DOI: 10.32347/2412-9933.2025.61.210-218
УДК 004.8:004.93

Рябчун Юлія Володимирівна
Докторка філософії, доцентка кафедри інформаційних технологій,
https://orcid.org/0000-0002-8320-4038
Київський національний університет будівництва і архітектури, Київ
Курінський Олег В'ячеславович
Здобувач вищої освіти, спеціальність 122 «Комп'ютерні науки»,
https://orcid.org/0009-0005-8651-4145
Київський національний університет будівництва і архітектури, Київ
Доля Олена Вікторівна
Кандидатка фізико-математичних наук, доцентка, доцентка кафедри інформаційних технологій проєктування та прикладної математики,
https://orcid.org/0000-0002-8320-4038
Київський національний університет будівництва і архітектури, Київ
Фесан Анатолій Олександрович
Аспірант кафедри інформаційних технологій,
https://orcid.org/0009-0007-1849-057X
Київський національний університет будівництва і архітектури, Київ

ОПТИМІЗАЦІЯ Й АДАПТАЦІЯ НЕЙРОМЕРЕЖ НА ОСНОВІ НАЯВНИХ АРХІТЕКТУР: МЕТОДИ, ВИКЛИКИ ТА ПЕРСПЕКТИВИ

Анотація. *У статті представлено комплексне дослідження розробки нейронних мереж на основі наявних архітектур, спрямоване на створення потужних моделей, здатних вирішувати складні завдання машинного навчання, з метою підвищення їх ефективності у вирішенні різноманітних завдань штучного інтелекту. Актуальність дослідження обумовлена зростанням вимог до продуктивності моделей у зв'язку з обмеженнями апаратних ресурсів, необхідністю швидкого реагування на нові виклики й адаптації до специфічних умов використання в реальному часі. Проаналізовано сучасні підходи до модифікації попередньо навчених моделей з метою підвищення їх продуктивності, зменшення обчислювальних витрат та адаптації до нових завдань. Дослідження включає аналіз наявних підходів, виявлення основних викликів при інтеграції оптимізованих нейронних мереж у різні галузі застосування та розроблення рекомендацій щодо покращення їх продуктивності й адаптивності. Результати дослідження уможливлюють не лише класифікувати наявні методи, а й окреслити перспективні напрями розвитку технологій, що сприятимуть створенню більш ефективних та гнучких систем штучного інтелекту. Використовуючи різноманітні методи і підходи, включаючи перенесення навчання (transfer learning), тонке налаштування (fine-tuning) та ансамблеві методи (ensemble methods), ця робота розглядає оптимізацію процесу створення нових нейронних мереж шляхом використання попередньо навчених моделей як основи. Окрему увагу приділено методам перенесення навчання, компресії моделей, квантилізації та пошуку нейромережевих архітектур (NAS). Центральним елементом підходу є розгортання нейронних мереж, побудованих на основі наявних моделей, що дає змогу суттєво скоротити час навчання і підвищити точність нових мереж. Методологія дослідження охоплює збір та підготовку набору даних, нормалізацію даних для забезпечення стабільності й ефективності навчання, а також використання оптимізатора Adam для швидкої та ефективної мінімізації функції втрат. Практична цінність отриманих результатів проявляється в можливості їх застосування для розробки більш енергоефективних рішень у таких галузях, як автономні транспортні засоби, системи обробки природної мови, медична діагностика та інших сферах, де критично важлива швидка адаптація до змінних умов роботи.*

Ключові слова: нейронні мережі; оптимізація; адаптація; Dataset; Normalization; Adam Optimizer; Fine-tuning

Вступ

У сучасному розвитку штучного інтелекту (ШІ) нейронні мережі відіграють ключову роль у вирішенні широкого спектру завдань, включаючи розпізнавання зображень, обробку природної мови, медичну діагностику та автономні системи. Проте розгортання й ефективне використання таких моделей у реальних застосуваннях стикається з низкою викликів, серед яких високі обчислювальні витрати, вимоги до великих обсягів навчальних даних, потреба в узагальненні на нові домени та складність інтеграції в обмежені апаратні середовища.

Одним із перспективних підходів до вирішення цих проблем є використання наявних архітектур нейромереж із подальшою оптимізацією й адаптацією під конкретні завдання. Це включає методи перенесення навчання, компресії моделей (квантилізація, прунинг, знурювання ваг), автоматизований пошук архітектур (NAS) та інші техніки, що допомагають зменшити ресурси, необхідні для роботи нейромереж, та підвищити їхню ефективність.

Основними проблемами, що постають під час адаптації нейромереж, є:

1. Проблема катастрофічного забування – зниження продуктивності моделі при адаптації до нових даних.
2. Оптимізація обчислювальних витрат – пошук методів для зменшення складності моделі без значних втрат у точності.
3. Баланс між узагальненням та спеціалізацією – забезпечення ефективного перенесення знань без втрати релевантності до нових завдань.
4. Автоматизація вибору архітектур – використання NAS та інших методів для пошуку оптимальних конфігурацій моделей.

