Experiment – 11

Genre-Based Movie Recommendations Using Convolutional Neural Networks (CNNs)

Aim: Capstone Project – 1

Introduction:

Objective: The goal of this project is to build an image classification model capable of Recommending different movies categories using CNNs. Image classification is an essential task in computer vision, aiding applications such as automatic content filtering, recommendation systems, and more.

Project Scope: This document provides a detailed walkthrough of the steps involved, including data preprocessing, model design, training, and testing, with code explanations at each stage.

Dataset Description: The dataset is composed of 80 training images and 80 validation images across eight movie genre classes. Classes are arranged in separate folders, which simplifies the dataset structure and facilitates the use of directory-based loading. Tools and Technologies Used: We leverage TensorFlow, Keras, and Google Colab for model building, training, and testing. Google Colab provides GPU support, which speeds up training.

Setting Up the Environment:

Importing the required libraries: Import necessary libraries at the start of your source code to access their functions, streamline development, and enhance code efficiency. This practice improves readability and organization, making collaboration easier and debugging simpler.

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers, models
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from google.colab import drive
import os
import matplotlib.pyplot as plt
import random
```

Mounting Google Drive: Since the dataset is hosted on Google Drive, we use drive.mount to access it in Colab.

```
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive;
```

This mounts Google Drive at the specified path, allowing us to load images directly for training and validation.

Organizing Dataset Directories: The dataset structure has folders for each class under train and validation directories. This organization aligns with Keras's flow_from_directory method for efficient data loading.

Data Preprocessing with ImageDataGenerator:

Image Rescaling: Since pixel values range between 0-255, rescaling to the range 0-1 helps improve model performance by standardizing input values.

```
train_dir = 'drive/MyDrive/movies project/genre/Train'
validation_dir = 'drive/MyDrive/movies project/genre/Test'
```

Batch Data Loading: The flow_from_directory function is used to load and label images in batches, improving memory efficiency. We specify parameters like target_size for resizing images, batch_size to define how many images are processed in each batch, and class_mode='categorical' since it's a multi-class classification problem.

```
train_datagen = ImageDataGenerator(rescale=1.0/255)
validation_datagen = ImageDataGenerator(rescale=1.0/255)
```

```
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=8,
    class_mode='categorical'
)
```

Expected Output: Upon execution, it reports the number of images found in each directory, confirming that all classes are loaded correctly.

Building the Convolutional Neural Network (CNN) :

Model Design: CNNs are widely used for image classification due to their ability to capture spatial hierarchies in images. Our model is designed with three convolutional blocks followed by pooling layers and dense layers.

1. Layer 1 (Convolution + MaxPooling): The first convolutional layer applies 32 filters, each of size (3x3), with ReLu activation to introduce non-linearity. Max pooling with a 2x2 filter reduces spatial dimensions, retaining the most important features.

2. Layer 2 & 3 (Deeper Convolutions): Subsequent layers have increasing filter sizes (64 and 128), enabling the model to learn more complex features. Pooling reduces dimensionality to prevent overfitting.

3. Flattening and Dense Layers: The Flatten layer converts 3D feature maps into a 1D vector, which feeds into Dense layers for classification. The dropout layer (0.5) reduces overfitting by randomly ignoring 50% of neurons during training.

```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(train_generator.num_classes, activation='softmax')
])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Expected Output Warning: A minor warning suggests not specifying input_shape within the Sequential model layers directly, as the Sequential model can infer it.

Compiling the Model:

Optimizer and Loss Function: The Adam optimizer combines the best properties of Adagrad and RMSProp to handle sparse gradients and adjust the learning rate. We use categorical_crossentropy as our loss function due to the multi-class nature of the problem.

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

Valuation Metric: Accuracy is chosen to evaluate how well the model classifies images into the correct categories.

Training the Model:

Model Fitting: The fit method trains the model using train_generator and validation_generator over 10 epochs. Steps per Epoch: Steps_per_epoch=100 indicates the number of batches per epoch. Epochs: Set to 10, though tuning this can optimize performance.

```
history = model.fit(
 train_generator,
 steps_per_epoch=100,
 epochs=10,
 validation_data=validation_generator,
 validation_steps=50
 )
```

```
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 24s 192ms/step - accuracy: 0.1417 - loss: 2.0868 - val_accuracy: 0.3250 - val_loss: 2.0033
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 22s 165ms/step - accuracy: 0.2842 - loss: 1.9686 - val_accuracy: 0.5500 - val_loss: 1.7720
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 146ms/step - accuracy: 0.2945 - loss: 1.7203 - val_accuracy: 0.6625 - val_loss: 1.2467
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 136ms/step - accuracy: 0.5748 - loss: 1.2205 - val_accuracy: 0.7250 - val_loss: 0.8599
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 21s 140ms/step - accuracy: 0.7014 - loss: 0.8451 - val_accuracy: 0.9000 - val_loss: 0.3962
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 137ms/step - accuracy: 0.8808 - loss: 0.4515 - val_accuracy: 0.9625 - val_loss: 0.2077
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 141ms/step - accuracy: 0.9649 - loss: 0.1923 - val_accuracy: 0.9625 - val_loss: 0.1210
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 24s 161ms/step - accuracy: 0.9147 - loss: 0.2328 - val_accuracy: 0.9250 - val_loss: 0.2239
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 158ms/step - accuracy: 0.9502 - loss: 0.2774 - val_accuracy: 0.9875 - val_loss: 0.1012
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 151ms/step - accuracy: 0.9877 - loss: 0.1837 - val_accuracy: 1.0000 - val_loss: 0.0273
```

Interpreting Outputs: At each epoch, we get metrics for training and validation accuracy and loss, showing how well the model learns.

