Practical 3

Name: Sharvari Kishor More
Class: D20B
Roll No.: 35

Aim:

To build a **cognitive-based application** that acquires knowledge from images and applies it in real-world domains such as **Customer Service, Insurance, Healthcare, Smarter Cities, and Government services**.

Theory:

- **Cognitive Computing** refers to systems that mimic human thought processes to analyze and interpret complex data. In this context, the system acquires knowledge from **medical images** (MRI scans) and helps in **decision support** for doctors.
- **Healthcare Application**: Early detection of strokes through MRI images is critical for treatment. Manual interpretation is time-consuming and prone to human error. A cognitive AI model can automate classification, assist radiologists, and reduce diagnosis time.
- **Deep Learning (CNN)**: Convolutional Neural Networks are widely used for **image classification**. CNNs extract patterns and features such as edges, textures, and shapes from MRI scans to differentiate between Normal, Ischemic, and Hemorrhagic conditions.
- **Knowledge Acquisition from Images**: By training on a labeled dataset of 750+ MRI images, the model "learns" medical patterns and applies them to new unseen cases, thereby **acquiring and applying knowledge cognitively**.

Libraries/Tools Used

1. **TensorFlow & Keras** – Used to build and train the CNN deep learning model for image classification.
2. **Matplotlib** – Used for visualization of training accuracy/loss curves.
3. **NumPy** – For numerical operations and handling image arrays.
4. **KaggleHub** – To download the MRI dataset directly from Kaggle into Colab.
5. **OS** – To handle dataset paths and directory management.
6. **ImageDataGenerator (from Keras)** – For image preprocessing, rescaling, and data augmentation (rotation, flipping, zoom).

Code:

Step 1: Install & Import Libraries

Before building any AI application, we need supporting libraries for deep learning, preprocessing, visualization, and dataset handling.

```
!pip install kagglehub --quiet
import os
import numpy as np
```

```
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models
import kagglehub
```

This step sets up the environment required for cognitive application development.

Step 2: Download Dataset

- **Theory**:
  Cognitive systems rely on **knowledge acquisition** from large datasets. In this experiment, we use **Brain MRI images** that belong to three categories: *Normal, Ischemic, and Hemorrhagic*.
  Using `kagglehub`, the dataset is fetched directly from Kaggle, ensuring easy access without manual uploads.

```
path = kagglehub.dataset_download("mitangshu11/brain-stroke-mri-images")
print("Dataset path:", path)
# Use preprocessed images (without text overlay)
data_dir = os.path.join(path,"dataset", "Stroke Classification")
print("Data directory:", data_dir)
```

Output:

```
Dataset path: /kaggle/input/brain-stroke-mri-images
Data directory: /kaggle/input/brain-stroke-mri-images/dataset/Stroke
Classification
```

- This step ensures the model has a structured dataset for learning.

Step 3: Data Preprocessing

- **Theory**:
  Preprocessing prepares raw MRI images into a form that a CNN can learn from:
  - **Rescaling**: Normalizes pixel values from 0–255 into 0–1 for faster convergence.
  - **Resizing**: Standardizes all images to the same size (128×128 pixels).
  - **Data Augmentation**: Applies random transformations (rotation, zoom, flipping) to artificially increase dataset size and prevent overfitting.
  - **Train-Validation Split**: Divides dataset into training (80%) and validation (20%) to evaluate model generalization.

```
img_size = (128,128)
batch_size = 32

train_datagen = ImageDataGenerator(
    rescale=1./255,
```

```
    validation_split=0.2,
    rotation_range=15,
    zoom_range=0.1,
    horizontal_flip=True
)

train_gen = train_datagen.flow_from_directory(
    data_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode="categorical",
    subset="training"
)

val_gen = train_datagen.flow_from_directory(
    data_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode="categorical",
    subset="validation"
)

print("Classes:", train_gen.class_indices)
```

Output:

```
Found 493 images belonging to 3 classes.
Found 122 images belonging to 3 classes.
Classes: {'Haemorrhagic': 0, 'Ischemic': 1, 'Normal': 2}
```

This step improves model performance and ensures fair evaluation.

Step 4: Build CNN Model:

A **Convolutional Neural Network (CNN)** is used because it mimics how humans visually recognize patterns:

- **Convolutional Layers** automatically extract low-level features (edges, corners) and high-level features (shapes, textures).
- **Pooling Layers** reduce dimensionality while retaining important information.
- **Dense Layers** combine extracted features to classify images.
- **Softmax Output Layer** produces probabilities for each class (Normal, Ischemic, Hemorrhagic).
- **Loss Function (Categorical Crossentropy)** measures how well the prediction matches the true label.

