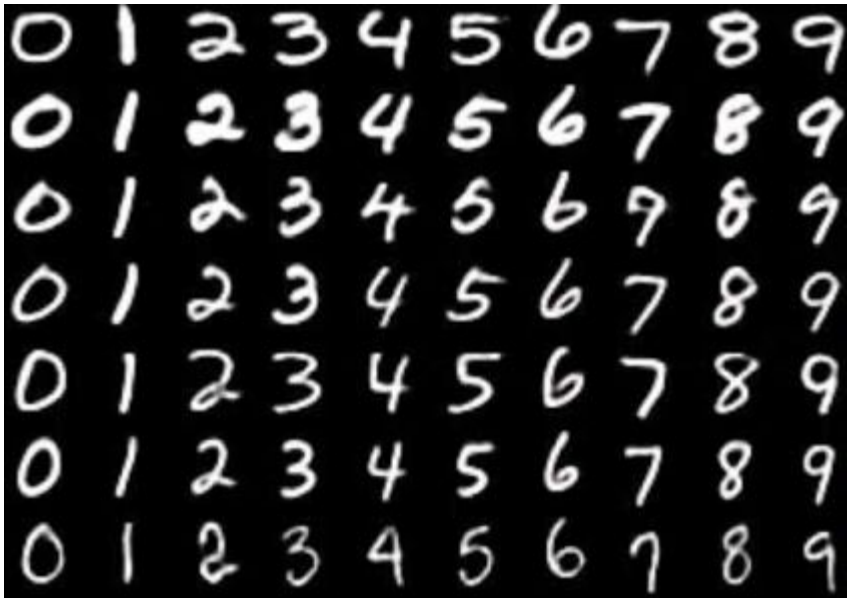
DeepLearning.AI

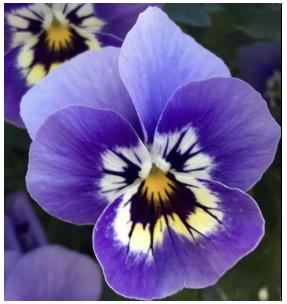
Introduction to Data Pipelines

Data Management in PyTorch

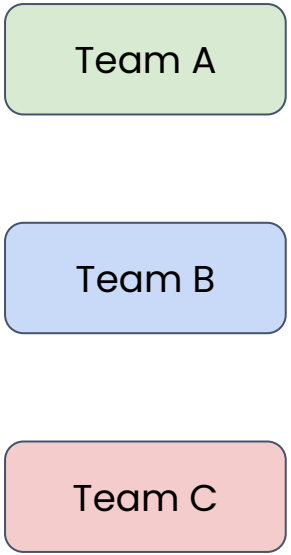
MNIST



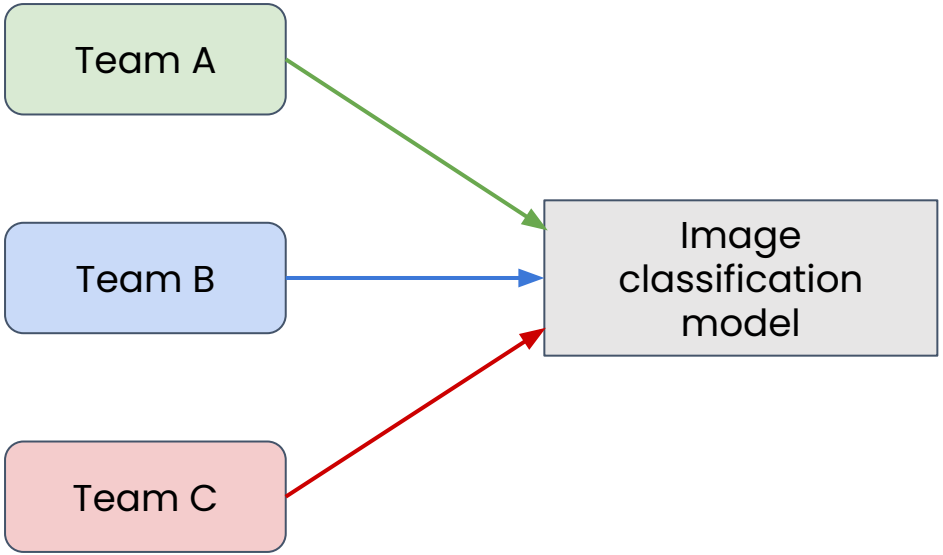
The Botanical Garden



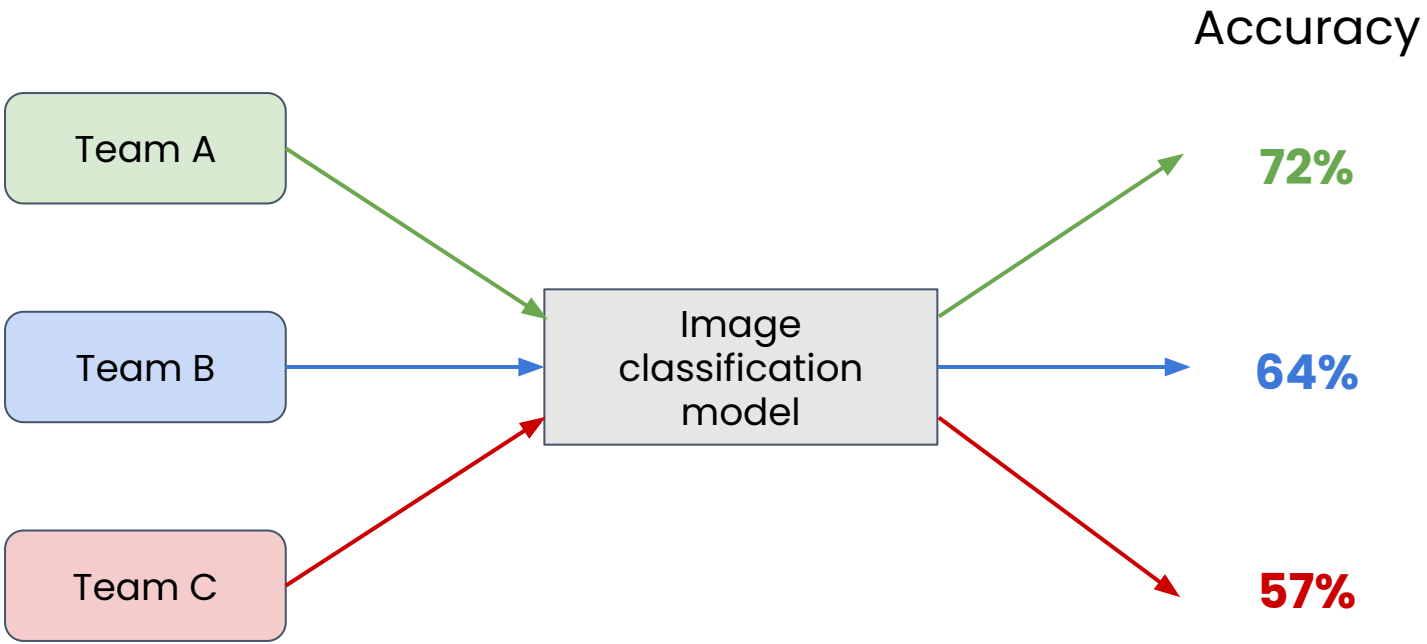
The Flowers app



The Flowers app



The Flowers app



Oxford 102 Flowers Dataset

image_00001.jpg

image_00002.jpg

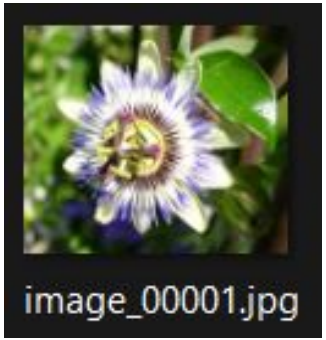
image_00003.jpg

image_00004.jpg

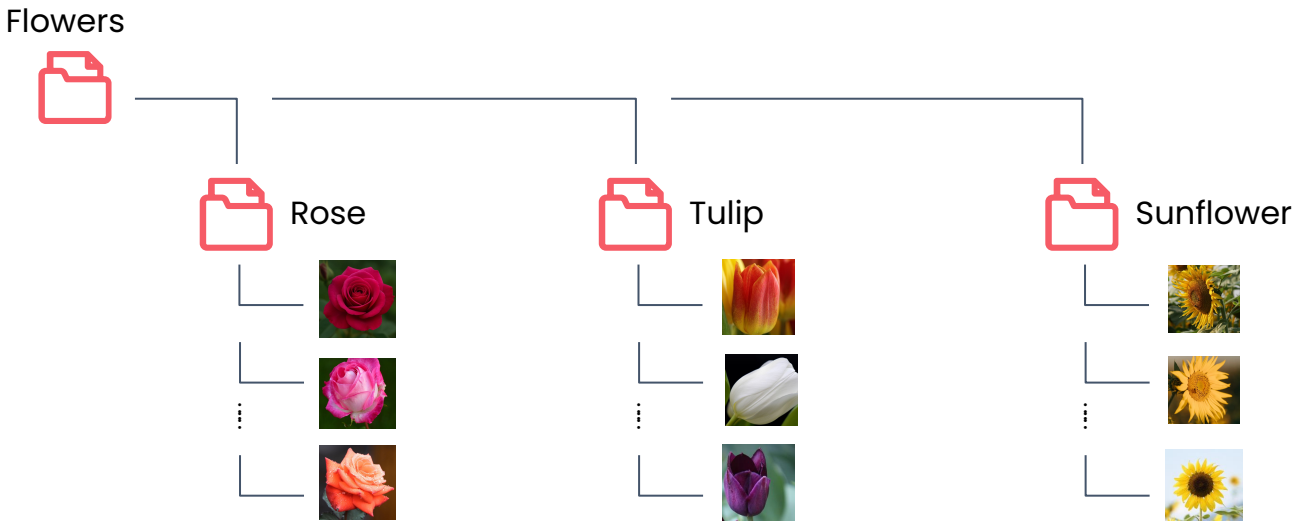
image_00005.jpg

image_00006.jpg

. . .