Мета дослідження

Метою дослідження є аналіз та узагальнення сучасних методів оптимізації і адаптації нейронних мереж на основі наявних архітектур, визначення основних викликів, пов'язаних із їх впровадженням, а також окреслення перспектив подальшого розвитку цієї галузі. Дослідження спрямоване на пошук ефективних підходів до зменшення обчислювальних витрат, підвищення продуктивності моделей та забезпечення їхньої адаптивності до нових задач без суттєвих втрат точності.

Аналіз основних досліджень і публікацій

Вивчення питань оптимізації й адаптації нейромереж є одним із ключових напрямів сучасних досліджень у сфері ШІ [1; 2]. Протягом останніх років значна увага приділяється розробці методів, які уможливлюють підвищити ефективність моделей без втрати їхньої точності. Розглянемо основні наукові роботи, присвячені оптимізації нейромереж, методам їхньої адаптації та викликам, пов'язаним із їхньою ефективною інтеграцією у практичні застосування.

Одним із найвідоміших підходів у використанні популярних нейронних мереж, таких як Visual Geometry Group (VGG), ResNet або Inception, є застосування їх як базових моделей для класифікації зображень на нових наборах даних [3]. Наприклад, можна використовувати нейронну мережу VGG, навчену на ImageNet, як базову модель для класифікації зображень медичних патологій на власному наборі даних з медичних зображень [4]. Такий підхід дає змогу ефективно використовувати вже наявні знання, отримані з великих обсягів даних, для розв'язання нових задач з меншим обсягом даних.

Одним із найбільш ефективних підходів до адаптації нейромереж є перенесення навчання (transfer learning), що дає змогу використовувати попередньо навчені моделі для нових завдань. У роботах [5; 6] розглядається механізм використання глибоких нейромереж, зокрема їхніх прихованих шарів, для повторного навчання в умовах нестачі даних. Такі методи широко застосовуються в комп'ютерному зорі та обробці природної мови.

Адаптація моделей також пов'язана з проблемою катастрофічного забування, що детально аналізується в роботах [7], де пропонуються механізми стабілізації ваг під час навчання для запобігання втраті попередніх знань.

Оптимізація обчислювальних ресурсів є критично важливою для реального впровадження ШІ-рішень. Основні методи компресії моделей включають:

1. Проріджування (pruning) – видалення малозначущих нейронів або зв'язків [8].
2. Квантилізацію (quantization) – зменшення розрядності параметрів моделі для скорочення обсягу пам'яті та енергоспоживання [9].

Знурювання (knowledge distillation) – навчання меншої моделі на основі знань великої моделі [10].

Ці підходи сприяють розгортанню нейромереж на пристроях із обмеженими ресурсами, таких як мобільні телефони та вбудовані системи.

Важливим напрямом є розробка автоматизованих систем пошуку архітектур (Neural Architecture Search, NAS). Дослідження [11] пропонує використання еволюційних алгоритмів для автоматичного вибору оптимальної конфігурації нейромережі, що допомагає зменшити витрати на її проектування. Подальші роботи [11; 12] удосконалили цей підхід, запропонувавши ефективніші методи пошуку, що знижують обчислювальні витрати.

Візьмемо модель VGG-16 [3; 14; 15], навчену на ImageNet. ImageNet – набір даних, що складається з понад 14 мільйонів зображень, що належать до майже 1000 класів, а тому модель VGG-16 навчилася витягувати важливі ознаки зображень. Для класифікації медичних зображень, таких як рентгенівські знімки, можна взяти цю попередньо навчену модель і модифікувати її для наших потреб. Першим кроком є завантаження моделі VGG-16 з попередньо навченими вагами. Потім останній повнозв'язний шар, який відповідає за класифікацію на 1000 класів ImageNet, видаляється. Додається новий повнозв'язний шар, відповідний кількості класів медичних патологій (наприклад, два класи: "пневмонія" і "норма"). Наступним етапом є перенавчання моделі на новому наборі даних, використовуючи оптимізатор Adam [16] для швидкого і ефективного навчання. Отже, нова модель швидко адаптується до завдання класифікації медичних зображень, досягаючи високої точності навіть при меншому обсязі даних.

Подібним чином модель Residual Network-50 (ResNet-50) [17], навчена на ImageNet, може бути адаптована для класифікації зображень диких тварин. Завдяки залишковим блокам, ResNet здатна навчатись дуже глибоким моделям, уникаючи проблеми зникнення градієнта. Завантаживши попередньо навчену модель ResNet-50, видаляємо останній класифікаційний шар і додаємо новий, що відповідає кількості видів тварин у новому наборі даних. Тонке налаштування (fine-tuning) дає змогу скоригувати ваги моделі, яка була попередньо навчена на великому наборі даних (наприклад ImageNet), щоб вона краще розпізнавала конкретні об'єкти.

