Assignment 3

November 26, 2024

1 Programming Assignment 3

1.1 Student ID: 916461653

[2]:
```
import json
import os

import cv2
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import torch
from datasets import Dataset
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Conv2D, Dense, Flatten, MaxPooling2D
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from torch.nn.functional import sigmoid
from transformers import (
    AutoModelForSequenceClassification,
    AutoTokenizer,
    Trainer,
    TrainingArguments,
)
```

```
/usr/local/lib/python3.10/dist-packages/tensorflow/lite/python/util.py:55:
DeprecationWarning: jax.xla_computation is deprecated. Please use the AOT APIs;
see https://jax.readthedocs.io/en/latest/aot.html. For example, replace
xla_computation(f)(*xs) with jit(f).lower(*xs).compiler_ir('hlo'). See
CHANGELOG.md for 0.4.30 for more examples.
  from jax import xla_computation as _xla_computation
/usr/local/lib/python3.10/dist-packages/tensorflow/lite/python/util.py:55:
DeprecationWarning: jax.xla_computation is deprecated. Please use the AOT APIs;
see https://jax.readthedocs.io/en/latest/aot.html. For example, replace
xla_computation(f)(*xs) with jit(f).lower(*xs).compiler_ir('hlo'). See
```

```
CHANGELOG.md for 0.4.30 for more examples.
  from jax import xla_computation as _xla_computation
```

2 Question 1: Association Rule Generation from Transaction Data

In this section, we'll analyze transaction data using association rule mining. The analysis will:

1. Load and process the grocery transaction dataset
2. Calculate basic statistics about the dataset
3. Generate association rules using specified support and confidence thresholds
4. Create a heatmap visualization of rule counts for different parameter combinations

[ ]:
```python
print("\n" + "=" * 80)
print("PART A & B: Loading Transaction Dataset")
print("=" * 80)
print("Loading Grocery_Items_24.csv...")


transaction_data = pd.read_csv("Grocery_Items_24.csv", header=0)
grocery_transactions = transaction_data.values.tolist()
grocery_transactions = [
    [item for item in transaction if isinstance(item, str)]
    for transaction in grocery_transactions
]

print("\n" + "=" * 80)
print("PART C: Dataset Statistics")
print("=" * 80)


all_grocery_items = [
    item for transaction in grocery_transactions for item in transaction
]
unique_grocery_items = len(set(all_grocery_items))
total_transactions = len(grocery_transactions)
item_frequency = pd.Series(all_grocery_items).value_counts()
most_common_item = item_frequency.index[0]
most_common_item_count = item_frequency.iloc[0]

print(f"Number of unique items: {unique_grocery_items}")
print(f"Number of records: {total_transactions}")
print(
    f"Most popular item: {most_common_item} (appears in␣
  ↪{most_common_item_count} transactions)"
)
```

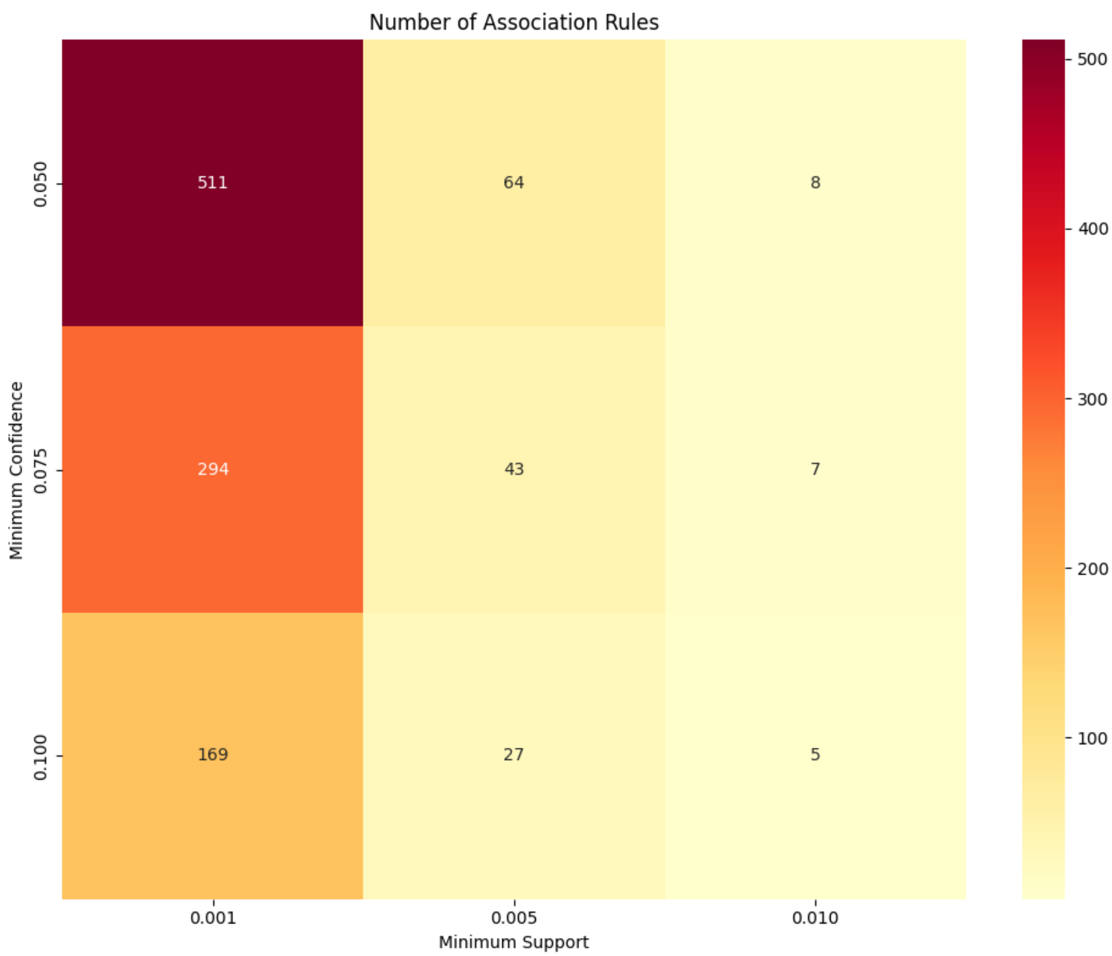
```
print("\n" + "=" * 80)
print("PART D: Association Rules Generation")
print("=" * 80)
print("Generating rules with minimum support = 0.01 and minimum confidence = 0.
↪08...")


transaction_encoder = TransactionEncoder()
encoded_array = transaction_encoder.fit_transform(grocery_transactions)
encoded_dataframe = pd.DataFrame(encoded_array, columns=transaction_encoder.
↪columns_)


initial_support = 0.01
initial_confidence = 0.08
frequent_itemsets = apriori(
    encoded_dataframe, min_support=initial_support, use_colnames=True
)
initial_rules = association_rules(
    frequent_itemsets,
    frequent_itemsets,
    metric="confidence",
    min_threshold=initial_confidence,
)

print("\nAssociation Rules:")
print(initial_rules)

print("\n" + "=" * 80)
print("PART E: Heatmap Generation")
print("=" * 80)
print("Generating heatmap for different support and confidence thresholds...")


support_thresholds = [0.001, 0.005, 0.01]
confidence_thresholds = [0.05, 0.075, 0.1]
rule_count_matrix = np.zeros((len(confidence_thresholds),␣
↪len(support_thresholds)))

for confidence_idx, confidence_value in enumerate(confidence_thresholds):
    for support_idx, support_value in enumerate(support_thresholds):
        print(
            f"\nCalculating rules for support={support_value},␣
↪confidence={confidence_value}"
        )
        current_frequent_items = apriori(
            encoded_dataframe, min_support=support_value, use_colnames=True
```

```python
        )
        current_rules = association_rules(
            current_frequent_items,
            current_frequent_items,
            metric="confidence",
            min_threshold=confidence_value,
        )
        rule_count_matrix[confidence_idx, support_idx] = len(current_rules)
        print(f"Number of rules found: {len(current_rules)}")


plt.figure(figsize=(10, 8))
sns.heatmap(
    rule_count_matrix,
    xticklabels=[f"{x:.3f}" for x in support_thresholds],
    yticklabels=[f"{x:.3f}" for x in confidence_thresholds],
    annot=True,
    fmt="g",
    cmap="YlOrRd",
)
plt.xlabel("Minimum Support")
plt.ylabel("Minimum Confidence")
plt.title("Number of Association Rules")
plt.tight_layout()

print("\nDisplaying heatmap...")
plt.show()
```

```
================================================================================
PART A & B: Loading Transaction Dataset
================================================================================
Loading Grocery_Items_24.csv…

================================================================================
PART C: Dataset Statistics
================================================================================
Number of unique items: 166
Number of records: 8000
Most popular item: whole milk (appears in 1321 transactions)

================================================================================
PART D: Association Rules Generation
================================================================================
Generating rules with minimum support = 0.01 and minimum confidence = 0.08…

Association Rules:
          antecedents         consequents  antecedent support  \
```

```
0        (rolls/buns)  (other vegetables)            0.111875
1  (other vegetables)        (rolls/buns)            0.122750
2        (whole milk)  (other vegetables)            0.156000
3  (other vegetables)        (whole milk)            0.122750
4        (whole milk)        (rolls/buns)            0.156000
5        (rolls/buns)        (whole milk)            0.111875
6              (soda)        (whole milk)            0.093625

   consequent support   support  confidence      lift  representativity  \
0            0.122750  0.011250    0.100559  0.819215               1.0
1            0.111875  0.011250    0.091650  0.819215               1.0
2            0.122750  0.016125    0.103365  0.842081               1.0
3            0.156000  0.016125    0.131365  0.842081               1.0
4            0.111875  0.013500    0.086538  0.773528               1.0
5            0.156000  0.013500    0.120670  0.773528               1.0
6            0.156000  0.011375    0.121495  0.778816               1.0

   leverage  conviction  zhangs_metric   jaccard  certainty  kulczynski
0 -0.002483    0.975328      -0.199025  0.050364  -0.025296    0.096104
1 -0.002483    0.977734      -0.200997  0.050364  -0.022773    0.096104
2 -0.003024    0.978381      -0.181802  0.061399  -0.022097    0.117365
3 -0.003024    0.971639      -0.176125  0.061399  -0.029189    0.117365
4 -0.003952    0.972263      -0.257551  0.053071  -0.028528    0.103604
5 -0.003952    0.959822      -0.247927  0.053071  -0.041860    0.103604
6 -0.003231    0.960723      -0.238580  0.047744  -0.040882    0.097206

================================================================================
PART E: Heatmap Generation
================================================================================
Generating heatmap for different support and confidence thresholds…

Calculating rules for support=0.001, confidence=0.05
Number of rules found: 511

Calculating rules for support=0.005, confidence=0.05
Number of rules found: 64

Calculating rules for support=0.01, confidence=0.05
Number of rules found: 8

Calculating rules for support=0.001, confidence=0.075
Number of rules found: 294

Calculating rules for support=0.005, confidence=0.075
Number of rules found: 43

Calculating rules for support=0.01, confidence=0.075
Number of rules found: 7
```

```
Calculating rules for support=0.001, confidence=0.1
Number of rules found: 169

Calculating rules for support=0.005, confidence=0.1
Number of rules found: 27

Calculating rules for support=0.01, confidence=0.1
Number of rules found: 5

Displaying heatmap…
```



