Image Data Handling in Deep Learning

Handling image data properly is crucial for training deep learning models efficiently. This includes **loading**, **resizing**, and **normalizing** images.

---

1. Loading Images

Using OpenCV (cv2)

```
import cv2
import numpy as np

# Load image in color mode
image = cv2.imread("image.jpg", cv2.IMREAD_COLOR)

# Convert image to RGB format (OpenCV loads images in BGR format)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Display image shape
print("Image shape:", image_rgb.shape)  # (Height, Width, Channels)
```

◆ OpenCV loads images in **BGR** format, but most deep learning libraries (TensorFlow, PyTorch) expect **RGB** format.

◆ cv2.imread("image.jpg", cv2.IMREAD_GRAYSCALE) loads the image in grayscale.

Using PIL (Pillow)

```
from PIL import Image

# Load image
image = Image.open("image.jpg")

# Convert image to NumPy array
image_np = np.array(image)
```

```
# Display image shape
print("Image shape:", image_np.shape)  # (Height, Width, Channels)
```

- PIL loads images in **RGB** format by default.
- It supports multiple image formats like **JPEG, PNG, BMP, and TIFF**.

Using TensorFlow (tf.keras.preprocessing.image)

```
import tensorflow as tf


# Load image using TensorFlow
image = tf.keras.preprocessing.image.load_img("image.jpg")


# Convert image to NumPy array
image_np = tf.keras.preprocessing.image.img_to_array(image)


print("Image shape:", image_np.shape)  # (Height, Width, Channels)
```

- TensorFlow loads images in **RGB** format.
- The function tf.keras.preprocessing.image.img_to_array() converts an image into a NumPy array with **float32** values.

Using PyTorch (torchvision)

```
from torchvision import transforms
from PIL import Image


# Load image
image = Image.open("image.jpg")


# Convert image to PyTorch tensor
transform = transforms.ToTensor()
image_tensor = transform(image)
```

```
print("Tensor shape:", image_tensor.shape)  # (Channels, Height, Width)
```

- PyTorch **expects images in (C, H, W) format**, whereas NumPy and TensorFlow use (H, W, C).
- transforms.ToTensor() automatically **normalizes pixel values to [0,1]**.

---

2. Resizing Images

Deep learning models expect a fixed input size. Resizing is necessary to maintain uniformity.

Using OpenCV

```
resized_image = cv2.resize(image_rgb, (224, 224))  # Resize to 224x224

print("Resized shape:", resized_image.shape)
```

Using PIL

```
resized_image = image.resize((224, 224))
```

Using TensorFlow

```
resized_image = tf.image.resize(image_np, (224, 224))
```

Using PyTorch

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])

image_resized = transform(image)
```

---

3. Normalizing Images

Normalization scales pixel values to a specific range, improving model convergence.

Methods:

- **Rescaling to [0,1]:** pixel_value = pixel_value / 255.0
- **Standardization (Z-score normalization):** pixel_value = (pixel_value - mean) / std
- **Mean subtraction (for ImageNet models):**
  - **Mean:** [0.485, 0.456, 0.406]
  - **Standard Deviation:** [0.229, 0.224, 0.225]

Using OpenCV

```
image_normalized = image_rgb / 255.0  # Normalize to [0,1]
```

Using TensorFlow

```
image_normalized = tf.keras.applications.vgg16.preprocess_input(image_np)
```

- This function applies ImageNet-specific normalization.

Using PyTorch

```
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])
])

image_normalized = transform(image)
```

- The Normalize transform applies **(pixel - mean) / std** channel-wise.

Final Notes

Library	**Format**	**Normalization**
OpenCV (cv2)	BGR (convert to RGB)	Divide by 255
PIL (Image)	RGB	Convert to NumPy and normalize
TensorFlow (tf.image)	RGB	preprocess_input for models
PyTorch (torchvision)	(C, H, W)	transforms.Normalize()