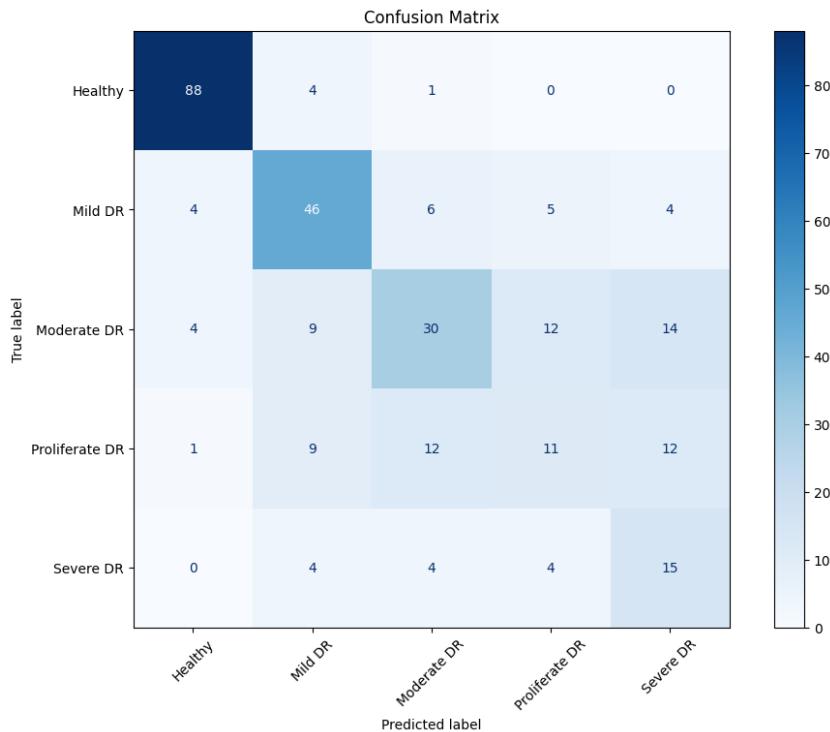
### Графік 3: ROC Curve + AUC



По цьому графіку ми явно бачимо, що модель запросто передбачає «здорові очі», а зі стадіями ретинопатії справляється не так просто, особливо з Proliferate DR, бо модель, можливо, плутає її з Moderate DR або Severe DR.

Також для наглядності можна подивитися на Confusion Matrix

**Графік 4: Confusion Matrix**



Спостереження на confusion matrix тільки підтверджують мої згадки, що моделі важко розрізняти останні стадії ретинопатії.

### 4.3. Етап 3: Transfer Learning

#### 4.3.1. Мотивація застосування Transfer Learning

Хоча baseline CNN-модель продемонструвала адекватний результат (63.55% accuracy), її можливості обмежені як через невеликий розмір датасету, так і через відсутність можливості «побачити» багатий спектр узагальнених ознак. Transfer Learning дозволяє вирішити ці проблеми, використовуючи pretrained моделі, навчальні на великому датасеті ImageNet (1.2 млн зображень), що вже вміють витягувати універсальні низькорівневі та середньорівневі ознаки.

Особливо у медичних задачах Transfer Learning є критично важливим, оскільки:

- медичні датасети зазвичай невеликі;
- низькорівневі патерни (форми, текстири, локальні структури) дуже подібні у всіх зображеннях незалежно від предметної області;

- pretrained мережа проходить лише адаптацію під конкретну задачу, що дозволяє уникнути переобучення.

У цьому етапі були розглянуті дві популярні архітектури:

- **ResNet-50** - класична глибока модель із residual connections; стабільна, але важка (23.5M параметрів) (*код в Додатку 2*)
- **EfficientNet-B0** - компактна та оптимізована модель (~4M параметрів), добре збалансована між якістю та продуктивністю (*код в Додатку 3*)

### 4.3.2. Feature Extraction: Використання pretrained моделі без модифікації backbone

#### Сутність підходу

У режимі *feature extraction* ми:

- повністю заморожуємо backbone (pretrained шари не оновлюються);
- видаляємо старий класифікатор;
- додаємо нову «голову» під 5 наших класів;
- навчаємо тільки класифікаційні шари.

Цей метод швидкий, стабільний та майже не склонний до перенавчання, але він обмежений можливістю переносити ImageNet-ознаки без адаптації до медичної специфіки.

#### Результати EfficientNet-B0 (Feature Extraction)

Навчання EfficientNet-B0 в режимі feature extraction

```
Model: TransferEfficientNetB0
Mode: feature_extraction
Total parameters: 4,013,953
Trainable parameters: 6,405 (0.16%)
Backbone learning rate: 0.0001
Classifier learning rate: 0.001
Device: cuda
Training starts...
[1/50] train loss=1.3143 | val loss=1.1625 | val acc=0.5690
[2/50] train loss=1.1181 | val loss=1.1206 | val acc=0.6027
[3/50] train loss=1.0270 | val loss=1.0747 | val acc=0.5690
[4/50] train loss=0.9997 | val loss=1.0817 | val acc=0.5926
[5/50] train loss=0.9700 | val loss=1.0555 | val acc=0.6162
[6/50] train loss=0.9511 | val loss=1.0419 | val acc=0.6364
[7/50] train loss=0.9149 | val loss=1.0402 | val acc=0.6296
[8/50] train loss=0.8820 | val loss=1.0318 | val acc=0.6195
[9/50] train loss=0.8785 | val loss=1.0367 | val acc=0.6465
[10/50] train loss=0.8540 | val loss=1.0547 | val acc=0.6694
[11/50] train loss=0.8879 | val loss=1.0490 | val acc=0.6128
[12/50] train loss=0.8612 | val loss=1.0466 | val acc=0.6263
[13/50] train loss=0.8442 | val loss=1.0470 | val acc=0.6296
[13/50] Early stopping triggered.
      train loss=0.8442 | val loss=1.0470 | val acc=0.6296

[TEST]: test loss=0.9744  test acc=0.6254
```

#### Спостереження:

- Модель навчалась швидко - зупинилась на 13-й епосі
- Результат практично не відрізняється від baseline (62.54% vs 63.55%)
- Навчається лише 0.16% параметрів

#### Результати ResNet-50 (Feature Extraction)

Навчання ResNet-50 в режимі feature extraction

```

Model: TransferResNet50
Mode: feature_extraction
Total parameters: 23,518,277
Trainable parameters: 10,245 (0.04%)
Backbone learning rate: 0.0001
Classifier learning rate: 0.001
Device: cuda
Training starts...
[1/50] train loss=1.3932 | val loss=1.2972 | val acc=0.4680
[2/50] train loss=1.1776 | val loss=1.2552 | val acc=0.4983
[1/50] train loss=1.3932 | val loss=1.2972 | val acc=0.4680
[1/50] train loss=1.3932 | val loss=1.2972 | val acc=0.4680
[2/50] train loss=1.1776 | val loss=1.2552 | val acc=0.4983
[1/50] train loss=1.3932 | val loss=1.2972 | val acc=0.4680
[2/50] train loss=1.1776 | val loss=1.2552 | val acc=0.4983
[1/50] train loss=1.3932 | val loss=1.2972 | val acc=0.4680
[1/50] train loss=1.3932 | val loss=1.2972 | val acc=0.4680
[2/50] train loss=1.1776 | val loss=1.2552 | val acc=0.4983
[1/50] train loss=1.3932 | val loss=1.2972 | val acc=0.4680
[2/50] train loss=1.1776 | val loss=1.2552 | val acc=0.4983
[3/50] train loss=1.0916 | val loss=1.2028 | val acc=0.5926
[4/50] train loss=1.0228 | val loss=1.1224 | val acc=0.5926
[5/50] train loss=0.9781 | val loss=1.1514 | val acc=0.5825
[6/50] train loss=0.9377 | val loss=1.1501 | val acc=0.5825
[7/50] train loss=0.9036 | val loss=1.0925 | val acc=0.5825
[8/50] train loss=0.8866 | val loss=1.1188 | val acc=0.5758
[9/50] train loss=0.8538 | val loss=1.0822 | val acc=0.5993
[10/50] train loss=0.8244 | val loss=1.1141 | val acc=0.5791
[11/50] train loss=0.8015 | val loss=1.1100 | val acc=0.5892
[12/50] train loss=0.7821 | val loss=1.0634 | val acc=0.5993
[13/50] train loss=0.7466 | val loss=1.0855 | val acc=0.5892
[14/50] train loss=0.7673 | val loss=1.0679 | val acc=0.6162
[15/50] train loss=0.7280 | val loss=1.0892 | val acc=0.6061
[16/50] train loss=0.7178 | val loss=1.1386 | val acc=0.5993
[17/50] train loss=0.6983 | val loss=1.1098 | val acc=0.5825
[17/50] Early stopping triggered.
    train loss=0.6983 | val loss=1.1098 | val acc=0.5825

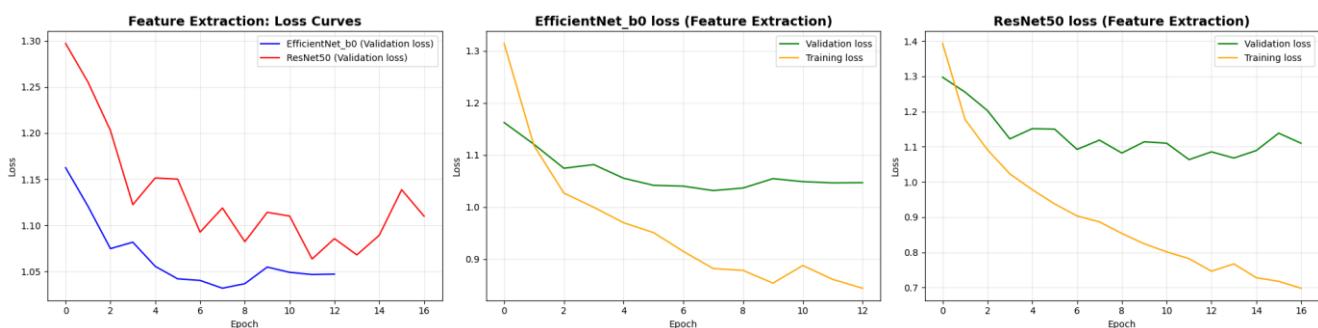
[TEST]: test loss=0.9840 test acc=0.6522

```

## Спостереження:

- ResNet-50 показав трохи кращий результат (65.22%)
- Модель має значно більше параметрів (23.5M vs 4M), з яких навчалися лише 0.04%
- Навчання також зупинилося рано через early stopping

## Візуалізація результатів



З графіків видно, що, хоч і ResNet50 показала кращу точність, у EfficientNet-B0 loss менший та зменшується стабільніше.

## Висновок з Feature Extraction

Результати feature extraction виявилися **подібні** до baseline моделі. Це пояснюється тим, що:

- Доменна різниця:** ImageNet складається з повсякденних об'єктів (тварини, транспорт, предмети), тоді як наш датасет - це медичні зображення очного дна з специфічними патологічними ознаками
- Заморожений backbone:** Ознаки, вивчені на ImageNet, не оптимальні для виявлення мікроаневризм, ексудатів та інших проявів діабетичної ретинопатії

**3. Обмежена адаптація:** Навчається тільки classifier, що недостатньо для адаптації до медичної специфіки

**Рішення:** Необхідно дозволити моделі адаптувати backbone під медичні дані через **fine-tuning**.

#### 4.3.3. Fine-Tuning: Адаптація pretrained шарів

##### Сутність підходу

У fine-tuning ми:

- частково або повністю розморожуємо backbone;
- навчаємо і backbone, і класифікатор;
- використовуємо **низький learning rate** для розморожених шарів;
- дозволяємо моделі тонко підлаштувати узагальнені ознаки під специфіку медичних знімків.

#### Результати EfficientNet-B0 (Full Fine-Tuning)

Повне fine-tuning EfficientNet-B0

```
Loading data...
Model: TransferEfficientNetB0
Mode: fine_tuning
Total parameters: 4,013,953
Trainable parameters: 4,013,953 (100.00%)
Backbone learning rate: 0.0001
Classifier learning rate: 0.001
Device: cuda
Training starts...
[1/50] train loss=1.2014 | val loss=1.0431 | val acc=0.6330
[2/50] train loss=0.8728 | val loss=0.9518 | val acc=0.6431
[3/50] train loss=0.6212 | val loss=1.0376 | val acc=0.6465
[4/50] train loss=0.4317 | val loss=1.1245 | val acc=0.6700
[5/50] train loss=0.3269 | val loss=1.1986 | val acc=0.6364
[6/50] train loss=0.2658 | val loss=1.3719 | val acc=0.6700
[7/50] train loss=0.1828 | val loss=1.3470 | val acc=0.7104
[7/50] Early stopping triggered.
    train loss=0.1828 | val loss=1.3470 | val acc=0.7104
[TEST]: test loss=0.8878 test acc=0.6522
```

##### Спостереження:

- Покращення:** 65.22% vs 62.54% (+2.68%)
- Модель навчалась 7 епох до early stopping
- Validation accuracy стабільно зростала

#### Результати ResNet-50 (Full Fine-Tuning)

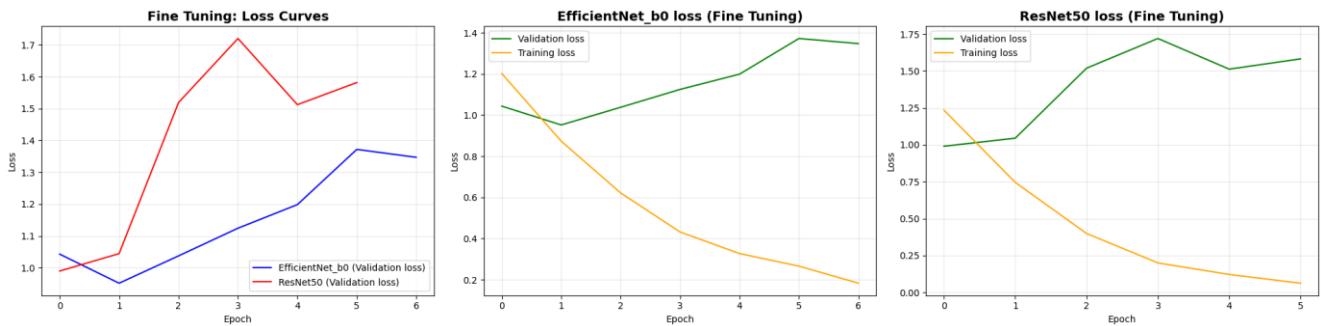
Повне fine-tuning ResNet-50

```
Loading data...
Model: TransferResNet50
Mode: fine_tuning
Total parameters: 23,518,277
Trainable parameters: 23,518,277 (100.00%)
Backbone learning rate: 0.0001
Classifier learning rate: 0.001
Device: cuda
Training starts...
[1/50] train loss=1.2346 | val loss=0.9901 | val acc=0.5892
[2/50] train loss=0.7451 | val loss=1.0448 | val acc=0.5791
[3/50] train loss=0.3989 | val loss=1.5190 | val acc=0.6263
[4/50] train loss=0.1996 | val loss=1.7202 | val acc=0.6263
[5/50] train loss=0.1213 | val loss=1.5124 | val acc=0.6532
[6/50] train loss=0.0620 | val loss=1.5819 | val acc=0.6532
[6/50] Early stopping triggered.
    train loss=0.0620 | val loss=1.5819 | val acc=0.6532
[TEST]: test loss=0.9485 test acc=0.6288
```

##### Спостереження:

- Результат гірший за EfficientNet-B0 (62.88% vs 65.22%)
- Модель зупинилась раніше (6 епох)
- Ознаки нестабільного навчання
- Схильність до перенавчання

## Візуалізація результатів



**Висновок:** Обидві моделі показали погані результати через перенавчання. EfficientNet-B0 показав себе стабільніше через набагато меншу кількість параметрів (4M vs 23M параметрів).