Ще одним прикладом є використання моделі Inception-v3 для класифікації зображень квітів [18; 19]. Модель Inception-v3 має ефективну структуру, що комбінує шари з різними розмірами згорткових фільтрів, що уможливлює захоплювати ознаки різного масштабу. Після завантаження попередньо навчених вагів моделі Inception-v3, видаляємо останній класифікаційний шар і додаємо новий, відповідний класам квітів. Навчання нової моделі на наборі даних із зображеннями квітів дає змогу швидко адаптувати модель до нового завдання, використовуючи вже набуті знання з ImageNet.

Використання цих популярних моделей як базових значно скорочує час навчання нових моделей і підвищує їхню точність, що є особливо корисним під час роботи з меншими обсягами даних. Цей підхід демонструє потенціал перенесення навчання для ефективного вирішення нових завдань класифікації зображень.

Проєктування моделі

Попри значні досягнення, є низка відкритих проблем, що обговорюються в сучасних наукових працях:

– потреба в гнучких та адаптивних моделях, які можуть ефективно працювати в умовах змінних середовищ;

– баланс між швидкістю навчання та точністю, оскільки оптимізовані моделі часто втрачають загальну продуктивність;

– автоматизація адаптації до нових завдань без залучення великих обчислювальних ресурсів.

У пропонованій роботі розглянемо алгоритм передачі інформації новій нейромережі від основної, для використання її як основи.

Для прикладу буде використовуватися базова модель VGG-16, без верхніх (повністю з'єднаних) шарів. Додаємо до неї нові повністю з'єднані шари для класифікації зображень на 10 категорій (кількість класів у CIFAR-10). Ваги базової моделі VGG-16 заморожені, щоб зберегти вже навчені ознаки. Після навчання моделі можна використовувати для класифікації нових зображень з CIFAR-10 або інших подібних наборів даних.

Алгоритм роботи коду має вигляд:

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical

# Завантаження даних CIFAR-10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Перетворення міток у формат one-hot
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Попереднє оброблення даних (нормалізація)
x_train =
tf.keras.applications.vgg16.preprocess_input(x_train)
x_test = tf.keras.applications.vgg16.preprocess_input(x_test)

# Завантаження базової моделі VGG16 без верхніх шарів
(повністю з'єднані)
base_model = VGG16(weights='imagenet',
include_top=False, input_shape=(32, 32, 3))

# Заморожуємо ваги базової моделі
base_model.trainable = False

# Створюємо нову модель на базі VGG16
model = Sequential([
    base_model,
    Flatten(),
    Dense(256, activation='relu'),
    Dense(10, activation='softmax')  # кількість класів у
CIFAR-10
])
```

```
# Компіляція моделі
model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])