2.1 Dataset Characteristics (C)

- 166 unique items across 8,000 transactions
- Most frequent item: whole milk (1,321 occurrences, 16.5% of transactions)

2.2 Association Rules Analysis (D)

With minimum support=0.01 and confidence=0.08, key associations found:

1. {rolls/buns} → {other vegetables} (conf: 10.1%, lift: 0.82)
2. {whole milk} → {other vegetables} (conf: 10.3%, lift: 0.84)
3. {soda} → {whole milk} (conf: 12.1%, lift: 0.78)

Lift values <1 indicate these associations occur less frequently than expected under independence.

2.3 Parameter Sensitivity Analysis (E)

The heatmap reveals:

1. Rule count decreases with increasing thresholds:
   - Max rules (511): support=0.001, confidence=0.05
   - Min rules (5): support=0.01, confidence=0.1
2. Most significant drop occurs when increasing support threshold (0.001→0.005)
3. Increasing confidence has less impact on rule reduction compared to support

Optimal thresholds appear to be support=0.005, confidence=0.075 (43 rules), balancing rule quantity with significance.

3 Question 2: Image Classification using Convolutional Neural Networks

This section implements a 4-class CNN classifier with the specified architecture:

- First convolutional layer (8 3×3 filters)
- Max pooling (2×2)
- Second convolutional layer (4 3×3 filters)
- Max pooling (2×2)
- Flatten layer
- Hidden layer (8 nodes)
- Output layer (4 nodes with softmax activation)

Since my Rowan ID ends in 3, I will experiment with different numbers of nodes (4 and 16) in the hidden layer.