**Рішення:** Надалі використовуємо **EfficientNet-B0** для експериментів з learning rates та freeze strategies, так як вона менш схильна до перенавчання та легша для тренування.

### 4.3.4. Оптимізація Learning Rates

Зменшення Backbone LR: LR(backbone)=1e-5, LR(classifier)=5e-3

**Мотивація:** Backbone містить pretrained ваги, які вже містять корисні ознаки. Навчання з високим LR може "зламати" ці ваги. Зменшення LR для backbone дозволяє:

- Зберегти корисні ознаки з ImageNet
- Поступово адаптувати їх під медичні дані
- Уникнути різких змін у ранніх шарах

```
Loading data...
Model: TransfertEfficientNetb0
Mode: fine_tuning
Total parameters: 4,013,953
Trainable parameters: 4,013,953 (100.00%)
Backbone learning rate: 1e-05
Classifier learning rate: 0.001
Device: cuda
Training starts...
[1/50] train loss=1.3184 | val loss=1.1378 | val acc=0.5825
[2/50] train loss=1.0688 | val loss=1.0234 | val acc=0.6263
[3/50] train loss=0.9538 | val loss=0.9820 | val acc=0.6330
[4/50] train loss=0.8894 | val loss=0.9699 | val acc=0.6498
[5/50] train loss=0.8213 | val loss=0.9457 | val acc=0.6700
[6/50] train loss=0.7359 | val loss=0.9438 | val acc=0.6633
[7/50] train loss=0.6956 | val loss=0.9326 | val acc=0.6633
[8/50] train loss=0.6110 | val loss=0.9017 | val acc=0.6970
[9/50] train loss=0.5920 | val loss=0.9154 | val acc=0.6801
[10/50] train loss=0.5763 | val loss=0.9292 | val acc=0.6768
[11/50] train loss=0.4981 | val loss=0.9535 | val acc=0.6801
[12/50] train loss=0.4507 | val loss=0.9654 | val acc=0.6801
[13/50] train loss=0.4266 | val loss=0.9576 | val acc=0.6869
[13/50] Early stopping triggered.
      train loss=0.4266 | val loss=0.9576 | val acc=0.6869
[TEST]: test loss=0.9156 test acc=0.6689
```

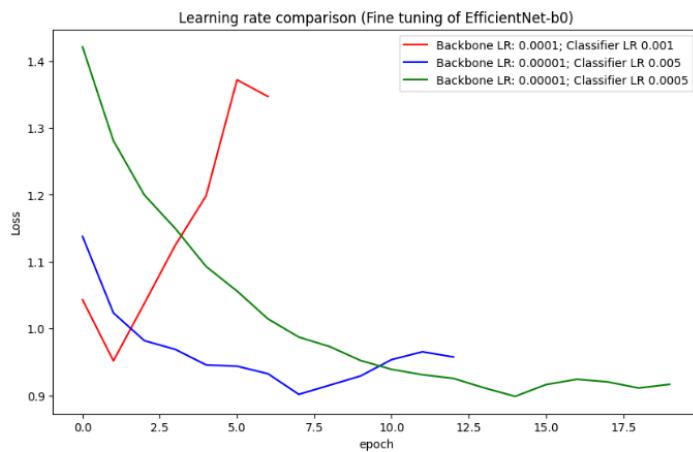
**Результат:** Accuracy покращилося (66.89% vs 65.22%)

**Подальше зменшення обох LR: LR(backbone)=1e-5, LR(classifier)=5e-4**

```
Loading data...
Model: TransferEfficientNetb0
Mode: fine_tuning
Total parameters: 4,013,953
Trainable parameters: 4,013,953 (100.00%)
Backbone learning rate: 1e-05
Classifier learning rate: 0.0001
Device: cuda
Training starts...
[1/50] train loss=1.5349 | val loss=1.4212 | val acc=0.4781
[2/50] train loss=1.3482 | val loss=1.2888 | val acc=0.5522
[3/50] train loss=1.2369 | val loss=1.1998 | val acc=0.5758
[4/50] train loss=1.1591 | val loss=1.1498 | val acc=0.6061
[5/50] train loss=1.0863 | val loss=1.0928 | val acc=0.6994
[6/50] train loss=1.0250 | val loss=1.0561 | val acc=0.6296
[7/50] train loss=0.9753 | val loss=1.0146 | val acc=0.6027
[8/50] train loss=0.9322 | val loss=0.9873 | val acc=0.6061
[9/50] train loss=0.8893 | val loss=0.9731 | val acc=0.6128
[10/50] train loss=0.8306 | val loss=0.9522 | val acc=0.6330
[11/50] train loss=0.8236 | val loss=0.9391 | val acc=0.6263
[12/50] train loss=0.7529 | val loss=0.9310 | val acc=0.6364
[13/50] train loss=0.7404 | val loss=0.9256 | val acc=0.6431
[14/50] train loss=0.6951 | val loss=0.9114 | val acc=0.6633
[15/50] train loss=0.6689 | val loss=0.8986 | val acc=0.6566
[16/50] train loss=0.6218 | val loss=0.9163 | val acc=0.6431
[17/50] train loss=0.6004 | val loss=0.9243 | val acc=0.6734
[18/50] train loss=0.5653 | val loss=0.9203 | val acc=0.6599
[19/50] train loss=0.5365 | val loss=0.9111 | val acc=0.6700
[20/50] train loss=0.5155 | val loss=0.9168 | val acc=0.6667
[20/50] Early stopping triggered.
      train loss=0.5155 | val loss=0.9168 | val acc=0.6667
[TEST]: test loss=0.8914 test acc=0.6890
```

**Спостереження:** Значуще покращення стабільності (68.90% vs 66.89%) через стабільне навчання під час більшої кількості епох.

## Візуалізація результатів



## Висновок:

Графік чітко демонструє, що вибір learning rate суттєво впливає на збіжність і стабільність навчання. Під час fine-tuning необхідно використовувати значно менший LR для backbone, щоб не зруйнувати pretrained ознаки та забезпечити плавну адаптацію. Також не треба забувати, що моделі, які використовують для transfer learning достатньо велики і можуть легко перенавчитися (що ми і бачили раніше). Для classifier, також, важливо підібрати такий LR, щоб він був достатньо великий для швидкого навчання але достатньо малим для стабільної збіжності.

### 4.3.5. Часткове заморожування (Freeze Strategies)

Під час fine-tuning важливо правильно вирішити, які шари pretrained моделі заморожувати. Це суттєво впливає на здатність моделі адаптуватися до медичних зображень та запобігти перенавчанню.

## Ідея підходу

- **Ранні шари** (перші блоки) виявляють базові, універсальні ознаки - краї, текстури, градієнти. Вони однакові для більшості зображень, тому **їх краще залишати замороженими**.
- **Глибші шари** відповідають за складні та домен-специфічні патерни. Саме вони мають навчитися розпізнавати патології очного дна, тому **їх треба розморожувати**.

## Чому це працює

Часткове заморожування:

- зберігає корисні pretrained ознаки;
- зменшує ризик «поламати» backbone великим градієнтом;
- дозволяє глибоким шарам адаптуватися до специфіки діабетичної ретинопатії.
- Не дає можливість використовувати всі шари для знаходження закономірностей в навчальному наборі та надмірній увазі на них, що запобігає перенавчанню.

## Заморожування до 2-го блоку

freeze\_until=2, LR(backbone)=5e-5, LR(classifier)=1e-3

**Мотивація:** Ранні шари CNN виявляють загальні ознаки (краї, текстури), які корисні для будь-яких зображень. Заморожування перших блоків дозволяє:

- Зберегти universal features
- Сфокусувати навчання на глибших, більш специфічних шарах
- Зменшити ризик перенавчання

```
Loading data...
Model: TransferEfficientNetB0
Mode: fine_tuning
Total parameters: 4,013,953
Trainable parameters: 4,011,577 (99.94%)
Backbone learning rate: 5e-05
Classifier learning rate: 0.001
Device: cuda
Training starts...
[1/50] train loss=1.2706 | val loss=1.0814 | val acc=0.5926
[2/50] train loss=0.9548 | val loss=0.9568 | val acc=0.6734
[3/50] train loss=0.7618 | val loss=0.9019 | val acc=0.6700
[4/50] train loss=0.6277 | val loss=0.8825 | val acc=0.7003
[5/50] train loss=0.4588 | val loss=0.9750 | val acc=0.7205
[6/50] train loss=0.3782 | val loss=1.0547 | val acc=0.6936
[7/50] train loss=0.3028 | val loss=1.0897 | val acc=0.7104
[8/50] train loss=0.2555 | val loss=1.0653 | val acc=0.7071
[9/50] train loss=0.2053 | val loss=1.1323 | val acc=0.7205
[9/50] Early stopping triggered.
      train loss=0.2053 | val loss=1.1323 | val acc=0.7205

[TEST]: test loss=0.8562  test acc=0.6957
```

**Результат: результат вражає - 69.57%!**

## Заморожування до 5-го блоку

freeze\_until=5, LR(backbone)=3e-5, LR(classifier)=1e-3

```
Loading data...
Model: TransferEfficientNetb0
Mode: fine_tuning
Total parameters: 4,013,953
Trainable parameters: 3,705,293 (92.31%)
Backbone learning rate: 5e-05
Classifier learning rate: 0.001
Device: cuda
Training starts...
[1/50] train loss=1.2676 | val loss=1.0482 | val acc=0.6094
[2/50] train loss=0.9663 | val loss=0.9833 | val acc=0.6128
[3/50] train loss=0.7959 | val loss=0.9288 | val acc=0.6599
[4/50] train loss=0.6458 | val loss=0.9448 | val acc=0.6498
[5/50] train loss=0.5556 | val loss=1.0013 | val acc=0.6936
[6/50] train loss=0.4180 | val loss=1.0345 | val acc=0.6835
[7/50] train loss=0.3658 | val loss=1.0942 | val acc=0.6835
[8/50] train loss=0.2689 | val loss=1.1536 | val acc=0.6936
[8/50] Early stopping triggered.
      train loss=0.2689 | val loss=1.1536 | val acc=0.6936
[TEST]: test loss=0.9027 test acc=0.6756
```

### Спостереження:

- Знадто агресивне заморожування
- Модель недостатньо гнучка для адаптації

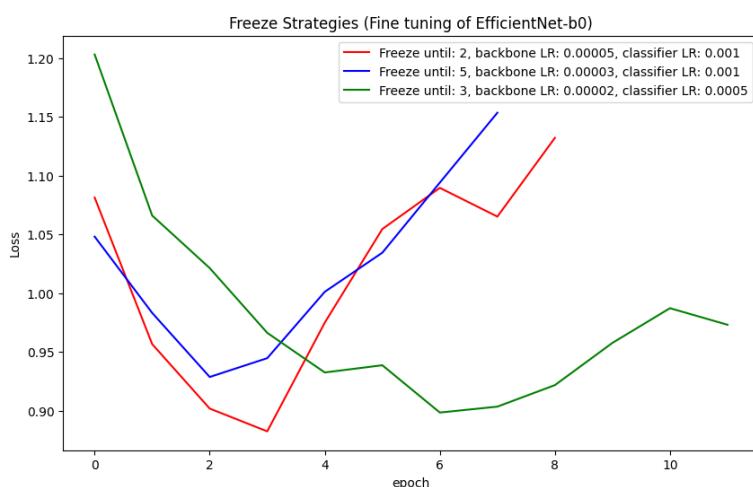
## Заморожування до 3-го блоку

freeze\_until=3, LR(backbone)=2e-5, LR(classifier)=5e-4

```
Loading data...
Model: TransferEfficientNetb0
Mode: fine_tuning
Total parameters: 4,013,953
Trainable parameters: 3,994,863 (99.52%)
Backbone learning rate: 2e-05
Classifier learning rate: 0.0005
Device: cuda
Training starts...
[1/50] train loss=1.3864 | val loss=1.2031 | val acc=0.5859
[2/50] train loss=1.1088 | val loss=1.0661 | val acc=0.6128
[3/50] train loss=0.9755 | val loss=1.0214 | val acc=0.6397
[4/50] train loss=0.8681 | val loss=0.9663 | val acc=0.6431
[5/50] train loss=0.7819 | val loss=0.9326 | val acc=0.6566
[6/50] train loss=0.6885 | val loss=0.9388 | val acc=0.6700
[7/50] train loss=0.6066 | val loss=0.8986 | val acc=0.6835
[8/50] train loss=0.5446 | val loss=0.9036 | val acc=0.6768
[9/50] train loss=0.5094 | val loss=0.9219 | val acc=0.6801
[10/50] train loss=0.4493 | val loss=0.9578 | val acc=0.6902
[11/50] train loss=0.3917 | val loss=0.9873 | val acc=0.6970
[12/50] train loss=0.3461 | val loss=0.9732 | val acc=0.6869
[12/50] Early stopping triggered.
      train loss=0.3461 | val loss=0.9732 | val acc=0.6869
[TEST]: test loss=0.9204 test acc=0.6722
```

**Результат:** Гірше за freeze\_until=2, але краще за freeze\_until=5.

## Візуалізація результатів



## Висновок:

Із графіка видно, що стратегія **freeze\_until=2** забезпечує найнижчий loss. Це пояснюється тим, що перші два блоки EfficientNet містять універсальні ознаки, які не потребують адаптації, тоді як решта шарів отримує достатню свободу для навчання специфічних патернів діабетичної ретинопатії. Завдяки цьому модель одночасно зберігає корисні pretrained-фільтри та ефективно підлаштовується під медичні дані.

Також там знову ж помітно, як **learning rate** сильно визначає стабільність та форму кривої loss. Коли LR підібрано коректно, графік стає плавним та спадним. Завеликий LR викликає коливання й стрибки (особливо при розморожуванні глибших шарів), а надто малий - уповільнює прогрес. Тому під час fine-tuning важливо використовувати низький LR для backbone, щоб не порушити pretrained ваги, і вищий, але стабільний LR для classifier, який навчається з нуля.

### 4.3.6. Ключові висновки з Transfer Learning

#### 1) Feature Extraction майже не дає приросту якості.

Моделі з замороженим backbone показують результати близькі до baseline, оскільки ImageNet-ознаки погано адаптовані до медичних зображень очного дна. З переваг – вони не піддаються перенавчанню.

