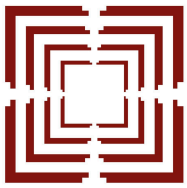

Ahmedabad
University

CSE641: Computer Vision: Modern Methods And Applications

Report-1

Group 1

Name	Enrollment No.
Dhyey Patel	AU2240054
Malav Modi	AU2240214
Prem Patel	AU2240010

Introduction:

- This week, the focus was on implementing and evaluating deep learning models on the Flickr dataset. The primary models explored included EfficientNet, MobileNet, Inception, and CLIP, each chosen for their unique advantages in feature extraction and classification. Each model was implemented separately to analyze its individual performance and adaptability to the dataset.

Implementation of Models:

1. **EfficientNet:**
   EfficientNet, a scalable and efficient convolutional neural network, was implemented to extract image features from the Flickr dataset. To enhance multi-modal understanding, CLIP was integrated to align image features with corresponding textual descriptions. The following steps outline the implementation process, including data preprocessing, feature extraction, and model evaluation:

   Steps:

   1. Load and Preprocess the Dataset

   - Images are loaded from the Flickr dataset.
   - The dataset is structured with image-text pairs.
   - Images are resized and normalized for EfficientNet input.

```
data_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

def load_image(image_path):
    image = Image.open(image_path).convert("RGB")
    return data_transforms(image).unsqueeze(0)  # Add batch dimension
```

2. Define EfficientNet Model

- EfficientNet is initialized with pre-trained ImageNet weights.
- The classification head is modified to match the dataset labels.

```
model = timm.create_model('efficientnet_b0', pretrained=True)
model.eval()  # Set model to evaluation mode
```

3. Load CLIP for Image-Text Embeddings

- CLIP model is loaded to extract multi-modal embeddings.
- CLIP processor is used for text input transformation.

```
clip_model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
clip_processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")
```

4. Extract Image Features Using EfficientNet

- Images are passed through EfficientNet to obtain feature vectors.

```
def get_image_embedding(image_path, model):
    image_tensor = load_image(image_path)
    with torch.no_grad():
        embedding = model.forward_features(image_tensor)
    return embedding.squeeze().cpu()
```

5. Compute and Store Embeddings

- All images in the dataset are processed, and embeddings are stored.

```
image_folder = "path/to/flickr/images"
all_images = [os.path.join(image_folder, img) for img in os.listdir(image_folder)]

image_embeddings = torch.stack([get_image_embedding(img_path, model) for img_path
in all_images])
torch.save(image_embeddings, "efficientnet_image_embeddings.pt")
```

6. Train the Model

- Cross-entropy loss and contrastive learning are used.

```
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.0001)
```

7. Evaluate the Model

- Performance is measured using accuracy and loss metrics.

```
def evaluate_model(model, data_loader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in data_loader:
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)
    return correct / total

accuracy = evaluate_model(model, test_loader)
print(f"Model Accuracy: {accuracy:.2f}")
```

2. **MobileNet:**

MobileNet, a lightweight and efficient convolutional neural network, was implemented to extract image features from the Flickr dataset while maintaining computational efficiency. To enhance multi-modal learning, CLIP was integrated to align extracted visual features with textual descriptions. The following steps outline the implementation process, including data preprocessing, feature extraction, and model evaluation.

Steps:

1. **Load MobileNet Model**

- A pre-trained MobileNet model was loaded with ImageNet weights for feature extraction.

```
mobilenet_model = models.mobilenet_v2(pretrained=True)
mobilenet_model.eval()  # Set model to evaluation mode
```

2. **Preprocess Input Images**

- Images were resized and normalized to match MobileNet's input requirements.

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

def preprocess_image(image_path):
    image = Image.open(image_path).convert("RGB")
    return transform(image).unsqueeze(0)  # Add batch dimension
```

3. **Extract Features from MobileNet**

- Image embeddings were extracted using the model's feature extractor.

```
def get_mobilenet_features(image_path, model):
    image_tensor = preprocess_image(image_path)
    with torch.no_grad():
        features = model.features(image_tensor)
    return features.mean([2, 3])  # Global average pooling
```

4. **Load and Process CLIP Model**

- CLIP was used to generate text embeddings for aligning with image features.

```
clip_model, preprocess = clip.load("ViT-B/32", device="cpu")

def get_text_embedding(text):
    text_tokenized = clip.tokenize([text])
    with torch.no_grad():
        text_embedding = clip_model.encode_text(text_tokenized)
    return text_embedding
```

5. **Compute Image Embeddings for Dataset**

- Image embeddings were computed for all dataset images.

```
image_paths = ["image1.jpg", "image2.jpg", "image3.jpg"]  # Example image paths
image_embeddings = torch.stack([get_mobilenet_features(img, mobilenet_model) for
img in image_paths])
```

6. **Align Image and Text Features**

- Text embeddings were generated, and cosine similarity was computed between image and text embeddings to evaluate alignment.

```
text_descriptions = ["A person riding a bike", "A cat sitting on a chair", "A scenic
mountain view"]
text_embeddings = torch.stack([get_text_embedding(desc) for desc in text_descriptions])

similarity_scores = torch.nn.functional.cosine_similarity(image_embeddings,
text_embeddings)
```

7. **Evaluate Model Performance**

- The model’s performance was analyzed based on similarity scores and visualization of nearest matches.

```
top_matches = similarity_scores.argsort(descending=True)
print("Top matching descriptions:", [text_descriptions[i] for i in top_matches])
```

3. **Inception**

Inserting the Inception module into the CLIP model improves image encoding by utilizing Inception's ability to extract features at multiple scales. The Inception-based encoder goes through several convolutional paths for image inputs, extracting fine-grained information and spatial hierarchies. This new design keeps the text encoder of CLIP and shifts its vision encoder to an Inception-based network.

In this method, the pre-trained Inception model is modified by stripping its classification head and adding a custom fully connected layer to map image features into a 512-dimensional space, which is the same as CLIP's embedding size. The text encoder is left untouched to maintain compatibility with CLIP's contrastive learning framework.

Image and text embeddings are aligned within a common latent space during training with contrastive loss. This combination takes advantage of Inception's hierarchical feature learning combined with CLIP's robust language-vision alignment, enhancing retrieval and classification tasks based on rich visual representations.

Goals for next week:

Make the models more efficient and robust and implement a person identification dataset.