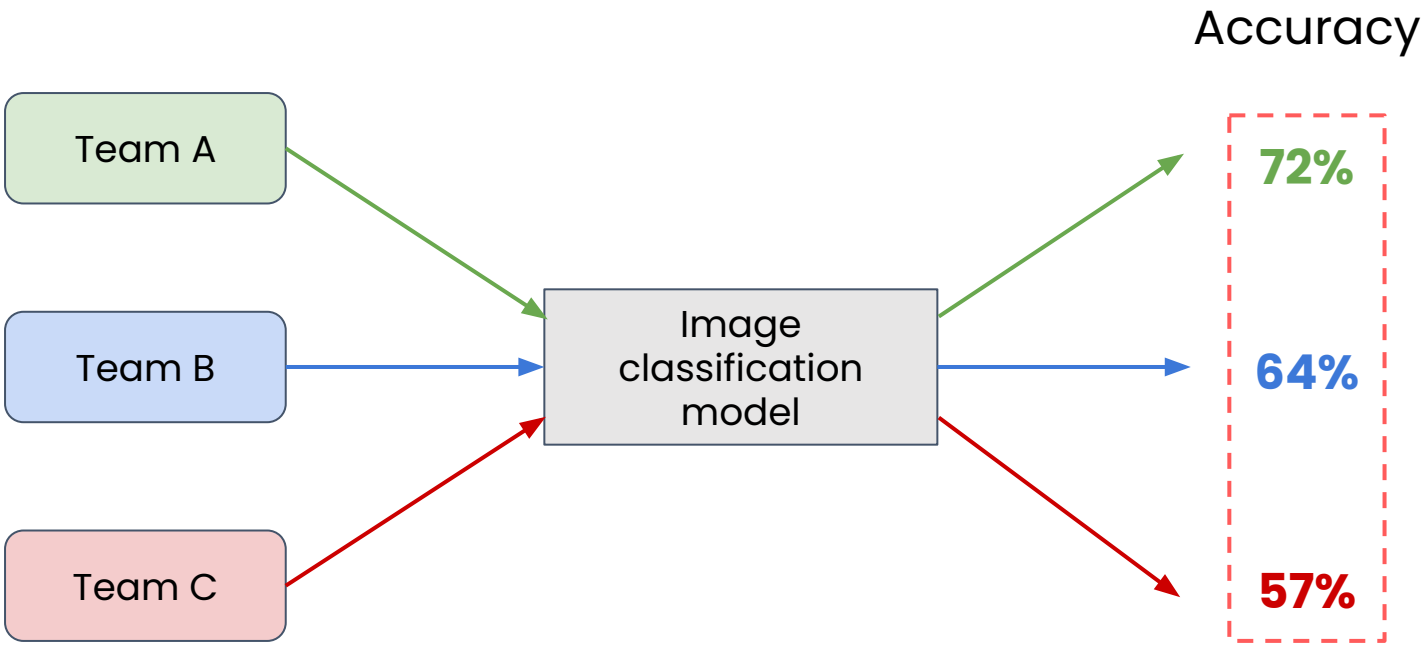
Structured Dataset



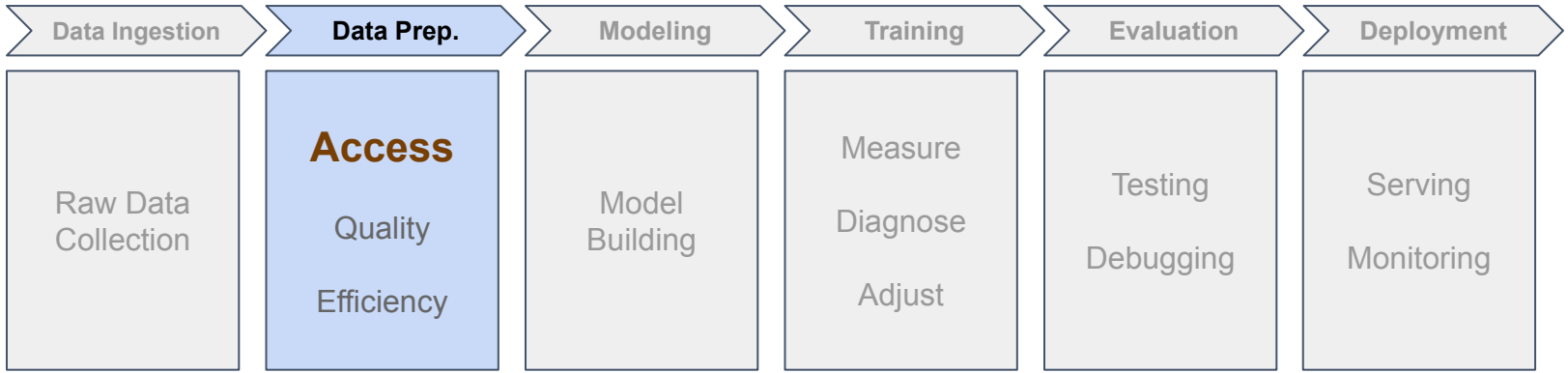
Oxford 102 Flowers Dataset

imagelabels.mat

The Flowers app

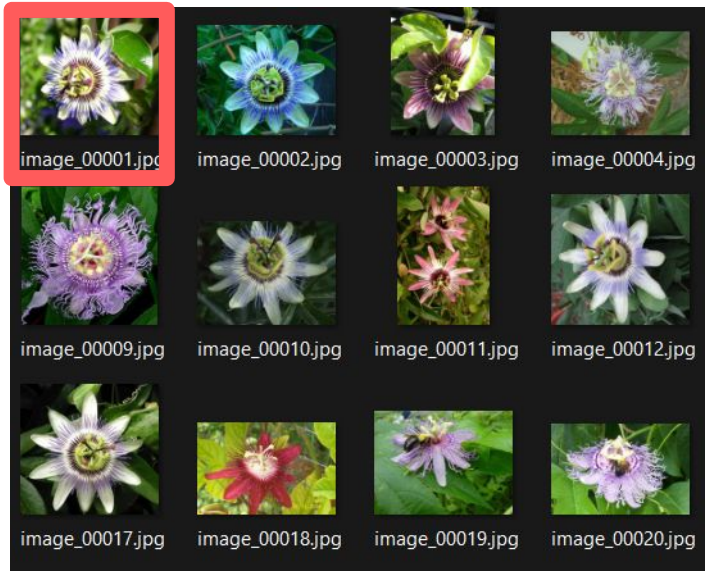


The ML Pipeline



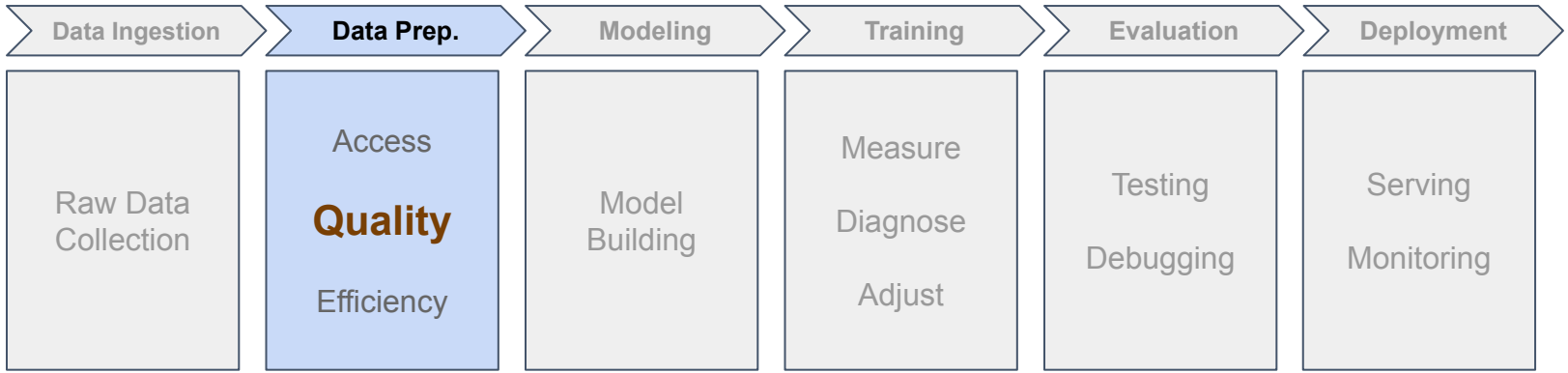
Access

Data Ingestion | **Data Prep.** | Modeling | Training | Evaluation | Deployment



imagelabels.mat

The ML Pipeline



Oxford 102 Flowers Dataset

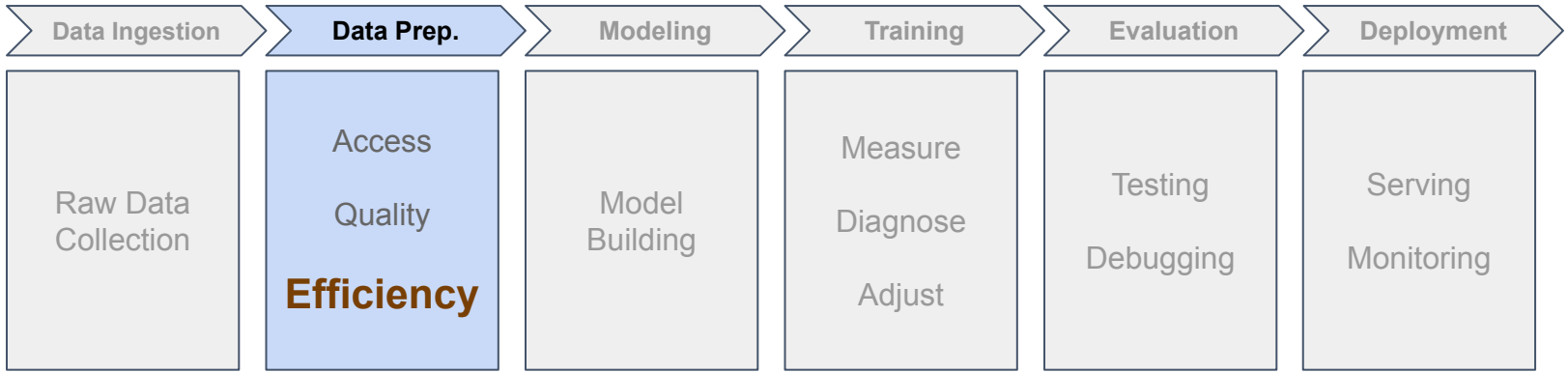
Data Ingestion | **Data Prep.** | Modeling | Training | Evaluation | Deployment