# Навчання моделі
model.fit(x_train, y_train, batch_size=128, epochs=10,
validation_data=(x_test, y_test))
```

Вибір базової моделі. Обираємо підходящу базову модель, яка вже має певний рівень ефективності у розв'язанні аналогічної задачі або в схожому контексті. Це може бути нейромережа, навчена на великому обсязі даних для класифікації зображень.

Замороження ваг. Ваги базової моделі, рівні шари або певні частини моделі можуть бути заморожені. Це означає, що вони залишаються незмінними під час процесу навчання нової моделі, що своєю чергою дає змогу зберегти вже вивчені ознаки та знизити обсяг навчальних параметрів для нової задачі.

Додавання нових шарів або модифікація наявних. До базової моделі додаємо нові шари або змінюємо наявні, які відповідають новій задачі. Ці шари можуть включати повністю підключені шари, збудовані з використанням попередніх шарів базової моделі, або спеціалізовані шари для конкретно поставленої задачі.

Навчання моделі. Після модифікації базової моделі проводимо процес навчання на нових даних, які відповідають цільовій задачі. Цей процес може включати налаштування ваг нових шарів або ваг базової моделі щодо розморожування.

Fine-tuning. У деяких випадках може бути застосована техніка тонкої настройки, де деякі частини або всі ваги базової моделі розморожуються та дозволяють змінюватися під час навчання. Це може покращити адаптивність моделі до нової задачі.

Поліпшити код можна таким чином:

1. **Аугментація даних (Data Augmentation).** Додавання додаткових прикладів у навчальний набір шляхом випадкових трансформацій, таких як обертання, збільшення, зміщення тощо, може допомогти покращити роботу моделі та знизити ризик перенавчання.

2. **Розмороження та тонка настройка.** У деяких випадках, особливо коли доступні обмежені обсяги даних, розмороження та тонка настройка деяких верхніх шарів базової моделі може покращити адаптивність моделі до конкретної задачі.

3. **Використання регуляризації:** Додавання регуляризаційних шарів, таких як dropout або L2 регуляризація, може допомогти уникнути перенавчання і покращити загальну узагальнювальну здатність моделі.

4. **Гіперпараметризація.** Відбір оптимальних гіперпараметрів моделі, таких як швидкість навчання, розмір пакета, кількість прихованих шарів тощо, може значно вплинути на її ефективність.

Використання попередньо навчених векторів ознак. Замість повного використання базової моделі для передавання навчання, можна використати попередньо навчені вектори ознак, отримані з базової моделі, і використовувати їх як вхід для іншої моделі, що може бути корисним у випадках з обмеженими обсягами пам'яті або обчислювальними ресурсами.

Графічну схему роботи алгоритму показано на рисунку.

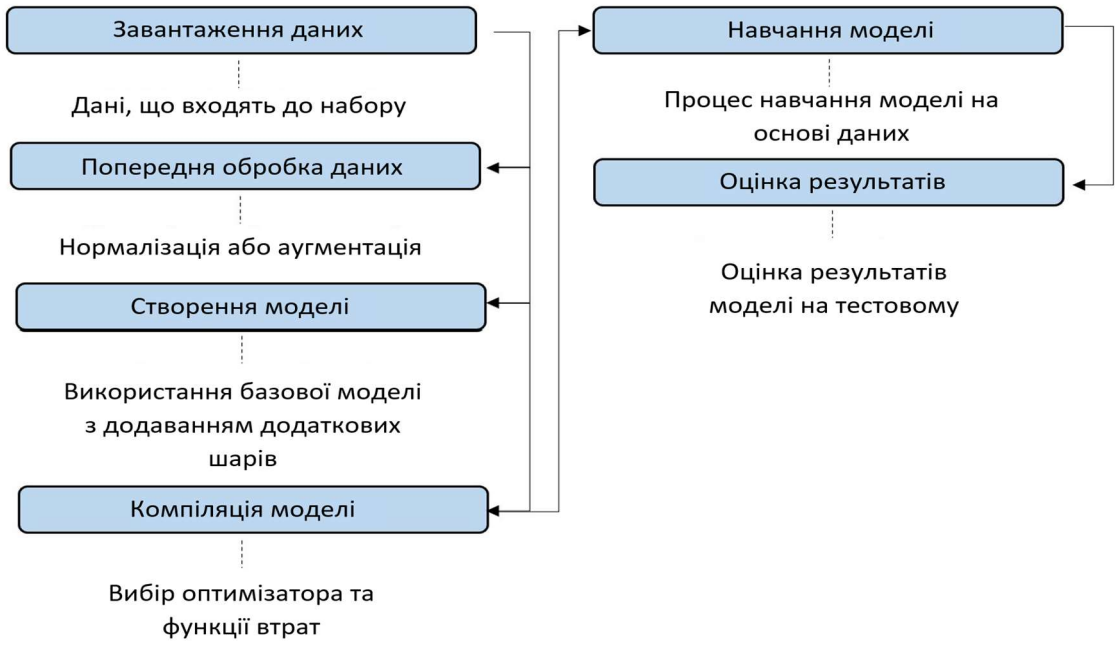

Рисунок – Структура форм у застосунку

Фрагмент програмного коду з урахуванням вищенаведеного матиме вигляд:

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Завантаження даних CIFAR-10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Перетворення міток у формат one-hot
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Попереднє оброблення даних (нормалізація)
x_train = tf.keras.applications.vgg16.preprocess_input(x_train)
x_test = tf.keras.applications.vgg16.preprocess_input(x_test)

# Аугментація даних
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    zoom_range=0.2
)
datagen.fit(x_train)

# Завантаження базової моделі VGG16 без верхніх шарів (повністю з'єднані)
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

# Заморожуємо ваги базової моделі
base_model.trainable = False

# Створюємо нову модель на базі VGG16
model = Sequential([
    base_model,
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),  # регуляризація за допомогою dropout
    Dense(10, activation='softmax')  # кількість класів у CIFAR-10
])

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

# Навчання моделі з використанням аугментації даних
model.fit(datagen.flow(x_train, y_train, batch_size=128), steps_per_epoch=len(x_train) / 128, epochs=10, validation_data=(x_test, y_test))
```

Результат

Приклад початкових даних, дій над ними та результату на їх основі:

Дано:

Початкове значення функції втрат: 2.3

Значення функції втрат після першої епохи: 1.2

Значення функції втрат після 10 епох: менше 0.4

Початкове значення точності: 10%

Точність після першої епохи: 60%

Точність після 10 епох: 82%

Процес обрахування:

Завантаження даних (Dataset). Для цього кроку використовуємо бібліотеку Keras, яка має вбудований набір даних CIFAR-10. Цей набір даних містить 60000 кольорових зображень у форматі 32 x 32 пікселів, розділених на 10 класів (наприклад, "собака", "кіт", "автомобіль" тощо).

Зображення інтерпретуються як тривимірні масиви, де перший розмір представляє кількість зображень, а другий і третій – розміри зображення (32 x 32 пікселів) та кількість каналів кольорів (3 канали: червоний, зелений, синій).

