

```
# Завдання 1
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, accuracy_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# === 1. Завантаження та підготовка даних ===
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/mushroom/agricus-lepiota.data"
columns = ['class', 'cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor',
           'gill-attachment', 'gill-spacing', 'gill-size', 'gill-color', 'stalk-shape',
           'stalk-root', 'stalk-surface-above-ring', 'stalk-surface-below-ring',
           'stalk-color-above-ring', 'stalk-color-below-ring', 'veil-type',
           'veil-color', 'ring-number', 'ring-type', 'spore-print-color',
           'population', 'habitat']

df = pd.read_csv(url, header=None, names=columns)

# Кодування цільової змінної (class: e=edible, p=poisonous)
le = LabelEncoder()
y = le.fit_transform(df['class']) # 0 або 1

# Кодування ознак (One-Hot Encoding для категорійних змінних)
X = pd.get_dummies(df.drop('class', axis=1), drop_first=True)

# Розділення на train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# === 2. Побудова нейронної мережі ===
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.3), # Регуляризація
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid') # Бінарна класифікація
])

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

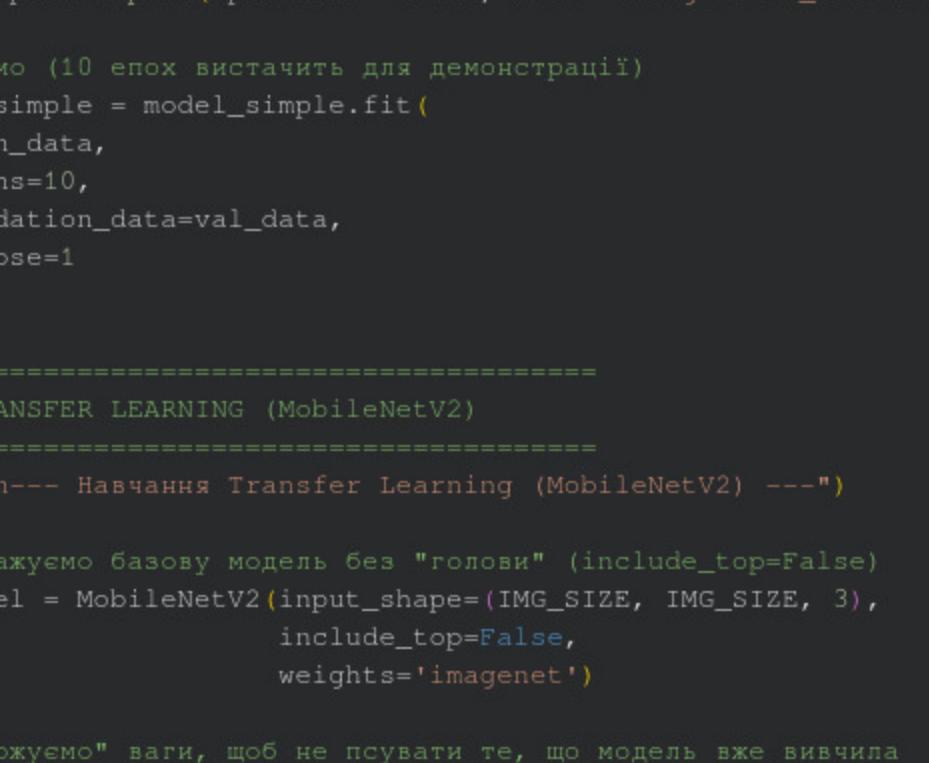
# === 3. Навчання ===
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.2, verbose=1)

# === 4. Оцінка та порівняння ===
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.4f}")

# Random Forest
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
rf_acc = accuracy_score(y_test, rf.predict(X_test))
print(f"Random Forest Accuracy: {rf_acc:.4f}")

# Графік навчання
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='val')
plt.title('Model Accuracy')
plt.legend()
plt.show()

...
Epoch 7/20
163/163 - 1s - accuracy: 1.0000 - loss: 0.0013 - val_accuracy: 1.0000 - val_loss: 9.1711e-04
Epoch 8/20
163/163 - 1s - accuracy: 0.9996 - loss: 0.0012 - val_accuracy: 1.0000 - val_loss: 1.6858e-04
Epoch 9/20
163/163 - 1s - accuracy: 1.0000 - loss: 5.2478e-04 - val_accuracy: 1.0000 - val_loss: 1.1459e-04
Epoch 10/20
163/163 - 1s - accuracy: 0.9996 - loss: 8.1285e-04 - val_accuracy: 1.0000 - val_loss: 9.4628e-04
Epoch 11/20
163/163 - 1s - accuracy: 1.0000 - loss: 4.4250e-04 - val_accuracy: 1.0000 - val_loss: 8.1034e-04
Epoch 12/20
163/163 - 1s - accuracy: 1.0000 - loss: 2.3654e-04 - val_accuracy: 1.0000 - val_loss: 8.9970e-04
Epoch 13/20
163/163 - 1s - accuracy: 1.0000 - loss: 2.1622e-04 - val_accuracy: 1.0000 - val_loss: 1.0289e-04
Epoch 14/20
