Name : Naikwadi Yash Shivdas

AI&DS II Experiment 03

AIM: To build a Cognitive based application to acquire knowledge through images for a Customer service application / Insurance / Healthcare Application / Smarter Cities / Government etc.

THEORY:

Overview

This experiment aims to design and implement a cognitive application that uses image-based knowledge acquisition to assist customer service operations. Specifically, the focus is on a food delivery service where the system can analyze food images uploaded by users to recognize dishes, suggest menu items, and improve user engagement.

Cognitive Computing in Image Analysis

- Cognitive computing mimics human thought processes by enabling machines to interpret and understand information cognitively rather than just following pre-programmed rules.
- In this experiment, image analysis serves as the primary mode for data acquisition, allowing the system to "see" and understand user-submitted images.
- This capability is built through state-of-the-art deep learning models trained on large labeled datasets to classify and identify food items accurately.

Image Classification Using Deep Learning

- The core technology used is Convolutional Neural Networks (CNNs), which excel in extracting features from images for classification.
- Pre-trained models like ResNet are often used and fine-tuned with domain-specific datasets (e.g., Food-101) to leverage learned visual features while adapting to specific food categories.
- The model assigns probabilities to each food class, detecting what food item is present in the image.

Dataset Utilization (Food-101)

- The Food-101 dataset contains 101,000 food images across 101 categories, providing a comprehensive and diverse training base for the model to learn from.
- This large-scale dataset captures variations in cooking styles, plating, and regional foods, improving the robustness of the recognition system.
- Using this dataset enables the model to generalize better to real-world food images uploaded during customer interactions.

Application in Customer Service

- By integrating image recognition, the customer service system becomes multimodal, accepting both text queries and images.
- When a user uploads a food image, the cognitive system identifies the dish and then provides personalized menu suggestions, order tracking, and related customer support actions.

- This enhances the customer experience by making interactions more intuitive and efficient, supporting contactless and faster service.

Challenges and Solutions

- Image recognition models may face challenges like varying lighting conditions, image quality, and food appearance variations.
- To improve accuracy, data augmentation techniques (rotations, flips, scaling) are applied during training to make the model resilient.
- Hyperparameter tuning (adjusting learning rates, batch sizes) further refines model performance and prevents issues like loss not converging.
- Continuous feedback and retraining with new food images from users can dynamically improve the model post-deployment.

Benefits of Cognitive Image-Based Customer Service

- Reduces human effort in manually identifying and categorizing customer queries related to food items.
- Speeds up problem resolution by auto-detecting customer preferences and issues directly from images.
- Opens up new interactive channels for customer engagement, such as image uploading for quick menu recommendations.
- Provides valuable business insights into popular dishes and customer preferences through data analytics on recognized images.

CODE:

Libraries Import and Setup

```
import numpy as np
import pandas as pd
from pathlib import Path
import matplotlib.image as mpimg
import os.path
import os
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix, classification_report
```

```
from google.colab import files
files.upload()
```

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

Download and Extract Dataset

```
!kaggle datasets download -d kmader/food41
```

```
Dataset URL: https://www.kaggle.com/datasets/kmader/food41
License(s): copyright-authors
Downloading food41.zip to /content
100% 5.30G/5.30G [01:07<00:00, 178MB/s]
100% 5.30G/5.30G [01:07<00:00, 83.9MB/s]
```

```
!unzip food41.zip -d food41
```

```
Streaming output truncated to the last 5000 lines.
  inflating: food41/images/tacos/1030289.jpg
  inflating: food41/images/tacos/1033196.jpg
  inflating: food41/images/tacos/1036030.jpg
…
…
```

```
#creating a directory
image_dir = Path('/content/food41/images')
```

```
class_names = sorted(os.listdir(image_dir))
n_classes = len(class_names)
```