- **Optimizer (Adam)** updates weights efficiently during training.

```
model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(128,128,3)),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(128, (3,3), activation='relu'),
    layers.MaxPooling2D(2,2),

    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(3, activation='softmax')   # 3 classes: Normal, Ischemic,
Hemorrhagic
])

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

model.summary()
```

Output:
Model: "sequential_1"

Layer (type)	**Output Shape**	**Param**
conv2d_3 (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_3 (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_4 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_4 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_5 (Conv2D)	(None, 28, 28, 128)	73,856
max_pooling2d_5 (MaxPooling2D)	(None, 14, 14, 128)	0

flatten_1 (Flatten)	(None, 25088)	0
dense_2 (Dense)	(None, 128)	3,211,392
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 3)	387

```
Total params: 3,305,027 (12.61 MB)
Trainable params: 3,305,027 (12.61 MB)
Non-trainable params: 0 (0.00 B)
```

This step provides the cognitive model that “learns” from MRI data.

Step 5: Train Model

- **Theory**:
  Training is the process where the CNN repeatedly adjusts its internal weights using **backpropagation** to minimize loss.
  - **Epochs**: Number of times the model sees the entire dataset.
  - **Batch Size**: Number of samples processed before updating weights.
  - During training, the model learns stroke-related features from MRI images.

This step represents the **knowledge acquisition phase** of cognitive computing.

```
history = model.fit(
    train_gen,
    validation_data=val_gen,
    epochs=20
)
```

Step 6: Evaluate Performance

- **Theory**:
  After training, the model is evaluated on unseen validation data to check generalization ability.
  - **Accuracy Curve** shows how well the model is learning.
  - **Loss Curve** shows how much error remains during training.
    Comparing training and validation performance helps detect **overfitting** or **underfitting**.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(len(acc))

plt.figure(figsize=(12,5))
```

```
plt.subplot(1,2,1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Accuracy')

plt.subplot(1,2,2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Loss')
plt.show()
```

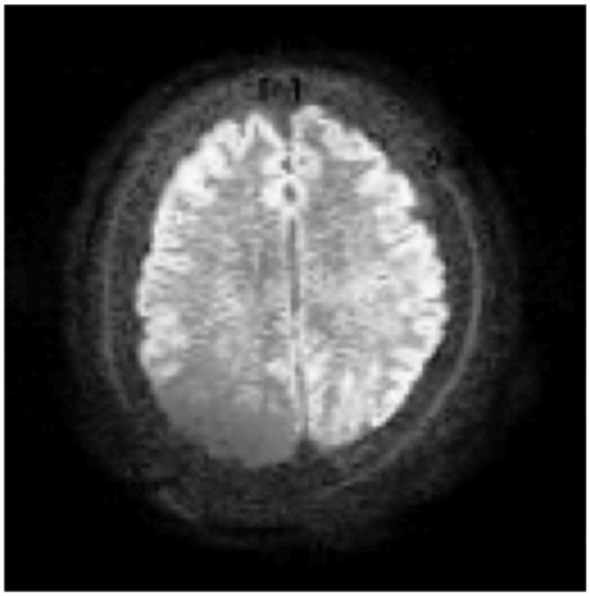
This step ensures the model is reliable and can be trusted for decision support.

Step 7: Test Prediction

- **Theory**:
  To simulate a **real-world application**, we test the trained model on a random MRI image.
  - The model predicts whether the image belongs to *Normal, Ischemic, or Hemorrhagic*.
  - Comparing prediction with true label shows practical effectiveness.

```
import random

# Correct way to get a batch from generator
test_img, test_label = next(val_gen)

i = random.randint(0, test_img.shape[0]-1)
img = test_img[i]
```

```
true_label = np.argmax(test_label[i])

plt.imshow(img)
plt.axis("off")
plt.show()

pred = model.predict(np.expand_dims(img, axis=0))
print("Predicted:", np.argmax(pred), "| True:", true_label)
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
Predicted: 2 | True: 2
```

This step demonstrates how the cognitive application can assist healthcare professionals by classifying MRI images automatically.

Conclusion

This experiment successfully demonstrates the use of **cognitive AI systems** for knowledge acquisition from images. By applying **deep learning (CNN)** on MRI scans, the system was able to classify brain images into **Normal, Ischemic, and Hemorrhagic Stroke categories**.

The results prove that **cognitive-based applications** can:

- Aid **Healthcare** by assisting radiologists in faster and more accurate diagnosis.
- Be extended to other domains such as **Insurance (automated claim verification using images), Customer Service (visual query resolution), Smarter Cities (traffic monitoring), and Government (identity verification, public health monitoring)**.

Thus, the aim of building a **cognitive-based application for knowledge acquisition through images** is achieved.

Here is the attached Colab file.