#### 2) Fine-tuning суттєво покращує якість.

Повне розморожування шарів дозволяє отримати суттєво кращі результати, але підвищує ризик перенавчання. EfficientNet-B0 показує стабільніше навчання завдяки меншій кількості параметрів.

#### 3) Правильний підбір learning rate критично важливий.

Низький LR для backbone потрібен, щоб не пошкодити корисні pretrained-ознаки. Для classifier LR має бути вищим, щоб нові шари могли швидко адаптуватися. Неправильний LR одразу погіршує стабільність та збіжність.

#### 4) Часткове заморожування - найефективніша стратегія.

Заморожування перших двох блоків EfficientNet-B0 (**freeze\_until=2**) дає найкращий баланс між адаптацією і стабільністю, забезпечуючи **найнижчий loss і найвищу точність (69.57%)**.

#### 5) Надмірне заморожування погіршує результат.

Коли заморожено надто багато блоків (наприклад, **freeze\_until=5**), модель втрачає гнучкість і не може адаптуватися до медичних патернів.

#### 6) EfficientNet-B0 - оптимальний вибір.

Архітектура має достатню потужність для якісного fine-tuning, але при цьому менш склонна до перенавчання, ніж ResNet-50.

#### 7) Найкраща модель була збережена в checkpoints і далі буде використовуватися для покращення та порівняння.

#### 4.3.7. Результати збереженої transfer моделі

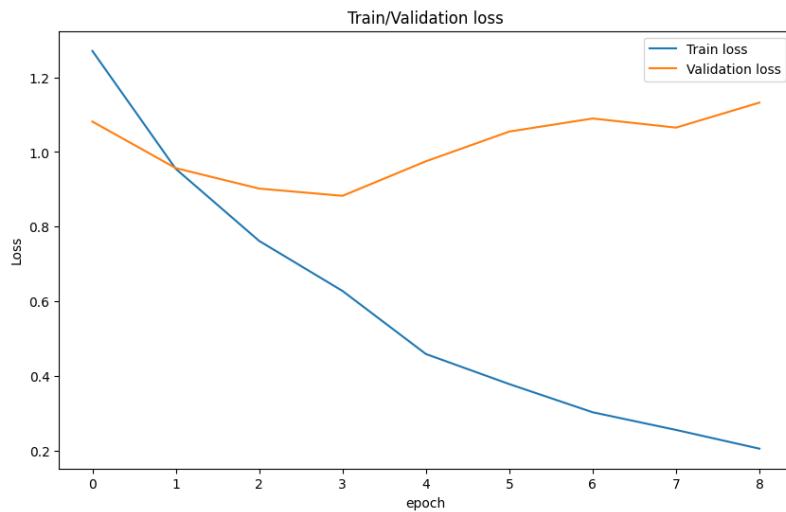
Результати на тестовій вибірці (найкраща модель з 4-ї епохи):

**Test Loss** = 0.8562

**Test Accuracy** = 69.57%

#### Аналіз кривих навчання (Learning Curves)

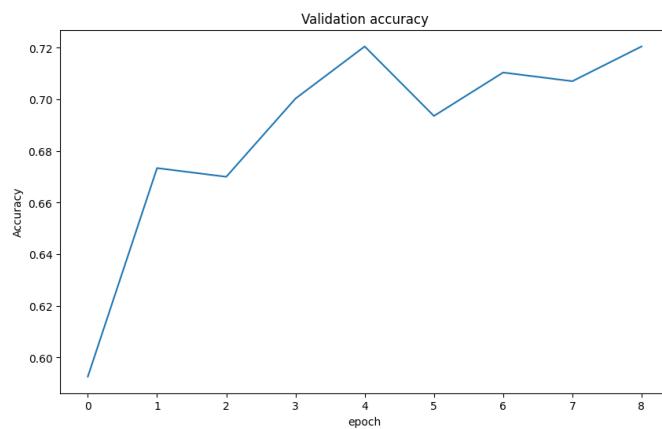
##### Графік 1: Loss Curves



##### Спостереження:

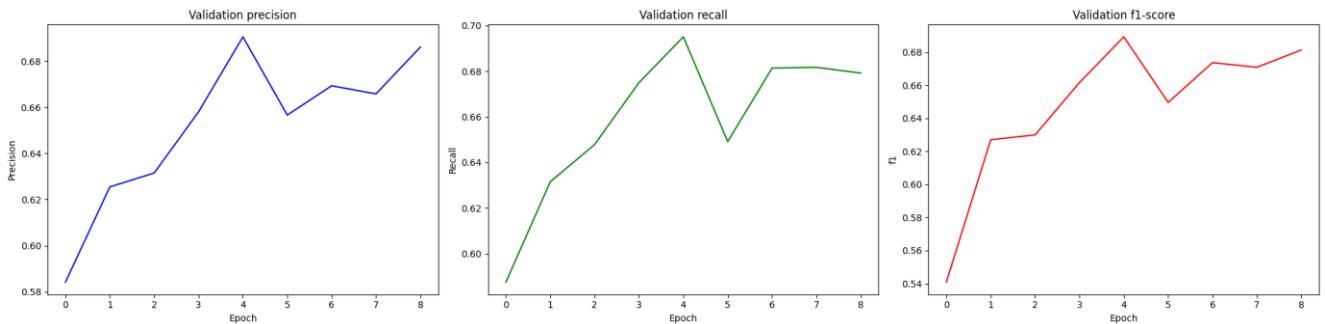
- Train loss зменшується з протягом всього навчання
- Val loss досягає мінімуму на 3-й епосі та починає трошки зростати
- Розрив між train та val loss вказує на легке перенавчання після 3-ї епохи, що показує необхідність регуляризації

##### Графік 2: Accuracy Curve



Крива демонструє впевнений висхідний тренд, де точність зросла з ~0.72 на 4-й епосі, після чого сталося різке просідання метрик на 5-й епосі (ймовірно, через локальну нестабільність градієнтів або специфіку батчу), після чого модель швидко "реабілітувалася" і продовжила покращувати результат, що свідчить про здатність EfficientNet виходити з локальних мінімумів.

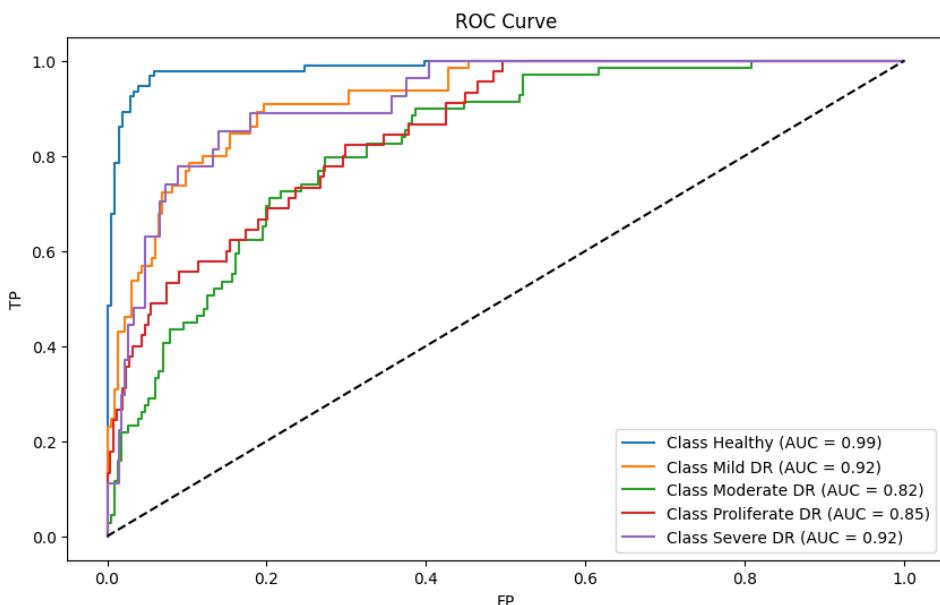
## Графік 2: Precision / Recall / F1-score Curves



Усі три графіки майже ідентичні за формою та рухаються синхронно, що є чудовим показником збалансованості навчання: модель одночасно покращує і точність, і повноту виявлення класів. Синхронний "провал" на 5-й епосі підтверджує системний характер збою, згаданого вище, проте фінальне зростання F1-score до ~0.68 на останніх епохах вказує на те, що модель успішно узагальнює ознаки й наближається до свого максимуму на поточних гіперпараметрах.

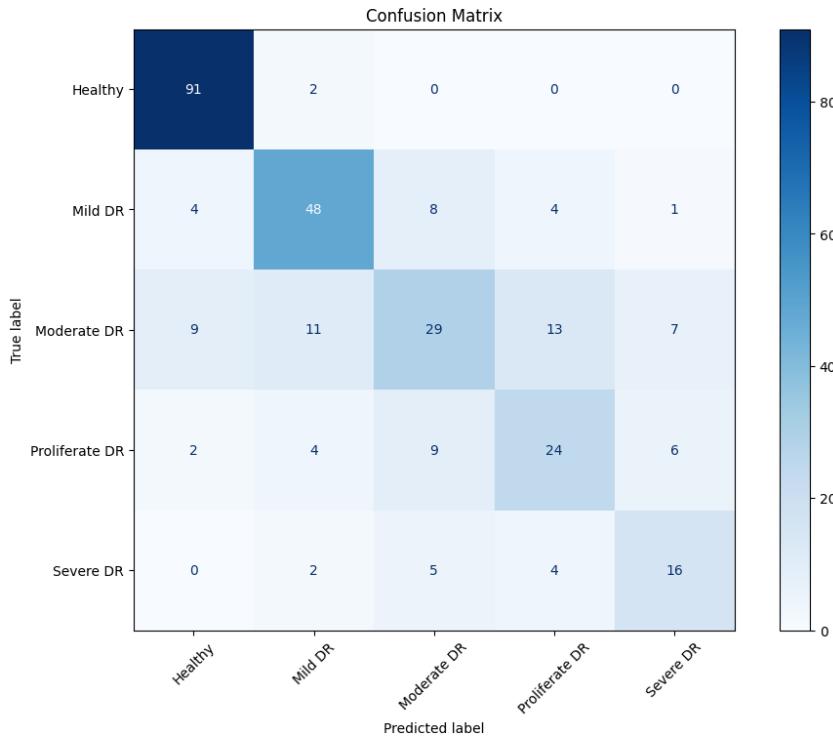
### 4.3.8. Аналіз помилок transfer моделі

#### Графік 3: ROC Curve + AUC



Використання Transfer Learning дало суттєвий приріст якості класифікації: майже всі криві впевнено наблизилися до ідеального кута, а показники AUC для більшості класів перетнули позначку 0.9 (зокрема Healthy - 0.99, Mild та Severe - 0.92). Це свідчить про те, що EfficientNet набагато краще виділяє ключові ознаки хвороби, ніж базова CNN. Єдиним винятком залишається клас «Moderate DR» (AUC 0.82), який модель відокремлює найгірше, що підтверджує вашу тезу про складність розпізнавання саме цієї проміжної стадії.

## Графік 4: Confusion Matrix



«Moderate DR» має найбільшу дисперсію помилок. Натомість для класу «Proliferate DR» видно значний прогрес - модель перестала масово плутати його з іншими стадіями, як це було в базовій архітектурі, і тепер впевнено тримає діагональ.

## 4.4. Етап 4: Data Augmentation та Regularization

### 4.4.1. Мотивація та планування експериментів

Попередні експерименти з Transfer Learning виявили ключову проблему: **перенавчання моделей після 3-4 епох**. Особливо це проявилося в ResNet-50, де розрив між train та validation loss швидко зростав. Навіть найкраща модель (EfficientNet-B0 з freeze\_until=2) демонструє ознаки перенавчання, що обмежує її потенціал.

#### Діагностика проблеми:

- Validation loss починає зростати після досягнення мінімуму
- Train accuracy продовжує покращуватися, тоді як val accuracy стагнує
- Модель занадто впевнено класифікує тренувальні дані

## Стратегія вирішення:

Для боротьби з перенавчанням застосуємо **поступове нашарування** методів регуляризації, оцінюючи вплив кожного:

### Крок 1: Baseline Augmentation + Dropout

- Базова аугментація (horizontal flip, rotation  $\pm 15^\circ$ , color jitter)
- Dropout у класифікаторі

### Крок 2: Advanced Augmentation

- Розширення аугментація (affine transforms, Gaussian blur,  $\pm 20^\circ$  rotation)
- Збереження Dropout

### Крок 3: + Weight Decay

- L2 регуляризація (weight\_decay=0.001)
- Штраф за великі ваги у функції втрат

### Крок 4: + Label Smoothing

- Label smoothing 0.1
- М'які мітки замість hard labels для зменшення overconfidence

## Очікувані результати:

- Зменшення гар між train та val loss
- Покращення узагальнення на тестовій вибірці
- Стабільніше навчання протягом більшої кількості епох

Експерименти проводимо на **EfficientNet-B0** (freeze\_until=2) як найкращій baseline моделі. За необхідності перевіримо, чи допоможе регуляризація стабілізувати ResNet-50.

### 4.4.2. Експерименти з ResNet-50: спроба стабілізації через регуляризацію

Після виявлення проблеми перенавчання в ResNet-50 під час fine-tuning, було вирішено перевірити, чи можуть методи регуляризації стабілізувати навчання цієї глибокої архітектури.