Dataset Overview and Visualization

```
# Discover names of all 101 food classes in the dataset
print(f"Total Number of Classes : {n_classes} \nClass Names : {class_names}")
```

```
Total Number of Classes : 101
Class Names : ['apple_pie', 'baby_back_ribs', 'baklava', 'beef_carpaccio',
'beef_tartare', 'beet_salad', 'beignets', 'bibimbap', 'bread_pudding',
'breakfast_burrito', 'bruschetta', 'caesar_salad', 'cannoli', 'caprese_salad',
'carrot_cake', 'ceviche', 'cheese_plate', 'cheesecake', 'chicken_curry',
'chicken_quesadilla', 'chicken_wings', 'chocolate_cake', 'chocolate_mousse',
'churros', 'clam_chowder', 'club_sandwich', 'crab_cakes', 'creme_brulee',
'croque_madame', 'cup_cakes', 'deviled_eggs', 'donuts', 'dumplings',
'edamame', 'eggs_benedict', 'escargots', 'falafel', 'filet_mignon',
'fish_and_chips', 'foie_gras', 'french_fries', 'french_onion_soup',
'french_toast', 'fried_calamari', 'fried_rice', 'frozen_yogurt',
'garlic_bread', 'gnocchi', 'greek_salad', 'grilled_cheese_sandwich',
'grilled_salmon', 'guacamole', 'gyoza', 'hamburger', 'hot_and_sour_soup',
'hot_dog', 'huevos_rancheros', 'hummus', 'ice_cream', 'lasagna',
'lobster_bisque', 'lobster_roll_sandwich', 'macaroni_and_cheese', 'macarons',
'miso_soup', 'mussels', 'nachos', 'omelette', 'onion_rings', 'oysters',
'pad_thai', 'paella', 'pancakes', 'panna_cotta', 'peking_duck', 'pho',
'pizza', 'pork_chop', 'poutine', 'prime_rib', 'pulled_pork_sandwich', 'ramen',
'ravioli', 'red_velvet_cake', 'risotto', 'samosa', 'sashimi', 'scallops',
'seaweed_salad', 'shrimp_and_grits', 'spaghetti_bolognese',
'spaghetti_carbonara', 'spring_rolls', 'steak', 'strawberry_shortcake',
'sushi', 'tacos', 'takoyaki', 'tiramisu', 'tuna_tartare', 'waffles']
```

```
#checking how many photos are in each class - 100 as is expected
image_df['Label'].value_counts()
```

	count
Label	
spaghetti_bolognese	100
donuts	100
cheesecake	100
french_onion_soup	100
bread_pudding	100
...	...
pizza	100
lobster_bisque	100
baby_back_ribs	100
nachos	100
fried_rice	100

101 rows × 1 columns

dtype: int64

```
# display the first 10 images and their labels
fig, axs = plt.subplots(2, 5, figsize=(10, 5))
axs = axs.flatten()

for i in range(10):
    img_name = image_df['Filepath'].iloc[i]
    img_label = image_df['Label'].iloc[i]

    img = cv2.imread(img_name)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    axs[i].imshow(img)
    axs[i].set_title(img_label)
    axs[i].axis('off')
```

```
plt.show()
```

Train-Test Split and Image Data Generators

```
#creating two image data generators - first one for train and validation
datasets and the second one for test dataset
train_df, test_df = train_test_split(image_df, train_size=0.7, shuffle=True,
random_state=1)

train_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2
)

test_generator = tf.keras.preprocessing.image.ImageDataGenerator(
     rescale=1./255,
)

# setup dataset

train_images = train_generator.flow_from_dataframe(
    dataframe=train_df,
    x_col='Filepath',
    y_col='Label',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=32,
    shuffle=True,
    seed=42,
    subset='training'
)

val_images = train_generator.flow_from_dataframe(
    dataframe=train_df,
    x_col='Filepath',
    y_col='Label',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=32,
    shuffle=True,
    seed=42,
    subset='validation'
)

test_images = test_generator.flow_from_dataframe(
    dataframe=test_df,
    x_col='Filepath',
    y_col='Label',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=32,
    shuffle=False
)
```

```
Found 5656 validated image filenames belonging to 101 classes.
Found 1414 validated image filenames belonging to 101 classes.
Found 3030 validated image filenames belonging to 101 classes.
```

Model Architecture Definition

```
#defining a CNN using the MobileNetV2 architecture pre-trained on the ImageNet
dataset
pretrained = tf.keras.applications.mobilenet_v2.MobileNetV2(
    input_shape=[224,224,3], include_top=False,
    weights='imagenet'
    )

pretrained.trainable = False

model = tf.keras.models.Sequential([
    pretrained,
    tf.keras.layers.GlobalAveragePooling2D(),
    layers.Dropout(0.25),
    layers.Dense(256, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.1),
    layers.Dense(128, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.1),
    layers.Dense(n_classes, activation='softmax')
])
```

```
Downloading data from
```

```
https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h5
9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

Model Compilation

```
#let's configure the learning process using Adam optimizer and
#look at the architecture of the model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss=tf.keras.losses.CategoricalCrossentropy(),
    metrics=[ 'AUC']
)
print(model.summary())
```