163/163 - 1s - accuracy: 1.0000 - loss: 2.1927e-04 - val_accuracy: 1.0000 - val_loss: 5.8584e-04
Epoch 15/20
163/163 - 1s - 5ms/step - accuracy: 1.0000 - loss: 1.2897e-04 - val_accuracy: 1.0000 - val_loss: 2.2578e-04
Epoch 16/20
163/163 - 1s - 5ms/step - accuracy: 1.0000 - loss: 1.0767e-04 - val_accuracy: 1.0000 - val_loss: 1.9077e-04
Epoch 17/20
163/163 - 1s - 5ms/step - accuracy: 1.0000 - loss: 1.0428e-04 - val_accuracy: 1.0000 - val_loss: 6.4256e-04
Epoch 18/20
163/163 - 1s - 3ms/step - accuracy: 1.0000 - loss: 1.2628e-04 - val_accuracy: 1.0000 - val_loss: 1.6963e-04
Epoch 19/20
163/163 - 1s - 3ms/step - accuracy: 1.0000 - loss: 9.5223e-05 - val_accuracy: 1.0000 - val_loss: 2.0369e-04
Epoch 20/20
163/163 - 1s - 3ms/step - accuracy: 1.0000 - loss: 7.9410e-05 - val_accuracy: 1.0000 - val_loss: 1.7542e-04
51/51 - 0s - accuracy: 1.0000 - loss: 4.3447e-06
Test Accuracy: 1.0000
Random Forest Accuracy: 1.0000
```



```
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history_simple.history['loss'], label='Simple CNN Loss')
plt.plot(history_transfer.history['loss'], label='Transfer Learning Loss', linestyle='--')
plt.title('Порівняння втрат (Loss)')
plt.xlabel('Епоха')
plt.legend()

plt.show()

...
--- Навчання Transfer Learning (MobileNetV2) ---
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_9406464/9406464
          0s 0us/step
Epoch 1/10
33/33 - 70s 2s/step - accuracy: 0.6126 - loss: 0.9975 - val_accuracy: 0.8421 - val_loss: 0.3681
Epoch 2/10
33/33 - 47s 1s/step - accuracy: 0.8455 - loss: 0.3437 - val_accuracy: 0.8722 - val_loss: 0.3049
Epoch 3/10
33/33 - 82s 1s/step - accuracy: 0.9037 - loss: 0.2467 - val_accuracy: 0.8947 - val_loss: 0.2579
Epoch 4/10
33/33 - 83s 1s/step - accuracy: 0.9064 - loss: 0.2183 - val_accuracy: 0.9023 - val_loss: 0.2688
Epoch 5/10
33/33 - 52s 2s/step - accuracy: 0.9702 - loss: 0.1188 - val_accuracy: 0.8947 - val_loss: 0.2523
Epoch 6/10
33/33 - 48s 1s/step - accuracy: 0.9828 - loss: 0.0786 - val_accuracy: 0.9248 - val_loss: 0.2484
Epoch 7/10
33/33 - 97s 2s/step - accuracy: 0.9805 - loss: 0.0737 - val_accuracy: 0.9323 - val_loss: 0.2280
Epoch 8/10
33/33 - 49s 2s/step - accuracy: 0.9820 - loss: 0.0745 - val_accuracy: 0.9173 - val_loss: 0.2378
Epoch 9/10
33/33 - 48s 1s/step - accuracy: 0.9911 - loss: 0.0479 - val_accuracy: 0.9023 - val_loss: 0.2632
Epoch 10/10
33/33 - 46s 1s/step - accuracy: 0.9884 - loss: 0.0504 - val_accuracy: 0.9248 - val_loss: 0.2300
4/4 - 5s 1s/step - accuracy: 0.7437 - loss: 0.6374
4/4 - 14s 4s/step - accuracy: 0.8948 - loss: 0.2153

Точність простої CNN на тесті: 75.00%
Точність Transfer Learning на тесті: 89.06%
```