[ ]:
```
cropped_images_dir = "./Processed"

image_data = []
image_labels = []

dog_breed_classes = [
    "n02089078-black-and-tan_coonhound",
    "n02091831-Saluki",
    "n02092002-Scottish_deerhound",
```

```
    "n02095314-wire-haired_fox_terrier",
]


for breed_index, breed_class in enumerate(dog_breed_classes):
    breed_directory = os.path.join(cropped_images_dir, breed_class)
    if not os.path.isdir(breed_directory):
        print(f"Directory not found: {breed_directory}")
        continue
    for image_file in os.listdir(breed_directory):
        if image_file.endswith(".jpg"):
            image_path = os.path.join(breed_directory, image_file)
            image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
            resized_image = cv2.resize(image, (6, 6))
            image_data.append(resized_image)
            image_labels.append(breed_index)

image_data_array = np.array(image_data)
image_labels_array = np.array(image_labels)

reshaped_image_data = image_data_array.reshape(-1, 6, 6, 1)
one_hot_labels = to_categorical(image_labels_array, num_classes=4)


X_train_initial, X_test, y_train_initial, y_test = train_test_split(
    reshaped_image_data, one_hot_labels, test_size=0.2, random_state=42
)
X_train, X_validation, y_train, y_validation = train_test_split(
    X_train_initial, y_train_initial, test_size=0.2, random_state=42
)


base_cnn_model = Sequential(
    [
        Conv2D(8, (3, 3), activation="relu", padding="same", input_shape=(6, 6,␣
↪1)),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(4, (3, 3), activation="relu", padding="same"),
        MaxPooling2D(pool_size=(2, 2)),
        Flatten(),
        Dense(8, activation="relu"),
        Dense(4, activation="softmax"),
    ]
)

base_cnn_model.compile(
    optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"]
```

```
)
base_model_history = base_cnn_model.fit(
    X_train,
    y_train,
    epochs=100,
    batch_size=32,
    validation_data=(X_validation, y_validation),
)


model_5x5 = Sequential(
    [
        Conv2D(8, (3, 3), activation="relu", padding="same", input_shape=(6, 6,␣
 ↪1)),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(4, (5, 5), activation="relu", padding="same"),
        MaxPooling2D(pool_size=(2, 2)),
        Flatten(),
        Dense(8, activation="relu"),
        Dense(4, activation="softmax"),
    ]
)

model_5x5.compile(
    optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"]
)
model_5x5_history = model_5x5.fit(
    X_train,
    y_train,
    epochs=100,
    batch_size=32,
    validation_data=(X_validation, y_validation),
)


model_7x7 = Sequential(
    [
        Conv2D(8, (3, 3), activation="relu", padding="same", input_shape=(6, 6,␣
 ↪1)),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(4, (7, 7), activation="relu", padding="same"),
        MaxPooling2D(pool_size=(2, 2)),
        Flatten(),
        Dense(8, activation="relu"),
        Dense(4, activation="softmax"),
    ]
)
```

```
model_7x7.compile(
    optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"]
)
model_7x7_history = model_7x7.fit(
    X_train,
    y_train,
    epochs=100,
    batch_size=32,
    validation_data=(X_validation, y_validation),
)


plt.figure(figsize=(12, 4))

plt.subplot(1, 3, 1)
plt.plot(base_model_history.history["accuracy"], label="Training Accuracy")
plt.plot(base_model_history.history["val_accuracy"], label="Validation␣
↪Accuracy")
plt.title("Base Model (3x3 filter)")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()

plt.subplot(1, 3, 2)
plt.plot(model_5x5_history.history["accuracy"], label="Training Accuracy")
plt.plot(model_5x5_history.history["val_accuracy"], label="Validation Accuracy")
plt.title("Model with 5x5 filter")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()

plt.subplot(1, 3, 3)
plt.plot(model_7x7_history.history["accuracy"], label="Training Accuracy")
plt.plot(model_7x7_history.history["val_accuracy"], label="Validation Accuracy")
plt.title("Model with 7x7 filter")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()

plt.tight_layout()
plt.show()

print("\nFinal Accuracies:")
print(
    f"Base Model (3x3) - Training: {base_model_history.history['accuracy'][-1]:.
↪4f}, Validation: {base_model_history.history['val_accuracy'][-1]:.4f}"
```

```
)
print(
    f"5x5 Filter Model - Training: {model_5x5_history.history['accuracy'][-1]:.
↪4f}, Validation: {model_5x5_history.history['val_accuracy'][-1]:.4f}"
)
print(
    f"7x7 Filter Model - Training: {model_7x7_history.history['accuracy'][-1]:.
↪4f}, Validation: {model_7x7_history.history['val_accuracy'][-1]:.4f}"
)
```

```
Epoch 1/100
```

```
/Users/rudra/Documents/Code/Freelancing/Python/Data
Mining/.venv/lib/python3.12/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
16/16           0s 5ms/step -
accuracy: 0.2853 - loss: 15.3956 - val_accuracy: 0.2520 - val_loss: 5.1526
Epoch 2/100
16/16           0s 1ms/step -
accuracy: 0.2256 - loss: 4.2694 - val_accuracy: 0.1575 - val_loss: 1.9203
Epoch 3/100
16/16           0s 1ms/step -
accuracy: 0.2404 - loss: 1.9045 - val_accuracy: 0.2520 - val_loss: 1.5187
Epoch 4/100
16/16           0s 1ms/step -
accuracy: 0.2640 - loss: 1.5503 - val_accuracy: 0.2677 - val_loss: 1.4922
Epoch 5/100
16/16           0s 1ms/step -
accuracy: 0.2533 - loss: 1.4825 - val_accuracy: 0.2677 - val_loss: 1.4777
Epoch 6/100
16/16           0s 1ms/step -
accuracy: 0.2596 - loss: 1.4559 - val_accuracy: 0.2598 - val_loss: 1.4682
Epoch 7/100
16/16           0s 1ms/step -
accuracy: 0.2629 - loss: 1.4389 - val_accuracy: 0.2598 - val_loss: 1.4609
Epoch 8/100
16/16           0s 1ms/step -
accuracy: 0.2610 - loss: 1.4261 - val_accuracy: 0.2598 - val_loss: 1.4546
Epoch 9/100
16/16           0s 1ms/step -
accuracy: 0.2628 - loss: 1.4212 - val_accuracy: 0.2520 - val_loss: 1.4485
Epoch 10/100
16/16           0s 1ms/step -
```

```
accuracy: 0.2628 - loss: 1.4106 - val_accuracy: 0.2520 - val_loss: 1.4421
Epoch 11/100
16/16            0s 1ms/step -
accuracy: 0.2649 - loss: 1.4006 - val_accuracy: 0.2520 - val_loss: 1.4371
Epoch 12/100
16/16            0s 4ms/step -
accuracy: 0.2690 - loss: 1.3938 - val_accuracy: 0.2520 - val_loss: 1.4333
Epoch 13/100
16/16            0s 1ms/step -
accuracy: 0.2638 - loss: 1.3904 - val_accuracy: 0.2520 - val_loss: 1.4304
Epoch 14/100
16/16            0s 1ms/step -
accuracy: 0.2638 - loss: 1.3895 - val_accuracy: 0.2520 - val_loss: 1.4279
Epoch 15/100
16/16            0s 1ms/step -
accuracy: 0.2640 - loss: 1.3885 - val_accuracy: 0.2598 - val_loss: 1.4254
Epoch 16/100
16/16            0s 1ms/step -
accuracy: 0.2640 - loss: 1.3875 - val_accuracy: 0.2598 - val_loss: 1.4235
Epoch 17/100
16/16            0s 1ms/step -
accuracy: 0.2640 - loss: 1.3865 - val_accuracy: 0.2598 - val_loss: 1.4217
Epoch 18/100
16/16            0s 1ms/step -
accuracy: 0.2640 - loss: 1.3855 - val_accuracy: 0.2598 - val_loss: 1.4200
Epoch 19/100
16/16            0s 1ms/step -
accuracy: 0.2640 - loss: 1.3846 - val_accuracy: 0.2598 - val_loss: 1.4184
Epoch 20/100
16/16            0s 1ms/step -
accuracy: 0.2640 - loss: 1.3838 - val_accuracy: 0.2598 - val_loss: 1.4170
Epoch 21/100
16/16            0s 1ms/step -
accuracy: 0.2640 - loss: 1.3830 - val_accuracy: 0.2598 - val_loss: 1.4157
Epoch 22/100
16/16            0s 1ms/step -
accuracy: 0.2640 - loss: 1.3822 - val_accuracy: 0.2598 - val_loss: 1.4144
Epoch 23/100
16/16            0s 1ms/step -
accuracy: 0.2640 - loss: 1.3815 - val_accuracy: 0.2598 - val_loss: 1.4132
Epoch 24/100
16/16            0s 4ms/step -
accuracy: 0.2640 - loss: 1.3809 - val_accuracy: 0.2598 - val_loss: 1.4120
Epoch 25/100
16/16            0s 1ms/step -
accuracy: 0.2658 - loss: 1.3803 - val_accuracy: 0.2598 - val_loss: 1.4109
Epoch 26/100
16/16            0s 1ms/step -
```

```
accuracy: 0.2658 - loss: 1.3800 - val_accuracy: 0.2598 - val_loss: 1.4101
Epoch 27/100
16/16            0s 1ms/step -
accuracy: 0.2658 - loss: 1.3799 - val_accuracy: 0.2598 - val_loss: 1.4094
Epoch 28/100
16/16            0s 1ms/step -
