**ASSIGNEMENT 2**

**Link To Github**

<https://github.com/huma-naveed/assignment_2>

**Task 1**

**Hand written digits dataset was given to build CNN from scratch**

**Step 1: Environment Setup**

* **Google Colab**: Utilized Google Colab for its GPU capabilities, ensuring faster model training.
* **Drive Mounting**: Mounted Google Drive to access the dataset stored there.

**Step 2: Data Preparation**

* **Data Loading**: Loaded the dataset from the specified directory. The dataset consists of hand-written digit images in three classes (0, 1, 2).
* **Preprocessing**: Set up the data loaders with appropriate parameters for batch size, image dimensions, and grayscale color mode. Utilized a split of 80% for training and 20% for validation.
* **Data Augmentation**: Implemented data augmentation to increase the diversity of the training data without collecting new images. This helps improve the robustness of the model.

**Step 3: Model Setup**

* **CNN Architecture**: Defined a sequential model with the following layers:
  + **Convolutional Layers**: Two convolutional layers with ReLU activation to extract features from images.
  + **Pooling Layers**: Max pooling layers to reduce the spatial dimensions of the feature maps.
  + **Batch Normalization**: Normalized the inputs to each layer to stabilize and speed up training.
  + **Flatten Layer**: Flattened the feature maps to feed into the dense layers.
  + **Dense Layers**: Two dense layers, including a dropout layer to prevent overfitting and a final softmax layer for classification.

**Step 4: Compilation and Training**

* **Compilation**: Compiled the model with the Adam optimizer and sparse categorical cross-entropy loss function. This setup is typical for multi-class classification problems.
* **Training**: Trained the model for 10 epochs, monitoring both training and validation accuracy and loss. This process updates the model weights to minimize the loss function.

**Step 5: Evaluation and Visualization**

* **Evaluation**: Evaluated the model on the validation dataset to assess its performance.
* **Visualization**: Plotted training and validation accuracy and loss over epochs to visualize the learning process and identify any issues like overfitting or underfitting.

**Step 6: Conclusion**

* **Results Interpretation**: Discussed the model's performance, including any insights or anomalies observed during training and evaluation.
* **Further Steps**: Suggested potential improvements or further experiments, such as adjusting the model architecture, trying different hyperparameters, or using additional data augmentation techniques.

**Task 2**

Enhancing Handwritten Digit Classification with Transfer Learning and Fine Tuning

The primary objective of Task 2 is to significantly improve the handwritten digit classification model from Task 1 by implementing transfer learning and fine-tuning techniques using the VGG16 model pre-trained on the ImageNet dataset.

1. **Transfer Learning with VGG16**:
   * **VGG16 as Feature Extractor**: Utilized the VGG16 model, renowned for its effectiveness in image recognition tasks, as a starting point. Loaded it without the top classification layer to serve as a powerful feature extractor.
   * **Custom Top Layers**: Added new layers on top of VGG16 to tailor it to our specific task of classifying 3 classes of handwritten digits. This included a Flatten layer to convert the 2D feature maps into a 1D vector, followed by Dense layers for classification.
   * **Initial Training**: Compiled the model with the Adam optimizer and sparse categorical cross-entropy loss function, reflecting the integer nature of our labels. Trained the model for a predefined number of epochs, allowing only the weights of the new layers to adjust.
2. **Fine-Tuning the Model**:
   * **Unfreezing Layers**: After the initial training, unfroze the top layers of the VGG16 model to allow them to adjust during further training. This fine-tuning process helps the model better adapt to the specifics of our data.
   * **Recompilation**: Recompiled the model with a significantly lower learning rate. This cautious approach is crucial to avoid disrupting the pre-learned features during the fine-tuning phase.
   * **Continued Training**: Continued training the model for additional epochs, allowing the unfrozen layers to learn from our specific dataset.

**Results**

* **Improved Accuracy**: Transfer learning with VGG16 provided a substantial boost in accuracy compared to the basic CNN model from Task 1. The model's ability to generalize and accurately classify unseen data improved significantly.
* **Fine-Tuning Impact**: Fine-tuning further enhanced the model's performance, making the pre-trained layers more relevant to the specific task of classifying handwritten digits and thereby improving the validation accuracy.

Task 2 demonstrated the power of transfer learning and fine-tuning in significantly improving the performance of deep learning models, especially when dealing with limited datasets. By leveraging the knowledge gained from a model pre-trained on a large and diverse dataset (ImageNet), we were able to achieve high accuracy in classifying handwritten digits, a task with a much smaller and specific dataset

**Task 3**

1. **Environment and Data Setup**:
   * **Google Colab**: The task is performed in a Google Colab environment to leverage its GPU support for faster processing.
   * **Drive Mounting**: Google Drive is mounted to access the dataset stored in the specified path.
   * **Directory Listing**: Lists the contents of the specified directory to verify access to the dataset.
2. **Model Setup**:
   * **Pre-Trained VGG16 Model**: The VGG16 model pre-trained on the ImageNet dataset is loaded without its top classification layer. This model serves as the feature extractor.
   * **Model Modification**: The output of the 'fc1' layer of the VGG16 model is set as the new output, focusing on feature extraction relevant to image retrieval.
3. **Feature Extraction Function**:
   * **extract\_features Function**: A function is defined to preprocess input images (resizing and preprocessing) and then use the modified VGG16 model to extract and normalize features.
4. **Image Retrieval Function**:
   * **retrieve\_similar\_images Function**: This function takes a query image, extracts its features, computes the Euclidean distance between this query image's features and the features of all images in the database, and then retrieves the images with the closest features.
5. **Main Processing**:
   * **Database and Query Image Paths**: Paths to the image database and query images are defined.
   * **Feature Extraction for Database Images**: Features for each image in the database are extracted and stored.
   * **Query Image Processing**: Each query image is processed to find and display the top N similar images from the database. For each query image, the similar images and their paths are printed.
6. **Display and Summary**:
   * **Image Display**: The query image and the retrieved similar images are displayed using IPython's display function.
   * **Summary Generation**: A summary is generated for each query, detailing the number of similar images found and confirming the completion of the retrieval process.
7. **Execution**: The main function is called to execute the program.

**Conclusion and Summary**: After processing all query images, a final summary is printed, providing an overview of the retrieval results for each query.