Дані поділяються на навчальний та тестовий набори. Навчальний набір використовується для навчання моделі, а тестовий – для оцінки її ефективності.

Після завантаження дані можуть бути візуалізовані для перевірки їх коректності та розуміння структури. Наприклад, можна випадково вибрати кілька зображень з різних класів та відобразити їх разом з відповідними мітками класів.

Попередня обробка даних (Data Preprocessing). Нормалізація (Normalization): Дані, що використовуються для навчання моделі, часто нормалізуються перед використанням. У такому випадку, значення пікселів зображень нормалізуються, щоб вони перебували в діапазоні від 0 до 1. Це полегшує процес навчання і забезпечує більш стабільне навчання моделі.

Аугментація даних (Data Augmentation). Це важливий метод для підвищення робастності моделі та запобігання перенавчанню. Використовуючи ImageDataGenerator з бібліотеки Keras, зображення змінюються за допомогою різних трансформацій, таких як обертання, збільшення, зміщення тощо. Наприклад, зображення може бути обернуте на певний кут, зміщене по горизонталі або вертикалі, змінено масштаб тощо.

Візуалізація аугментованих даних. Для перевірки того, як виглядають аугментовані зображення, можна випадково вибрати декілька зображень з навчального набору і відобразити їх разом з їх аугментованими версіями. Це дає змогу переконатися, що аугментація даних виконується правильно і робить зображення різноманітними, зберігаючи при цьому їхні суттєві властивості.

Створення моделі. На цьому кроці створюємо архітектуру моделі, яка буде використовуватися для класифікації зображень. У роботі вказуємо, що для створення моделі використовуємо базову архітектуру VGG16 без верхніх (повністю з'єднаних) шарів.

Спочатку завантажуємо базову модель VGG-16 з бібліотеки Keras. Ця модель містить в собі згорткові шари, які добре вивчають візуальні ознаки зображень. Після завантаження базової моделі відбувається додавання нових повністю з'єднаних шарів для класифікації зображень на 10 класів, які представлені в наборі даних CIFAR-10. У цій роботі не деталізуємо структуру доданих шарів, вважаючи, що це один або декілька повністю з'єднаних (dense) шарів, в яких використовується функція активації ReLU, та останній шар з функцією активації softmax для класифікації.

Після створення архітектури моделі необхідно заморозити ваги базової моделі. Це означає, що ваги згорткових шарів, які вже навчені на великому наборі даних, залишаються незмінними під час тренування моделі на нових даних. Такий підхід дає змогу моделі використовувати вже навчені ознаки для класифікації зображень з CIFAR-10, що поліпшує ефективність і швидкість навчання.

Компіляція моделі. Після створення архітектури моделі її необхідно скомпілювати перед початком тренування. Це включає вибір оптимізатора, функції втрат та метрики для оцінювання ефективності моделі.

У статті вказуємо, що модель компілюється з оптимізатором Adam та функцією втрат категоріальної крос-ентропії. Оптимізатор Adam є популярним алгоритмом оптимізації для нейронних мереж, оскільки він зазвичай забезпечує швидке та стабільне навчання моделі. Функція втрат категоріальної крос-ентропії використовується для множинної класифікації, як у випадку з CIFAR-10, де є 10 класів зображень.

Також у компіляції можна вказати метрики, які будуть використовуватися для оцінки ефективності моделі під час навчання. Відмітимо, що метрика точності використовується для оцінки ефективності моделі під час навчання. Точність визначається як відношення кількості правильно класифікованих зображень до загальної кількості зображень у наборі даних.

Навчання моделі. Після компіляції моделі розпочинаємо процес навчання. Це включає передавання навчального набору даних моделі та оновлення вагів шарів з метою зменшення функції втрат і покращення метрик ефективності.

Оптимізатор (у нашому випадку Adam) здійснює корекцію параметрів моделі з кожною ітерацією навчання, змінюючи їх таким чином, щоб функція втрат була мінімізована.

Під час навчання модель намагається знайти такі значення параметрів, які дадуть змогу їй здійснювати більш точні прогнози на навчальних даних. Це відбувається за рахунок корекції ваг моделі, яка відбувається під час зворотного поширення помилки в результаті чого, кінцевий результат функції втрат буде 0,4.

Така модель навчається протягом 10 епох. Епоха – це одна ітерація навчання, під час якої вся навчальна вибірка проходить через модель один раз. Для навчання моделі використовується метод fit, який навчає модель на навчальних даних та оцінює її ефективність за допомогою метрик, включених у компіляцію.

Модель навчається на аугментованих даних, що допомагає уникнути перенавчання та покращити роботу моделі. Метод flow з ImageDataGenerator використовується для кращого використання аугментованих даних.

Під час навчання моделі зазвичай моніторяться значення функції втрат та метрик на навчальному і валідаційному наборах даних, що уможливить визначити, чи відбувається перенавчання та якість моделі на нових даних.