accuracy: 0.2658 - loss: 1.3797 - val_accuracy: 0.2598 - val_loss: 1.4088
Epoch 29/100
16/16            0s 1ms/step -
accuracy: 0.2658 - loss: 1.3795 - val_accuracy: 0.2598 - val_loss: 1.4082
Epoch 30/100
16/16            0s 1ms/step -
accuracy: 0.2658 - loss: 1.3794 - val_accuracy: 0.2598 - val_loss: 1.4074
Epoch 31/100
16/16            0s 1ms/step -
accuracy: 0.2658 - loss: 1.3793 - val_accuracy: 0.2598 - val_loss: 1.4068
Epoch 32/100
16/16            0s 1ms/step -
accuracy: 0.2658 - loss: 1.3792 - val_accuracy: 0.2598 - val_loss: 1.4064
Epoch 33/100
16/16            0s 1ms/step -
accuracy: 0.2658 - loss: 1.3791 - val_accuracy: 0.2598 - val_loss: 1.4059
Epoch 34/100
16/16            0s 1ms/step -
accuracy: 0.2658 - loss: 1.3790 - val_accuracy: 0.2598 - val_loss: 1.4055
Epoch 35/100
16/16            0s 1ms/step -
accuracy: 0.2651 - loss: 1.3790 - val_accuracy: 0.2598 - val_loss: 1.4051
Epoch 36/100
16/16            0s 4ms/step -
accuracy: 0.2658 - loss: 1.3789 - val_accuracy: 0.2598 - val_loss: 1.4049
Epoch 37/100
16/16            0s 1ms/step -
accuracy: 0.2658 - loss: 1.3788 - val_accuracy: 0.2598 - val_loss: 1.4045
Epoch 38/100
16/16            0s 1ms/step -
accuracy: 0.2670 - loss: 1.3788 - val_accuracy: 0.2677 - val_loss: 1.4042
Epoch 39/100
16/16            0s 1ms/step -
accuracy: 0.2670 - loss: 1.3788 - val_accuracy: 0.2677 - val_loss: 1.4036
Epoch 40/100
16/16            0s 1ms/step -
accuracy: 0.2670 - loss: 1.3788 - val_accuracy: 0.2677 - val_loss: 1.4032
Epoch 41/100
16/16            0s 1ms/step -
accuracy: 0.2658 - loss: 1.3788 - val_accuracy: 0.2677 - val_loss: 1.4029
Epoch 42/100
16/16            0s 1ms/step -
```

```
accuracy: 0.2670 - loss: 1.3788 - val_accuracy: 0.2677 - val_loss: 1.4026
Epoch 43/100
16/16            0s 1ms/step -
accuracy: 0.2658 - loss: 1.3788 - val_accuracy: 0.2677 - val_loss: 1.4024
Epoch 44/100
16/16            0s 1ms/step -
accuracy: 0.2651 - loss: 1.3788 - val_accuracy: 0.2677 - val_loss: 1.4020
Epoch 45/100
16/16            0s 1ms/step -
accuracy: 0.2658 - loss: 1.3788 - val_accuracy: 0.2677 - val_loss: 1.4017
Epoch 46/100
16/16            0s 1ms/step -
accuracy: 0.2670 - loss: 1.3788 - val_accuracy: 0.2677 - val_loss: 1.4016
Epoch 47/100
16/16            0s 4ms/step -
accuracy: 0.2750 - loss: 1.3788 - val_accuracy: 0.3622 - val_loss: 1.4014
Epoch 48/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3788 - val_accuracy: 0.3622 - val_loss: 1.4013
Epoch 49/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3789 - val_accuracy: 0.3622 - val_loss: 1.4010
Epoch 50/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3789 - val_accuracy: 0.3622 - val_loss: 1.4008
Epoch 51/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3789 - val_accuracy: 0.3622 - val_loss: 1.4009
Epoch 52/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3789 - val_accuracy: 0.3622 - val_loss: 1.4008
Epoch 53/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3789 - val_accuracy: 0.3622 - val_loss: 1.4008
Epoch 54/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3789 - val_accuracy: 0.3622 - val_loss: 1.4008
Epoch 55/100
16/16            0s 3ms/step -
accuracy: 0.2950 - loss: 1.3791 - val_accuracy: 0.3622 - val_loss: 1.4002
Epoch 56/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3790 - val_accuracy: 0.3622 - val_loss: 1.4001
Epoch 57/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3790 - val_accuracy: 0.3622 - val_loss: 1.4002
Epoch 58/100
16/16            0s 1ms/step -
```

```
accuracy: 0.2950 - loss: 1.3790 - val_accuracy: 0.3622 - val_loss: 1.4008
Epoch 59/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3791 - val_accuracy: 0.3622 - val_loss: 1.4002
Epoch 60/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3791 - val_accuracy: 0.3622 - val_loss: 1.3998
Epoch 61/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3791 - val_accuracy: 0.3622 - val_loss: 1.3999
Epoch 62/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3792 - val_accuracy: 0.3622 - val_loss: 1.4000
Epoch 63/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3792 - val_accuracy: 0.3622 - val_loss: 1.4000
Epoch 64/100
16/16            0s 2ms/step -
accuracy: 0.2950 - loss: 1.3792 - val_accuracy: 0.3622 - val_loss: 1.4005
Epoch 65/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3792 - val_accuracy: 0.3622 - val_loss: 1.4005
Epoch 66/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3793 - val_accuracy: 0.3622 - val_loss: 1.3998
Epoch 67/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3793 - val_accuracy: 0.3622 - val_loss: 1.4001
Epoch 68/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3793 - val_accuracy: 0.3622 - val_loss: 1.3996
Epoch 69/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3793 - val_accuracy: 0.3622 - val_loss: 1.3996
Epoch 70/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3793 - val_accuracy: 0.3622 - val_loss: 1.4000
Epoch 71/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3793 - val_accuracy: 0.3622 - val_loss: 1.4011
Epoch 72/100
16/16            0s 2ms/step -
accuracy: 0.2950 - loss: 1.3795 - val_accuracy: 0.3622 - val_loss: 1.3997
Epoch 73/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3794 - val_accuracy: 0.3622 - val_loss: 1.3997
Epoch 74/100
16/16            0s 2ms/step -
```

```
accuracy: 0.2950 - loss: 1.3793 - val_accuracy: 0.3622 - val_loss: 1.4005
Epoch 75/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3793 - val_accuracy: 0.3622 - val_loss: 1.4008
Epoch 76/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3793 - val_accuracy: 0.3622 - val_loss: 1.4009
Epoch 77/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3795 - val_accuracy: 0.3622 - val_loss: 1.4011
Epoch 78/100
16/16            0s 2ms/step -
accuracy: 0.2956 - loss: 1.3791 - val_accuracy: 0.3622 - val_loss: 1.4011
Epoch 79/100
16/16            0s 2ms/step -
accuracy: 0.2950 - loss: 1.3791 - val_accuracy: 0.3622 - val_loss: 1.4022
Epoch 80/100
16/16            0s 2ms/step -
accuracy: 0.2950 - loss: 1.3792 - val_accuracy: 0.3622 - val_loss: 1.4010
Epoch 81/100
16/16            0s 2ms/step -
accuracy: 0.2950 - loss: 1.3791 - val_accuracy: 0.3622 - val_loss: 1.4018
Epoch 82/100
16/16            0s 2ms/step -
accuracy: 0.2950 - loss: 1.3789 - val_accuracy: 0.3622 - val_loss: 1.4027
Epoch 83/100
16/16            0s 4ms/step -
accuracy: 0.2950 - loss: 1.3790 - val_accuracy: 0.3622 - val_loss: 1.4017
Epoch 84/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3789 - val_accuracy: 0.3622 - val_loss: 1.4015
Epoch 85/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3788 - val_accuracy: 0.3622 - val_loss: 1.4025
Epoch 86/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3787 - val_accuracy: 0.3622 - val_loss: 1.4032
Epoch 87/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3788 - val_accuracy: 0.3622 - val_loss: 1.4028
Epoch 88/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3786 - val_accuracy: 0.3622 - val_loss: 1.4041
Epoch 89/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3786 - val_accuracy: 0.3622 - val_loss: 1.4039
Epoch 90/100
16/16            0s 2ms/step -
```

```
accuracy: 0.2950 - loss: 1.3786 - val_accuracy: 0.3622 - val_loss: 1.4031
Epoch 91/100
16/16            0s 1ms/step -
accuracy: 0.2952 - loss: 1.3785 - val_accuracy: 0.3622 - val_loss: 1.4040
Epoch 92/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3784 - val_accuracy: 0.3622 - val_loss: 1.4043
Epoch 93/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3779 - val_accuracy: 0.3622 - val_loss: 1.4035
Epoch 94/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3777 - val_accuracy: 0.3622 - val_loss: 1.4041
Epoch 95/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3771 - val_accuracy: 0.3622 - val_loss: 1.4050
Epoch 96/100
16/16            0s 4ms/step -
accuracy: 0.2950 - loss: 1.3767 - val_accuracy: 0.3622 - val_loss: 1.4067
Epoch 97/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3764 - val_accuracy: 0.3622 - val_loss: 1.4050
Epoch 98/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3766 - val_accuracy: 0.3622 - val_loss: 1.4063