Correct Size

Correct Format

Correct Structure

The ML Pipeline



Data Preparation

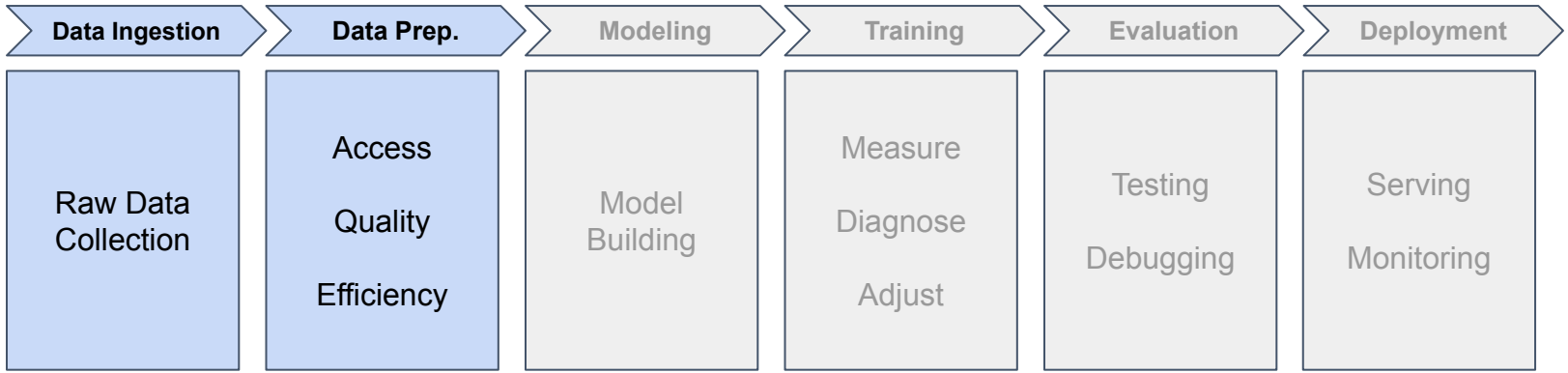
https://www.robots.ox.ac.uk/~vgg/data/flowers/102/

- **Access Problems**
- **Quality Problems**
- **Efficiency Problems**

Data Pipelines


```
from torch.utils.data import Dataset

from torch.utils.data import DataLoader
```

The Data Pipeline

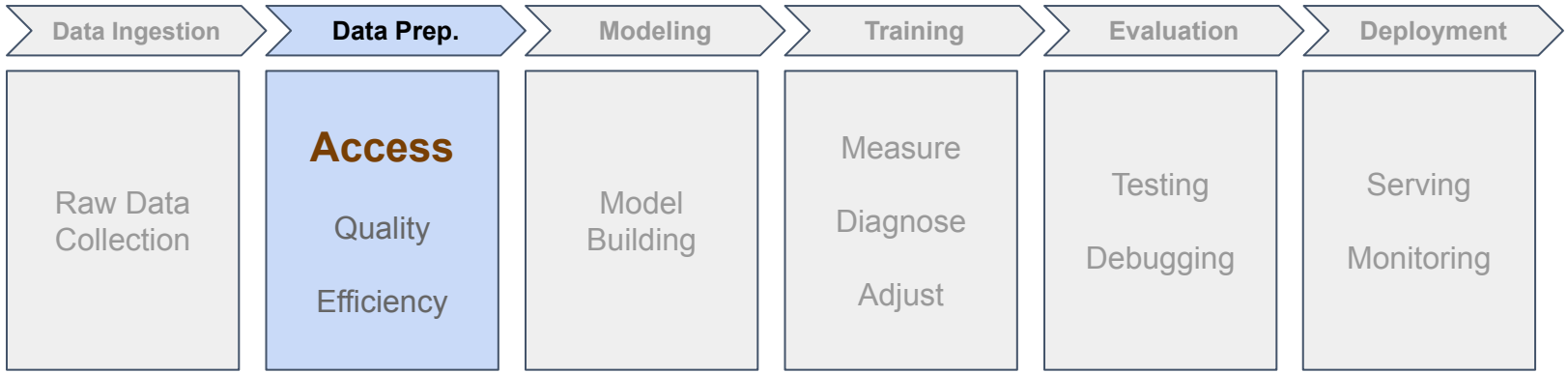


DeepLearning.AI

Data Access

Data Management in PyTorch

The ML Pipeline



Oxford 102 Flowers Dataset

https://www.robots.ox.ac.uk/~vgg/data/flowers/102/

- 102 classes
- 40 to 258 image per class
- Large scale, pose and light variations

Download Function

```
def download_dataset():
    """Download the Oxford 102 Flowers dataset"""
    # URLs for the dataset
    image_url = "https://www.robots.ox.ac.uk/~vgg/data/flowers/102/102flowers.tgz"
    labels_url = "https://www.robots.ox.ac.uk/~vgg/data/flowers/102/imagelabels.mat"

    # Create directory
    os.makedirs("flower_data", exist_ok=True)
```

Oxford 102 Flowers Dataset

Oxford 102 Flowers Dataset

- 8189 images

Oxford 102 Flowers Dataset

- 8189 images
- Named -> `image_00001.jpg` to `image_08189.jpg`

Oxford 102 Flowers Dataset

- 8189 images
- Named -> `image_00001.jpg` to `image_08189.jpg`

Labels stored in `imagelabels.mat` file

Oxford 102 Flowers Dataset

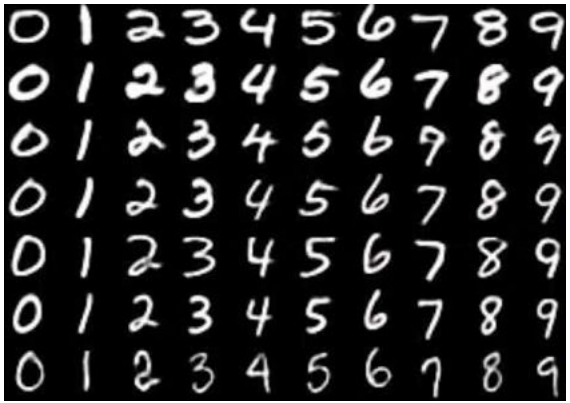
- 8189 images
- Named -> `image_00001.jpg` to `image_08189.jpg`

Labels stored in `imagelabels.mat` file

Different sources

Loading Data

```
dataset = datasets.MNIST(root='./data', train=True, download=True)

dataloader = DataLoader(dataset, batch_size=32)
```



Oxford 102 Flowers Dataset

- 8189 images
- Named -> `image_00001.jpg` to `image_08189.jpg`

Labels stored in `imagelabels.mat` file

The Three Methods

```
class OxfordFlowersDataset(Dataset):
    # Setup: where to find images and labels
    def __init__(self):

    # How many total samples
    def __len__(self):

    # How to get image and label number 'idx'
    def __getitem__(self, idx):
```

The Three Methods

```
class OxfordFlowersDataset(Dataset):
    # Setup: where to find images and labels
    def __init__(self):

    # How many total samples
    def __len__(self):

    # How to get image and label number 'idx'
    def __getitem__(self, idx):
```

The Three Methods

```
class OxfordFlowersDataset(Dataset):
    # Setup: where to find images and labels
    def __init__(self):

    # How many total samples
    def __len__(self):

    # How to get image and label number 'idx'
    def __getitem__(self, idx):
```

The Three Methods

```
class OxfordFlowersDataset(Dataset):
    # Setup: where to find images and labels
    def __init__(self):

    # How many total samples
    def __len__(self):

    # How to get image and label number 'idx'
    def __getitem__(self, idx):
```

The Three Methods

```
class OxfordFlowersDataset(Dataset):
    # Setup: where to find images and labels
    def __init__(self):

    # How many total samples
    def __len__(self):

    # How to get image and label number 'idx'
    def __getitem__(self, idx):
```

The __init__ Method

```
class OxfordFlowersDataset(Dataset):
    def __init__(self, root_dir):
        self.root_dir = root_dir
        self.img_dir = os.path.join(root_dir, 'jpg')

        # Load Matlab labels
        labels_mat = scipy.io.loadmat(os.path.join(root_dir, 'imagelabels.mat'))

        self.labels = labels_mat['labels'][0]
```

The `__init__` Method

```
class OxfordFlowersDataset(Dataset):
    def __init__(self, root_dir):
        self.root_dir = root_dir
        self.img_dir = os.path.join(root_dir, 'jpg')

        # Load Matlab labels
        labels_mat = scipy.io.loadmat(os.path.join(root_dir, 'imagelabels.mat'))

        self.labels = labels_mat['labels'][0]
```

The `__init__` Method

```
class OxfordFlowersDataset(Dataset):
    def __init__(self, root_dir):
        ...
        self.labels = labels_mat['labels'][0]
```

The `__init__` Method

```
class OxfordFlowersDataset(Dataset):
    def __init__(self, root_dir):
        ...

        self.labels = labels_mat['labels'][0]
```

```
print(len(self.labels)) # Number of image labels
```

Output:

```
C:>
8189
```

The __init__ Method

```
class OxfordFlowersDataset(Dataset):
    def __init__(self, root_dir):
        ...

        self.labels = labels_mat['labels'][0]
```

```
print(len(self.labels)) # Number of image labels
print(self.labels[:10]) # First 10 image labels
```

Output:

```
C:>
8189
```

The __init__ Method

```
class OxfordFlowersDataset(Dataset):
    def __init__(self, root_dir):
        ...

        self.labels = labels_mat['labels'][0]
```

```
print(len(self.labels)) # Number of image labels
print(self.labels[:10]) # First 10 image labels
```

Output:

```
C:>
8189
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

The __init__ Method

```
# MIN LABEL
print(f"Min label: {self.labels.min()}")

# MAX LABEL
print(f"Max label: {self.labels.max()}")
```

Output:

```
C:>
1
102
```

The `__init__` Method

```
# MIN LABEL
print(f"Min label: {self.labels.min()}")

# MAX LABEL
print(f"Max label: {self.labels.max()}")
```

Output:

```
C:>
1
102
```

Not zero indexed!