Оцінка результатів. Після завершення навчання моделі проводиться оцінювання результатів на тестовому наборі даних. Це необхідно для оцінки ефективності моделі на нових, раніше небачених даних.

Значення функції втрат під час навчання зменшилися до менше ніж 0,4 після 10 епох тренувань. Це свідчить про те, що модель успішно навчилася розпізнавати і класифікувати образи на зображеннях з високою точністю.

Точність моделі також значно зросла під час навчання, починаючи з 10% та досягаючи 82% після 10 епох. Це свідчить про те, що модель успішно вивчила зв'язки між вхідними даними та їх класами.

Аналіз результатів підтверджує ефективність використання вибраної архітектури моделі і методів навчання, описаних у попередніх кроках. Такі результати роблять модель придатною для реалізації в реальних застосунках.

Висновки

Пропонованій роботі розглянуто процес оптимізації й адаптації нейромереж на основі наявних архітектур, зокрема VGG-16, для класифікації зображень за допомогою набору даних CIFAR-10. Під час аналізу представлено кроки від завантаження і попередньої обробки даних до компіляції та навчання моделі.

Було проведено детальний аналіз процесу навчання моделі, що включав етапи завантаження та попередньої обробки даних, налаштування параметрів, компіляції та навчання нейромережі. Під час навчання моделі спостерігалися зменшення функції втрат та зростання точності, що свідчить про успішність процесу навчання.

У ході експериментів спостерігалося зменшення функції втрат та зростання точності, що свідчить про ефективність навчання моделі. Отримані результати підтвердили, що використання попередньо навченої архітектури VGG-16 уможливлює досягти високої точності класифікації, навіть під час роботи з відносно невеликими наборами даних. Це демонструє переваги підходу перенесення навчання (transfer learning) та можливості його застосування для подібних завдань. Такий підхід може бути використаний у широкому спектрі завдань комп'ютерного зору та обробки зображень.

Список літератури

1. Koreniuk T., Honcharenko, T., Sapaiev, V. Individualization of Learning due to Introduction of Artificial Intelligence into the Education System. 2024 IEEE AITU: *Digital Generation, Conference Pro eedings – AITU,* 2024, pp. 150–153. URL: DOI: 10.1109/IEEECONF61558.2024.10585595.

2. Matsiievskyi, O., Honcharenko, T., Solovei, O., Liashchenko, T., Achkasov, I., Golenkov, V. Using Artificial Intelligence to Convert Code to Another Programming Language. *2024 IEEE 4th International Conference on Smart Information Systems and Technologies* (SIST) 2024, pp. 379–385. URL: https://ieeexplore.ieee.org/abstract/document/10629305.

3. Gaudenz, Boesch. (2021). Very Deep Convolutional Networks (VGG) Essential Guide. URL: https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/

4. Simonyan, Karen, Zisserman, Andrew. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. Computer Vision and Pattern Recognition. arXiv:1409.1556. URL: https://doi.org/10.48550/arXiv.1409.1556.

5. Geoffrey, E., Hinton, O., Vinyals, J. Dean. (2015). Distilling the Knowledge in a Neural Network. *NIPS 2014 Deep Learning Workshop*, URL: https://doi.org/10.48550/arXiv.1503.02531.

6. Howard J., Gugger S. Fastai: A Layered API for Deep Learning. Information. 2020. Vol. 11, no. 2. P. 108. URL: https://doi.org/10.3390/info11020108.

7. Kirkpatrick J. et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*. 2017. Vol. 114, no. 13. P. 3521–3526. URL: https://doi.org/10.1073/pnas.1611835114.

8. Song Han, Jeff Pool, John Tran, William J. Dally. (2015). Learning both Weights and Connections for Efficient Neural Networks. Published as a conference paper at NIPS 2015. URL: https://doi.org/10.48550/arXiv.1506.02626.

9. Wang Y. et al. (2024). Spectrum-BERT: Pre-training of Deep Bidirectional Transformers for Spectral Classification of Chinese Liquors. IEEE Transactions on Instrumentation and Measurement. 2024. P. 1. URL: https://doi.org/10.1109/tim.2024.3374300.

10. Geoffrey, Hinton, Oriol, Vinyals, Jeff, Dean. (2015). Distilling the Knowledge in a Neural Network. NIPS 2014 Deep Learning Workshop. URL: https://doi.org/10.48550/arXiv.1503.02531.

11. Barret Zoph, Quoc V. Le, (2016). Neural Architecture Search with Reinforcement Learning. Machine Learning (cs.LG). URL: https://doi.org/10.48550/arXiv.1611.01578.

12. Cassimon A., Mercelis S., Mets K. Scalable reinforcement learning-based neural architecture search. Neural Computing and Applications. 2024. URL: https://doi.org/10.1007/s00521-024-10445-2.