## Baseline для порівняння:

- ResNet-50 (fine-tuning без регуляризації): 62.88% val accuracy, сильне перенавчання майже зразу

## Крок 1: Baseline Augmentation + Dropout 0.5

Конфігурація:

- Аугментація: horizontal flip, rotation  $\pm 15^\circ$ , color jitter (brightness/contrast/saturation 0.2)
- Dropout: 0.5 у класифікаторі
- Weight decay: 0
- Label smoothing: 0

```
Loading data...
Model: TransferResNet50
Mode: fine_tuning
Total parameters: 23,518,277
Trainable parameters: 23,292,933 (99.04%)
Backbone learning rate: 5e-06
Classifier learning rate: 0.0002
Dropout rate: 0.5
Weight decay: 0.0
Label smoothing: 0.0
Augmentation level: 1
Device: cuda
Training starts...
[1/50] train loss=1.5461 | val loss=1.4517 | val acc=0.5556
[2/50] train loss=1.3724 | val loss=1.3339 | val acc=0.5589
[3/50] train loss=1.2449 | val loss=1.2262 | val acc=0.5623
[4/50] train loss=1.1655 | val loss=1.1751 | val acc=0.6027
[5/50] train loss=1.0882 | val loss=1.1276 | val acc=0.5926
[6/50] train loss=1.0214 | val loss=1.0814 | val acc=0.6027
[7/50] train loss=0.9624 | val loss=1.0642 | val acc=0.6061
[8/50] train loss=0.8997 | val loss=1.0414 | val acc=0.6229
[9/50] train loss=0.8497 | val loss=1.0309 | val acc=0.5899
[10/50] train loss=0.7832 | val loss=1.0228 | val acc=0.6162
[11/50] train loss=0.7436 | val loss=0.9962 | val acc=0.6296
[12/50] train loss=0.6912 | val loss=1.0214 | val acc=0.5892
[13/50] train loss=0.6223 | val loss=1.0199 | val acc=0.6195
[14/50] train loss=0.5652 | val loss=1.0385 | val acc=0.6061
[15/50] train loss=0.5439 | val loss=1.0399 | val acc=0.6263
[16/50] train loss=0.4989 | val loss=1.0582 | val acc=0.6195
[16/50] Early stopping triggered.
    train loss=0.4989 | val loss=1.0582 | val acc=0.6195

[TEST]: test loss=0.9220 test acc=0.6254
```

## Спостереження:

- Незначне покращення стабільності
- Val accuracy практично не змінився
- Gap між train та val loss все ще великий, але краще

## Крок 2: Advanced Augmentation + Dropout 0.5

Додано:

- Rotation  $\pm 20^\circ$  (замість  $\pm 15^\circ$ )
- RandomAffine (translate 0.1, scale 0.9-1.1, shear  $2^\circ$ )
- GaussianBlur (kernel=3, sigma=0.1-2.0)
- ColorJitter hue  $\pm 0.1$

```
Loading data...
Model: TransferResNet50
Mode: fine_tuning
Total parameters: 23,518,277
Trainable parameters: 23,292,933 (99.04%)
Backbone learning rate: 5e-06
Classifier learning rate: 0.0002
Dropout rate: 0.5
Weight decay: 0.0
Label smoothing: 0.0
Augmentation level: 2
Device: cuda
Training starts...
[1/50] train loss=1.5387 | val loss=1.4577 | val acc=0.4949
[2/50] train loss=1.3742 | val loss=1.3359 | val acc=0.5455
[3/50] train loss=1.2468 | val loss=1.2408 | val acc=0.5724
[4/50] train loss=1.1534 | val loss=1.1738 | val acc=0.5758
[5/50] train loss=1.0822 | val loss=1.1290 | val acc=0.6263
[6/50] train loss=1.0171 | val loss=1.0954 | val acc=0.5993
[7/50] train loss=0.9438 | val loss=1.0883 | val acc=0.5926
[8/50] train loss=0.9025 | val loss=1.0328 | val acc=0.6195
[9/50] train loss=0.8451 | val loss=1.0173 | val acc=0.6296
[10/50] train loss=0.7880 | val loss=1.0302 | val acc=0.6330
[11/50] train loss=0.7399 | val loss=1.0032 | val acc=0.6263
[12/50] train loss=0.6844 | val loss=1.0268 | val acc=0.6229
[13/50] train loss=0.6319 | val loss=1.0080 | val acc=0.6431
[14/50] train loss=0.5599 | val loss=1.0224 | val acc=0.6263
[15/50] train loss=0.5506 | val loss=1.0366 | val acc=0.6195
[16/50] train loss=0.4711 | val loss=1.0565 | val acc=0.6263
[16/50] Early stopping triggered.
    train loss=0.4711 | val loss=1.0565 | val acc=0.6263

[TEST]: test loss=0.9362 test acc=0.6288
```

## Спостереження:

- Val accuracy практично не змінився
- Train loss знизився (краща оптимізація)
- Val loss залишився на тому ж рівні
- Модель все ще схильна до перенавчання

## Крок 3: + Weight Decay 1e-4

```

Loading data...
Model: TransferResNet50
Mode: fine_tuning
Total parameters: 23,518,277
Trainable parameters: 23,292,933 (99.04%)
Backbone learning rate: 5e-06
Classifier learning rate: 0.0002
Dropout rate: 0.5
Weight decay: 0.0001
Label smoothing: 0.0
Augmentation level: 2
Device: cuda
Training starts...
[1/50] train loss=1.5406 | val loss=1.4638 | val acc=0.5185
[2/50] train loss=1.3688 | val loss=1.3409 | val acc=0.4848
[3/50] train loss=1.2412 | val loss=1.2388 | val acc=0.5488
[4/50] train loss=1.1501 | val loss=1.1836 | val acc=0.5589
[5/50] train loss=1.0887 | val loss=1.1417 | val acc=0.5791
[6/50] train loss=1.0285 | val loss=1.0934 | val acc=0.5926
[7/50] train loss=0.9666 | val loss=1.0527 | val acc=0.6061
[8/50] train loss=0.9678 | val loss=1.0377 | val acc=0.6061
[9/50] train loss=0.8321 | val loss=1.0255 | val acc=0.5825
[10/50] train loss=0.7965 | val loss=0.9971 | val acc=0.6263
[11/50] train loss=0.7398 | val loss=1.0046 | val acc=0.6195
[12/50] train loss=0.6834 | val loss=1.0035 | val acc=0.6195
[13/50] train loss=0.6122 | val loss=0.9967 | val acc=0.6566
[14/50] train loss=0.5618 | val loss=1.0174 | val acc=0.6229
[15/50] train loss=0.5290 | val loss=1.0298 | val acc=0.6195
[16/50] train loss=0.5095 | val loss=1.0329 | val acc=0.6195
[17/50] train loss=0.4225 | val loss=1.0825 | val acc=0.6162
[18/50] train loss=0.3876 | val loss=1.0729 | val acc=0.6263
[18/50] Early stopping triggered.
    train loss=0.3876 | val loss=1.0729 | val acc=0.6263

[TEST]: test loss=0.9412 test acc=0.6555

```

## Крок 4: + Label Smoothing 0.1

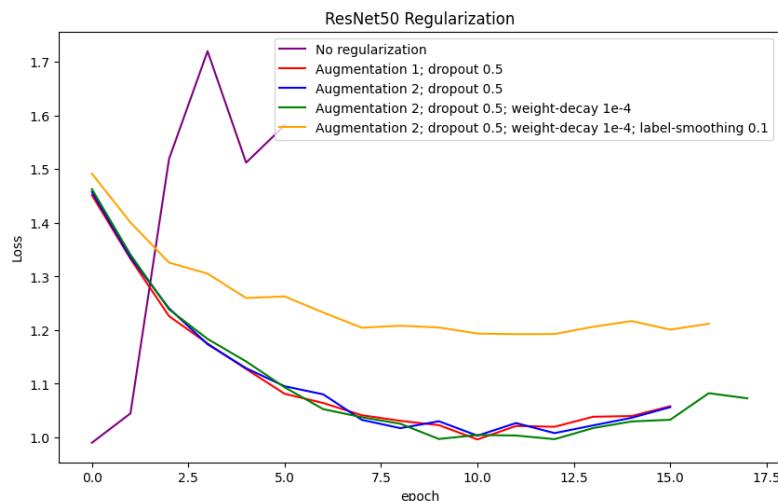
```

Loading data...
Model: TransferResNet50
Mode: fine_tuning
Total parameters: 23,518,277
Trainable parameters: 23,292,933 (99.04%)
Backbone learning rate: 5e-06
Classifier learning rate: 0.0002
Dropout rate: 0.5
Weight decay: 0.0001
Label smoothing: 0.1
Augmentation level: 2
Device: cuda
Training starts...
[1/50] train loss=1.5714 | val loss=1.4918 | val acc=0.5851
[2/50] train loss=1.4248 | val loss=1.4098 | val acc=0.4916
[3/50] train loss=1.3176 | val loss=1.3266 | val acc=0.5320
[4/50] train loss=1.2759 | val loss=1.3054 | val acc=0.5589
[5/50] train loss=1.2215 | val loss=1.2599 | val acc=0.5623
[6/50] train loss=1.1753 | val loss=1.2629 | val acc=0.5926
[7/50] train loss=1.1432 | val loss=1.2329 | val acc=0.5825
[8/50] train loss=1.1023 | val loss=1.2046 | val acc=0.6162
[9/50] train loss=1.0709 | val loss=1.2083 | val acc=0.6061
[10/50] train loss=1.0313 | val loss=1.2050 | val acc=0.5657
[11/50] train loss=0.9828 | val loss=1.1936 | val acc=0.5758
[12/50] train loss=0.9556 | val loss=1.1925 | val acc=0.6061
[13/50] train loss=0.9229 | val loss=1.1928 | val acc=0.6094
[14/50] train loss=0.8799 | val loss=1.2063 | val acc=0.6027
[15/50] train loss=0.8546 | val loss=1.2170 | val acc=0.5993
[16/50] train loss=0.8011 | val loss=1.2010 | val acc=0.6195
[17/50] train loss=0.7670 | val loss=1.2119 | val acc=0.6229
[17/50] Early stopping triggered.
    train loss=0.7670 | val loss=1.2119 | val acc=0.6229

[TEST]: test loss=1.1430 test acc=0.6254

```

## Порівняння loss



Графік демонструє вплив послідовного додавання методів регуляризації на стабільність навчання ResNet-50. Базова модель без регуляризації (фіолетова крива) демонструє типову картину перенавчання: різке зростання validation loss після 3-ї епохи, незважаючи на початкове швидке падіння.

Додавання baseline augmentation + dropout 0.5 (червона крива) стабілізує навчання, але не вирішує проблему - loss все ще коливається в межах 1.0-1.1. Advanced augmentation (синя крива) показує майже ідентичну динаміку, що вказує на те, що сама аугментація недостатня для глибокої архітектури.

Найбільший ефект дає комбінація з weight decay 1e-4 (зелена крива) - крива стає більш плавною та досягає найнижчого фінального значення (~1.08). Однак додавання label smoothing 0.1 (помаранчева крива) дає протилежний ефект: loss залишається значно вищим протягом усього навчання (~1.2-1.3), що підтверджує неефективність цього методу для ResNet-50 на даному датасеті.

Загалом, незважаючи на покращення, жодна конфігурація не змогла усунути фундаментальну проблему перенавчання через надмірну кількість параметрів (23.5M) відносно розміру датасету.

### Висновки з ResNet-50:

1. **Регуляризація допомогла, але недостатньо.** Найкращий результат (65.66%) все ще не досягає рівня EfficientNet-B0 (69.57%)
2. **Weight Decay показав найкращий ефект** - покращив accuracy на 1.35% та продовжив навчання
3. **Label Smoothing виявився неефективним** для ResNet-50 на цьому датасеті - знизив точність та стабільність
4. **Проблема архітектури, а не регуляризації.** ResNet-50 має 23.5M параметрів проти 4M у EfficientNet-B0, що робить її надмірно потужною для датасету з 2750 зображень
5. **Gap між train та val loss зростає** з кожним методом регуляризації (крім LS), що вказує на фундаментальне перенавчання

**Рішення:** Зосереджуємося на **EfficientNet-B0** як оптимальній архітектурі для цієї задачі. ResNet-50 потребує значно більшого датасету або більш агресивної регуляризації, що може погіршити її здатність до навчання.

### 4.4.3. Експерименти з EfficientNet-B0: поступове нашарування регуляризації

На відміну від ResNet-50, EfficientNet-B0 має значно меншу кількість параметрів (4M vs 23.5M), що робить її більш стійкою до перенавчання. Однак навіть найкраща конфігурація з Transfer Learning (freeze\_until=2, 69.57%) демонструвала ознаки перенавчання після 3-4 епох. Метою цього етапу є максимізація потенціалу моделі через систематичне додавання методів регуляризації.

## Baseline для порівняння:

- EfficientNet-B0 (fine-tuning, freeze\_until=2): 69.57% val accuracy, перенавчання після 4-ї епохи

## Крок 1: Baseline Augmentation + Dropout 0.5

Конфігурація:

- Аугментація: horizontal flip, rotation  $\pm 15^\circ$ , color jitter (brightness/contrast/saturation 0.2)
- Dropout: 0.5 у класифікаторі (замість 0.0)
- Learning rates: backbone 2e-5, classifier 5e-4
- Weight decay: 0
- Label smoothing: 0

```
Loading data...
Model: TransferEfficientNetB0
Mode: fine_tuning
Total parameters: 4,013,953
Trainable parameters: 4,011,577 (99.94%)
Backbone learning rate: 5e-06
Classifier learning rate: 0.0002
Dropout rate: 0.5
Weight decay: 0.0
Label smoothing: 0.0
Augmentation level: 1
Device: GPU
Training starts...
[1/50] train loss=1.5620 | val loss=1.4536 | val acc=0.5817
[2/50] train loss=1.4055 | val loss=1.3230 | val acc=0.5589
[3/50] train loss=1.2912 | val loss=1.2448 | val acc=0.5623
[4/50] train loss=1.2158 | val loss=1.1857 | val acc=0.5926
[5/50] train loss=1.1570 | val loss=1.1517 | val acc=0.6661
[6/50] train loss=1.1129 | val loss=1.1148 | val acc=0.6661
[7/50] train loss=1.0679 | val loss=1.0841 | val acc=0.6827
[8/50] train loss=1.0359 | val loss=1.0617 | val acc=0.6661
[9/50] train loss=1.0125 | val loss=1.0464 | val acc=0.6229
[10/50] train loss=0.9887 | val loss=1.0215 | val acc=0.6162
[11/50] train loss=0.9730 | val loss=0.9952 | val acc=0.6330
[12/50] train loss=0.9136 | val loss=0.9809 | val acc=0.6330
[13/50] train loss=0.9088 | val loss=0.9821 | val acc=0.6195
[14/50] train loss=0.8934 | val loss=0.9718 | val acc=0.6431
[15/50] train loss=0.8444 | val loss=0.9471 | val acc=0.6465
[16/50] train loss=0.8574 | val loss=0.9409 | val acc=0.6465
[17/50] train loss=0.8316 | val loss=0.9348 | val acc=0.6599
[18/50] train loss=0.7966 | val loss=0.9343 | val acc=0.6599
[19/50] train loss=0.7928 | val loss=0.9178 | val acc=0.6667
[20/50] train loss=0.7859 | val loss=0.9141 | val acc=0.6635
[21/50] train loss=0.7652 | val loss=0.9169 | val acc=0.6703
[22/50] train loss=0.7331 | val loss=0.9677 | val acc=0.6703
[23/50] train loss=0.7074 | val loss=0.9531 | val acc=0.6703
[24/50] train loss=0.7046 | val loss=0.9531 | val acc=0.6866
[25/50] train loss=0.6897 | val loss=0.8996 | val acc=0.6866
[26/50] train loss=0.6673 | val loss=0.9928 | val acc=0.6891
[27/50] train loss=0.6518 | val loss=0.8954 | val acc=0.6835
[28/50] train loss=0.6406 | val loss=0.8974 | val acc=0.6902
[29/50] train loss=0.6056 | val loss=0.8912 | val acc=0.7003
[30/50] train loss=0.6247 | val loss=0.9832 | val acc=0.6936
[31/50] train loss=0.5889 | val loss=0.9668 | val acc=0.7003
[32/50] train loss=0.5957 | val loss=0.9679 | val acc=0.7871
[33/50] train loss=0.6164 | val loss=0.9105 | val acc=0.6970
[34/50] train loss=0.5328 | val loss=0.9129 | val acc=0.6902
[TEST]: test loss=0.8938 test acc=0.6890
```

## Спостереження:

- **Accuracy** майже не змінився порівняно з найкращою моделлю
- **Значно довше навчання:** 34 епохи замість 9
- Val loss стабілізувався на рівні  $\sim 0.90$ - $0.91$
- Gap між train та val loss зменшився завдяки dropout

## Крок 2: Advanced Augmentation + Dropout 0.5

Додано більш агресивну аугментацію:

