**Лабораторна робота 10 ІАД**
**Transfer Learning з використанням TensorFlow**

**Мета** : Навчитися застосовувати transfer learning для класифікації зображень, використовуючи попередньо натреновані моделі у TensorFlow. Ознайомитися з основами fine-tuning моделі для покращення точності на нових даних.

**Крок 1: Імпорт бібліотек**

```
import tensorflow as tf
import tensorflow_hub as hub
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
```

**Крок 2: Завантаження та підготовка даних**

```
# Завантаження CIFAR-10
from tensorflow.keras.datasets import cifar10

# Розділення на тренувальні та тестові дані
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Нормалізація даних
x_train = x_train / 255.0
x_test = x_test / 255.0

# Назви класів
class_names = [
    "airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck"
]

# Перевірка розміру даних
print(f"Розмір тренувальних даних: {x_train.shape}, Тестових даних: {x_test.shape}")
```

**Крок 3: Візуалізація даних**

```
# Візуалізація зразків із тренувального набору
plt.figure(figsize=(10, 5))
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.imshow(x_train[i])
    plt.title(class_names[y_train[i][0]])
    plt.axis("off")
plt.show()
```

**Крок 4: Завантаження попередньо натренованої моделі**

```python
# Завантаження MobileNetV2 з TensorFlow Hub
mobilenet_url = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4"
feature_extractor = hub.KerasLayer(mobilenet_url, input_shape=(224, 224, 3), trainable=False)

# Побудова моделі
model = tf.keras.Sequential([
    tf.keras.layers.Resizing(224, 224),  # Зміна розміру зображень до вимог моделі
    feature_extractor,
    tf.keras.layers.Dense(len(class_names), activation='softmax')  # Кількість виходів дорівнює кількості класів
])

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

model.summary()
```

## Крок 5: Тренування моделі

```python
# Тренування моделі
history = model.fit(
    x_train, y_train,
    validation_split=0.2,
    epochs=5,
    batch_size=64
)
```

## Крок 6: Оцінка моделі

```python
# Оцінка моделі на тестових даних
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"Точність на тестових даних: {test_accuracy * 100:.2f}%")
```

## Крок 7: Матриця похибок

```python
# Передбачення на тестових даних
y_pred = np.argmax(model.predict(x_test), axis=1)

# Матриця похибок
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Передбачений клас")
plt.ylabel("Реальний клас")
plt.title("Матриця похибок")
plt.show()

# Звіт класифікації
print(classification_report(y_test, y_pred, target_names=class_names))
```

## Крок 8: Візуалізація результатів передбачення

```python
# Візуалізація результатів передбачення
plt.figure(figsize=(10, 5))
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.imshow(x_test[i])
    plt.title(f"Реальний: {class_names[y_test[i][0]]}\nПередбачений: {class_names[y_pred[i]]}")
    plt.axis("off")
plt.show()
```

## Крок 9: Fine-tuning (Донастроювання моделі)

```python
# Розморозка останніх кількох шарів для fine-tuning
feature_extractor.trainable = True

# Компіляція моделі з меншою швидкістю навчання
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Донастроювання моделі
fine_tune_history = model.fit(
    x_train, y_train,
    validation_split=0.2,
    epochs=5,
    batch_size=64
)

# Оцінка моделі після fine-tuning
fine_tune_loss, fine_tune_accuracy = model.evaluate(x_test, y_test)
print(f"Точність після fine-tuning: {fine_tune_accuracy * 100:.2f}%")
```

## Висновки

1. Точність моделі: Результати демонструють значне покращення після застосування fine-tuning.
2. Матриця похибок: Модель добре класифікує класи, але спостерігаються деякі плутанини між візуально схожими класами.
3. Використання Transfer Learning: Завдяки попередньо натренованій моделі MobileNetV2 вдалося швидко навчити модель на новому датасеті.
4. Оптимізація: Fine-tuning дозволив досягти більшої точності.