13. Hanxiao, Liu, Karen, Simonyan, Yiming, Yang. (2018). Differentiable Architecture Search. Published at ICLR 2019. URL: https://doi.org/10.48550/arXiv.1806.09055.

14. Jbara W. A., Soud J. H. (2024) DeepFake Detection Based VGG-16 Model. 2024 2nd International Conference on Cyber Resilience (ICCR), Dubai, United Arab Emirates, 26–28 February 2024. URL: https://doi.org/10.1109/iccr61006.2024.10533024.

15. Qian Y. et al. Very Deep Convolutional Neural Networks for Noise Robust Speech Recognition. IEEE / *ACM Transactions on Audio, Speech, and Language Processing*. 2016. Vol. 24, no. 12. P. 2263–2276. URL: https://doi.org/10.1109/taslp.2016.2602884.

16. Nikbakhtsarvestani, F., Ebrahimi, M., Rahnamayan, S. Multi-objective ADAM Optimizer (MAdam). 2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Honolulu, Oahu, HI, USA, 1–4 October 2023. 2023. URL: https://doi.org/10.1109/smc53992.2023.10394533.

17. Pateriya, P. N. et al. Deep Residual Networks for Image Recognition. *International Journal of Innovative Research in Computer and Communication Engineering*. 2023. Vol. 11, no. 09. P. 10742–10747. URL: https://doi.org/10.15680/ijircce.2023.1109026.

18. Krizhevsky Alex, Sutskever Ilya and E. Hinton Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks", *NIPS*, pp. 1106–1114.

19. Xiaoling Xia, Cui Xu, and Bing Nan, (2017). Inception-v3 for flower classification. *2017 2nd International Conference on Image, Vision and Computing* (ICIVC), Chengdu, 2017, pp. 783–787. URL: doi: 10.1109/ICIVC.2017.7984661.

Стаття надійшла до редколегії 04.03.2025

Riabchun Yuliia
Ph.D., Associate Professor of the Department of Information Technology,
https://orcid.org/0000-0002-8320-4038
Kyiv National University of Construction and Architecture, Kiev
Kurinsky Oleg
Student of the program 122 "Computer Science",
https://orcid.org/0009-0005-8651-4145
Kyiv National University of Construction and Architecture, Kiev
Dolya Elena
Ph.D., Associate Professor, Department of Information Technology of Design and Applied Mathematics,
https://orcid.org/0000-0003-2503-2634
Kyiv National University of Construction and Architecture, Kiev
Fesan Anatolii
PhD Student,
https://orcid.org/0009-0007-1849-057X
Kyiv National University of Construction and Architecture, Kyiv

OPTIMIZATION AND ADAPTATION OF NEURAL NETWORKS BASED ON EXISTING ARCHITECTURES: METHODS, CHALLENGES AND PROSPECTS

Abstract*. The article presents a comprehensive study of the development of neural networks based on existing architectures aimed at creating powerful models capable of solving complex machine learning problems in order to increase their effectiveness in solving various artificial intelligence tasks. The relevance of the study is due to the growing requirements for model performance due to hardware resource constraints, the need to respond quickly to new challenges and adapt to specific conditions of use in real time. The study analyzes modern approaches to modifying pre-trained models to improve their performance, reduce computational costs, and adapt to new tasks. The study includes an analysis of existing approaches, identification of the main challenges in integrating optimized neural networks into various applications, and development of recommendations for improving their performance and adaptability. The results of the study allow not only to classify existing methods but also to outline promising areas of technology development that will contribute to the creation of more efficient and flexible artificial intelligence systems. Using a variety of methods and approaches, including transfer learning, fine-tuning, and ensemble methods, this paper discusses the optimization of the process of creating new neural networks by using pre-trained models as a basis. Special attention is paid to methods of transfer learning, model compression, quantization, and neural network architecture search (NAS). The central element of the approach is the deployment of neural networks built on the basis of existing models, which can significantly reduce training time and improve the accuracy of new networks. The research methodology includes collecting and preparing the data set, normalizing the data to ensure stability and efficiency of training, and using the Adam optimizer to minimize the loss function quickly and efficiently. The practical value of the results obtained is manifested in the possibility of their application to develop more energy-efficient solutions in such areas as autonomous vehicles, natural language processing systems, medical diagnostics, and other areas where rapid adaptation to changing operating conditions is critical.*

Keywords: neural networks; optimization; adaptation; Dataset; Normalization; Adam Optimizer; Fine-tuning

References

1. Koreniuk T., Honcharenko, T., Sapaiev, V. (2024). Individualization of Learning due to Introduction of Artificial Intelligence into the Education System. *2024 IEEE AITU: Digital Generation, Conference Pro eedings – AITU 2024*, pp. 150–153. URL: DOI: 10.1109/IEEECONF61558.2024.10585595.

2. Matsiievskyi, O., Honcharenko, T., Solovei, O., Liashchenko, T., Achkasov, I., Golenkov, V. (2024). Using Artificial Intelligence to Convert Code to Another Programming Language. *2024 IEEE 4th International Conference on Smart Information Systems and Technologies* (SIST), pp. 379–385. URL: https://ieeexplore.ieee.org/abstract/document/10629305.