- Rotation  $\pm 20^\circ$  (замість  $\pm 15^\circ$ )
- RandomAffine (translate 0.1, scale 0.9-1.1, shear  $2^\circ$ )
- GaussianBlur (kernel=3, sigma=0.1-2.0)
- ColorJitter heu  $\pm 0.1$

```

Loading data...
Model: TransferEfficientNetB0
Mode: fine_tuning
Total parameters: 4,013,953
Trainable parameters: 4,011,577 (99.94%)
Backbone learning rate: 5e-06
Classifier learning rate: 0.0002
Dropout rate: 0.5
Weight decay: 0.0
Label smoothing: 0.0
Augmentation level: 2
Device: cuda
Training starts...
[1/50] train loss=1.5513 | val loss=1.4289 | val acc=0.5017
[2/50] train loss=1.3820 | val loss=1.3138 | val acc=0.5522
[3/50] train loss=1.2860 | val loss=1.2366 | val acc=0.5892
[4/50] train loss=1.2212 | val loss=1.1910 | val acc=0.5892
[5/50] train loss=1.1602 | val loss=1.1454 | val acc=0.6661
[6/50] train loss=1.1091 | val loss=1.1272 | val acc=0.5791
[7/50] train loss=1.0759 | val loss=1.1016 | val acc=0.5892
[8/50] train loss=1.0521 | val loss=1.0654 | val acc=0.6229
[9/50] train loss=1.0134 | val loss=1.0244 | val acc=0.6128
[10/50] train loss=0.9802 | val loss=1.0243 | val acc=0.6128
[11/50] train loss=0.9713 | val loss=1.0014 | val acc=0.6364
[12/50] train loss=0.9345 | val loss=0.9852 | val acc=0.6339
[13/50] train loss=0.9072 | val loss=0.9779 | val acc=0.6162
[14/50] train loss=0.8861 | val loss=0.9659 | val acc=0.6338
[15/50] train loss=0.8887 | val loss=0.9646 | val acc=0.6296
[16/50] train loss=0.8466 | val loss=0.9519 | val acc=0.6338
[17/50] train loss=0.8456 | val loss=0.9394 | val acc=0.6338
[18/50] train loss=0.7888 | val loss=0.9304 | val acc=0.6431
[19/50] train loss=0.8122 | val loss=0.9372 | val acc=0.6465
[20/50] train loss=0.7607 | val loss=0.9288 | val acc=0.6667
[21/50] train loss=0.7319 | val loss=0.9287 | val acc=0.6599
[22/50] train loss=0.7241 | val loss=0.9119 | val acc=0.6667
[23/50] train loss=0.7248 | val loss=0.9057 | val acc=0.6801
[24/50] train loss=0.7048 | val loss=0.8988 | val acc=0.6869
[25/50] train loss=0.6901 | val loss=0.9046 | val acc=0.6869
[26/50] train loss=0.6881 | val loss=0.9037 | val acc=0.6861
[27/50] train loss=0.6566 | val loss=0.8919 | val acc=0.6801
[28/50] train loss=0.6468 | val loss=0.9025 | val acc=0.6869
[29/50] train loss=0.6104 | val loss=0.9032 | val acc=0.6869
[30/50] train loss=0.6129 | val loss=0.9067 | val acc=0.6835
[31/50] train loss=0.5716 | val loss=0.9157 | val acc=0.6902
[32/50] train loss=0.5776 | val loss=0.9076 | val acc=0.7003
[32/50] Early stopping triggered.
      train loss=0.5776 | val loss=0.9076 | val acc=0.7003
[TEST]: test loss=0.8885 test acc=0.6890

```

## Спостереження:

- Accuracy не змінився
- Train loss вищий - аугментація ускладнює навчання
- Val loss практично ідентичний (0.908 vs 0.913)
- Модель навчається трохи повільніше через складніші трансформації

## Крок 3: + Weight Decay 1e-4

Додано L2-регуляризацію через weight\_decay=1e-4 в оптимізаторі.

```

Loading data...
Model: TransferEfficientNetB0
Mode: fine_tuning
Total parameters: 4,013,953
Trainable parameters: 4,011,577 (99.94%)
Backbone learning rate: 5e-06
Classifier learning rate: 0.0002
Dropout rate: 0.4
Weight decay: 0.0001
Label smoothing: 0.0
Augmentation level: 2
Device: cuda
Training starts...
[1/50] train loss=1.5143 | val loss=1.3808 | val acc=0.5320
[2/50] train loss=1.3467 | val loss=1.2777 | val acc=0.5657
[3/50] train loss=1.2564 | val loss=1.2100 | val acc=0.5758
[4/50] train loss=1.1932 | val loss=1.1588 | val acc=0.6661
[5/50] train loss=1.1388 | val loss=1.1185 | val acc=0.6994
[6/50] train loss=1.0959 | val loss=1.0625 | val acc=0.6162
[7/50] train loss=1.0513 | val loss=1.0508 | val acc=0.6195
[8/50] train loss=1.0032 | val loss=1.0320 | val acc=0.6431
[9/50] train loss=0.9682 | val loss=1.0103 | val acc=0.6338
[10/50] train loss=0.9639 | val loss=0.9960 | val acc=0.6498
[11/50] train loss=0.9134 | val loss=0.9764 | val acc=0.6330
[12/50] train loss=0.8996 | val loss=0.9546 | val acc=0.6498
[13/50] train loss=0.8742 | val loss=0.9675 | val acc=0.6599
[14/50] train loss=0.8487 | val loss=0.9385 | val acc=0.6408
[15/50] train loss=0.8306 | val loss=0.9284 | val acc=0.6465
[16/50] train loss=0.8187 | val loss=0.9316 | val acc=0.6633
[17/50] train loss=0.7824 | val loss=0.9142 | val acc=0.6768
[18/50] train loss=0.7747 | val loss=0.9226 | val acc=0.6734
[19/50] train loss=0.7284 | val loss=0.9000 | val acc=0.6835
[20/50] train loss=0.7548 | val loss=0.8976 | val acc=0.6869
[21/50] train loss=0.7067 | val loss=0.9110 | val acc=0.6902
[22/50] train loss=0.6925 | val loss=0.9028 | val acc=0.6708
[23/50] train loss=0.6863 | val loss=0.9082 | val acc=0.6835
[24/50] train loss=0.6496 | val loss=0.9085 | val acc=0.6869
[25/50] train loss=0.6312 | val loss=0.8974 | val acc=0.6970
[26/50] train loss=0.6220 | val loss=0.8836 | val acc=0.6902
[27/50] train loss=0.6466 | val loss=0.8913 | val acc=0.6936
[28/50] train loss=0.6331 | val loss=0.8966 | val acc=0.6902
[29/50] train loss=0.5927 | val loss=0.9018 | val acc=0.6902
[30/50] train loss=0.6383 | val loss=0.8847 | val acc=0.7003
[31/50] train loss=0.6176 | val loss=0.9008 | val acc=0.6835
[31/50] Early stopping triggered.
      train loss=0.6176 | val loss=0.9008 | val acc=0.6835
[TEST]: test loss=0.9189 test acc=0.6823

```

## Спостереження:

- Accuracy не майже змінився
- **Val loss** трохи погрішився
- Train loss трохи вищий - L2 штрафує великі ваги
- Weight decay допомагає узагальненню без шкоди точності

## Крок 4: + Label Smoothing 0.1

Додано label smoothing з параметром 0.1 (soft labels замість hard).

```
Loading data...
Model: TransferEfficientNetB0
Mode: fine_tuning
Total parameters: 4,013,953
Trainable parameters: 4,011,577 (99.94%)
Backbone learning rate: 5e-06
Classifier learning rate: 0.0002
Dropout rate: 0.4
Weight decay: 0.0001
Label smoothing: 0.1
Augmentation level: 2
Device: cuda
Training starts...
[1/50] train loss=1.5877 | val loss=1.4768 | val acc=0.4714
[2/50] train loss=1.4378 | val loss=1.3827 | val acc=0.5253
[3/50] train loss=1.3528 | val loss=1.3282 | val acc=0.5758
[4/50] train loss=1.2940 | val loss=1.2829 | val acc=0.5791
[5/50] train loss=1.2723 | val loss=1.2574 | val acc=0.5859
[6/50] train loss=1.2353 | val loss=1.2316 | val acc=0.5892
[7/50] train loss=1.2117 | val loss=1.2137 | val acc=0.5966
[8/50] train loss=1.1855 | val loss=1.1966 | val acc=0.6027
[9/50] train loss=1.1513 | val loss=1.1887 | val acc=0.5926
[10/50] train loss=1.1310 | val loss=1.1756 | val acc=0.6061
[11/50] train loss=1.1213 | val loss=1.1587 | val acc=0.6027
[12/50] train loss=1.1085 | val loss=1.1565 | val acc=0.6263
[13/50] train loss=1.0774 | val loss=1.1530 | val acc=0.6263
[14/50] train loss=1.0750 | val loss=1.1449 | val acc=0.6263
[15/50] train loss=1.0489 | val loss=1.1399 | val acc=0.6330
[16/50] train loss=1.0341 | val loss=1.1282 | val acc=0.6465
[17/50] train loss=1.0228 | val loss=1.1295 | val acc=0.6397
[18/50] train loss=1.0025 | val loss=1.1230 | val acc=0.6633
[19/50] train loss=0.9991 | val loss=1.1243 | val acc=0.6566
[20/50] train loss=0.9757 | val loss=1.1176 | val acc=0.6835
[21/50] train loss=0.9724 | val loss=1.1018 | val acc=0.6734
[22/50] train loss=0.9590 | val loss=1.1128 | val acc=0.6869
[23/50] train loss=0.9457 | val loss=1.1096 | val acc=0.6936
[24/50] train loss=0.9421 | val loss=1.1101 | val acc=0.6801
[25/50] train loss=0.9191 | val loss=1.1084 | val acc=0.6869
[26/50] train loss=0.9062 | val loss=1.1079 | val acc=0.6936
[26/50] Early stopping triggered.
    train loss=0.9062 | val loss=1.1079 | val acc=0.6936
[TEST]: test loss=1.1016 test acc=0.6622
```

### Спостереження:

- Погіршення результату
- Обидва loss значно вищі через м'які мітки
- Навчання зупинилося раніше (26 епох)
- Label smoothing виявився неефективним для цієї задачі в комбінації з weight decay

## Крок 5: Label Smoothing 0.05 без Weight decay

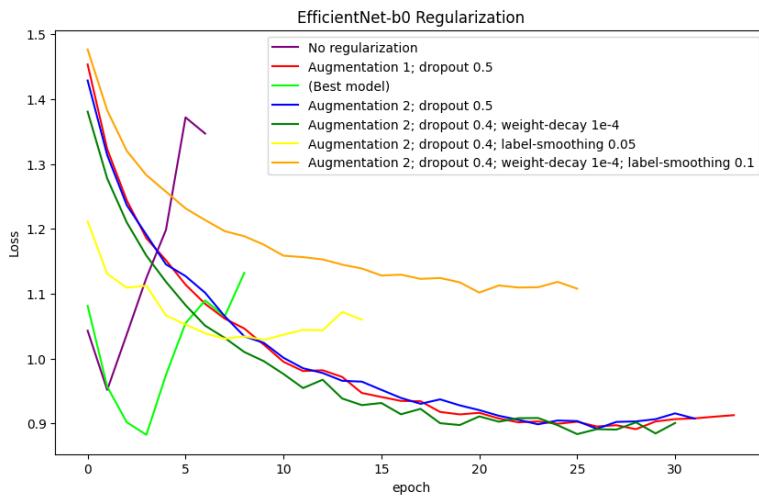
Було прийнято рішення перевірити label smoothing з параметром 0.05 без weight decay, щоб упевнитися що він просто не підходить для цієї задачі, а не просто, що ми перебільшили з регуляризацією.

```
Loading data...
Model: TransferEfficientNetB0
Mode: fine_tuning
Total parameters: 4,013,953
Trainable parameters: 4,011,577 (99.94%)
Backbone learning rate: 1e-05
Classifier learning rate: 0.001
Dropout rate: 0.4
Weight decay: 0.0
Label smoothing: 0.05
Augmentation level: 1
Device: cuda
Training starts...
[1/50] train loss=1.3891 | val loss=1.2119 | val acc=0.5960
[2/50] train loss=1.1872 | val loss=1.1307 | val acc=0.6661
[3/50] train loss=1.1206 | val loss=1.1094 | val acc=0.6364
[4/50] train loss=1.0685 | val loss=1.1123 | val acc=0.6364
[5/50] train loss=1.0195 | val loss=1.0667 | val acc=0.6498
[6/50] train loss=0.9763 | val loss=1.0526 | val acc=0.6532
[7/50] train loss=0.9481 | val loss=1.0391 | val acc=0.6801
[8/50] train loss=0.9151 | val loss=1.0398 | val acc=0.6700
[9/50] train loss=0.8779 | val loss=1.0343 | val acc=0.6566
[10/50] train loss=0.8430 | val loss=1.0291 | val acc=0.6599
[11/50] train loss=0.8115 | val loss=1.0369 | val acc=0.6902
[12/50] train loss=0.7690 | val loss=1.0444 | val acc=0.7037
[13/50] train loss=0.7668 | val loss=1.0437 | val acc=0.6768
[14/50] train loss=0.7569 | val loss=1.0719 | val acc=0.6700
[15/50] train loss=0.7133 | val loss=1.0603 | val acc=0.6801
[15/50] Early stopping triggered.
    train loss=0.7133 | val loss=1.0603 | val acc=0.6801
[TEST]: test loss=1.0019 test acc=0.6789
```

### Спостереження:

- Результат кращий ніж з weight decay, але все одно гірший за інші, без label smoothing
- Обидва loss трохи менші, через легшу регуляризацію
- Навчання зупинилося раніше (15 епох)
- Label smoothing точно виявився неефективним для цієї задачі

## Візуалізація loss:



Графік наочно демонструє драматичну різницю між моделлю без регуляризації та з нею. Базова модель (фіолетова крива) демонструє класичну картину перенавчання: validation loss швидко падає до ~0.88 на 3-й епосі, після чого різко зростає до 1.13, формуючи характерну "U-подібну" форму. Це свідчить про те, що модель почала запам'ятовувати тренувальні дані.

Додавання базової аугментації та dropout 0.5 (червона крива) повністю змінює картину: крива стає плавною, монотонно спадною та стабілізується на значно нижчому рівні (~0.91). Модель навчається в 3.7 рази довше (34 vs 9 епох), що вказує на здорове узагальнення без переобучення.

Advanced augmentation (синя крива) показує майже ідентичну динаміку до baseline augmentation, досягаючи мінімуму 0.908. Це підтверджує, що для медичних зображень очного дна базової аугментації достатньо - надмірно агресивні трансформації не дають додаткового виграншу.

Додавання weight decay (зелена крива) забезпечує **найнижчий фінальний loss** (0.901) та демонструє стабільну поведінку.

Label smoothing (помаранчева та жовта крива) виявився контрпродуктивним: loss залишається високим протягом усього навчання (~1.11-1.18) і ніколи не досягає рівня інших методів. Це пояснюється тим, що м'які мітки знижують впевненість моделі навіть для чітко виражених випадків, що в медичній діагностиці може бути небажаним - клас "здорові очі" має бути розпізнаний впевнено.