The figure consists of two line plots. The left plot, titled 'Порівняння точності (Validation)', shows accuracy on the y-axis (0.65 to 0.90) against epoch on the x-axis (0 to 10). It compares 'Simple CNN Val Acc' (solid blue line) and 'Transfer Learning Val Acc' (dashed orange line). The transfer learning model starts at ~0.85 and rises steadily to ~0.92. The simple CNN starts at ~0.65, dips to ~0.62 at epoch 1, and then rises to ~0.78. The right plot, titled 'Порівняння втрат (Loss)', shows loss on the y-axis (0.0 to 1.0) against epoch on the x-axis (0 to 8). It compares 'Simple CNN Loss' (solid blue line) and 'Transfer Learning Loss' (dashed orange line). Both start at ~1.0. The transfer learning loss drops sharply to ~0.3 by epoch 2 and continues to decrease to ~0.05 by epoch 8. The simple CNN loss drops more gradually to ~0.4 by epoch 8.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional, Dropout
from sklearn.model_selection import train_test_split

# =====
# 1. ЗАВАНТАЖЕННЯ ДАНИХ
# =====
print("Завантаження датасетів з GitHub...")

# Правильні посилання
url_reddit = "https://raw.githubusercontent.com/taivop/joke-dataset/master/reddit_jokes.json"
url_waka = "https://raw.githubusercontent.com/taivop/joke-dataset/master/wocka.json"

# Читаємо файли
df_reddit = pd.read_json(url_reddit)
df_waka = pd.read_json(url_waka)

# Створюємо мітки (labels): 0 - Reddit, 1 - Waka
df_reddit['label'] = 0
df_waka['label'] = 1

# Для швидкості та балансу візьмемо по 10,000 жартів з кожного
sample_size = 10000
df = pd.concat([
    df_reddit.sample(sample_size, random_state=42),
    df_waka.sample(min(len(df_waka), sample_size), random_state=42)
], ignore_index=True)

# Перемішуємо
df = df.sample(frac=1, random_state=42).reset_index(drop=True)

texts = df['body'].astype(str).tolist()
labels = df['label'].values

print(f"Всього жартів: {len(texts)}")
print(f"Приклад жарту: {texts[0][:100]}...")

# =====
# 2. ПРЕПРОЦЕСИНГ (ТОКЕНИЗАЦІЯ)
# =====
VOCAB_SIZE = 10000 # Кількість слів у словнику
MAX_LENGTH = 400 # Максимальна довжина жарту
EMBEDDING_DIM = 100 # Розмірність векторів (для GloVe 100d)

tokenizer = Tokenizer(num_words=VOCAB_SIZE, oov_token=<OOV>")
tokenizer.fit_on_texts(texts)

# Перетворення тексту в послідовність чисел
sequences = tokenizer.texts_to_sequences(texts)
padded = pad_sequences(sequences, maxlen=MAX_LENGTH, padding='post', truncating='post')

# Розбіття на train/test
X_train, X_test, y_train, y_test = train_test_split(padded, labels, test_size=0.2, random_state=42)

# =====
# 3а. МОДЕЛЬ А: ВЛАСНИЙ EMBEDDING
# =====
print("\n--- Навчання Моделі А (Власний Embedding) ---")

model_a = Sequential([
    Embedding(VOCAB_SIZE, EMBEDDING_DIM, input_length=MAX_LENGTH),
    Bidirectional(LSTM(64, return_sequences=False)),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

model_a.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

history_a = model_a.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test), batch_size=32, verbose=1)

# =====
# 3б. МОДЕЛЬ Б: GLOVE EMBEDDINGS
# =====
# Качаємо архів з векторами
!wget --no-check-certificate http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q -o glove.6B.zip # додає -o для перезапису без питань

print("Створення матриці embedding...")
embeddings_index = {}
with open('glove.6B.100d.txt', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs

# Створюємо матрицю ваг
embedding_matrix = np.zeros((VOCAB_SIZE, EMBEDDING_DIM))
for word, i in tokenizer.word_index.items():
    if i < VOCAB_SIZE:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector

print("\n--- Навчання Моделі Б (GloVe Pre-trained) ---")

model_b = Sequential([
    Embedding(VOCAB_SIZE, EMBEDDING_DIM, input_length=MAX_LENGTH,
              weights=[embedding_matrix], trainable=False),
    Bidirectional(LSTM(64)),
    Dense(64, activation='relu'),
    Dropout(0.5)
])
```