Epoch 99/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3760 - val_accuracy: 0.3622 - val_loss: 1.4075
Epoch 100/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3761 - val_accuracy: 0.3622 - val_loss: 1.4073
Epoch 1/100
16/16            0s 5ms/step -
accuracy: 0.2445 - loss: 11.0896 - val_accuracy: 0.3701 - val_loss: 3.2967
Epoch 2/100
16/16            0s 2ms/step -
accuracy: 0.3389 - loss: 2.2577 - val_accuracy: 0.2756 - val_loss: 1.3954
Epoch 3/100
16/16            0s 3ms/step -
accuracy: 0.2636 - loss: 1.4034 - val_accuracy: 0.2835 - val_loss: 1.3868
Epoch 4/100
16/16            0s 1ms/step -
accuracy: 0.2611 - loss: 1.3881 - val_accuracy: 0.2756 - val_loss: 1.3865
Epoch 5/100
16/16            0s 1ms/step -
accuracy: 0.2656 - loss: 1.3869 - val_accuracy: 0.2756 - val_loss: 1.3846
Epoch 6/100
16/16            0s 1ms/step -
```

```
accuracy: 0.2656 - loss: 1.3859 - val_accuracy: 0.2756 - val_loss: 1.3829
Epoch 7/100
16/16            0s 1ms/step -
accuracy: 0.2656 - loss: 1.3851 - val_accuracy: 0.2756 - val_loss: 1.3815
Epoch 8/100
16/16            0s 1ms/step -
accuracy: 0.2656 - loss: 1.3844 - val_accuracy: 0.2756 - val_loss: 1.3800
Epoch 9/100
16/16            0s 1ms/step -
accuracy: 0.2656 - loss: 1.3839 - val_accuracy: 0.2756 - val_loss: 1.3789
Epoch 10/100
16/16            0s 2ms/step -
accuracy: 0.2656 - loss: 1.3834 - val_accuracy: 0.2756 - val_loss: 1.3782
Epoch 11/100
16/16            0s 1ms/step -
accuracy: 0.2656 - loss: 1.3830 - val_accuracy: 0.2756 - val_loss: 1.3772
Epoch 12/100
16/16            0s 1ms/step -
accuracy: 0.2656 - loss: 1.3826 - val_accuracy: 0.2756 - val_loss: 1.3762
Epoch 13/100
16/16            0s 1ms/step -
accuracy: 0.2656 - loss: 1.3822 - val_accuracy: 0.2756 - val_loss: 1.3756
Epoch 14/100
16/16            0s 2ms/step -
accuracy: 0.2656 - loss: 1.3819 - val_accuracy: 0.2756 - val_loss: 1.3751
Epoch 15/100
16/16            0s 1ms/step -
accuracy: 0.2656 - loss: 1.3816 - val_accuracy: 0.2756 - val_loss: 1.3745
Epoch 16/100
16/16            0s 4ms/step -
accuracy: 0.2656 - loss: 1.3813 - val_accuracy: 0.2756 - val_loss: 1.3738
Epoch 17/100
16/16            0s 1ms/step -
accuracy: 0.2656 - loss: 1.3811 - val_accuracy: 0.2756 - val_loss: 1.3730
Epoch 18/100
16/16            0s 1ms/step -
accuracy: 0.2656 - loss: 1.3809 - val_accuracy: 0.2756 - val_loss: 1.3722
Epoch 19/100
16/16            0s 1ms/step -
accuracy: 0.2656 - loss: 1.3806 - val_accuracy: 0.2756 - val_loss: 1.3713
Epoch 20/100
16/16            0s 1ms/step -
accuracy: 0.2656 - loss: 1.3804 - val_accuracy: 0.2756 - val_loss: 1.3708
Epoch 21/100
16/16            0s 1ms/step -
accuracy: 0.2656 - loss: 1.3803 - val_accuracy: 0.2835 - val_loss: 1.3696
Epoch 22/100
16/16            0s 5ms/step -
```

```
accuracy: 0.2656 - loss: 1.3801 - val_accuracy: 0.2835 - val_loss: 1.3684
Epoch 23/100
16/16            0s 1ms/step -
accuracy: 0.2637 - loss: 1.3798 - val_accuracy: 0.2835 - val_loss: 1.3676
Epoch 24/100
16/16            0s 1ms/step -
accuracy: 0.2637 - loss: 1.3796 - val_accuracy: 0.2835 - val_loss: 1.3670
Epoch 25/100
16/16            0s 1ms/step -
accuracy: 0.2637 - loss: 1.3795 - val_accuracy: 0.2835 - val_loss: 1.3664
Epoch 26/100
16/16            0s 1ms/step -
accuracy: 0.2637 - loss: 1.3796 - val_accuracy: 0.2835 - val_loss: 1.3662
Epoch 27/100
16/16            0s 1ms/step -
accuracy: 0.2637 - loss: 1.3793 - val_accuracy: 0.2835 - val_loss: 1.3656
Epoch 28/100
16/16            0s 4ms/step -
accuracy: 0.2637 - loss: 1.3792 - val_accuracy: 0.2835 - val_loss: 1.3654
Epoch 29/100
16/16            0s 1ms/step -
accuracy: 0.2637 - loss: 1.3792 - val_accuracy: 0.2835 - val_loss: 1.3650
Epoch 30/100
16/16            0s 1ms/step -
accuracy: 0.2637 - loss: 1.3790 - val_accuracy: 0.2835 - val_loss: 1.3641
Epoch 31/100
16/16            0s 1ms/step -
accuracy: 0.2637 - loss: 1.3789 - val_accuracy: 0.2835 - val_loss: 1.3633
Epoch 32/100
16/16            0s 1ms/step -
accuracy: 0.2637 - loss: 1.3788 - val_accuracy: 0.2835 - val_loss: 1.3628
Epoch 33/100
16/16            0s 1ms/step -
accuracy: 0.2637 - loss: 1.3787 - val_accuracy: 0.2835 - val_loss: 1.3629
Epoch 34/100
16/16            0s 1ms/step -
accuracy: 0.2637 - loss: 1.3788 - val_accuracy: 0.2835 - val_loss: 1.3629
Epoch 35/100
16/16            0s 2ms/step -
accuracy: 0.2637 - loss: 1.3786 - val_accuracy: 0.2835 - val_loss: 1.3618
Epoch 36/100
16/16            0s 1ms/step -
accuracy: 0.2637 - loss: 1.3786 - val_accuracy: 0.2835 - val_loss: 1.3613
Epoch 37/100
16/16            0s 1ms/step -
accuracy: 0.2637 - loss: 1.3785 - val_accuracy: 0.2835 - val_loss: 1.3612
Epoch 38/100
16/16            0s 1ms/step -
```

```
accuracy: 0.2637 - loss: 1.3784 - val_accuracy: 0.2835 - val_loss: 1.3609
Epoch 39/100
16/16            0s 1ms/step -
accuracy: 0.2637 - loss: 1.3784 - val_accuracy: 0.2835 - val_loss: 1.3605
Epoch 40/100
16/16            0s 1ms/step -
accuracy: 0.2637 - loss: 1.3782 - val_accuracy: 0.2835 - val_loss: 1.3609
Epoch 41/100
16/16            0s 4ms/step -
accuracy: 0.2637 - loss: 1.3783 - val_accuracy: 0.2835 - val_loss: 1.3603
Epoch 42/100
16/16            0s 2ms/step -
accuracy: 0.2656 - loss: 1.3781 - val_accuracy: 0.2835 - val_loss: 1.3600
Epoch 43/100
16/16            0s 1ms/step -
accuracy: 0.2662 - loss: 1.3780 - val_accuracy: 0.2835 - val_loss: 1.3599
Epoch 44/100
16/16            0s 1ms/step -
accuracy: 0.2656 - loss: 1.3780 - val_accuracy: 0.2835 - val_loss: 1.3592
Epoch 45/100
16/16            0s 1ms/step -
accuracy: 0.2591 - loss: 1.3779 - val_accuracy: 0.2835 - val_loss: 1.3588
Epoch 46/100
16/16            0s 1ms/step -
accuracy: 0.2598 - loss: 1.3778 - val_accuracy: 0.3701 - val_loss: 1.3592
Epoch 47/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3778 - val_accuracy: 0.3701 - val_loss: 1.3584
Epoch 48/100
16/16            0s 5ms/step -
accuracy: 0.2997 - loss: 1.3777 - val_accuracy: 0.3701 - val_loss: 1.3581
Epoch 49/100
16/16            0s 1ms/step -
accuracy: 0.3003 - loss: 1.3775 - val_accuracy: 0.3701 - val_loss: 1.3586
Epoch 50/100
16/16            0s 1ms/step -
accuracy: 0.2950 - loss: 1.3774 - val_accuracy: 0.3701 - val_loss: 1.3577
Epoch 51/100
16/16            0s 1ms/step -
accuracy: 0.3003 - loss: 1.3774 - val_accuracy: 0.3701 - val_loss: 1.3584
Epoch 52/100
16/16            0s 1ms/step -
accuracy: 0.2904 - loss: 1.3773 - val_accuracy: 0.3701 - val_loss: 1.3571
Epoch 53/100
16/16            0s 2ms/step -
accuracy: 0.3003 - loss: 1.3772 - val_accuracy: 0.3701 - val_loss: 1.3573
Epoch 54/100
16/16            0s 1ms/step -
```

```
accuracy: 0.2950 - loss: 1.3771 - val_accuracy: 0.3701 - val_loss: 1.3574
Epoch 55/100
16/16            0s 2ms/step -
accuracy: 0.3003 - loss: 1.3770 - val_accuracy: 0.3701 - val_loss: 1.3569
Epoch 56/100
16/16            0s 5ms/step -
accuracy: 0.3003 - loss: 1.3769 - val_accuracy: 0.3701 - val_loss: 1.3580
Epoch 57/100
16/16            0s 2ms/step -
accuracy: 0.2930 - loss: 1.3767 - val_accuracy: 0.3701 - val_loss: 1.3588
Epoch 58/100
16/16            0s 2ms/step -
accuracy: 0.2909 - loss: 1.3769 - val_accuracy: 0.3701 - val_loss: 1.3563
Epoch 59/100
16/16            0s 2ms/step -
accuracy: 0.2952 - loss: 1.3765 - val_accuracy: 0.3701 - val_loss: 1.3572
Epoch 60/100
16/16            0s 2ms/step -
accuracy: 0.2909 - loss: 1.3765 - val_accuracy: 0.3701 - val_loss: 1.3551
Epoch 61/100
16/16            0s 1ms/step -
accuracy: 0.2978 - loss: 1.3766 - val_accuracy: 0.3622 - val_loss: 1.3552
Epoch 62/100
16/16            0s 1ms/step -
accuracy: 0.2983 - loss: 1.3762 - val_accuracy: 0.3701 - val_loss: 1.3561
Epoch 63/100
16/16            0s 1ms/step -
accuracy: 0.2901 - loss: 1.3762 - val_accuracy: 0.3701 - val_loss: 1.3538
Epoch 64/100
16/16            0s 2ms/step -
accuracy: 0.2978 - loss: 1.3764 - val_accuracy: 0.3622 - val_loss: 1.3537
Epoch 65/100
16/16            0s 2ms/step -
accuracy: 0.2952 - loss: 1.3759 - val_accuracy: 0.3701 - val_loss: 1.3567
Epoch 66/100
16/16            0s 5ms/step -
accuracy: 0.2834 - loss: 1.3758 - val_accuracy: 0.3701 - val_loss: 1.3537