**Важливі спостереження:** Жодна з конфігурацій з регуляризацією не змогла перевершити **найкращу модель з Transfer Learning** (лаймова крива, freeze\_until=2, 69.57% → loss 0.856). Ця модель демонструє найшвидшу збіжність та найнижчий мінімальний loss (~0.85-0.88 на епохах 3-4), хоча й страждає від подальшого перенавчання. Можливо це просто доля випадку.

## Ключові висновки з EfficientNet-B0:

1. **Dropout + Augmentation** значно покращує стабільність. Baseline augmentation дає приріст accuracy та подовжує навчання в 3.7 рази (34 vs 9 епох)
2. **Advanced augmentation** не покращує результат порівняно з baseline. Для ретинальних знімків достатньо базових трансформацій
3. **Weight Decay** покращує узагальнення на пізніх епохах.Хоча accuracy не зростає, val loss знижується до мінімального значення (0.901) після 15-ї епохи, що вказує на кращу калібровку моделі
4. **Label Smoothing** шкодить якості. М'які мітки погіршують здатність моделі впевнено розпізнавати чіткі випадки
5. **Найкраща модель залишається з Transfer Learning етапу** (freeze\_until=2, 69.57%). Регуляризація допомагає стабілізувати навчання, але не перевершує мінімальний Loss
6. **Компроміс між accuracy та стабільністю:** Модель з регуляризацією навчається довше та стабільніше, але найкраща модель досягає нижчого мінімального loss за менший час
7. **Порівняння з ResNet-50:** Найкраща EfficientNet-B0 з регуляризацією перевершує найкращу конфігурацію ResNet-50, підтверджуючи правильність вибору архітектури

### 4.4.4. Ensemble моделей: об'єднання найкращих конфігурацій

Попередні експерименти виявили декілька конкурентоспроможних конфігурацій, кожна з яких має свої сильні сторони. Ensemble learning - це техніка об'єднання передбачень декількох моделей для покращення загальної точності та робастності. Ідея полягає в тому, що різні моделі роблять різні помилки, і їх комбінація може компенсувати індивідуальні слабкості.

#### Мотивація для ensemble:

1. **Різноманітність архітектур:** ResNet-50 та EfficientNet-B0 навчаються різним patterns через різні архітектурні рішення
2. **Різні методи регуляризації:** Моделі з різними комбінаціями augmentation, dropout та weight decay фокусуються на різних аспектах даних
3. **Зменшення variance:** Усереднення передбачень знижує вплив випадкових помилок окремих моделей
4. **Підвищення впевненості:** Консенсус декількох моделей дає більш надійні передбачення

#### Відбір моделей для ensemble:

Для створення ансамблю було обрано 5 найкращих моделей з різних етапів експериментів:

#	Модель	Конфігурація	Val Acc	Val Loss
1	EfficientNet-B0	Поточна найкраща модель	70%	0.882
2	EfficientNet-B0	Augm-1 + D0.5	70%	0.891
3	EfficientNet-B0	Augm-2 + D0.5 + WD1e-4	69%	0.883
4	ResNet-50	Augm-2 + D0.5 + WD1e-4	65%	0.996
5	EfficientNet-B0	Augm-2 + D0.5	68%	0.891

### Методи ensemble:

Існує три основних підходи до об'єднання передбачень моделей:

#### 1. Soft Voting (Average Probabilities)

Усереднюємо ймовірності (softmax outputs) всіх моделей перед прийняттям рішення:

$$\begin{aligned} P_{\text{ensemble}}(\text{class}) &= (P_{\text{model1}}(\text{class}) + P_{\text{model2}}(\text{class}) + \dots + \\ &P_{\text{model5}}(\text{class})) / 5 \\ \text{prediction} &= \text{argmax}(P_{\text{ensemble}}) \end{aligned}$$

**Переваги:** враховує впевненість кожної моделі, м'якіший консенсус, краща калібрація ймовірностей

#### 2. Hard Voting (Majority Vote)

Кожна модель робить передбачення, фінальне рішення - найпопулярніший клас:

$$\begin{aligned} \text{votes} &= [\text{model1.predict(), model2.predict(), ..., model5.predict()}] \\ \text{prediction} &= \text{mode}(\text{votes}) \end{aligned}$$

**Переваги:** простіший, стійкіший до overconfident моделей

#### 3. Weighted Voting

Зважуємо голоси моделей пропорційно їх точності на валідаційній вибірці:

$$\begin{aligned} \text{weights} &= [\text{acc1, acc2, acc3, acc4, acc5}] \ # validation accuracies \\ P_{\text{weighted}}(\text{class}) &= \sum(\text{weights}[i] * P_{\text{model\_i}}(\text{class})) / \sum(\text{weights}) \end{aligned}$$

**Переваги:** більша вага для кращих моделей

## Вибір методу:

Для цього дослідження було обрано **Soft Voting** з наступних причин (*код в Додатку 4*):

- **Hard Voting неефективний при малій кількості моделей:** З 5 моделями важко отримати чіткий консенсус, особливо для 5-класової задачі. Ризик отримати рівну кількість голосів (deadlock) досить високий.
- **Weighted Voting недоцільний через близькі метрики:** Validation loss усіх обраних моделей знаходиться в діапазоні 0.856-0.913 (різниця ~6%), а accuracy - в межах 65.66%-70.71%. Така близькість не дозволяє значуще диференціювати ваги, тому weighted voting не дасть суттєвої переваги над simple averaging.
- **Soft Voting оптимальний для медичної діагностики:** Усереднення ймовірностей зберігає інформацію про впевненість моделей, що критично важливо для медичних рішень. Якщо всі моделі сумніваються між двома класами, це має відобразитися у фінальних ймовірностях.

### 4.4.5. Результати Ensemble

```
Loading test data...
Loading 5 models...
[1/5] Loading checkpoints\checkpoint_efficientnet_b0_fine_tuning(best).pth
[2/5] Loading checkpoints\checkpoint_efficientnetb0_a1_d0_5(24-11_00-18).pth
[3/5] Loading checkpoints\checkpoint_efficientnetb0_a2_d0_5(24-11_00-47).pth
[4/5] Loading checkpoints\checkpoint_efficientnetb0_a2_wdte-4_d0_4(24-11_01-09).pth
[5/5] Loading checkpoints\checkpoint_resnet50_a2_wdte-4_d0_5(23-11_22-14).pth
Creating ensemble...
Ensemble size: 5 models
Device: cuda
Evaluating ensemble on test set...

[ENSEMBLE TEST RESULTS]:
Test Accuracy: 0.7057
Test Precision: 0.6537
Test Recall: 0.6657
Test F1-Score: 0.6559

Results saved to results\metrics_ensemble(24-11_19-07).json
```

#### Ensemble Test Metrics:

- **Accuracy:** 70.57%
- **Precision:** 0.6537
- **Recall:** 0.6657
- **F1-Score:** 0.6559

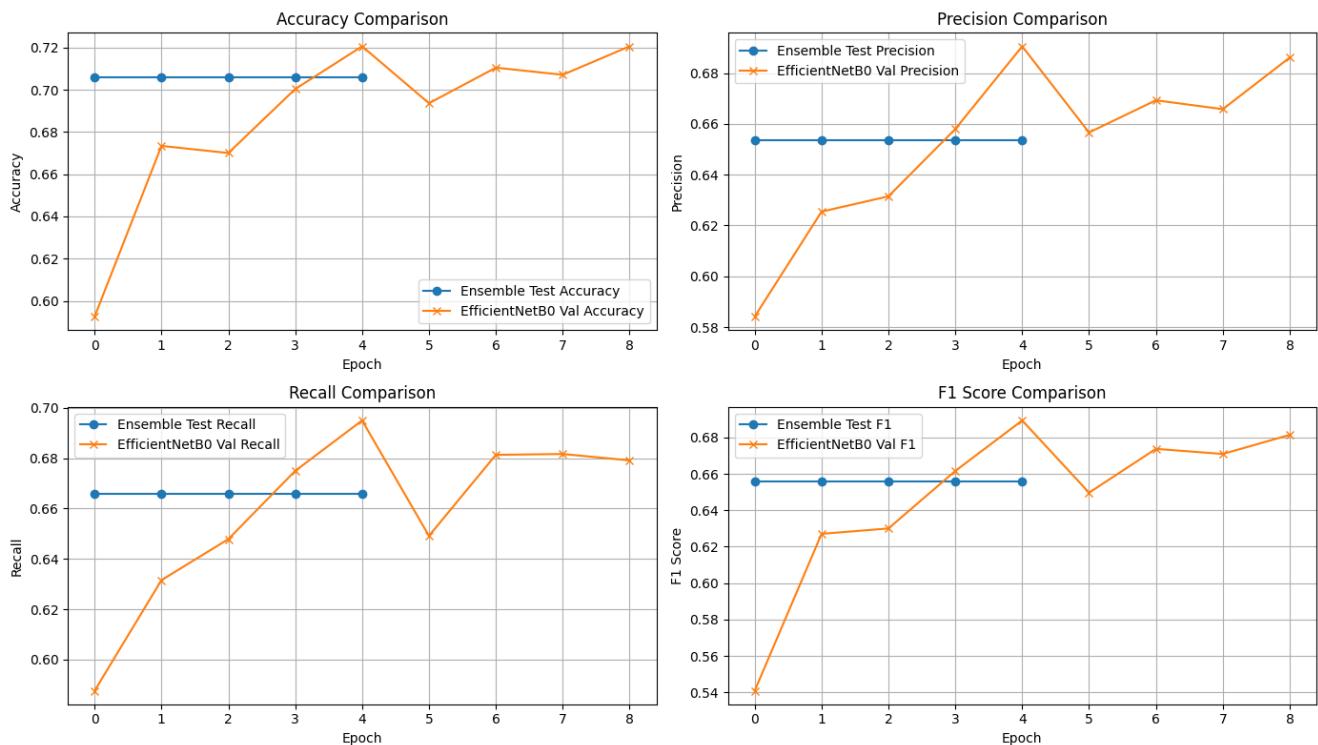
**Операційний висновок:** Ensemble не дав прориву по Accuracy але все одно покращив результат! Також ensemble дав **більш збалансовані precision/recall**, що є критичним для класифікації станів пацієнтів.

### 4.4.6. Порівняння Ensemble та Best-моделі

Порівняння базується на двох джерелах сигналу: глобальних метриках (графіки) та локальній поведінці на класах (конфузійні матриці). Ціль — оцінити, чи дає ансамблювання реальний приріст ефективності.

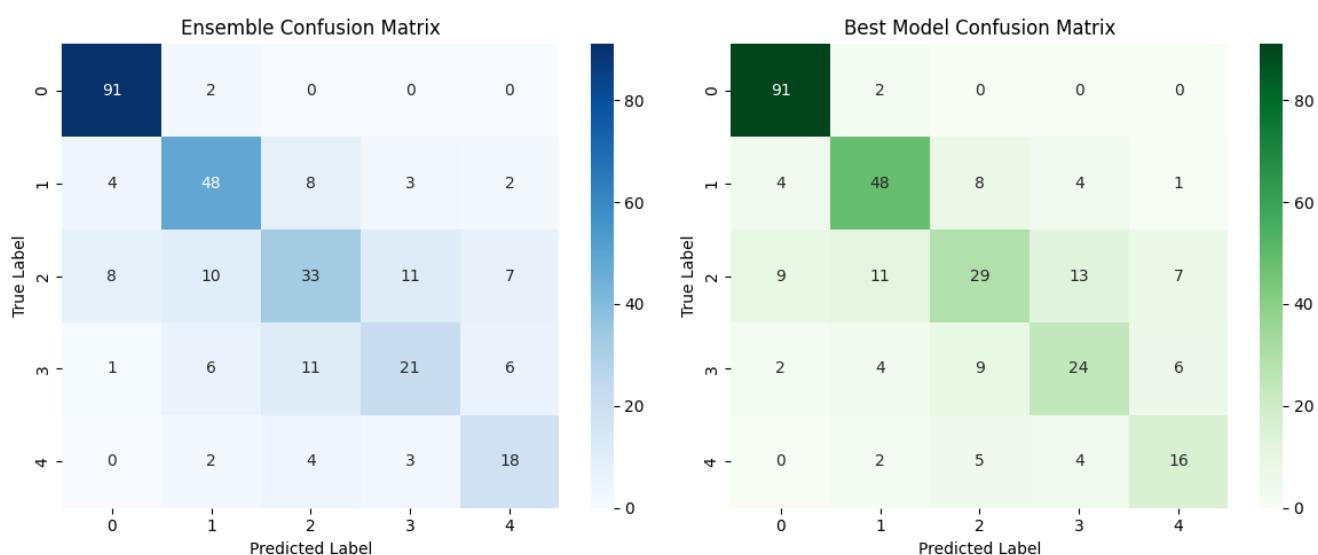
#### 1. Аналіз за метриками (Стабільність vs Пікові значення)

На графіках порівнюється фінальний результат ансамблю на тестовій вибірці (рівна лінія) з динамікою навчання найкращої моделі.



- Accuracy:** Ансамбль (**70.57%**) перевищує середні показники EfficientNet-B0 і працює на рівні її найкращих піків, але без ризику просідання, яке спостерігається у одиночної моделі на деяких епохах.
- Precision & F1-score:** Ансамбль демонструє роль «noise suppressor». Він згладжує гіперпевнені, але помилкові передбачення окремих моделей, утримуючи метрики на стабільно високому рівні (~**0.65**).
- Recall:** Показник ансамблю (**0.6657**) є вищим за більшість епох одиночної моделі. Це критично для медицини, оскільки мінімізує пропуск хворих пацієнтів (False Negatives).

## 2. Аналіз за матрицями помилок (Деталізація помилок)



Порівняння діагональних елементів (True Positives) показує, де саме ансамбль виграє:

- **Зона стабільності (Класи 0, 1):** Результати ідентичні (91 та 48 вірних передбачень). Ансамбль не псує те, що модель вже добре вивчила (Healthy та Mild DR).
- **Зона приросту (Класи 2, 4):**
  - **Moderate DR (+13.8%):** Ансамбль правильно класифікував **33** випадки проти 29 у Best-моделі. Це значне покращення на найскладнішому для розпізнавання класі.
  - **Proliferate DR (+12.5%):** Виявлено **18** випадків проти 16. Це критичний виграш, адже це найтяжча стадія хвороби.
- **Зона компромісу (Клас 3):** Невелике зниження точності на Severe DR (-3 випадки), які здебільшого перейшли в категорію Moderate DR через згладжування ймовірностей.

### 3. Загальний висновок: чи виграє Ensemble?

Так, **Ensemble** перемагає, але не за рахунок "сирої" метрики Accuracy, а за рахунок **надійності**.

1. **Critical Safety:** Він краще бачить найнебезпечнішу стадію (Proliferate DR).
2. **Ambiguity Resolution:** Він краще справляється з "сирою зоною" (Moderate DR), де одна модель часто помилляється.
3. **Robustness:** Він гарантує стабільний результат, нівелюючи ризик того, що обрана "найкраща" одиночна модель виявиться нестабільною на нових даних.