Evaluating Model Performance:

Training and Validation Accuracy: After training, we evaluate the model’s performance. The training accuracy reaches around 98.75%, while the validation accuracy is even higher at 100.00%. This result suggests the possibility of overfitting.

```
train_accuracy=history.history['accuracy'][-1]
val_accuracy=history.history['val_accuracy'][-1]
print(f"Final Training Accuracy:{train_accuracy*100:.2f}%")
print(f"Final Validation Accuracy:{val_accuracy*100:.2f}%")
```

Explanation: High training accuracy and lower validation accuracy often point to overfitting. Adding more data, adjusting dropout rates, or experimenting with regularization could improve generalization.

Making Recommendations on Given Genre:

Displaying Movie Images by Genre: The provided code defines a function to display random movie images based on user-selected genres. It allows users to input a genre and the number of images they wish to see, enhancing their experience by visually exploring movie options.

```
def display_images_from_genre(genre_dir, num_images):
    all_images = [os.path.join(genre_dir, img)
                  for img in os.listdir(genre_dir)
                  if img.endswith(('png', 'jpg', 'jpeg'))]
    if len(all_images) == 0:
        print(f"No images found in the directory: {genre_dir}")
        return
    selected_images = random.sample(all_images,min(num_images, len(all_images)))
    plt.figure(figsize=(15, 5))

    for i, img_path in enumerate(selected_images):
        img = plt.imread(img_path)
        plt.subplot(1, num_images, i + 1)
        plt.imshow(img)
        plt.axis('off')
        txt=os.path.basename(img_path).replace('.jpg','')
        plt.title(txt)
    plt.show()
```

This function efficiently retrieves and displays random movie images from a specified genre, enriching the user experience by providing a visual representation of their selections and facilitating easier movie discovery.

User Interaction for Movie Genre Selection: This code segment facilitates user input for selecting a movie genre and the number of images to display, enhancing the interactive experience of exploring movie options visually.

```
genre_list="(e.g.,Action,Comedy,Horror,Thriller,Romance,Sci-Fi,Fantasy,War)"
def main():
    base_dir_test='drive/MyDrive/movies project/genre/Test'
    base_dir_train='drive/MyDrive/movies project/genre/Train'
    genre_input = input(F"Enter the genre {genre_list.replace(',',' ,')}: ")
    num_images=int(input('Enter the number of movies (10 Maximum) : '))
    genre_input=genre_input.title()
    genre_dir_test = os.path.join(base_dir_test, genre_input)
    genre_dir_train = os.path.join(base_dir_train, genre_input)
    if os.path.exists(genre_dir_test):
        display_images_from_genre(genre_dir_test, num_images)
    elif os.path.exists(genre_dir_train):
        display_images_from_genre(genre_dir_train, num_images)
    else:
        print(f"No such genre: {genre_input}. Please check the folder name.")

main()
```

This code segment enhances user interaction by allowing genre selection and specifying the number of images to display. It effectively guides users in visually exploring movie options based on their preferences.

Displaying the Image: The image is loaded and resized to 150x150 pixels to match the model's expected input dimensions. The image is displayed with the Movie name class label as a title, making the output easily interpretable. This visualization helps users quickly assess the model's performance by observing both the image and its predicted category directly.

Source Code:

Input:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers, models
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from google.colab import drive
import os
import matplotlib.pyplot as plt
import random
```

```
drive.mount('/content/drive')
```

Ouput:

```
Mounted at /content/drive
```

Input:

```
train_dir = 'drive/MyDrive/movies project/genre/Train'
validation_dir = 'drive/MyDrive/movies project/genre/Test'
```

```
train_datagen = ImageDataGenerator(rescale=1.0/255)
validation_datagen = ImageDataGenerator(rescale=1.0/255)
```

Output:

Errors will be displayed if any issues occur during the execution process; otherwise, no messages will be shown.

Input:

```
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=8,
    class_mode='categorical'
)
```

Output:

```
Found 80 images belonging to 8 classes.
```

Input:

```
validation_generator = validation_datagen.flow_from_directory
    validation_dir,
    target_size=(150, 150),
    batch_size=8,
    class_mode='categorical'
)
```

Output:

```
Found 80 images belonging to 8 classes.
```

Input:

```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(train_generator.num_classes, activation='softmax')
])
```

Output:

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107:
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Input:

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

Output:

Errors will be displayed if any issues occur during the execution process; otherwise, no messages will be shown.

Input:

```
history = model.fit(
 train_generator,
 steps_per_epoch=100,
 epochs=10,
 validation_data=validation_generator,
 validation_steps=50
 )
```

Output:

```
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 24s 192ms/step - accuracy: 0.1417 - loss: 2.0868 - val_accuracy: 0.3250 - val_loss: 2.0033
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 22s 165ms/step - accuracy: 0.2842 - loss: 1.9686 - val_accuracy: 0.5500 - val_loss: 1.7720
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 146ms/step - accuracy: 0.2945 - loss: 1.7203 - val_accuracy: 0.6625 - val_loss: 1.2467
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 136ms/step - accuracy: 0.5748 - loss: 1.2205 - val_accuracy: 0.7250 - val_loss: 0.8599
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 21s 140ms/step - accuracy: 0.7014 - loss: 0.8451 - val_accuracy: 0.9000 - val_loss: 0.3962
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 137ms/step - accuracy: 0.8808 - loss: 0.4515 - val_accuracy: 0.9625 - val_loss: 0.2077
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 141ms/step - accuracy: 0.9649 - loss: 0.1923 - val_accuracy: 0.9625 - val_loss: 0.1210
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 24s 161ms/step - accuracy: 0.9147 - loss: 0.2328 - val_accuracy: 0.9250 - val_loss: 0.2239
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 158ms/step - accuracy: 0.9502 - loss: 0.2774 - val_accuracy: 0.9875 - val_loss: 0.1012
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 151ms/step - accuracy: 0.9877 - loss: 0.1837 - val_accuracy: 1.0000 - val_loss: 0.0273
```

Input:

```
train_accuracy=history.history['accuracy'][-1]
val_accuracy=history.history['val_accuracy'][-1]
print(f"Final Training Accuracy:{train_accuracy*100:.2f}%")
print(f"Final Validation Accuracy:{val_accuracy*100:.2f}%")
```

Output:

```
Final Training Accuracy:98.75%
Final Validation Accuracy:100.00%
```

Input:

```python
def display_images_from_genre(genre_dir, num_images):
    all_images = [os.path.join(genre_dir, img)
                  for img in os.listdir(genre_dir)
                  if img.endswith(('png', 'jpg', 'jpeg'))]
    if len(all_images) == 0:
        print(f"No images found in the directory: {genre_dir}")
        return
    selected_images = random.sample(all_images,min(num_images, len(all_images)))
    plt.figure(figsize=(15, 5))

    for i, img_path in enumerate(selected_images):
        img = plt.imread(img_path)
        plt.subplot(1, num_images, i + 1)
        plt.imshow(img)
        plt.axis('off')
        txt=os.path.basename(img_path).replace('.jpg','')
        plt.title(txt)
    plt.show()
```

Output:

Errors will be displayed if any issues occur during the execution process; otherwise, no messages will be shown.

Input:

```python
genre_list="(e.g.,Action,Comedy,Horror,Thriller,Romance,Sci-Fi,Fantasy,War)"
def main():
    base_dir_test='drive/MyDrive/movies project/genre/Test'
    base_dir_train='drive/MyDrive/movies project/genre/Train'
    genre_input = input(F"Enter the genre {genre_list.replace(',',' ,')}: ")
    num_images=int(input('Enter the number of movies (10 Maximum) : '))
    genre_input=genre_input.title()
    genre_dir_test = os.path.join(base_dir_test, genre_input)
    genre_dir_train = os.path.join(base_dir_train, genre_input)
    if os.path.exists(genre_dir_test):
        display_images_from_genre(genre_dir_test, num_images)
    elif os.path.exists(genre_dir_train):
        display_images_from_genre(genre_dir_train, num_images)
    else:
        print(f"No such genre: {genre_input}. Please check the folder name.")

main()
```

Output:

```
Enter the genre (e.g. ,Action ,Comedy ,Horror ,Thriller ,Romance ,Sci-Fi ,Fantasy ,War): action
Enter the number of movies (10 Maximum) : 1
```

Drive

Result Analysis:

After receiving inputs for the genre and the desired number of movies, the model efficiently generates a tailored list of movie recommendations. For each movie, the model presents only the image and title, ensuring a clean and focused display. The recommendations are limited to the specified number, aligning precisely with the user's preferences.

Training Accuracy: Approximately 98.75%, indicating the model has learned to classify training images accurately.

Validation Accuracy: Around 100.00%, which is significantly lower than the training accuracy, suggesting a potential overfitting issue where the model generalizes poorly to unseen data.

This streamlined approach enhances user experience by providing quick, visually engaging results that meet the requested quantity and genre. The model's CNN-powered recommendation engine further boosts the accuracy and relevance of the suggestions.

Conclusion:

This project successfully applies Convolutional Neural Networks (CNNs) to recommend movies based on genre and the specified number of movies requested. By analyzing genre-specific features in the training data, the model provides relevant movie names and images for each genre. This work demonstrates CNNs effectiveness in building genre-based, visually guided movie recommendation systems that deliver customized results according to user-defined preferences.