Epoch 67/100
16/16            0s 2ms/step -
accuracy: 0.2962 - loss: 1.3755 - val_accuracy: 0.3701 - val_loss: 1.3550
Epoch 68/100
16/16            0s 2ms/step -
accuracy: 0.2862 - loss: 1.3753 - val_accuracy: 0.3701 - val_loss: 1.3558
Epoch 69/100
16/16            0s 1ms/step -
accuracy: 0.2841 - loss: 1.3747 - val_accuracy: 0.3701 - val_loss: 1.3556
Epoch 70/100
16/16            0s 1ms/step -
```

```
accuracy: 0.2848 - loss: 1.3738 - val_accuracy: 0.3701 - val_loss: 1.3541
Epoch 71/100
16/16            0s 2ms/step -
accuracy: 0.2877 - loss: 1.3735 - val_accuracy: 0.3858 - val_loss: 1.3605
Epoch 72/100
16/16            0s 2ms/step -
accuracy: 0.2939 - loss: 1.3748 - val_accuracy: 0.3780 - val_loss: 1.3557
Epoch 73/100
16/16            0s 2ms/step -
accuracy: 0.2862 - loss: 1.3712 - val_accuracy: 0.3858 - val_loss: 1.3575
Epoch 74/100
16/16            0s 1ms/step -
accuracy: 0.2953 - loss: 1.3697 - val_accuracy: 0.3622 - val_loss: 1.3584
Epoch 75/100
16/16            0s 1ms/step -
accuracy: 0.3055 - loss: 1.3666 - val_accuracy: 0.3622 - val_loss: 1.3597
Epoch 76/100
16/16            0s 1ms/step -
accuracy: 0.3087 - loss: 1.3641 - val_accuracy: 0.3937 - val_loss: 1.3535
Epoch 77/100
16/16            0s 1ms/step -
accuracy: 0.3033 - loss: 1.3559 - val_accuracy: 0.4016 - val_loss: 1.3457
Epoch 78/100
16/16            0s 5ms/step -
accuracy: 0.3139 - loss: 1.3479 - val_accuracy: 0.4173 - val_loss: 1.3366
Epoch 79/100
16/16            0s 1ms/step -
accuracy: 0.3173 - loss: 1.3387 - val_accuracy: 0.4331 - val_loss: 1.3243
Epoch 80/100
16/16            0s 2ms/step -
accuracy: 0.3293 - loss: 1.3258 - val_accuracy: 0.4252 - val_loss: 1.3168
Epoch 81/100
16/16            0s 2ms/step -
accuracy: 0.3572 - loss: 1.3148 - val_accuracy: 0.4331 - val_loss: 1.3112
Epoch 82/100
16/16            0s 1ms/step -
accuracy: 0.3599 - loss: 1.3085 - val_accuracy: 0.4331 - val_loss: 1.3085
Epoch 83/100
16/16            0s 1ms/step -
accuracy: 0.3499 - loss: 1.3010 - val_accuracy: 0.4409 - val_loss: 1.3050
Epoch 84/100
16/16            0s 1ms/step -
accuracy: 0.3515 - loss: 1.3018 - val_accuracy: 0.4173 - val_loss: 1.2958
Epoch 85/100
16/16            0s 1ms/step -
accuracy: 0.3506 - loss: 1.2908 - val_accuracy: 0.4173 - val_loss: 1.2865
Epoch 86/100
16/16            0s 1ms/step -
```

```
accuracy: 0.3572 - loss: 1.2829 - val_accuracy: 0.4331 - val_loss: 1.2779
Epoch 87/100
16/16            0s 2ms/step -
accuracy: 0.3659 - loss: 1.2635 - val_accuracy: 0.4488 - val_loss: 1.2728
Epoch 88/100
16/16            0s 1ms/step -
accuracy: 0.3664 - loss: 1.2581 - val_accuracy: 0.4252 - val_loss: 1.2612
Epoch 89/100
16/16            0s 2ms/step -
accuracy: 0.3615 - loss: 1.2544 - val_accuracy: 0.4331 - val_loss: 1.2655
Epoch 90/100
16/16            0s 3ms/step -
accuracy: 0.3538 - loss: 1.2431 - val_accuracy: 0.4409 - val_loss: 1.2626
Epoch 91/100
16/16            0s 1ms/step -
accuracy: 0.3538 - loss: 1.2356 - val_accuracy: 0.4409 - val_loss: 1.2581
Epoch 92/100
16/16            0s 1ms/step -
accuracy: 0.3598 - loss: 1.2278 - val_accuracy: 0.4488 - val_loss: 1.2581
Epoch 93/100
16/16            0s 2ms/step -
accuracy: 0.3463 - loss: 1.2271 - val_accuracy: 0.4409 - val_loss: 1.2559
Epoch 94/100
16/16            0s 2ms/step -
accuracy: 0.3461 - loss: 1.2215 - val_accuracy: 0.4488 - val_loss: 1.2569
Epoch 95/100
16/16            0s 1ms/step -
accuracy: 0.3661 - loss: 1.2155 - val_accuracy: 0.4567 - val_loss: 1.2599
Epoch 96/100
16/16            0s 1ms/step -
accuracy: 0.3485 - loss: 1.2212 - val_accuracy: 0.4567 - val_loss: 1.2525
Epoch 97/100
16/16            0s 2ms/step -
accuracy: 0.3701 - loss: 1.2078 - val_accuracy: 0.4567 - val_loss: 1.2569
Epoch 98/100
16/16            0s 2ms/step -
accuracy: 0.3592 - loss: 1.2016 - val_accuracy: 0.4567 - val_loss: 1.2585
Epoch 99/100
16/16            0s 2ms/step -
accuracy: 0.3578 - loss: 1.2040 - val_accuracy: 0.4567 - val_loss: 1.2546
Epoch 100/100
16/16            0s 1ms/step -
accuracy: 0.3573 - loss: 1.1959 - val_accuracy: 0.4488 - val_loss: 1.2561
Epoch 1/100
16/16            0s 6ms/step -
accuracy: 0.2358 - loss: 4.4041 - val_accuracy: 0.2283 - val_loss: 3.1856
Epoch 2/100
16/16            0s 2ms/step -
```

```
accuracy: 0.2714 - loss: 2.7035 - val_accuracy: 0.2677 - val_loss: 2.6461
Epoch 3/100
16/16            0s 2ms/step -
accuracy: 0.2963 - loss: 2.2935 - val_accuracy: 0.2598 - val_loss: 2.2401
Epoch 4/100
16/16            0s 2ms/step -
accuracy: 0.2667 - loss: 2.0322 - val_accuracy: 0.3071 - val_loss: 2.0417
Epoch 5/100
16/16            0s 2ms/step -
accuracy: 0.2769 - loss: 1.8951 - val_accuracy: 0.2756 - val_loss: 1.8776
Epoch 6/100
16/16            0s 2ms/step -
accuracy: 0.2488 - loss: 1.7771 - val_accuracy: 0.2835 - val_loss: 1.7231
Epoch 7/100
16/16            0s 5ms/step -
accuracy: 0.2568 - loss: 1.5950 - val_accuracy: 0.2992 - val_loss: 1.6612
Epoch 8/100
16/16            0s 2ms/step -
accuracy: 0.2831 - loss: 1.4712 - val_accuracy: 0.3071 - val_loss: 1.6225
Epoch 9/100
16/16            0s 2ms/step -
accuracy: 0.3021 - loss: 1.4120 - val_accuracy: 0.3307 - val_loss: 1.5567
Epoch 10/100
16/16            0s 2ms/step -
accuracy: 0.3122 - loss: 1.3497 - val_accuracy: 0.3386 - val_loss: 1.5118
Epoch 11/100
16/16            0s 2ms/step -
accuracy: 0.3571 - loss: 1.2914 - val_accuracy: 0.3307 - val_loss: 1.4641
Epoch 12/100
16/16            0s 2ms/step -
accuracy: 0.3663 - loss: 1.2607 - val_accuracy: 0.2913 - val_loss: 1.4477
Epoch 13/100
16/16            0s 2ms/step -
accuracy: 0.3894 - loss: 1.2443 - val_accuracy: 0.2992 - val_loss: 1.4299
Epoch 14/100
16/16            0s 2ms/step -
accuracy: 0.3841 - loss: 1.2120 - val_accuracy: 0.3150 - val_loss: 1.4191
Epoch 15/100
16/16            0s 2ms/step -
accuracy: 0.4104 - loss: 1.1830 - val_accuracy: 0.2913 - val_loss: 1.4119
Epoch 16/100
16/16            0s 2ms/step -
accuracy: 0.4126 - loss: 1.1611 - val_accuracy: 0.2835 - val_loss: 1.4055
Epoch 17/100
16/16            0s 2ms/step -
accuracy: 0.4317 - loss: 1.1419 - val_accuracy: 0.2913 - val_loss: 1.3967
Epoch 18/100
16/16            0s 2ms/step -
```

```
accuracy: 0.4327 - loss: 1.1285 - val_accuracy: 0.2677 - val_loss: 1.4014
Epoch 19/100
16/16            0s 3ms/step -
accuracy: 0.4522 - loss: 1.1134 - val_accuracy: 0.2677 - val_loss: 1.3965
Epoch 20/100
16/16            0s 2ms/step -
accuracy: 0.4650 - loss: 1.0965 - val_accuracy: 0.2756 - val_loss: 1.3943
Epoch 21/100
16/16            0s 2ms/step -
accuracy: 0.4838 - loss: 1.0865 - val_accuracy: 0.2835 - val_loss: 1.3959
Epoch 22/100
16/16            0s 2ms/step -
accuracy: 0.4727 - loss: 1.0764 - val_accuracy: 0.2913 - val_loss: 1.3876
Epoch 23/100
16/16            0s 2ms/step -
accuracy: 0.4902 - loss: 1.0630 - val_accuracy: 0.3228 - val_loss: 1.3739
Epoch 24/100
16/16            0s 2ms/step -
accuracy: 0.5046 - loss: 1.0504 - val_accuracy: 0.3465 - val_loss: 1.3792
Epoch 25/100
16/16            0s 2ms/step -
accuracy: 0.5145 - loss: 1.0342 - val_accuracy: 0.3307 - val_loss: 1.3965
Epoch 26/100
16/16            0s 2ms/step -
accuracy: 0.5289 - loss: 1.0258 - val_accuracy: 0.3307 - val_loss: 1.3994
Epoch 27/100
16/16            0s 2ms/step -
accuracy: 0.5170 - loss: 1.0197 - val_accuracy: 0.3150 - val_loss: 1.4060
Epoch 28/100
16/16            0s 2ms/step -
accuracy: 0.5277 - loss: 1.0104 - val_accuracy: 0.2992 - val_loss: 1.4132
Epoch 29/100
16/16            0s 2ms/step -
accuracy: 0.5375 - loss: 1.0022 - val_accuracy: 0.3228 - val_loss: 1.4182
Epoch 30/100
16/16            0s 3ms/step -
accuracy: 0.5453 - loss: 0.9956 - val_accuracy: 0.3228 - val_loss: 1.4235
Epoch 31/100
16/16            0s 2ms/step -
accuracy: 0.5475 - loss: 0.9859 - val_accuracy: 0.3228 - val_loss: 1.4271
Epoch 32/100
16/16            0s 2ms/step -
accuracy: 0.5491 - loss: 0.9771 - val_accuracy: 0.3228 - val_loss: 1.4368
Epoch 33/100
16/16            0s 2ms/step -
accuracy: 0.5623 - loss: 0.9684 - val_accuracy: 0.3307 - val_loss: 1.4419
Epoch 34/100
16/16            0s 2ms/step -
```