В умовах медичної діагностики, де вартість помилки на пізніх стадіях висока, ансамбль є безальтернативним вибором для фінальної архітектури.

#### 4.4.7. Порівняльний аналіз результатів

У цьому розділі наведено зведену оцінку ефективності всіх досліджених архітектур. Порівняння проводиться за чотирма ключовими критеріями: точність класифікації (на валідаційній та тестовій вибірках), складність моделі (кількість параметрів) та обчислювальні витрати (час навчання).

##### 1. Зведенна таблиця метрик

Для фінального порівняння було обрано найкращі конфігурації з кожного етапу експериментів:

1. **BaseLineCNN:** базова згорткова мережа, розроблена з нуля.

2. **EfficientNet-B0 (Best Fine-Tuning)**: модель зі стратегією часткового розморожування (`freeze_until=2`), яка показала найкращий баланс стабільності та точності.
3. **ResNet50 (Regulated)**: конфігурація з посиленою регуляризацією (Augmentation Level 2 + Weight Decay  $1e-4$  + Dropout 0.5), що дозволила частково стабілізувати навчання глибокої мережі.
4. **Ensemble (Soft Voting)**: ансамбль із 5 найкращих моделей, що об'єднує їх передбачення методом усереднення ймовірностей.

### Підсумкове порівняння моделей

Модель	Accuracy (Validation)	Accuracy (Test)	Розмір (Параметри)	Час навчання (Епохи)
BaseLineCNN	64%	63.5%	~3.67 млн	21 епоха
EfficientNet-B0 ( <code>freeze_until=2</code> )	72%	69.6%	~4.0 млн	9 епох
ResNet50 ( $wd: 1e-4$ , $d: 0.5$ , <code>freeze_until=3</code> )	62%	65.6%	~23.5 млн	17 епох
Ensemble (Soft Voting)	70%	70.6%	N/A (сума моделей)	N/A

## 2. Аналіз ефективності (Quality vs Efficiency)

### Точність класифікації:

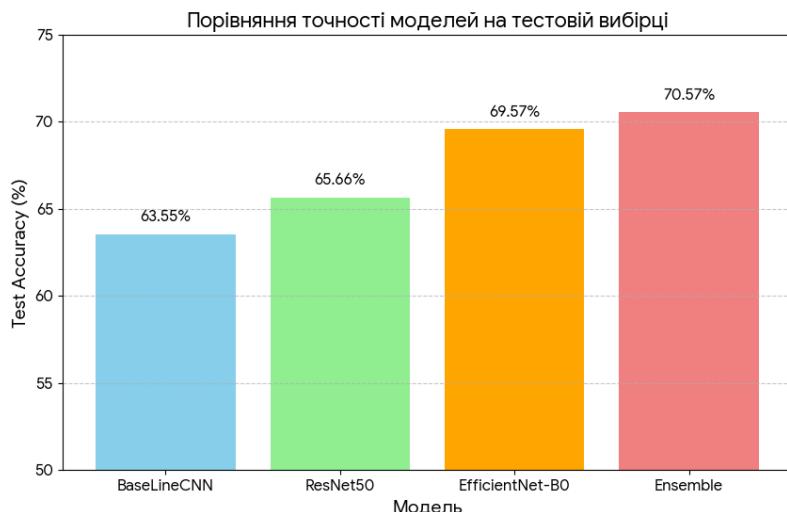
- Найвищу точність на тестовій вибірці продемонстрував підхід **Ensemble**, досягнувши рівня **70.57%**.
- Використання Transfer Learning (EfficientNet-B0) дозволило отримати приріст у **6.02%** порівняно з базовою CNN, що підтверджує важливість використання попередньо навчених ознак для медичних завдань з обмеженими даними.
- ResNet50, попри значно більшу глибину, поступився EfficientNet-B0. Це пояснюється надмірною ємністю моделі (23.5 млн параметрів) для датасету

з 2750 зображень, що призводило до перенавчання навіть за умов агресивної регуляризації.

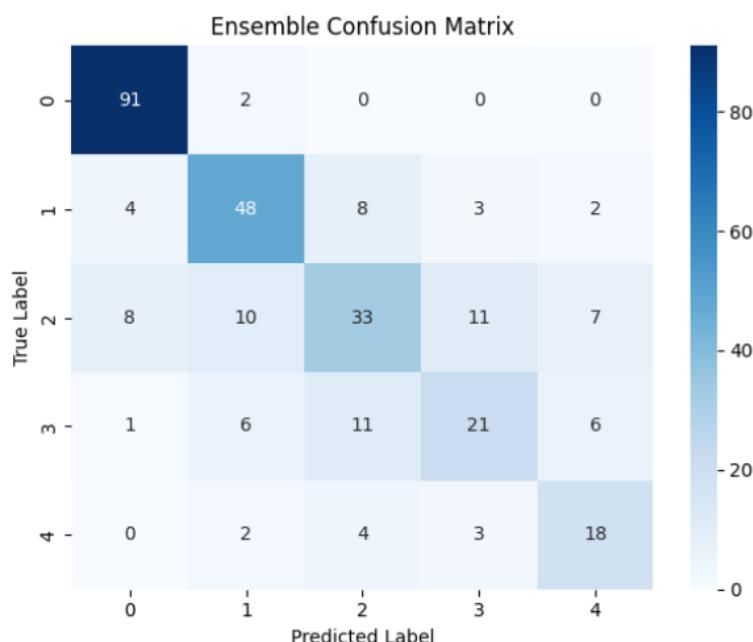
### Обчислювальна ефективність:

- **EfficientNet-B0** виявилася найбільш ефективною архітектурою. Маючи в ~6 разів менше параметрів ніж ResNet50 (4 млн проти 23.5 млн), вона не лише показала вищу точність, але й навчалася значно швидше та стабільніше.
- **BaseLineCNN** послужила хорошим бенчмарком, але потребувала більше епох для збіжності порівняно з pre-trained моделями.

### 3. Візуалізація порівняння



### 4. Фінальна матриця помилок (Ensamble моделей):



## 5.4. Висновки до розділу

1. **Оптимальна архітектура:** EfficientNet-B0 визначена як найбільш збалансоване рішення для задачі класифікації діабетичної ретинопатії на поточному датасеті. Вона поєднує високу точність з низькими вимогами до ресурсів.
2. **Роль Ансамблювання:**Хоча EfficientNet-B0 є найкращою одиночною моделлю, використання Ensemble дозволило підвищити надійність системи, особливо для критично важливих класів (Proliferate DR), нівелюючи індивідуальні помилки моделей.
3. **Обмеження ResNet:** Експерименти підтвердили, що просте збільшення глибини мережі (ResNet50) без відповідного збільшення обсягу даних не призводить до покращення результатів, а навпаки - ускладнює процес навчання через over-fitting.

## 4.5. Етап 5: Розробка Autoencoder та аналіз структури даних

### 4.5.1. Мотивація та постановка задачі

У попередніх етапах дослідження увага була зосереджена на **навченні з учителем (supervised learning)**, де головною метою була мінімізація помилки класифікації на основі відомих міток. Однак, для глибокого розуміння природи датасету та підвищення надійності системи діагностики, критично важливо дослідити саму структуру вхідних даних без прив'язки до міток.

Для цього використовується підхід **навчання без учителя (unsupervised learning)**, а саме - архітектура **автоенкодера (Autoencoder)**. Це нейронна мережа, яка навчається стискати вхідне зображення у компактний латентний вектор (кодування) і відновлювати його з максимальною точністю (декодування).

Впровадження автоенкодера в рамках цього проекту має дві практичні мети:

1. Детекція аномалій (Anomaly Detection):

Медичні датасети часто містять зображення низької якості, артефакти сканування або помилкові дані, які можуть негативно впливати на навчання класифікатора. Автоенкодер, навчений на репрезентативній вибірці "нормальних" зображень очного дна, матиме високу помилку реконструкції (MSE) на нестандартних або пошкоджених зображеннях. Це дозволить автоматизовано виявити та відфільтрувати "дивні" зразки.

2. Зменшення розмірності та візуалізація (Dimensionality Reduction):

Зображення очного дна мають високу розмірність (256x256x3), що унеможливлює їх пряму візуалізацію. Стиснення даних до латентного

простору ( $16 \times 16 \times 256$ ) з подальшою проекцією у 2D (за допомогою алгоритмів t-SNE або PCA) дозволить візуально оцінити:

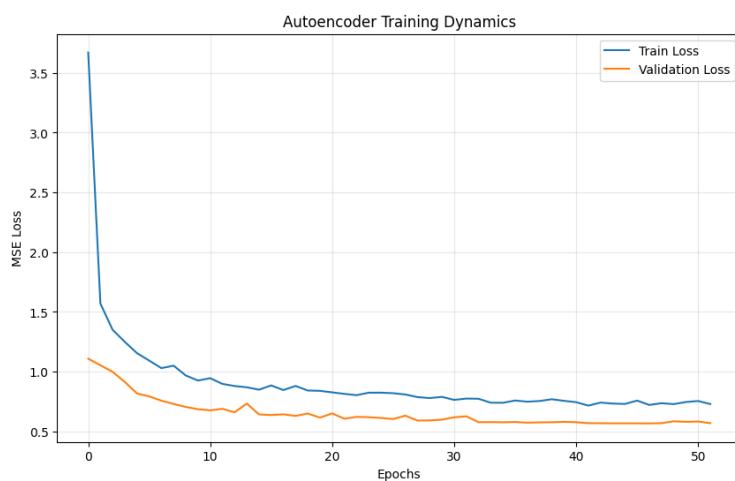
- Чи формують класи діабетичної ретинопатії чіткі кластери?
- Наскільки сильно перетинаються ознаки здорових пацієнтів та пацієнтів з початковою стадією хвороби?

#### 4.5.2. Тренування автоенкодера

**Конфігурація навчання** Для тренування моделі ConvAutoencoder (*код в Додатку 5*) було використано підхід реконструкції зображення. На відміну від задач класифікації, тут функція втрат вимірює, наскільки точно вихідне зображення відповідає входному після проходження через стиснуте латентне представлення.

- **Функція втрат:** `MSELoss` (Mean Squared Error) - обчислює середньоквадратичну різницю між пікселями оригіналу та реконструкції.
- **Оптимізатор:** Adam (з параметрами за замовчуванням).
- **Дані:** Використовувався повний набір даних для того, щоб модель вивчила загальні ознаки структури очного дна.

**Динаміка навчання** На графіку нижче зображено динаміку функції втрат (Loss) для тренувальної та валідаційної вибірок протягом 50 епох.



**Аналіз графіка:**

1. **Швидка збіжність:** Основне падіння помилки відбувається протягом перших 5-7 епох, де модель вивчає глобальні ознаки (форму диска зорового нерва, загальний контур сітківки).
2. **Стабільність:** Після 10-ї епохи криві виходять на плато. Важливо відзначити, що розрив між Train Loss та Validation Loss є мінімальним. Це свідчить про те, що автоенкодер добре узагальнює дані й не страждає від перенавчання (overfitting), успішно кодуючи навіть ті зображення, яких він не бачив.

### 4.5.3. Виявлення аномалій

Одним із ключових застосувань автоДенкодера в цьому проекті є автоматизована детекція аномалій. Принцип роботи базується на гіпотезі, що модель, навчена на загальному розподілі даних, матиме значно вищу помилку реконструкції (Reconstruction Error) на зображеннях, які містять нетипові артефакти або сильні патології.

#### Статистичний підхід

Для визначення порогу аномальності було використано статистичний аналіз помилок MSE на тестовій вибірці (299 зображень).

- **Середня помилка ( $\mu$ ):** 0.65125
- **Стандартне відхилення ( $\sigma$ ):** 0.35360

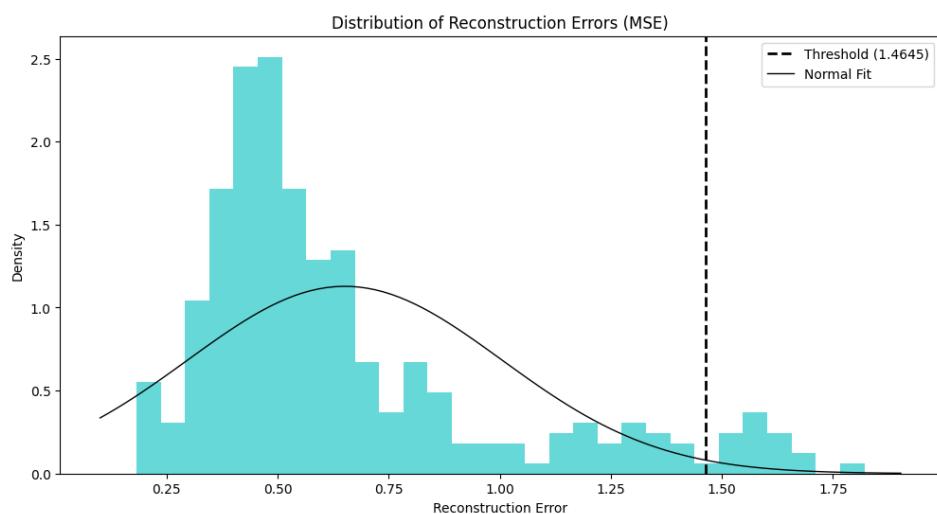
Поріг (Threshold) було встановлено за формулою ( $\mu + 2.3 * \sigma$ ), що охоплює приблизно 99% нормального розподілу даних.

- **Розрахований поріг:**  $0.65125 + 2.3 * 0.35360 = 1.46454$

#### Результати детекції

Застосувавши цей поріг до тестової вибірки, система ідентифікувала 17 аномалій з 299 зображень.

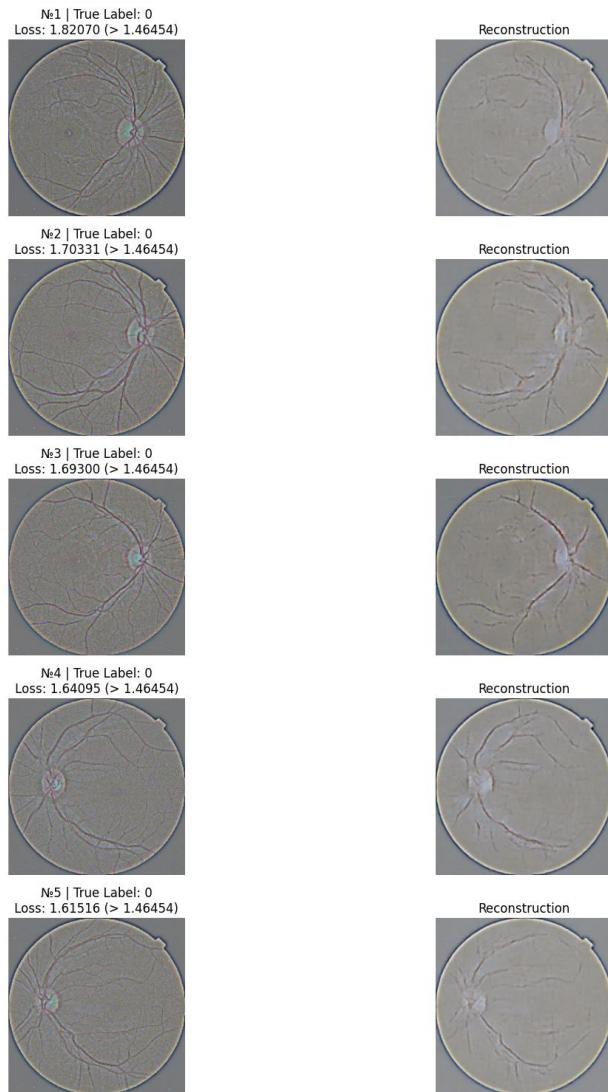
Розподіл помилок реконструкції показано на гістограмі нижче. Чорна пунктирна лінія позначає поріг відсікання.