3. Gaudenz, Boesch. (2021). Very Deep Convolutional Networks (VGG) Essential Guide. URL: https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/

4. Simonyan, Karen, Zisserman, Andrew. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. Computer Vision and Pattern Recognition. arXiv:1409.1556. URL: https://doi.org/10.48550/arXiv.1409.1556.

5. Geoffrey, E., Hinton, O., Vinyals, J. Dean. (2015). Distilling the Knowledge in a Neural Network. NIPS 2014 Deep Learning Workshop. URL: https://doi.org/10.48550/arXiv.1503.02531.

6. Howard, J., Gugger, S. Fastai: A Layered API for Deep Learning. Information. 2020. Vol. 11, no. 2. P. 108. URL: https://doi.org/10.3390/info11020108.

7. Kirkpatrick, J., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*. Vol. 114, no. 13. P. 3521–3526. URL: https://doi.org/10.1073/pnas.1611835114.

8. Song, Han, Jeff, Pool, John, Tran, William, J. Dally. (2015). Learning both Weights and Connections for Efficient Neural Networks. Published as a conference paper at NIPS 2015. URL: https://doi.org/10.48550/arXiv.1506.02626.

9. Wang Y., et al. (2024). Spectrum-BERT: Pre-training of Deep Bidirectional Transformers for Spectral Classification of Chinese Liquors. IEEE Transactions on Instrumentation and Measurement. 2024. P. 1. URL: https://doi.org/10.1109/tim.2024.3374300.

10. Geoffrey, Hinton, Oriol, Vinyals, Jeff, Dean. (2015). Distilling the Knowledge in a Neural Network. NIPS 2014 Deep Learning Workshop. URL: https://doi.org/10.48550/arXiv.1503.02531.

11. Barret Zoph, Quoc V. Le (2016). Neural Architecture Search with Reinforcement Learning. Machine Learning (cs.LG). URL: https://doi.org/10.48550/arXiv.1611.01578.

12. Cassimon, A., Mercelis, S., Mets, K. (2024). Scalable reinforcement learning-based neural architecture search. Neural Computing and Applications. URL: https://doi.org/10.1007/s00521-024-10445-2.

13. Hanxiao Liu, Karen Simonyan, Yiming Yang. (2018). Differentiable Architecture Search. Published at ICLR 2019. URL: https://doi.org/10.48550/arXiv.1806.09055

14. Jbara, W. A., Soud, J. H. (2024). DeepFake Detection Based VGG-16 Model. 2024 2nd International Conference on Cyber Resilience (ICCR), Dubai, United Arab Emirates, 26–28 February 2024. URL: https://doi.org/10.1109/iccr61006.2024.10533024.

15. Qian Y. et al. (2016). Very Deep Convolutional Neural Networks for Noise Robust Speech Recognition. IEEE/ACM Transactions on Audio, Speech, and Language Processing. 2016. Vol. 24, no. 12. P. 2263–2276. URL: https://doi.org/10.1109/taslp.2016.2602884.

16. Nikbakhtsarvestani F., Ebrahimi M., Rahnamayan S. Multi-objective ADAM Optimizer (MAdam). *2023 IEEE International Conference on Systems, Man, and Cybernetics* (SMC), Honolulu, Oahu, HI, USA, 1–4 October 2023. 2023. URL: https://doi.org/10.1109/smc53992.2023.10394533

17. Pateriya P. N. et al. (2023). Deep Residual Networks for Image Recognition. *International Journal of Innovative Research in Computer and Communication Engineering*. Vol. 11, no. 09. P. 10742–10747. URL: https://doi.org/10.15680/ijircce.2023.1109026

18. Krizhevsky, Alex, Sutskever, Ilya, and E., Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks", *NIPS*, pp. 1106–1114.

19. Xiaoling Xia, Cui Xu and Bing Nan. (2017). Inception-v3 for flower classification. *2017 2nd International Conference on Image, Vision and Computing* (ICIVC), Chengdu, 2017, pp. 783–787. URL: doi: 10.1109/ICIVC.2017.7984661.

Посилання на публікацію

APA Riabchun, Yu., Kurinsky, O., Dolya, E., & Fesan, A. (2025). Optimization and adaptation of neural networks based on existing architectures: methods, challenges and prospects. *Management of Development of Complex Systems*, 61, 210–218, dx.doi.org\10.32347/2412-9933.2025.61.210-218.

ДСТУ Рябчун Ю. В., Курінський О. В., Доля О. В., Фесан А. О. Оптимізація й адаптація нейромереж на основі наявних архітектур: методи, виклики та перспективи. *Управління розвитком складних систем.* Київ, 2025. № 61. С. 210 – 218, dx.doi.org\10.32347/2412-9933.2025.61.210-218.