VLM(ZERO SHOT CLASSIFICATION) PROJECT REPORT

VLM Zero-Shot Classification

1. What is VLM (Vision-Language Model)?

A **Vision-Language Model (VLM)** is an AI system that processes **both images and text** to understand and generate meaningful associations between them. VLMs, like **CLIP (Contrastive Language-Image Pretraining)** by OpenAI, are trained on large datasets of images and their corresponding text descriptions.

---

2. What is Zero-Shot Classification?

Zero-shot learning means the model can classify images **without being explicitly trained** on those categories. Instead of learning from labeled examples, it relies on **textual descriptions** to infer the correct classification.

Example:

- If a model has never seen a "zebra" before, but knows the **text description of a zebra**, it can still classify an image of a zebra correctly.
- It does this by computing **similarities between the image and text labels**.

---

3. How VLM Enables Zero-Shot Classification?

In **CLIP-based zero-shot classification**, the process involves:

1. **Text Encoding** → Converts a list of class labels (e.g., "dog," "cat," "car") into embeddings.
2. **Image Encoding** → Converts an uploaded image into an embedding.
3. **Similarity Matching** → Measures the **cosine similarity** between image and text embeddings.
4. **Prediction** → The label with the highest similarity score is chosen as the predicted class.

---

4. Applications of VLM Zero-Shot Classification

- **Image Search Engines** – Identify images based on textual queries.
- **Content Moderation** – Detect inappropriate or harmful images without retraining.
- **Medical Diagnostics** – Classify unseen medical conditions using textual descriptions.
- **Autonomous Vehicles** – Recognize traffic signs, pedestrians, and obstacles.
- **Retail & E-commerce** – Find similar products by uploading an image.

Project Breakdown & Work Division-

Since me (**Ranjan) and my friend Monotosh**, are working together on the **Vision-Language Model (VLM) project**, here's a structured breakdown of tasks for efficient collaboration.

Part 1: Model Implementation & Data Processing (Ranjan's Contribution)

1. Environment Setup & Dependencies

- Install required libraries (`torch`, `torchvision`, `CLIP`, etc.).
- Ensure GPU/CPU compatibility for faster execution.

2. Data Loading & Preprocessing

- Load and preprocess images using CLIP's standard pipeline.
- Parse `labels.csv` and tokenize labels for CLIP's text encoder.
- Implement batch processing instead of manual image uploads.

3. Model Execution & Feature Extraction

- Load the CLIP model (`ViT-B/32`, with possible future upgrades).
- Encode image and text inputs into feature vectors.
- Compute cosine similarity scores to determine the best match.

Part 2: Evaluation, Optimization & UI Development (Manotosh's Contribution)-

4. Results & Visualization

- Display the selected image along with classification results.
- Implement bar charts to visualize similarity scores.
- Store results in a structured format (CSV/JSON for analysis).

5. Model Performance & Accuracy Improvements

- Experiment with different CLIP models (`ViT-L/14`, `ViT-H/14`).
- Fine-tune CLIP on domain-specific datasets if required.
- Implement confidence thresholds for better classification reliability.

6. Deployment & UI Development

- Develop a **web-based UI** using **Streamlit or Gradio** for interactive use.
- Allow users to upload images and see real-time classifications.
- Deploy the model as an **API** using Flask/FastAPI for scalability.

Collaboration & Integration

- I (Ranjan) ensures the model pipeline is robust and optimized for processing.
- Monotosh enhances usability, visualization, and deployment.
- Both of us collaborate on testing and debugging.

Flow of Executionof the project:-

Step 1: Install Required Libraries

```
!pip install torch torchvision
!pip install git+https://github.com/openai/CLIP.git
```

Ensures that required dependencies are installed.

Step 2: Import Required Libraries

```
import torch
import clip
import os
import random
import pandas as pd
from PIL import Image
from IPython.display import display
```

- Loads necessary libraries for deep learning (`torch`), image processing (`PIL`), data handling (`pandas`), and display functions.

Step 3: Load CLIP Model

```
device = "cuda" if torch.cuda.is_available() else "cpu"
model, preprocess = clip.load("ViT-B/32", device=device)
```

- Checks for GPU availability and loads the **CLIP ViT-B/32 model** for image processing.

Step 4: Load Labels from CSV

```
csv_path = "/content/labels.csv"
df = pd.read_csv(csv_path)
labels = df['label_column'].tolist()  # Assuming the CSV contains a column
named 'label_column'
```

- Reads a CSV file (`labels.csv`) containing labels for image classification.

Step 5:Upload and Select Images:-

```
print("Please upload images (JPG, JPEG, PNG, WEBP, GIF, BMP) for
classification...")
uploaded_files = files.upload()
```

- **Prompts user to upload images** via Colab.
- Accepts **various image formats**.

```
supported_formats = (".jpg", ".jpeg", ".png", ".webp", ".gif", ".bmp")
image_files = [file for file in uploaded_files.keys() if
file.lower().endswith(supported_formats)]
```

- Filters **valid image files** from uploaded files.

```
if not image_files:
    raise ValueError("No valid image files were uploaded. Please upload at
least one.")
```

- **Checks if images were uploaded**; raises an error if none found.

