IMPORTANT: Train the CNN Model First

Before running the main application, you MUST train the PyTorch CNN model:

Step 1: Prepare Training Data

Ensure you have the training data file:

```
data/CNN_Model_Train_Data_FIXED.csv
```

The CSV should contain columns:

- `image_path`: Path to product images
- `product`: Product category/class labels

Step 2: Train the CNN Model

bash

```
python train_pytorch_model.py
```

Training Process:

- The script will automatically create the `models/` directory
- Training typically takes 15-30 minutes depending on your hardware
- The model will be saved as `models/pytorch_product_cnn.pth`
- Training logs will show progress, loss, and validation accuracy

Expected Output:

```
Using device: cuda  # or cpu
Original dataset size: 1000. Filtered dataset size: 950.
Epoch 1/50 | Train Loss: 2.1234 | Val Loss: 1.8765 | Val Acc: 0.3456
...
Best model saved to models/pytorch_product_cnn.pth with validation accuracy: 0.8234
```

Step 3: Set Up Vector Database

bash

```
python -c "
from services.vector_database import VectorDatabase
import os
vdb = VectorDatabase(os.environ.get('PINECONE_API_KEY'))
vdb.create_index()
```

```
print('Vector database initialized!')
"
```

Step 4: Run the Flask Application

bash

```
python app.py
```

Expected Output:

```
Initializing services...
Vector database initialized successfully.
PyTorch CNN model loaded successfully. Classes: 11

Starting Flask server...
 * Running on http://0.0.0.0:5000
```

Step 5: Access the Application

Open your web browser and navigate to:

```
http://localhost:5000
```

Using the Application

1. Text Query Interface

- Enter natural language queries (e.g., "heart decoration", "red lamp")
- System uses semantic search to find relevant products
- Results displayed in a table format

2. Image Query Interface (OCR)

- Upload images containing handwritten or printed text
- System extracts text using OCR and searches for related products
- Supports both EasyOCR (handwriting) and Tesseract (printed text)

3. Product Image Upload (CNN)

- Upload product images for classification
- CNN model identifies the product category
- Displays confidence score and predicted class

Configuration

Model Configuration

Edit `train_pytorch_model.py` to modify:

python

```
BATCH_SIZE = 32        # Adjust based on GPU memory
EPOCHS = 50            # Number of training epochs
IMAGE_SIZE = (128, 128)  # Input image dimensions
LEARNING_RATE = 0.001    # Learning rate for optimizer
```

OCR Configuration

Edit `services/ocr_service.py` to modify OCR settings:

python

```
# Tesseract configuration
config='--psm 6'  # Page segmentation mode


# EasyOCR configuration
self.easyocr_reader = easyocr.Reader(['en'])  # Languages
```

Vector Database Configuration

Edit `services/vector_database.py`:

python

```
dimension = 384        # Vector dimension
metric = "cosine"      # Similarity metric
top_k = 5              # Number of results to return
```

Troubleshooting

Common Issues

1. "No module named 'torch'"

bash

```
pip install torch torchvision torchaudio
```

2. "Tesseract not found"

- Install Tesseract OCR system-wide
- Add Tesseract to your system PATH
- On Windows, you might need to specify the path:

python

```
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'
```

3. "Model file not found"

Make sure you've run the training script:

bash

```
python train_pytorch_model.py
```

4. "Vector database is not available"

- Check your Pinecone API key
- Ensure you've created the index
- Verify your internet connection

5. "CUDA out of memory"

Reduce batch size in training script:

python

```
BATCH_SIZE = 16  # or 8
```

6. "Failed to perform OCR query"

- Check image format (JPG, PNG supported)
- Ensure image contains readable text
- Try both handwritten and printed text modes

Debug Mode

Enable Flask debug mode for detailed error messages:

python

```
app.run(host="0.0.0.0", port=5000, debug=True)
```

Logging

Add logging to track issues:

python

```
import logging
logging.basicConfig(level=logging.DEBUG)
```

Performance Optimization

Model Performance

- Use GPU for training: Install CUDA-compatible PyTorch
- Increase batch size if you have more GPU memory
- Use data augmentation for better generalization

Application Performance

- Use Redis for caching frequent queries
- Implement batch processing for multiple images
- Optimize image preprocessing pipeline

Security Considerations

- Keep your Pinecone API key secure
- Validate file uploads (size, format)
- Sanitize user inputs
- Use HTTPS in production
- Implement rate limiting

Deployment

Local Development

bash

```
python app.py
```