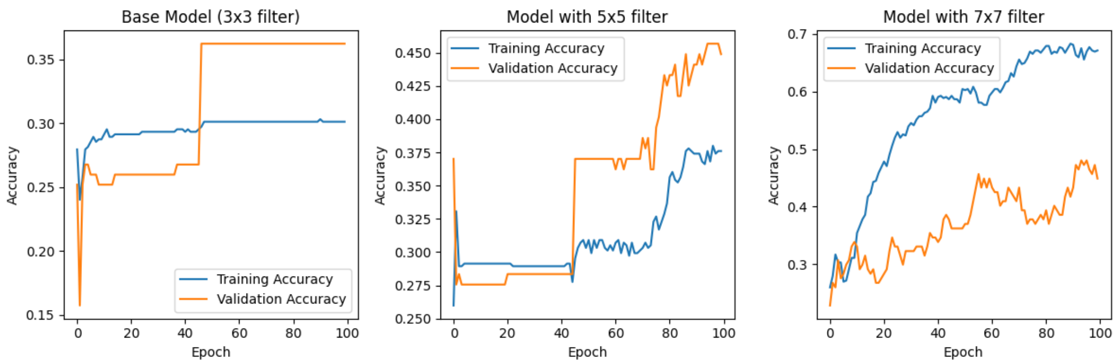
```
accuracy: 0.5666 - loss: 0.9606 - val_accuracy: 0.3307 - val_loss: 1.4434
Epoch 35/100
16/16            0s 3ms/step -
accuracy: 0.5658 - loss: 0.9543 - val_accuracy: 0.3307 - val_loss: 1.4550
Epoch 36/100
16/16            0s 2ms/step -
accuracy: 0.5681 - loss: 0.9503 - val_accuracy: 0.3150 - val_loss: 1.4490
Epoch 37/100
16/16            0s 5ms/step -
accuracy: 0.5650 - loss: 0.9431 - val_accuracy: 0.3307 - val_loss: 1.4200
Epoch 38/100
16/16            0s 2ms/step -
accuracy: 0.5791 - loss: 0.9335 - val_accuracy: 0.3543 - val_loss: 1.4109
Epoch 39/100
16/16            0s 2ms/step -
accuracy: 0.5954 - loss: 0.9274 - val_accuracy: 0.3465 - val_loss: 1.4185
Epoch 40/100
16/16            0s 2ms/step -
accuracy: 0.5897 - loss: 0.9211 - val_accuracy: 0.3465 - val_loss: 1.4316
Epoch 41/100
16/16            0s 2ms/step -
accuracy: 0.5999 - loss: 0.9147 - val_accuracy: 0.3386 - val_loss: 1.4409
Epoch 42/100
16/16            0s 2ms/step -
accuracy: 0.6120 - loss: 0.9140 - val_accuracy: 0.3465 - val_loss: 1.4497
Epoch 43/100
16/16            0s 4ms/step -
accuracy: 0.6014 - loss: 0.9065 - val_accuracy: 0.3780 - val_loss: 1.4673
Epoch 44/100
16/16            0s 5ms/step -
accuracy: 0.6112 - loss: 0.9023 - val_accuracy: 0.3858 - val_loss: 1.4814
Epoch 45/100
16/16            0s 2ms/step -
accuracy: 0.6036 - loss: 0.9001 - val_accuracy: 0.3780 - val_loss: 1.4905
Epoch 46/100
16/16            0s 1ms/step -
accuracy: 0.6171 - loss: 0.8962 - val_accuracy: 0.3622 - val_loss: 1.4953
Epoch 47/100
16/16            0s 2ms/step -
accuracy: 0.6069 - loss: 0.8964 - val_accuracy: 0.3622 - val_loss: 1.5025
Epoch 48/100
16/16            0s 2ms/step -
accuracy: 0.6112 - loss: 0.8885 - val_accuracy: 0.3622 - val_loss: 1.5269
Epoch 49/100
16/16            0s 2ms/step -
accuracy: 0.5990 - loss: 0.8879 - val_accuracy: 0.3622 - val_loss: 1.5139
Epoch 50/100
16/16            0s 2ms/step -
```

```
accuracy: 0.6183 - loss: 0.8778 - val_accuracy: 0.3622 - val_loss: 1.5086
Epoch 51/100
16/16            0s 2ms/step -
accuracy: 0.6222 - loss: 0.8785 - val_accuracy: 0.3701 - val_loss: 1.4817
Epoch 52/100
16/16            0s 1ms/step -
accuracy: 0.6266 - loss: 0.8665 - val_accuracy: 0.3701 - val_loss: 1.4823
Epoch 53/100
16/16            0s 1ms/step -
accuracy: 0.6115 - loss: 0.8645 - val_accuracy: 0.3858 - val_loss: 1.4495
Epoch 54/100
16/16            0s 1ms/step -
accuracy: 0.6299 - loss: 0.8699 - val_accuracy: 0.4094 - val_loss: 1.4457
Epoch 55/100
16/16            0s 2ms/step -
accuracy: 0.6219 - loss: 0.8687 - val_accuracy: 0.4331 - val_loss: 1.4418
Epoch 56/100
16/16            0s 1ms/step -
accuracy: 0.6133 - loss: 0.8770 - val_accuracy: 0.4567 - val_loss: 1.4481
Epoch 57/100
16/16            0s 1ms/step -
accuracy: 0.6036 - loss: 0.8774 - val_accuracy: 0.4331 - val_loss: 1.4611
Epoch 58/100
16/16            0s 4ms/step -
accuracy: 0.6082 - loss: 0.8821 - val_accuracy: 0.4488 - val_loss: 1.4773
Epoch 59/100
16/16            0s 1ms/step -
accuracy: 0.6031 - loss: 0.8929 - val_accuracy: 0.4331 - val_loss: 1.4254
Epoch 60/100
16/16            0s 2ms/step -
accuracy: 0.6163 - loss: 0.8708 - val_accuracy: 0.4488 - val_loss: 1.4233
Epoch 61/100
16/16            0s 2ms/step -
accuracy: 0.6202 - loss: 0.8550 - val_accuracy: 0.4331 - val_loss: 1.4274
Epoch 62/100
16/16            0s 1ms/step -
accuracy: 0.6286 - loss: 0.8465 - val_accuracy: 0.4252 - val_loss: 1.4497
Epoch 63/100
16/16            0s 1ms/step -
accuracy: 0.6367 - loss: 0.8360 - val_accuracy: 0.4252 - val_loss: 1.4750
Epoch 64/100
16/16            0s 1ms/step -
accuracy: 0.6243 - loss: 0.8327 - val_accuracy: 0.4016 - val_loss: 1.4833
Epoch 65/100
16/16            0s 2ms/step -
accuracy: 0.6333 - loss: 0.8324 - val_accuracy: 0.4094 - val_loss: 1.4973
Epoch 66/100
16/16            0s 4ms/step -
```

```
accuracy: 0.6357 - loss: 0.8303 - val_accuracy: 0.4094 - val_loss: 1.4764
Epoch 67/100
16/16            0s 1ms/step -
accuracy: 0.6378 - loss: 0.8269 - val_accuracy: 0.4331 - val_loss: 1.4806
Epoch 68/100
16/16            0s 1ms/step -
accuracy: 0.6463 - loss: 0.8084 - val_accuracy: 0.4252 - val_loss: 1.4877
Epoch 69/100
16/16            0s 1ms/step -
accuracy: 0.6561 - loss: 0.8006 - val_accuracy: 0.4173 - val_loss: 1.4840
Epoch 70/100
16/16            0s 1ms/step -
accuracy: 0.6714 - loss: 0.7842 - val_accuracy: 0.4094 - val_loss: 1.5025
Epoch 71/100
16/16            0s 2ms/step -
accuracy: 0.6812 - loss: 0.7734 - val_accuracy: 0.4331 - val_loss: 1.5098
Epoch 72/100
16/16            0s 1ms/step -
accuracy: 0.6731 - loss: 0.7670 - val_accuracy: 0.3937 - val_loss: 1.5154
Epoch 73/100
16/16            0s 4ms/step -
accuracy: 0.6629 - loss: 0.7630 - val_accuracy: 0.3937 - val_loss: 1.5241
Epoch 74/100
16/16            0s 1ms/step -
accuracy: 0.6796 - loss: 0.7606 - val_accuracy: 0.3701 - val_loss: 1.5465
Epoch 75/100
16/16            0s 1ms/step -
accuracy: 0.6896 - loss: 0.7541 - val_accuracy: 0.3780 - val_loss: 1.5406
Epoch 76/100
16/16            0s 2ms/step -
accuracy: 0.6849 - loss: 0.7527 - val_accuracy: 0.3780 - val_loss: 1.5656
Epoch 77/100
16/16            0s 2ms/step -
accuracy: 0.6938 - loss: 0.7433 - val_accuracy: 0.3701 - val_loss: 1.5541
Epoch 78/100
16/16            0s 2ms/step -
accuracy: 0.6885 - loss: 0.7443 - val_accuracy: 0.3780 - val_loss: 1.5856
Epoch 79/100
16/16            0s 1ms/step -
accuracy: 0.6924 - loss: 0.7384 - val_accuracy: 0.3858 - val_loss: 1.5692
Epoch 80/100
16/16            0s 4ms/step -
accuracy: 0.6930 - loss: 0.7395 - val_accuracy: 0.3780 - val_loss: 1.5666
Epoch 81/100
16/16            0s 2ms/step -
accuracy: 0.7026 - loss: 0.7330 - val_accuracy: 0.3937 - val_loss: 1.5447
Epoch 82/100
16/16            0s 1ms/step -
```

```
accuracy: 0.7007 - loss: 0.7334 - val_accuracy: 0.3701 - val_loss: 1.5764
Epoch 83/100
16/16            0s 2ms/step -
accuracy: 0.6827 - loss: 0.7299 - val_accuracy: 0.3858 - val_loss: 1.5687
Epoch 84/100
16/16            0s 1ms/step -
accuracy: 0.6924 - loss: 0.7274 - val_accuracy: 0.4016 - val_loss: 1.5885
Epoch 85/100
16/16            0s 2ms/step -
accuracy: 0.6897 - loss: 0.7308 - val_accuracy: 0.3937 - val_loss: 1.5856
Epoch 86/100
16/16            0s 2ms/step -
accuracy: 0.7031 - loss: 0.7244 - val_accuracy: 0.3858 - val_loss: 1.6176
Epoch 87/100
16/16            0s 3ms/step -
accuracy: 0.6999 - loss: 0.7249 - val_accuracy: 0.3858 - val_loss: 1.6101
Epoch 88/100
16/16            0s 1ms/step -
accuracy: 0.6871 - loss: 0.7211 - val_accuracy: 0.4173 - val_loss: 1.5464
Epoch 89/100
16/16            0s 1ms/step -
accuracy: 0.7069 - loss: 0.7292 - val_accuracy: 0.4331 - val_loss: 1.5485
Epoch 90/100
16/16            0s 1ms/step -
accuracy: 0.7175 - loss: 0.7260 - val_accuracy: 0.4173 - val_loss: 1.5467
Epoch 91/100
16/16            0s 1ms/step -
accuracy: 0.7086 - loss: 0.7162 - val_accuracy: 0.4331 - val_loss: 1.5431
Epoch 92/100
16/16            0s 1ms/step -
accuracy: 0.6994 - loss: 0.7247 - val_accuracy: 0.4724 - val_loss: 1.5275
Epoch 93/100
16/16            0s 1ms/step -
accuracy: 0.6941 - loss: 0.7279 - val_accuracy: 0.4646 - val_loss: 1.5378
Epoch 94/100
16/16            0s 4ms/step -
accuracy: 0.7008 - loss: 0.7228 - val_accuracy: 0.4803 - val_loss: 1.5408
Epoch 95/100
16/16            0s 1ms/step -
accuracy: 0.6826 - loss: 0.7314 - val_accuracy: 0.4724 - val_loss: 1.5701
Epoch 96/100
16/16            0s 2ms/step -
accuracy: 0.6902 - loss: 0.7309 - val_accuracy: 0.4803 - val_loss: 1.5679
Epoch 97/100
16/16            0s 2ms/step -
accuracy: 0.6954 - loss: 0.7364 - val_accuracy: 0.4646 - val_loss: 1.5731
Epoch 98/100
16/16            0s 2ms/step -
```

```
accuracy: 0.6820 - loss: 0.7410 - val_accuracy: 0.4567 - val_loss: 1.5536
Epoch 99/100
16/16              0s 1ms/step -
accuracy: 0.6812 - loss: 0.7325 - val_accuracy: 0.4724 - val_loss: 1.5493
Epoch 100/100
16/16              0s 1ms/step -
accuracy: 0.6762 - loss: 0.7279 - val_accuracy: 0.4488 - val_loss: 1.5328
```




```
Final Accuracies:
Base Model (3x3) - Training: 0.3012, Validation: 0.3622
5x5 Filter Model - Training: 0.3760, Validation: 0.4488
7x7 Filter Model - Training: 0.6713, Validation: 0.4488
```