Step 6. Select a Random Image:-

```
random_image_path = random.choice(image_files)
print(f"\nSelected Random Image: {random_image_path}")
image = Image.open(random_image_path).convert("RGB")
image_processed = preprocess(image).unsqueeze(0).to(device)
```

- **Randomly selects an image** from uploaded files.
- Converts it to **RGB format**.
- **Preprocesses** the image (resizing, normalizing) for CLIP.

```
print(f"\nDisplaying Selected Image: {random_image_path}")
display(image)
```

- Displays the selected image in **Colab**.

Step 7. Perform Zero-Shot Classification:-

```
text_inputs = clip.tokenize(labels).to(device)
```

- Converts **label list into tokenized text** for CLIP processing.

```
with torch.no_grad():
    image_features = model.encode_image(image_processed)
    text_features = model.encode_text(text_inputs)
    similarity = torch.cosine_similarity(image_features, text_features,
dim=-1)
```

- **Encodes** image and text into feature vectors.
- **Computes cosine similarity** between image features and each label.

Step 8. Display Predicted Label:-

```
best_match = labels[similarity.argmax().item()]
print(f"\nPredicted Label: {best_match}")
```

- **Finds the label with the highest similarity score**.
- Prints the **best-matching category** for the image.

```
print("\nSimilarity Scores:")
for label, score in zip(labels, similarity.tolist()):
    print(f"{label}: {score:.4f}")
```

- **Prints similarity scores** for all labels.

Paths & approaches to complete the project –

1. Setup & Installation

- Installs necessary libraries: `torch`, `torchvision`, and CLIP from OpenAI.
- Ensures the correct environment (`cuda` or `cpu`).

2. Data Loading

- Loads a CSV file (`labels.csv`) that contains classification labels.(https://github.com/manotoshmaity/VLM-zero-shot-classification/raw/refs/heads/main/labels.csv)
- Prompts users to upload images manually in Google Colab.
- Filters valid image formats (JPG, PNG, WEBP, GIF, BMP, etc.).
- Selects a random image for classification.

Next Steps:-

- Automate image input by defining a fixed dataset instead of requiring user uploads.
- Consider a batch processing method instead of random single-image classification.

3. Preprocessing:-

- Uses CLIP's built-in preprocessing pipeline to prepare images.
- Tokenizes text labels for CLIP's text encoder.

4. Model Implementation:-

- Loads the **ViT-B/32** variant of CLIP.
- Encodes both image and text inputs into feature vectors.
- Uses cosine similarity to find the best matching label.

🔧 Next Steps

- Add support for multiple CLIP models (e.g., ViT-L/14 for better accuracy).
- Implement confidence thresholds for classification.
- Allow user-defined labels instead of relying on `labels.csv`.

Evaluation & Results

- Displays the uploaded image.
- Prints similarity scores for all labels.
- Outputs the **best predicted label**.

🔧 Next Steps

- Improve result visualization (e.g., bar charts for similarity scores).
- Save results automatically (JSON/CSV output for large-scale processing).

Future Scope for the Vision-Language Model (VLM) Project

As this project leverages **CLIP (Contrastive Language-Image Pretraining)** for zero-shot classification, there are several exciting directions to extend its capabilities.

---

1. Enhancing Model Performance

- **Use Larger CLIP Models**
  - Instead of `ViT-B/32`, experiment with `ViT-L/14` or `ViT-H/14` for better accuracy.
  - Fine-tune CLIP for specific datasets using techniques like **Adapter Layers** or **LoRA**.
- **Use Multi-Modal Fusion**
  - Combine CLIP with **LLMs (e.g., GPT, LLaVA, BLIP)** for richer image-text understanding.
  - Explore **Vision Transformers (Swin, ViT-G/14)** for better visual representation.

---

2. Expanding Dataset & Use Cases

- **Custom Training on Domain-Specific Data**
  - Fine-tune CLIP for specialized fields like **medical imaging, satellite imagery, or retail product classification**.
- **Multi-Language Support**
  - Extend text labels beyond English using **Multilingual CLIP** or **translation models**.
- **Real-Time Applications**
  - Deploy in **real-time applications** like **surveillance, autonomous vehicles, or augmented reality**.

---

3. Improving Classification & Output

- **Better Confidence Metrics**
  - Implement **top-k predictions** instead of only the best match.
  - Use **logits-based calibration** to improve classification reliability.
- **Interactive UI for Results**
  - Develop a **web dashboard** using **Streamlit or Gradio** for easy visualization.
  - Allow users to **upload images and compare multiple models** interactively.

4. Integration with Other AI Models

- **Text-to-Image Models**
  - Combine CLIP with **DALL·E, Stable Diffusion, or MidJourney** for **text-based image generation**.
- **Object Detection + CLIP**
  - Use **YOLO or DETR** to detect objects and then classify them using CLIP.
  - Useful for **scene understanding and autonomous systems**.
- **Integration with Speech & NLP**
  - Allow users to **describe an image using voice commands**.
  - Combine with **LLMs like GPT-4V** for contextual image descriptions.

---

5. Deployment & Scalability

- **Deploy as an API**
  - Host as a **Flask/FastAPI server** for cloud-based access.
  - Integrate with **mobile apps or robotics**.
- **Edge AI & IoT**
  - Optimize for **low-power devices (e.g., Jetson Nano, Raspberry Pi)**.
  - Use quantization & pruning for efficient inference.

Conclusion

This project has immense potential for real-world applications. The future scope includes **improving accuracy, expanding dataset coverage, integrating with other AI models, and deploying as an interactive tool or API**.