```
C:>
0
101
```

The `__init__` Method

```
class OxfordFlowersDataset(Dataset):
    def __init__(self, root_dir):
        ...

        self.labels = labels_mat['labels'][0]
```

The `__init__` Method

```
class OxfordFlowersDataset(Dataset):
    def __init__(self, root_dir):
        ...

        self.labels = labels_mat['labels'][0] - 1
```

The `__init__` Method

```
class OxfordFlowersDataset(Dataset):
    def __init__(self, root_dir):
        self.root_dir = root_dir
        self.img_dir = os.path.join(root_dir, 'jpg')

        # Load Matlab labels
        labels_mat = scipy.io.loadmat(os.path.join(root_dir, 'imagelabels.mat'))

        self.labels = labels_mat['labels'][0] - 1
```

What **NOT** to do

```
class OxfordFlowersDataset(Dataset):
    self.all_images = []
    for i in range(8189):
        img = Image.open(f'image_{i:05d}.jpg')
        self.all_images.append(img)
```

4GB of RAM - gone instantly!

Lazy Loading

```
class OxfordFlowersDataset(Dataset):
    self.all_images = []
    for i in range(8189):
        img = Image.open(f'image_{i:05d}.jpg')
        self.all_images.append(img)
```

```
class OxfordFlowersDataset(Dataset):
    def __init__(self, root_dir):
        self.root_dir = root_dir
        self.img_dir = os.path.join(root_dir, 'jpg')

        # Load Matlab labels
        labels_mat = scipy.io.loadmat(os.path.join(root_dir, 'imagelabels.mat'))

        self.labels = labels_mat['labels'][0] - 1
```

Path

Labels

The `__len__` Method

```
def __len__(self):
    return len(self.labels)  # 8,189 samples
```

The `__len__` Method

```
def __len__(self):
    return len(self.labels)  # 8,189 samples
```

The __getitem__ Method

```
from PIL import Image

def __getitem__(self, idx):
    # Build the image filename
    img_name = f'image_{idx:05d}.jpg'
    img_path = os.path.join(self.img_dir, img_name)

    # Load the image
    image = Image.open(img_path)
    label = self.labels[idx]

    return image, label
```

The `__getitem__` Method

```
from PIL import Image

def __getitem__(self, idx):
    # Build the image filename
    img_name = f'image_{idx:05d}.jpg'
    img_path = os.path.join(self.img_dir, img_name)

    # Load the image
    image = Image.open(img_path)
    label = self.labels[idx]

    return image, label
```

The `__getitem__` Method

```
from PIL import Image

def __getitem__(self, idx):
    # Build the image filename
    img_name = f'image_{idx:05d}.jpg'
    img_path = os.path.join(self.img_dir, img_name)

    # Load the image
    image = Image.open(img_path)
    label = self.labels[idx]

    return image, label
```

The `__getitem__` Method

```
from PIL import Image

def __getitem__(self, idx):
    # Build the image filename
    img_name = f'image_{idx:05d}.jpg'
    img_path = os.path.join(self.img_dir, img_name)

    # Load the image
    image = Image.open(img_path)
    label = self.labels[idx]

    return image, label
```

The `__getitem__` Method

```
from PIL import Image

def __getitem__(self, idx):
    # Build the image filename
    img_name = f'image_{idx:05d}.jpg'
    img_path = os.path.join(self.img_dir, img_name)

    # Load the image
    image = Image.open(img_path)
    label = self.labels[idx]

    return image, label
```

Testing The Code

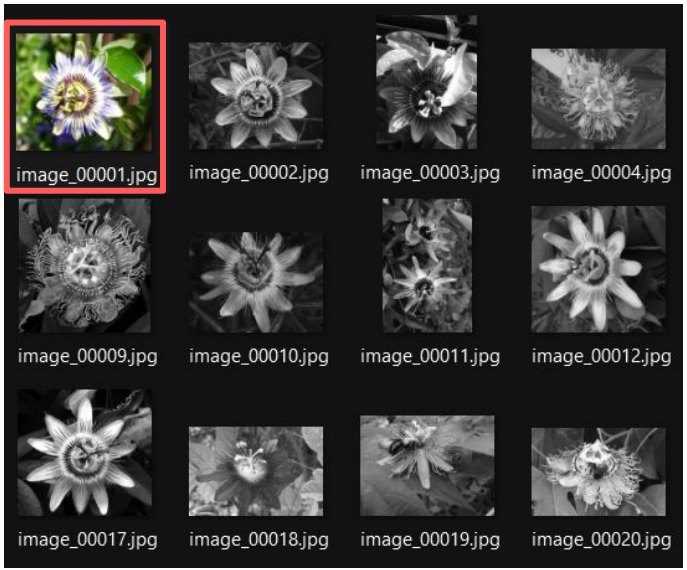
```
# Create the dataset
dataset = OxfordFlowersDataset('./flower_data')
print(f"Total samples: {len(dataset)}")  # Shows: 8189
# Try loading some images
img, label = dataset[0]
print(f"First image: {img.size}, Label: {label}")  # Success!
```

Output:

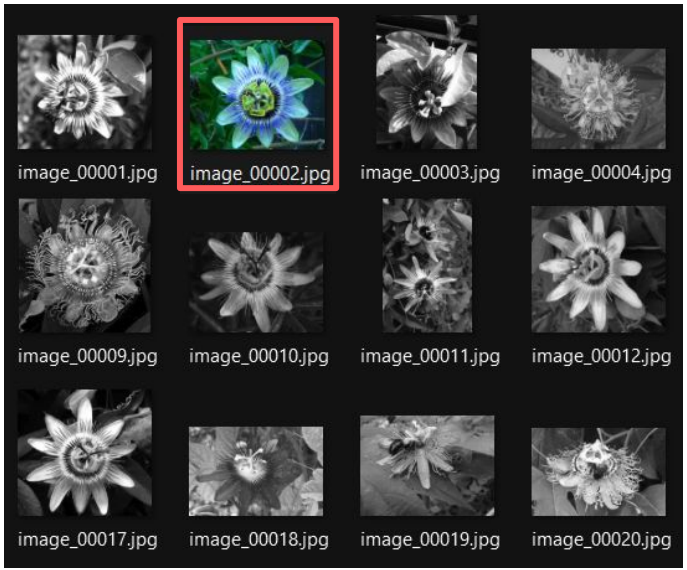
```
C:>
Total samples: 8189
First image: (591, 500), Label: 76
```

Oxford 102 Flowers Dataset

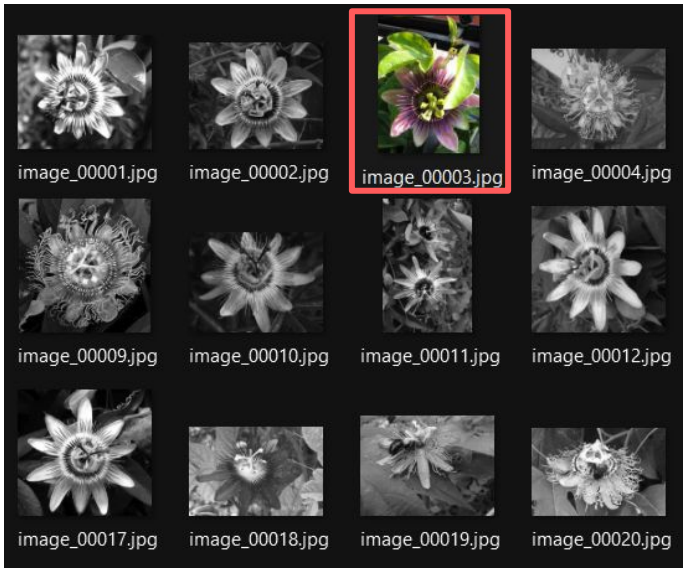
Oxford 102 Flowers Dataset



Oxford 102 Flowers Dataset



Oxford 102 Flowers Dataset



Oxford 102 Flowers Dataset

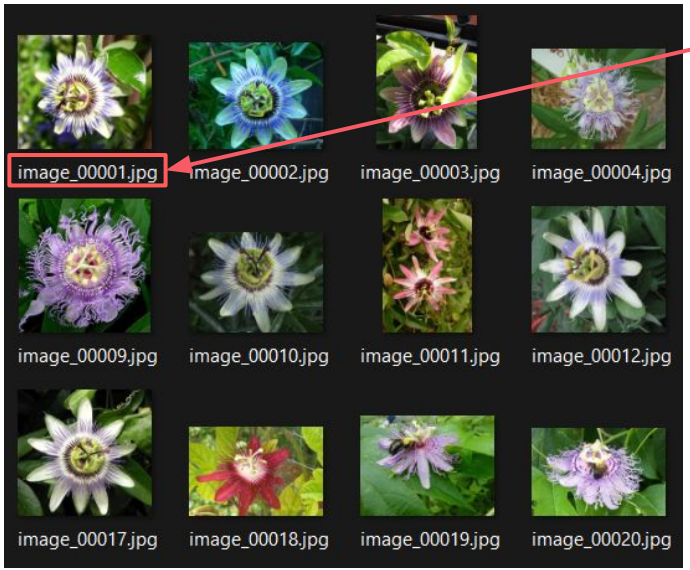


Image names start at number 1, not 0.

Fixing The `__getitem__` Method

```
from PIL import Image

def __getitem__(self, idx):
    # Build the filename
    img_name = f'image_{idx+1:05d}.jpg'  # Add 1 to filename
    img_path = os.path.join(self.img_dir, img_name)

    . . .
```

Output:

```
C:>
Total samples: 8189
First image: (591, 500), Label: 76
```

Loading Any Data

```
class SomeDataset(Dataset):
    # Setup: where to find images and labels
    def __init__(self):

    # How many total samples
    def __len__(self):

    # How to get image and label number 'idx'
    def __getitem__(self, idx):
```

Oxford 102 Flowers Dataset

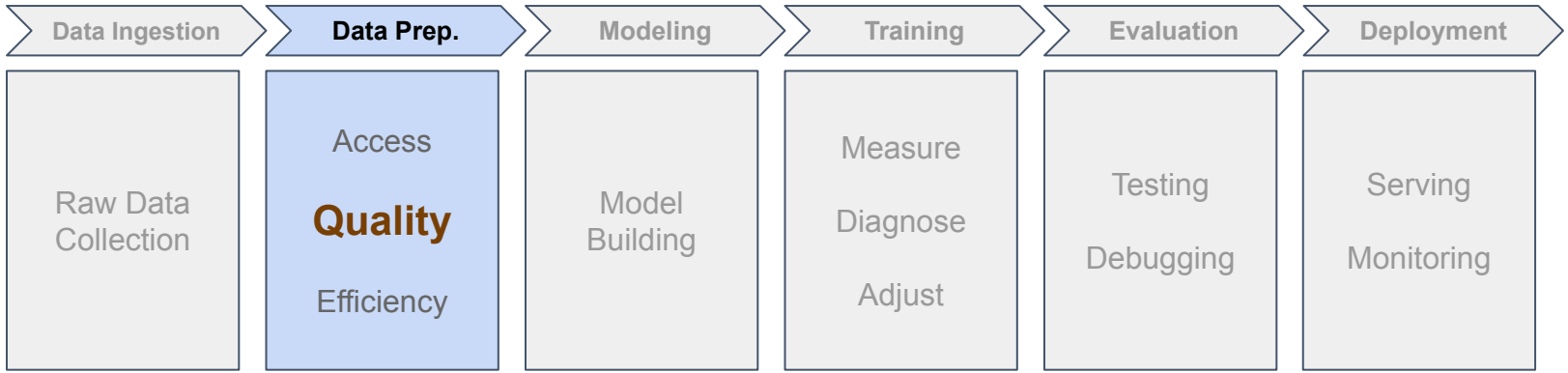
DeepLearning.AI

Transform Pipelines

Data Management in PyTorch

Oxford 102 Flowers Dataset

The ML Pipeline



Batches

```
# Create the dataset
dataset = OxfordFlowersDataset('./flower_data')
dataloader = DataLoader(dataset, batch_size=4, shuffle=True)

# Try to get a batch
for images, labels in dataloader:
    print(f"Batch shape: {images.shape}")
    break
```

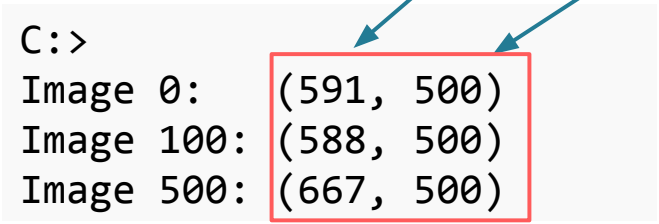
Output:

```
C:>
RuntimeError: stack expects each tensor to be equal size
```

Different Sized Images

Batch Format

[batch_size, channels, height, width]



Runtime Error

Checking Images Sizes

```
# Check a few images
for i in [0, 100, 500]:
    img, _ = dataset[i]
    print(f"Image {i}: {img.size}")
```

Output:

```
C:>
Image 0: (591, 500)
Image 100: (588, 500)
Image 500: (667, 500)
```

Checking Images Sizes

```python
# Check a few images
for i in [0, 100, 500]:
    img, _ = dataset[i]
    print(f"Image {i}: {img.size}")
```

Output:

```
C:>
Image 0: (591, 500)
Image 100: (588, 500)
Image 500: (667, 500)
```

All different sizes!

Checking Data Types

```
# Check a few images
for i in [0, 100, 500]:
    img, _ = dataset[i]
    print(f"Type: {type(img)}")
```

Output:

```
C:>
Type: <class 'PIL.JpegImagePlugin.JpegImageFile'>
Type: <class 'PIL.JpegImagePlugin.JpegImageFile'>
Type: <class 'PIL.JpegImagePlugin.JpegImageFile'>
```

Checking Data Types

```
# Check a few images
for i in [0, 100, 500]:
    img, _ = dataset[i]
    print(f"Type: {type(img)}")
```

Output:

```
C:>
Type: <class 'PIL.JpegImagePlugin.JpegImageFile'>
Type: <class 'PIL.JpegImagePlugin.JpegImageFile'>
Type: <class 'PIL.JpegImagePlugin.JpegImageFile'>
```

Changing Image Sizes

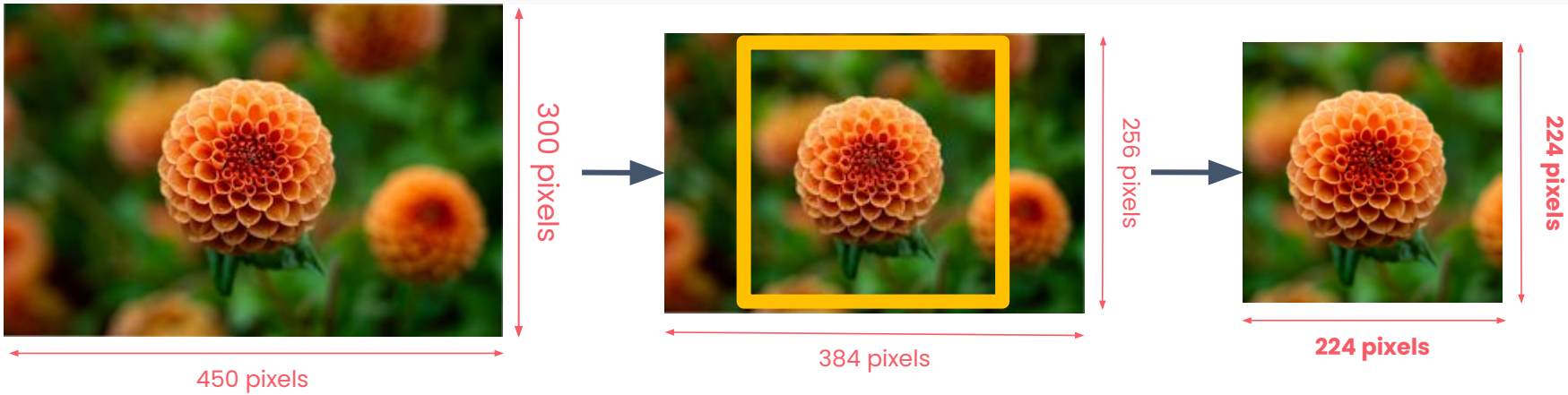
```
# First attempt - resize everything to 224x224:
transform = transforms.Resize((224, 224))
```

Changing Image Sizes

```
# Second attempt - resize the shorter edge to 256, then crop a square from the center:
transform = transforms.Compose([
    transforms.Resize(256),       # Resize shorter edge to 256
    transforms.CenterCrop(224),  # Extract 224x224 center square
])
```

Changing Image Sizes

```
# Second attempt - resize the shorter edge to 256, then crop a square from the center:
transform = transforms.Compose([
    transforms.Resize(256),       # Resize shorter edge to 256
    transforms.CenterCrop(224),   # Extract 224x224 center square
])
```



Changing Image Sizes

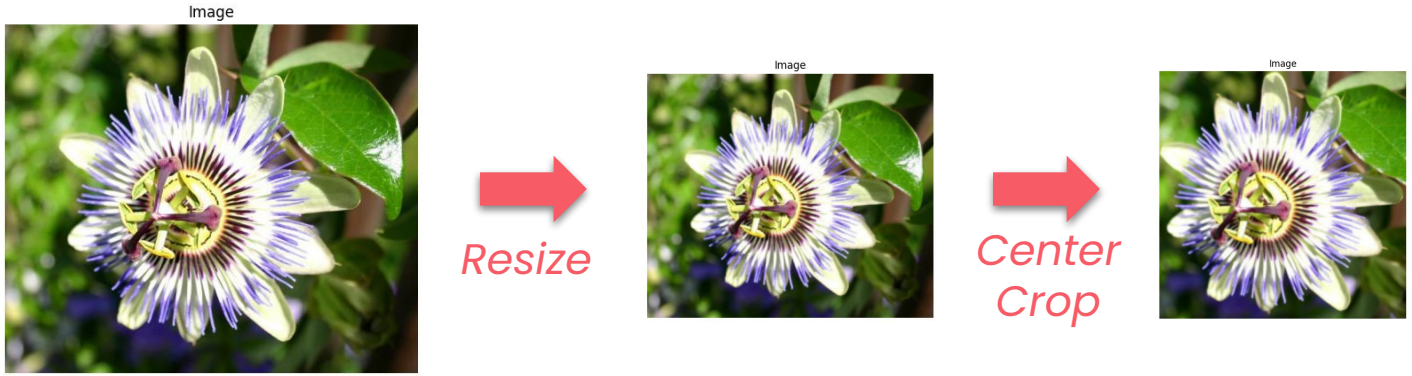
```
# Second attempt - resize the shorter edge to 256, then crop a square from the center:
transform = transforms.Compose([
    transforms.Resize(256),      # Resize shorter edge to 256
    transforms.CenterCrop(224),  # Extract 224x224 center square
])
```

Changing Image Sizes

```
# Second attempt – resize the shorter edge to 256, then crop a square from the center:
transform = transforms.Compose([
    transforms.Resize(256),      # Resize shorter edge to 256
    transforms.CenterCrop(224),  # Extract 224x224 center square
])
```

Changing Image Sizes

```
img, _ = dataset[0]
resized = transforms.Resize(256)(img)
print(f"After resize: {resized.size}")  # (302, 256) - keeps aspect ratio!
cropped = transforms.CenterCrop(224)(resized)
print(f"After crop: {cropped.size}")    # (224, 224) - perfect square
```



To Tensor