```
Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ mobilenetv2_1.00_224            │ (None, 7, 7, 1280)     │     2,257,984 │
│ (Functional)                    │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ global_average_pooling2d        │ (None, 1280)           │             0 │
│ (GlobalAveragePooling2D)        │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 1280)           │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 256)            │       327,936 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization             │ (None, 256)            │         1,024 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_1 (Dropout)             │ (None, 256)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 128)            │        32,896 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_1           │ (None, 128)            │           512 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_2 (Dropout)             │ (None, 128)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 101)            │        13,029 │
└─────────────────────────────────┴────────────────────────┴───────────────┘

Total params: 2,633,381 (10.05 MB)

Trainable params: 374,629 (1.43 MB)

Non-trainable params: 2,258,752 (8.62 MB)

None
```

Model Training with Early Stopping

```
#training the model with a batch size of 64 and saving the training history for
further analysis
```

```
EPOCHS = 50
BATCH_SIZE = 64
early_stop = EarlyStopping(monitor='val_loss',min_delta=0.0001, patience=5,
restore_best_weights = True)
history2 = model.fit(train_images,
                    steps_per_epoch=train_images.samples // BATCH_SIZE // 2,
                    epochs=EPOCHS,
                    validation_data=val_images,
                    validation_steps= val_images.samples // BATCH_SIZE // 2,
                    verbose=1,
                    callbacks=[early_stop],
                    shuffle=True)
```

```
Epoch 1/50
44/44 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 34s 286ms/step - AUC: 0.5842 - loss:
4.8672 - val_AUC: 0.6988 - val_loss: 4.3484
..
..
..
Epoch 23/50
44/44 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 125ms/step - AUC: 0.9753 - loss:
1.5440 - val_AUC: 0.9141 - val_loss: 2.5908
```

Evaluation on Test Dataset

```
#assessing the model's performance on the test dataset
loss, auc = model.evaluate(test_images)
```

```
95/95 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21s 223ms/step - AUC: 0.9136 - loss:
2.5181
```

```
#checking the available metrics
history_dict2 = history2.history

print(history_dict2.keys())
```

```
dict_keys(['AUC', 'loss', 'val_AUC', 'val_loss'])
```

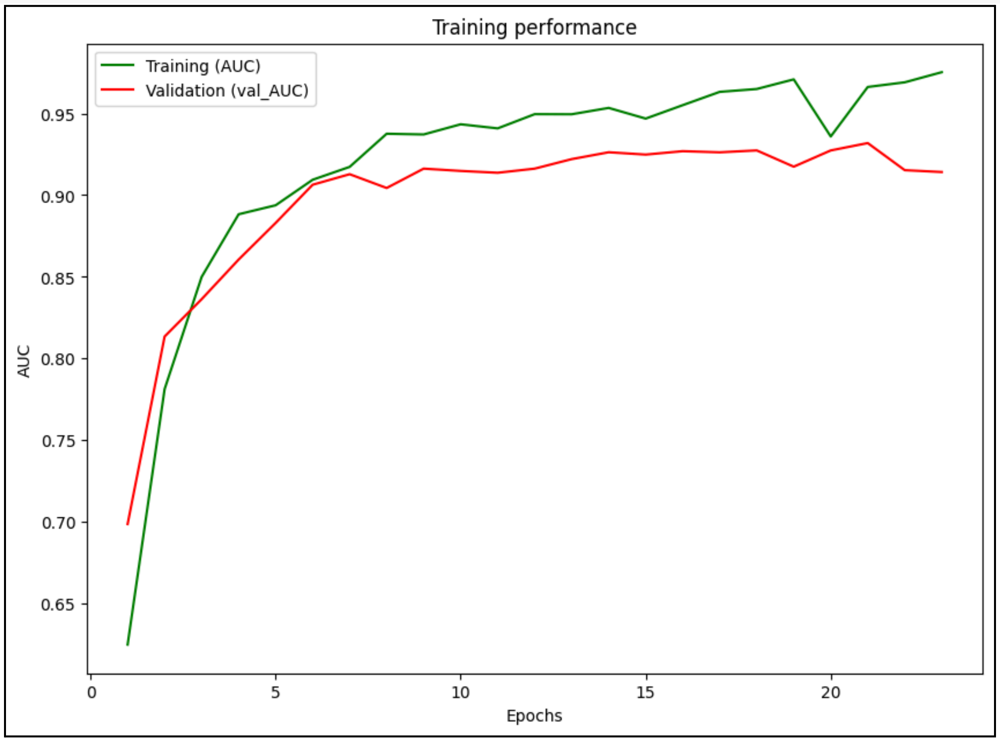
Plotting Training and Validation Metrics

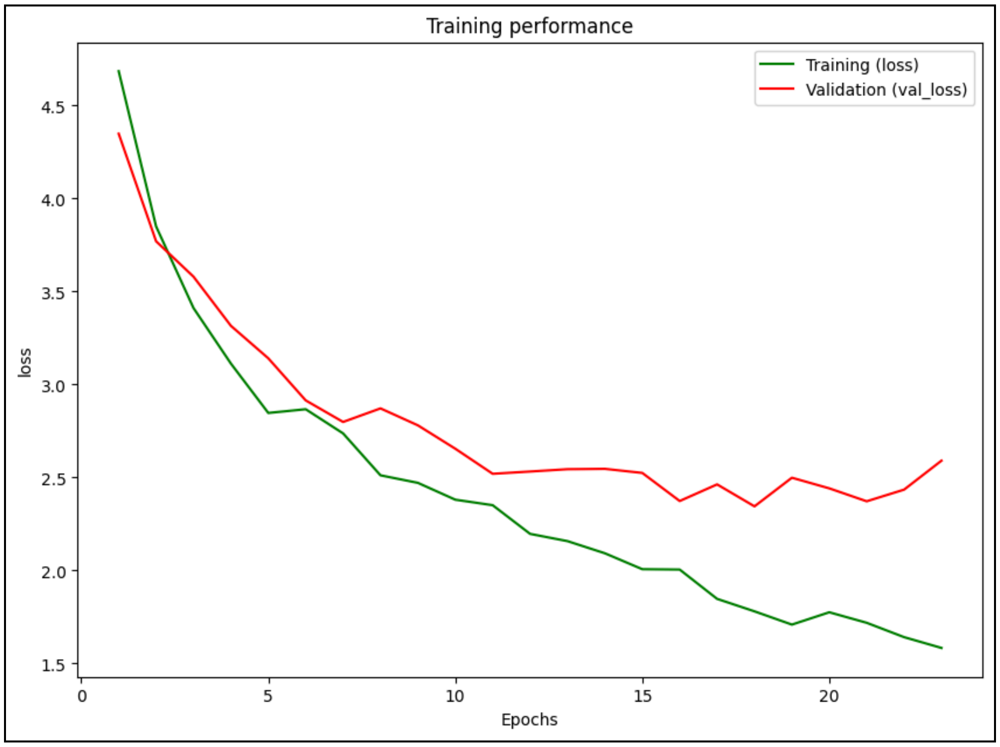
```
def plot_train_instrumentation(epochs, data, train_param, val_param):