3.1 Model Architecture & Performance

3.1.1 Base Model (3×3 filter)

- Training: 0.3012, Validation: 0.3622
- Shows consistent but slow improvement over epochs
- Higher validation than training suggests underfitting
- Stable learning curve with good generalization
- Limited by small receptive field for large features

3.1.2 5×5 Filter Model

- Training: 0.3760, Validation: 0.4488
- Performance improvement from base model
- Shows unstable validation accuracy
- Training curve shows limited learning capacity
- Filter size possibly too large for 6×6 input images

3.1.3 7×7 Filter Model

- Training: 0.6713, Validation: 0.4488
- Best overall performance

- Strongest learning progression
- Shows significant overfitting (train > val)
- Captures global image features effectively

3.2 Model Fitness Analysis

1. **Base Model (3×3)**:
   - Underfits the data
   - Receptive field too small for global patterns
   - Good stability but insufficient learning capacity
2. **5×5 Model**:
   - Moderate underfitting
   - Filter size mismatched with input dimensions
   - Better feature extraction than base model
3. **7×7 Model**:
   - Shows significant overfitting
   - Highest training accuracy but validation matches 5x5
   - Most effective at capturing training patterns

3.3 Key Finding

Larger filter size (7×7) demonstrates superior training performance but overfits, while the 5×5 filter achieves the same validation accuracy with better generalization. This suggests that intermediate filter sizes may provide better balance between feature capture and model generalization for small-resolution images.

4 Question 3: Text Classification using BERT

This section implements a multi-label text classification system by fine-tuning BERT. The task involves:

- Processing tweet data with 11 emotion classes
- Fine-tuning BERT base model for multi-label classification
- Evaluating performance using both strict and flexible matching criteria
- Visualizing the learning process through training curves

[3]:
```
computation_device = torch.device("cuda" if torch.cuda.is_available() else␣
 ↪"cpu")
print(f"Using device: {computation_device}")

emotion_labels = [
    "anger",
    "anticipation",
    "disgust",
```

```
    "fear",
    "joy",
    "love",
    "optimism",
    "pessimism",
    "sadness",
    "surprise",
    "trust",
]

emotion_index_to_label = {idx: label for idx, label in␣
 ↪enumerate(emotion_labels)}
emotion_label_to_index = {label: idx for idx, label in␣
 ↪enumerate(emotion_labels)}

print("Loading datasets...")
training_data = [json.loads(line) for line in open("./train.json", "r")]
validation_data = [json.loads(line) for line in open("./validation.json", "r")]
testing_data = [json.loads(line) for line in open("./test.json", "r")]

training_dataframe = pd.DataFrame(training_data)
validation_dataframe = pd.DataFrame(validation_data)
testing_dataframe = pd.DataFrame(testing_data)

print("Initializing tokenizer...")
bert_tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

print("Converting to HuggingFace datasets...")
training_dataset = Dataset.from_pandas(training_dataframe)
validation_dataset = Dataset.from_pandas(validation_dataframe)
testing_dataset = Dataset.from_pandas(testing_dataframe)

print("Preprocessing datasets...")
tokenized_training = bert_tokenizer(
    training_dataset["Tweet"], padding="max_length", truncation=True,␣
 ↪max_length=128
)
training_label_matrix = np.zeros((len(training_dataset["Tweet"]),␣
 ↪len(emotion_labels)))
for idx, label in enumerate(emotion_labels):
    training_label_matrix[:, idx] = training_dataset[label]
tokenized_training["labels"] = training_label_matrix.tolist()
training_dataset = Dataset.from_dict(tokenized_training)

tokenized_validation = bert_tokenizer(
    validation_dataset["Tweet"], padding="max_length", truncation=True,␣
 ↪max_length=128
```

```
)
validation_label_matrix = np.zeros(
    (len(validation_dataset["Tweet"]), len(emotion_labels))
)
for idx, label in enumerate(emotion_labels):
    validation_label_matrix[:, idx] = validation_dataset[label]
tokenized_validation["labels"] = validation_label_matrix.tolist()
validation_dataset = Dataset.from_dict(tokenized_validation)

tokenized_testing = bert_tokenizer(
    testing_dataset["Tweet"], padding="max_length", truncation=True,␣
 ↪max_length=128
)
testing_label_matrix = np.zeros((len(testing_dataset["Tweet"]),␣
 ↪len(emotion_labels)))
for idx, label in enumerate(emotion_labels):
    testing_label_matrix[:, idx] = testing_dataset[label]
tokenized_testing["labels"] = testing_label_matrix.tolist()
testing_dataset = Dataset.from_dict(tokenized_testing)

training_dataset.set_format("torch")
validation_dataset.set_format("torch")
testing_dataset.set_format("torch")

print("Initializing model...")
emotion_classifier = AutoModelForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    problem_type="multi_label_classification",
    num_labels=len(emotion_labels),
    id2label=emotion_index_to_label,
    label2id=emotion_label_to_index,
)

print("Setting up training arguments...")
training_configuration = TrainingArguments(
    output_dir="./bert_output",
    learning_rate=2e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=5,
    weight_decay=0.01,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    logging_dir="./logs",
    logging_strategy="steps",
```

```
    logging_steps=10,
    remove_unused_columns=False,
    report_to="none",
    save_total_limit=2,
)

print("Initializing trainer...")
model_trainer = Trainer(
    model=emotion_classifier,
    args=training_configuration,
    train_dataset=training_dataset,
    eval_dataset=validation_dataset,
    compute_metrics=lambda eval_pred: {
        "accuracy": accuracy_score(
            (sigmoid(torch.tensor(eval_pred[0])).numpy() > 0.5).astype(np.
 ↪float32),
            eval_pred[1],
        )
    },
)

print("Starting training...")
training_results = model_trainer.train()

training_history = model_trainer.state.log_history
training_metrics = [
    (log["epoch"], log["loss"])
    for log in training_history
    if "loss" in log and "eval_loss" not in log
]
validation_metrics = [
    (log["epoch"], log["eval_loss"]) for log in training_history if "eval_loss"␣
 ↪in log
]
training_metrics.sort(key=lambda x: x[0])
validation_metrics.sort(key=lambda x: x[0])
training_epochs, training_losses = zip(*training_metrics)
validation_epochs, validation_losses = zip(*validation_metrics)

plt.figure(figsize=(10, 6))
plt.plot(training_epochs, training_losses, "b-", label="Training Loss")
plt.plot(validation_epochs, validation_losses, "r-", label="Validation Loss")
plt.title("Training and Validation Loss Curves")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
```

```python
plt.xticks(range(0, int(max(training_epochs)) + 1))
plt.close()

print("\nEvaluating with strict accuracy...")
strict_test_results = model_trainer.evaluate(testing_dataset)
print("\nTest Results (Strict Accuracy - all labels must match):")
print(f"Accuracy: {strict_test_results['eval_accuracy']:.4f}")

model_trainer.compute_metrics = lambda eval_pred: {
    "accuracy": (
        (sigmoid(torch.tensor(eval_pred[0])).numpy() > 0.5).astype(np.float32)
        == eval_pred[1]
    )
    .any(axis=1)
    .mean()
}
any_match_test_results = model_trainer.evaluate(testing_dataset)
print("\nTest Results (Any-Match Accuracy - at least one label must match):")
print(f"Accuracy: {any_match_test_results['eval_accuracy']:.4f}")
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

```
Using device: cuda
Loading datasets…
Initializing tokenizer…
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_auth.py:94:
UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
(https://huggingface.co/settings/tokens), set it as secret in your Google Colab
and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
public models or datasets.
  warnings.warn(
```

```
tokenizer_config.json:   0%|          | 0.00/48.0 [00:00<?, ?B/s]
```

```
config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]
```

```
vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]
```

```
tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
```

```
Converting to HuggingFace datasets…
```

```
Preprocessing datasets…
Initializing model…

model.safetensors:   0%|          | 0.00/440M [00:00<?, ?B/s]

Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1568:
FutureWarning: `evaluation_strategy` is deprecated and will be removed in
version 4.46 of  Transformers. Use `eval_strategy` instead
  warnings.warn(

Setting up training arguments…
Initializing trainer…
Starting training…

<IPython.core.display.HTML object>


Evaluating with strict accuracy…

<IPython.core.display.HTML object>


Test Results (Strict Accuracy - all labels must match):
Accuracy: 0.2580

Test Results (Any-Match Accuracy - at least one label must match):
Accuracy: 1.0000
```