```
print(f"Image size: PIL {img.size}")
```

```
C:>
Image Size: PIL (224, 224)
```

```
img_tensor = transforms.ToTensor()(img)

print(f"Tensor Size: {tensor.shape}")
```

```
C:>
Tensor Size: torch.Size([3, 224, 224])
```

```
# red channel, top 3x3 values
print(img_tensor[0, :3, :3])
```

```
C:>
tensor([[0.3569, 0.4157, 0.5020],
        [0.3922, 0.4784, 0.5490],
        [0.3725, 0.4314, 0.5176]])
```

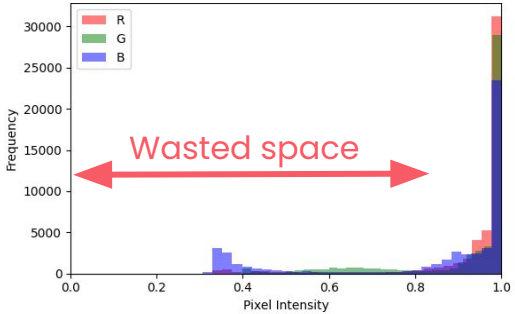
Normalization

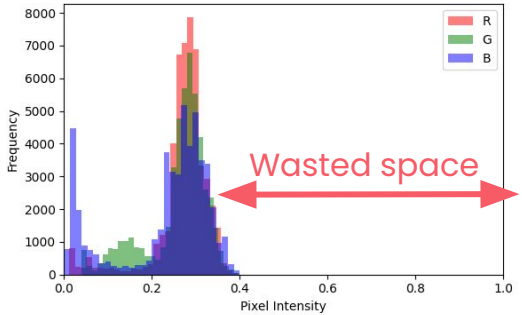
```
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

Normalization

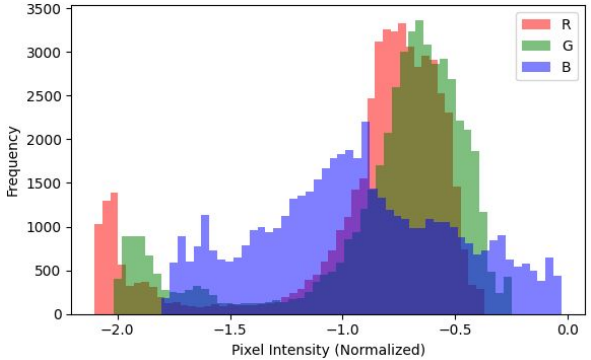
```
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

Normalization





Normalized Distribution




```
transforms.Normalize(mean=[0.485, 0.456, 0.406],
                     std=[0.229, 0.224, 0.225])
```

The Tensor Bridge

```
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize( . . .)
])
```

The Tensor Bridge

```
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize( . . .)
])
```

Images

Tensor Bridge

Tensors

The Tensor Bridge

```
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize( . . .)
])
```

Images

Tensor Bridge

The Tensor Bridge

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.Normalize( . . .)
])
```

Tensor Bridge

Tensors

The Tensor Bridge

```
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize( . . .)
])
```

Images

Tensor Bridge

Tensors

The Tensor Bridge

```
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize( . . .)
])
```

Images

Tensor Bridge

Tensors

The Tensor Bridge

```
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize( . . .)
])
```

Images

Tensor Bridge

Tensors

Transform Pipeline

```python
class OxfordFlowersDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        # ... previous code ...
        self.transform = transform

    def __getitem__(self, idx):
        # ... previous code ...
        if self.transform:
            image = self.transform(image)
        return image, label

# Create dataset with transforms
dataset = OxfordFlowersDataset('./flower_data', transform=transform)
```

```python
# Complete pipeline:
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

Batching

```
# show_tensor_histogram(img)
dataloader = DataLoader(dataset, batch_size=4, shuffle=True)

for images, labels in dataloader:
    print(f"Success! Batch shape: {images.shape}")
    break
```

Batching

```
# show_tensor_histogram(img)
dataloader = DataLoader(dataset, batch_size=4, shuffle=True)

for images, labels in dataloader:
    print(f"Success! Batch shape: {images.shape}")
    break
```

Output:

```
C:>
Success! Batch shape: torch.Size([4, 3, 224, 224])
```

Batching

```
# show_tensor_histogram(img)
dataloader = DataLoader(dataset, batch_size=4, shuffle=True)

for images, labels in dataloader:
    print(f"Success! Batch shape: {images.shape}")
    break
```

Output:

```
C:>
Success! Batch shape: torch.Size([4, 3, 224, 224])
```

Quick Debugging Tip

```
# Quick sanity check:
img, label = dataset[0]
print(f"Shape: {img.shape}")
print(f"Type: {img.dtype}")
print(f"Range: [{img.min():.1f}, {img.max():.1f}]")
```

Output:

```
C:>
Shape: torch.Size([3, 224, 224])
Type: torch.float32
Range: [-2.1, 2.6]
```

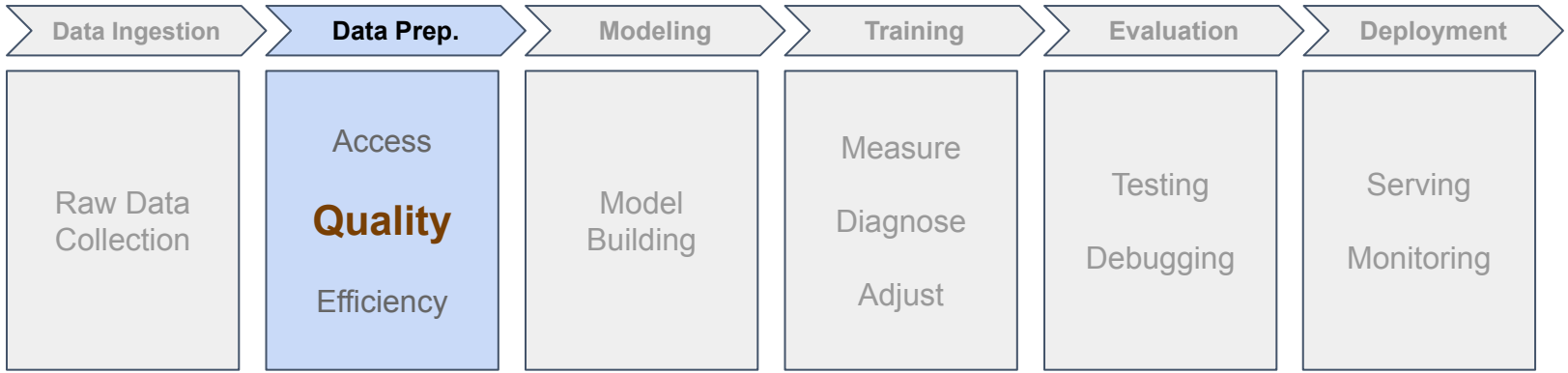
Quick Debugging Tip

```
raw_dataset = OxfordFlowersDataset('./flower_data', transform=None)
raw_img, label = raw_dataset[0]

img = transforms.Resize(256)(img)
print(f"After resize: {img.size}")

img = transforms.CenterCrop(224)(img)
print(f"After crop: {img.size}")

img = transforms.ToTensor()(img)
print(f"After ToTensor: {img.shape}, range [{img.min():.1f}, {img.max():.1f}]")
```

Output:

```
C:>
After resize: (302, 256)
After crop: (224, 224)
After ToTensor: torch.Size([3, 224, 224]), range [0.0, 1.0]
```

The ML Pipeline



Transform Pipeline