    plt.figure(figsize=(10,7))

    plt.plot(epochs, data[train_param], 'g', label=f'Training ({train_param})')
    plt.plot(epochs, data[val_param], 'red', label=f'Validation ({val_param})')

    plt.title("Training performance")
    plt.xlabel('Epochs')
    plt.ylabel(train_param)

    plt.legend()
    plt.show()
```

```
print(history_dict2.keys())
```

```
dict_keys(['AUC', 'loss', 'val_AUC', 'val_loss'])
```

```
#using the previously defined function to see the model's performance
#the loss functions don't converge in one point at the end
#there is a need to apply data augmentation or tune hypermarameters, but so far
this is the best version we've been able to get
epochs = range(1, len(history_dict2['AUC'])+1)

plot_train_instrumentation(epochs, history_dict2, 'AUC', 'val_AUC')
plot_train_instrumentation(epochs, history_dict2, 'loss', 'val_loss')
```





Generating Classification Report

```
#let's generate a classification report for the predictions made by the trained
model on the test dataset
#to summarise the model's performance on each class
predictions = np.argmax(model.predict(test_images), axis=1)


report = classification_report(test_images.labels, predictions,
target_names=test_images.class_indices, zero_division=0)
```

```
95/95 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16s 129ms/step
```

```
#let's look at the report
#the metrics vary between classes: the lowest F1 score is 0 for steak class and
the highest is 0.97 for edamame class
print(report)
```

```
                   precision    recall  f1-score   support

       apple_pie       0.08      0.04      0.05        28
  baby_back_ribs       0.21      0.22      0.22        27
         baklava       0.31      0.32      0.32        28
                                                                          …
                                                                          …
                                                                          …
       tiramisu      0.37      0.17      0.24        40
     tuna_tartare       0.15      0.28      0.20        25
         waffles       0.62      0.38      0.47        34

        accuracy                           0.39      3030
       macro avg       0.41      0.39      0.38      3030
    weighted avg       0.41      0.39      0.38      3030
```

CONCLUSION:

This experiment successfully demonstrated the use of cognitive computing with image recognition to enhance customer service in food delivery. By leveraging deep learning models to identify food items from images, the system can provide quick, personalized recommendations and improve customer engagement. This approach shows great potential for making food delivery services more efficient, intuitive, and user-friendly through intelligent image-based interactions.

39_colab_03