Видно, що більшість зображень групуються в лівій частині графіка (низька помилка), тоді як "хвіст" розподілу праворуч від лінії відсікання представляє знайдені аномалії.

## Візуальний аналіз аномалій

Нижче наведено топ-5 виявлених аномалій з найбільшою помилкою реконструкції ( $\text{Loss} > 1.6$ ). Зліва - оригінал, справа - спроба реконструкції автоенкодером.



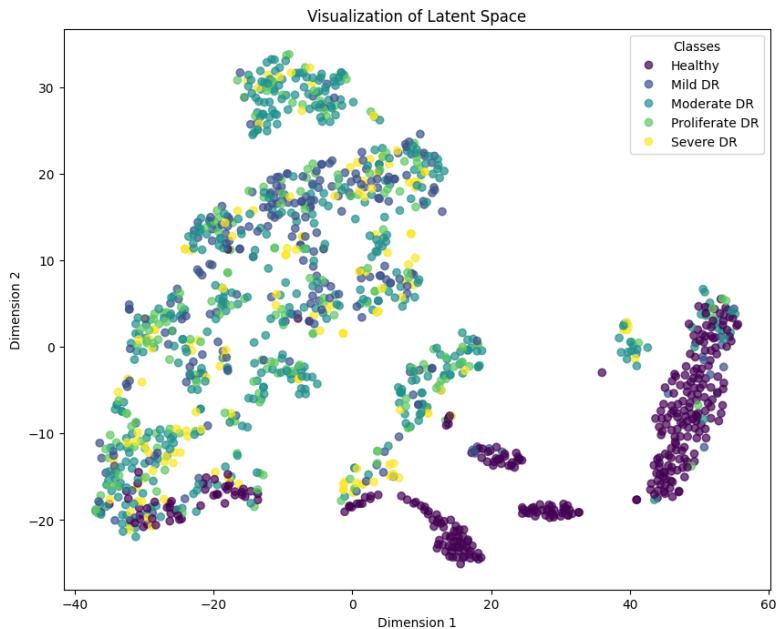
### Інтерпретація результатів:

На наведених прикладах видно, що автоенкодер працює як "фільтр низьких частот":

1. Він успішно відтворює загальну геометрію ока.
2. Він **не може** чітко відтворити дрібні високочастотні деталі, такі як тонкі судини або специфічні шуми/артефакти.
3. Саме нездатність відтворити ці складні деталі призводить до високого MSE. Це підтверджує, що метод ефективний для виявлення зображень, які суттєво відрізняються від "усередненого" стандарту (наприклад, знімки з поганим освітленням, шумами або важкими патологіями, що змінюють структуру судин).

#### 4.5.4. Візуалізація латентного простору

Для аналізу того, як модель структурує дані, було вилучено вектори ознак (embeddings) з "пляшкового горлечка" автоенкодера. За допомогою алгоритму t-SNE ці багатовимірні вектори були спроектовані у 2D простір.



**Аналіз кластеризації:** Графік демонструє складну структуру даних діабетичної ретинопатії:

1. **Клас "Healthy" (Фіолетовий):** Формує доволі щільний кластер (переважно в нижній правій частині). Це означає, що здорові очі візуально схожі між собою.
2. **Змішування класів:** Класи "Mild DR" (Синій) та "Moderate DR" (Бірюзовий) сильно перемішані з класом здорових очей. Це візуально підтверджує висновок попередніх розділів: ранні стадії хвороби важко відрізнити від норми, оскільки зміни в латентному просторі є неперервними, а не дискретними.
3. **Важкі стадії:** Класи "Proliferate DR" (Зелений) та "Severe DR" (Жовтий) розкидані ширше, часто знаходяться на периферії хмари точок. Це свідчить про високу варіативність ознак на пізніх стадіях хвороби.

**Висновок до розділу:** Автоенкодер виявився потужним інструментом для аналізу даних без учителя. Він дозволив автоматично виявляти аномальні зразки (17 випадків) та продемонстрував через візуалізацію t-SNE, що складність класифікації ретинопатії зумовлена плавним, градієнтним переходом між стадіями хвороби, а не чіткими межами між класами.

## 5. Висновок

У даній роботі було проведено комплексне дослідження методів глибокого навчання для задачі автоматизованої класифікації діабетичної ретинопатії на основі зображень очного дна. В умовах обмеженого набору даних (2750 зображень) та високої внутрішньокласової варіабельності було протестовано різні архітектурні підходи: від власної згорткової мережі до сучасних методів Transfer Learning та ансамблювання.

### **Основні результати дослідження:**

#### **1. Ефективність архітектур:**

- Базова CNN (Baseline) продемонструвала здатність до навчання (точність **63.5%**), але її можливості були обмежені відсутністю попередньо вивчених ознак.
- Використання **Transfer Learning** стало ключовим фактором покращення якості. Модель **EfficientNet-B0** виявилася оптимальним вибором, досягнувши точності **69.57%** при значно меншій кількості параметрів (~4 млн) порівняно з ResNet-50 (~23.5 млн).
- ResNet-50 продемонструвала схильність до швидкого перенавчання через надмірну ємність моделі відносно розміру датасету, навіть за умов агресивної регуляризації (Dropout, Weight Decay).

#### **2. Оптимальні стратегії навчання:**

- Дослідження показало, що просте вилучення ознак (Feature Extraction) з замороженим backbone є неефективним для медичних зображень через специфіку домену.
- Найкращою стратегією виявився **Fine-Tuning з частковим розморожуванням** (стратегія freeze\_until=2 для EfficientNet), що дозволило зберегти універсальні ознаки перших шарів та адаптувати глибокі шари до специфіки ретинопатії.
- Використання диференційованих learning rates (нижчого для backbone і вищого для класифікатора) було критично важливим для стабільності навчання.

#### **3. Роль ансамблювання:**

- Розроблений **Ensemble** з 5 найкращих моделей (Soft Voting) підвищив фінальну точність до **70.57%**.
- Найважливішим досягненням ансамблю стало не лише зростання Accuracy, а й покращення **Recall** та стабілізація розпізнавання критичних класів (Proliferate DR), що є пріоритетом для медичних діагностичних систем.

#### 4. Аналіз структури даних (Unsupervised Learning):

- Розроблений згортковий автоенкодер дозволив реалізувати модуль **детекції аномалій**, який успішно виявив 17 нетипових зразків у тестовій вибірці.
- Візуалізація латентного простору за допомогою t-SNE пояснила природу помилок класифікації: класи діабетичної ретинопатії не формують чітко розділених кластерів, а являють собою неперервний спектр змін (continuum). Значне перекриття класів "Healthy", "Mild" та "Moderate" підтверджує складність ранньої діагностики навіть для експертних систем.

## 6. Використані джерела

- 1) Goodfellow I. Deep Learning / I. Goodfellow, Y. Bengio, A. Courville. — Cambridge: MIT Press, 2016. — 800 p.
- 2) Chollet F. Deep Learning with Python / F. Chollet. — Manning Publications, 2017. — 384 p.
- 3) Diabetic Retinopathy Detection [Електронний ресурс] // Kaggle. — 2015. — Режим доступу: <https://www.kaggle.com/datasets/jockeroika/diabetic-retinopathy/data>. - Назва з екрана.
- 4) Tan M. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks / M. Tan, Q. V. Le // International Conference on Machine Learning (ICML). — 2019. — P. 6105–6114.
- 5) He K. Deep Residual Learning for Image Recognition / K. He, X. Zhang, S. Ren, J. Sun // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). — 2016. — P. 770–778.
- 6) Paszke A. PyTorch: An Imperative Style, High-Performance Deep Learning Library / A. Paszke et al. // Advances in Neural Information Processing Systems 32. — 2019. — P. 8024–8035.
- 7) Kingma D. P. Adam: A Method for Stochastic Optimization / D. P. Kingma, J. Ba // International Conference on Learning Representations (ICLR). — 2015.
- 8) Srivastava N. Dropout: A Simple Way to Prevent Neural Networks from Overfitting / N. Srivastava et al. // Journal of Machine Learning Research. — 2014. — Vol. 15. — P. 1929–1958.
- 9) Суботін С. О. Нейронні мережі: теорія та практика: навч. посіб. / С. О. Суботін. — Житомир: Вид-во ЖДУ ім. І. Франка, 2020. — 184 с.
- 10) Яровий А. А. Комп’ютерний зір: навч. посіб. / А. А. Яровий. — Вінниця: ВНТУ, 2017. — 165 с.
- 11) Simonyan K. Very Deep Convolutional Networks for Large-Scale Image Recognition / K. Simonyan, A. Zisserman // International Conference on Learning Representations (ICLR). — 2015.
- 12) Bank D. Autoencoders / D. Bank, N. Koenigstein, R. Giryes // arXiv preprint arXiv:2003.05991. — 2020.

## 7. Додатки

*Додаток 1. Код Baseline CNN*

```
class BaseLineCNN(nn.Module):
    def __init__(self, n_classes: int = 5, dropout_rate: float = 0):
        super().__init__()
        # conv layers
        self.featuresExtraction = nn.Sequential(
            # 1
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),

            # 2
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),

            # 3
            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )
        # avg pool
        self.avgpool = nn.AdaptiveAvgPool2d((4, 4))
        # classification
        self.classifier = nn.Sequential(
            nn.Linear(512 * 4 * 4, 256),
            nn.ReLU(),
            nn.Dropout(dropout_rate),
            nn.Linear(256, n_classes),
        )

    def forward(self, X: torch.Tensor) -> torch.Tensor:
        X = self.featuresExtraction(X)
        X = self.avgpool(X)
        X = torch.flatten(X, 1)
        X = self.classifier(X)
        return X
```

*Додаток 2. Код TransferResNet50*

```
class TransferResNet50(BaseTransferModel):
    def __init__(self, num_classes: int, mode: Literal['feature_extraction', 'fine_tuning'] = "feature_extraction",
                 freeze_until_layer: int = -1, dropout_rate: float = 0) -> None:
        super().__init__(num_classes, mode, dropout_rate)
        self._backbone = self._build_backbone()
        self._collect_backbone_layers()
        self._apply_freezing(freeze_until_layer)

    def _build_backbone(self) -> models.ResNet:
        model = models.resnet50(weights=models.ResNet50_Weights.IMAGENET1K_V2)
        in_features = model.fc.in_features
        if self.dropout_rate > 0:
            model.fc = nn.Sequential(
                nn.Dropout(self.dropout_rate),
                nn.Linear(in_features, self.num_classes)
            )
        else:
            model.fc = nn.Linear(in_features, self.num_classes)
        return model

    def _get_classifier_params(self) -> List:
        return self._backbone.fc.parameters()

    def _get_backbone_params(self) -> List:
        return [param for name, param in self._backbone.named_parameters() if "fc" not in name]

    def _collect_backbone_layers(self) -> None:
        self._backbone_layers = [
            self._backbone.conv1,
            self._backbone.bn1,
            self._backbone.layer1,
            self._backbone.layer2,
            self._backbone.layer3,
            self._backbone.layer4,
        ]
```

### Додаток 3. Код TransferEfficientNetB0

```
class TransferEfficientNetB0(BaseTransferModel):
    def __init__(self, num_classes: int, mode: Literal['feature_extraction', 'fine_tuning'] = "feature_extraction",
                 freeze_until_layer: int = -1, dropout_rate: float = 0) -> None:
        super().__init__(num_classes, mode, dropout_rate)
        self._backbone = self._build_backbone()
        self._collect_backbone_layers()
        self._apply_freezing(freeze_until_layer)

    def _build_backbone(self) -> models.EfficientNet:
        model = models.efficientnet_b0(weights=models.EfficientNet_B0_Weights.IMAGENET1K_V1)
        in_features = model.classifier[1].in_features
        if self.dropout_rate > 0:
            model.classifier = nn.Sequential(
                model.classifier[0],
                nn.Dropout(self.dropout_rate),
                nn.Linear(in_features, self.num_classes)
            )
        else:
            model.classifier = nn.Sequential(
                model.classifier[0],
                nn.Linear(in_features, self.num_classes)
            )
        return model

    def _get_classifier_params(self) -> List:
        return self._backbone.classifier.parameters()

    def _get_backbone_params(self) -> List:
        return [param for name, param in self._backbone.named_parameters() if "classifier" not in name]

    def _collect_backbone_layers(self) -> None:
        self._backbone_layers = list(self._backbone.features.children())
        self._backbone_layers.append(self._backbone.avgpool)
```

### Додаток 4. Код Ensemble

```
class EnsembleModel(nn.Module):
    def __init__(self, models: List[nn.Module], device: str = "cuda"):
        super().__init__()
        self.models = models
        self.device = device
        for model in self.models:
            model.to(self.device)
            model.eval()

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = x.to(self.device)

        all_probs = []
        with torch.no_grad():
            for model in self.models:
                logits = model(x)
                probs = torch.softmax(logits, dim=1)
                all_probs.append(probs)

        avg_probs = torch.stack(all_probs).mean(dim=0)
        return avg_probs
```

## Додаток 5. Код ConvAutoencoder

```

class ConvAutoencoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.encoder = nn.Sequential(
            # (batch_size x 256 x 256 x 3)
            # -> 256x256
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            # -> 128x128
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            # -> 64x64
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            # -> 32x32
            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            # -> 16x16
            # (batch_size x 16 x 16 x 256)
        )
        self.decoder = nn.Sequential(
            # (batch_size x 16 x 16 x 256)
            # -> 16x16
            nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2),
            nn.BatchNorm2d(128),
            nn.ReLU(True),
            # -> 32x32
            nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2),
            nn.BatchNorm2d(64),
            nn.ReLU(True),
            # -> 64x64
            nn.ConvTranspose2d(64, 32, kernel_size=2, stride=2),
            nn.BatchNorm2d(32),
            nn.ReLU(True),
            # -> 128x128
            nn.ConvTranspose2d(32, 3, kernel_size=2, stride=2)
            # -> 256x256
            # (batch_size x 256 x 256 x 3)
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.encoder(x)
        x = self.decoder(x)
        return x

    def get_embedding(self, x: torch.Tensor) -> torch.Tensor:
        with torch.no_grad():
            encoded = self.encoder(x) # (b, 256, 16, 16)
            embedding = F.adaptive_avg_pool2d(encoded, (1, 1))
        return torch.flatten(embedding, 1)

```