```
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),          # The tensor bridge
    transforms.Normalize(...)
])
```

DeepLearning.AI

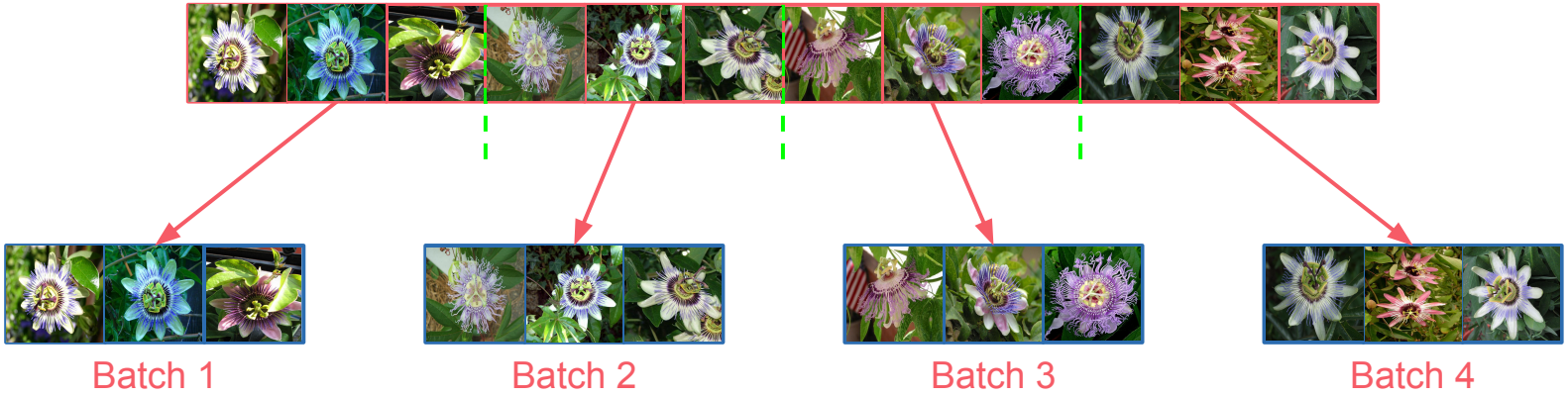
DataLoader

Data Management in PyTorch

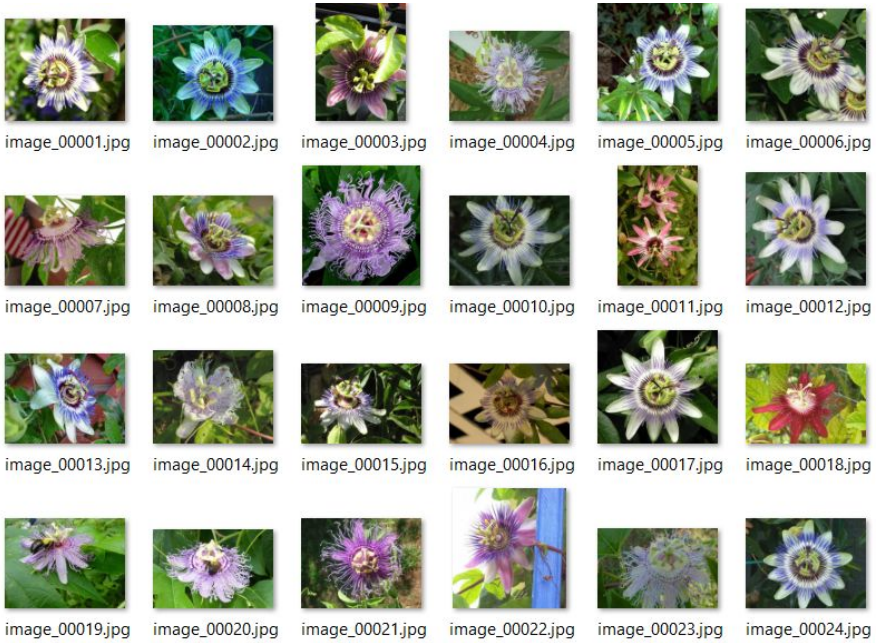
Splitting Dataset

- **Training**
- **Validation**
- **Test**

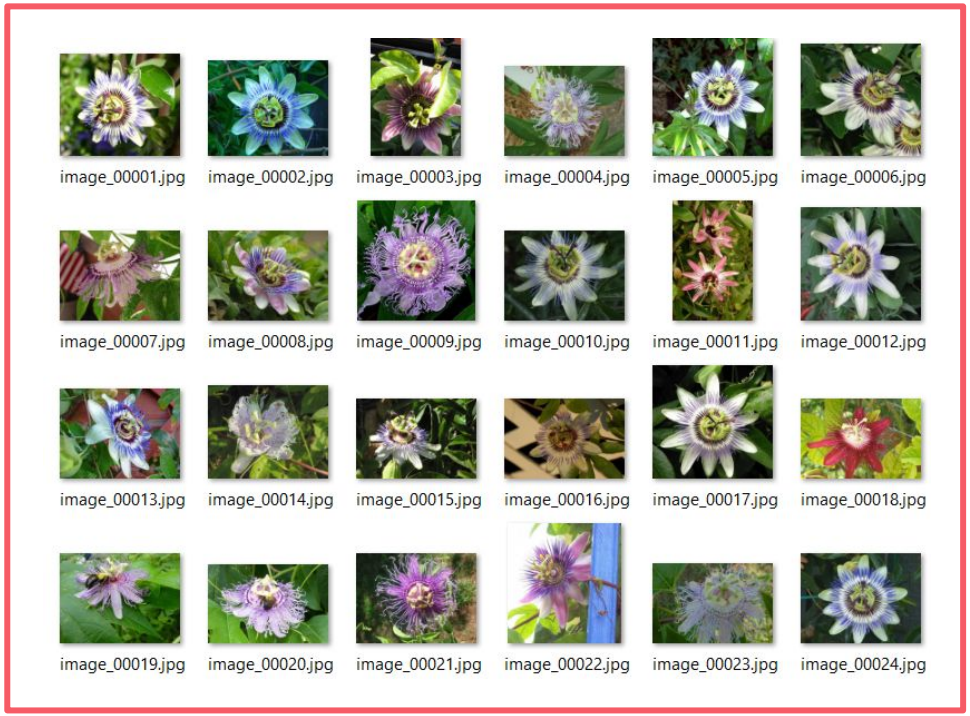
Batching -> DataLoader



Splitting Dataset



Splitting Dataset

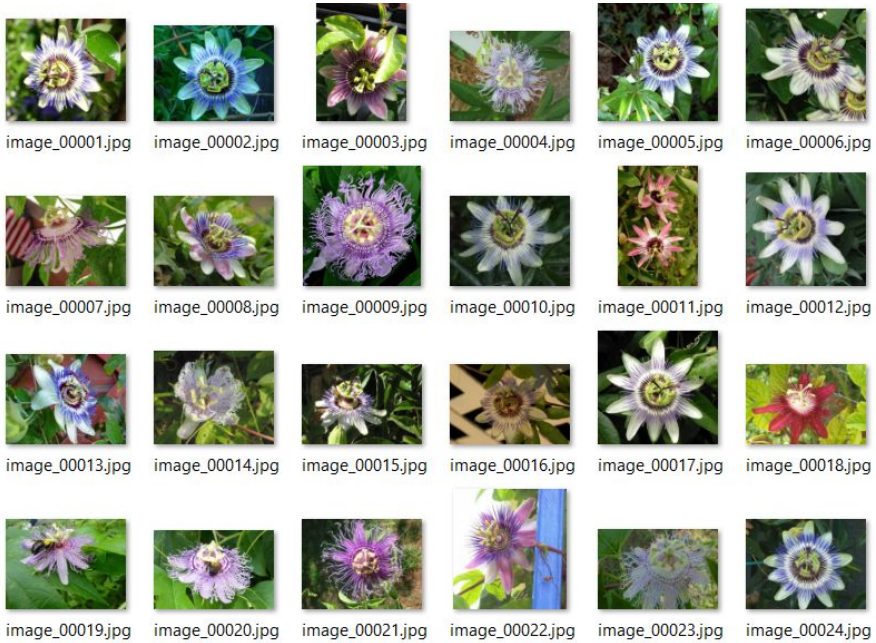


Full Dataset

New Data

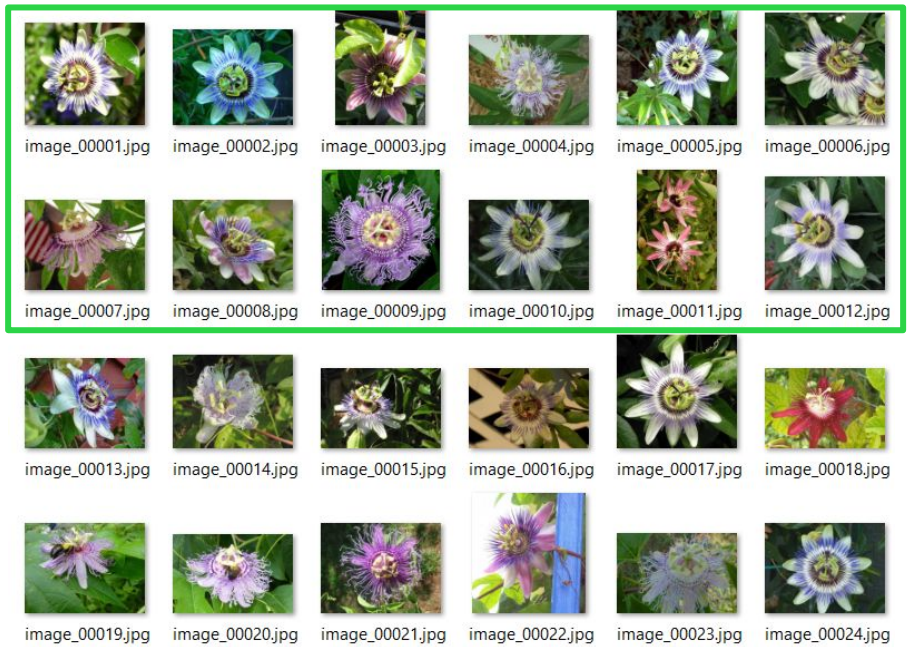
New image

Splitting Dataset



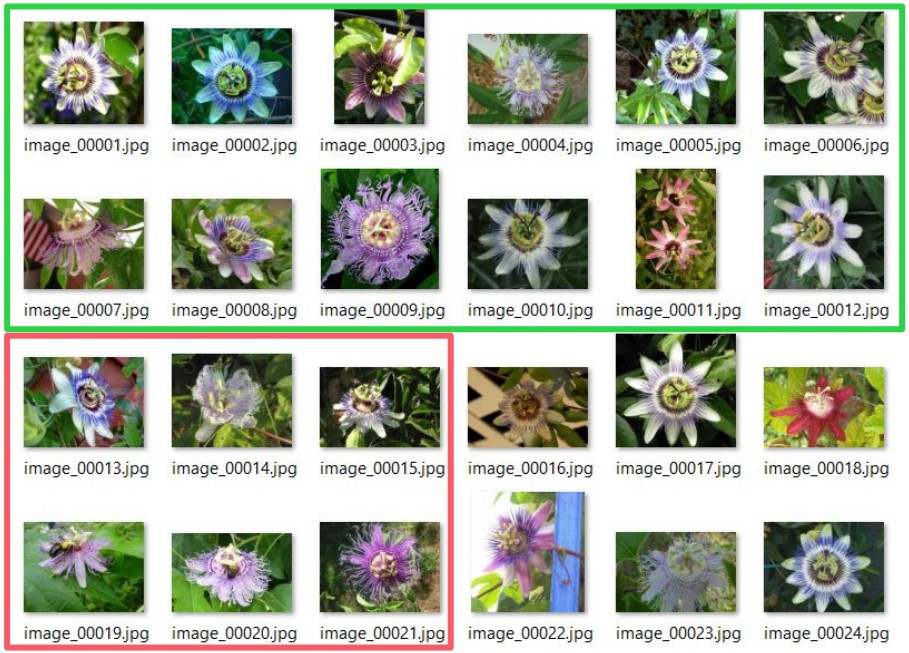
- Training
- Validation
- Test

Splitting Dataset



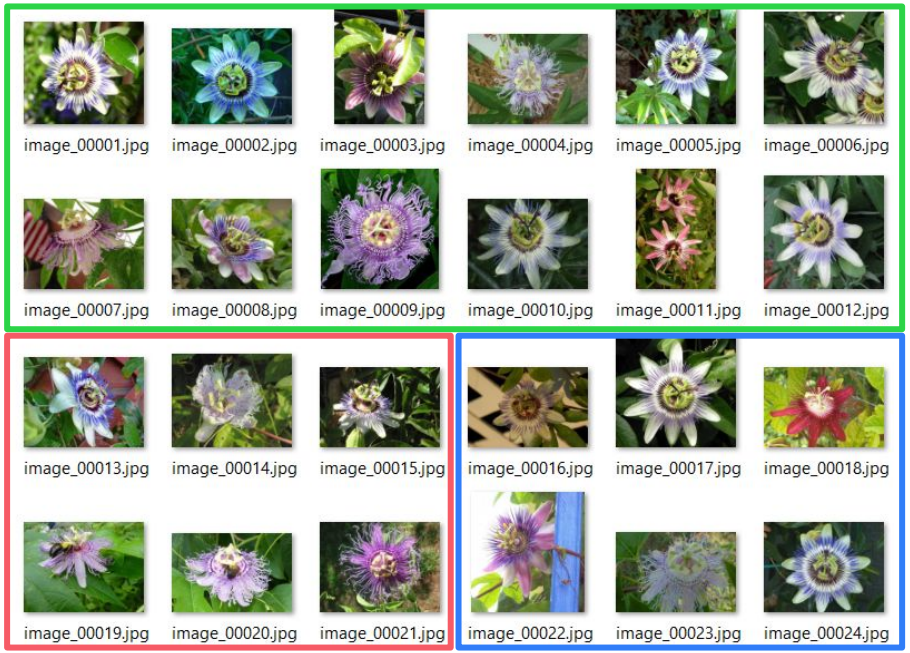
- **Training ~ 5700 (70%)**
- Validation
- Test

Splitting Dataset



- **Training ~ 5700 (70%)**
- **Validation ~ 1200 (15%)**
- Test

Splitting Dataset



- **Training ~ 5700 (70%)**
- **Validation ~ 1200 (15%)**
- **Test ~ 1200 (15%)**

Splitting in PyTorch

```
from torch.utils.data import random_split

# Split into train/val/test: 70/15/15
train_size = int(0.7 * len(dataset))
val_size = int(0.15 * len(dataset))
test_size = len(dataset) - train_size - val_size

train_dataset, val_dataset, test_dataset = random_split(
    dataset, [train_size, val_size, test_size]
)

print(f"Training: {len(train_dataset)} images")
print(f"Validation: {len(val_dataset)} images")
print(f"Test: {len(test_dataset)} images")
```

Splitting in PyTorch

```
from torch.utils.data import random_split

# Split into train/val/test: 70/15/15
train_size = int(0.7 * len(dataset))
val_size = int(0.15 * len(dataset))
test_size = len(dataset) - train_size - val_size

train_dataset, val_dataset, test_dataset = random_split(
    dataset, [train_size, val_size, test_size]
)

print(f"Training: {len(train_dataset)} images")
print(f"Validation: {len(val_dataset)} images")
print(f"Test: {len(test_dataset)} images")
```

Output:

```
C:>
Training: 5732 images
Validation: 1228 images
Test: 1229 images
```

Splitting in PyTorch

```
from torch.utils.data import random_split

# Split into train/val/test: 70/15/15
train_size = int(0.7 * len(dataset))
val_size = int(0.15 * len(dataset))
test_size = len(dataset) - train_size - val_size

train_dataset, val_dataset, test_dataset = random_split(
    dataset, [train_size, val_size, test_size]
)

print(f"Training: {len(train_dataset)} images")
print(f"Validation: {len(val_dataset)} images")
print(f"Test: {len(test_dataset)} images")
```

Output:

```
C:>
Training: 5732 images
Validation: 1228 images
Test: 1229 images
```

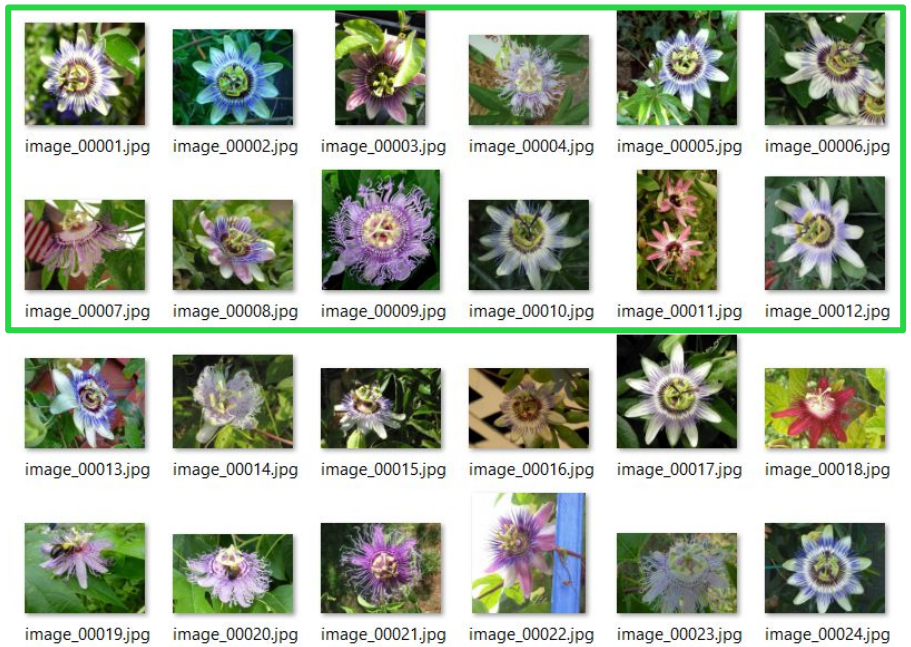
Original Data is Not Modified

```
# random_split creates view not copies
print(f"Original dataset still has: {len(dataset)} images")
```

Output:

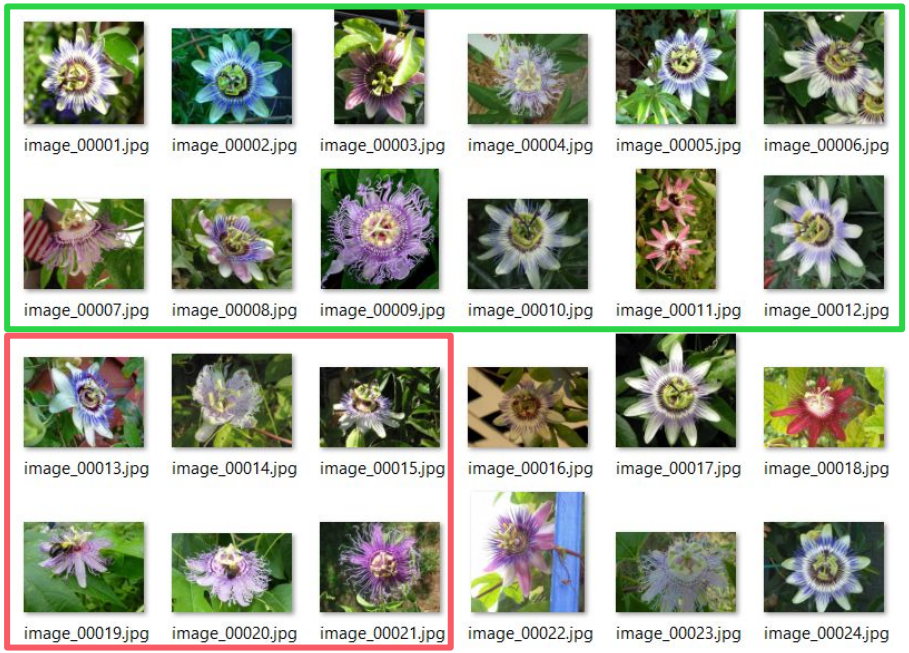
```
C:>
Original dataset still has: 8189 images
```

Splitting Dataset



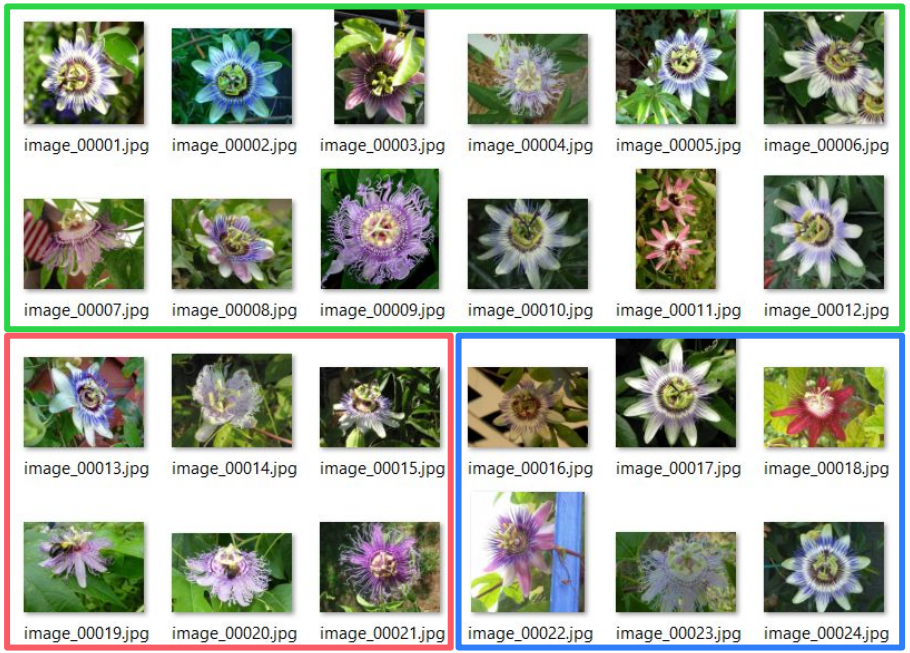
- **Training ~ 5700 (70%)**
- Validation
- Test

Splitting Dataset



- **Training ~ 5700 (70%)**
- **Validation ~ 1200 (15%)**
- Test

Splitting Dataset



- **Training ~ 5700 (70%)**
- **Validation ~ 1200 (15%)**
- **Test ~ 1200 (15%)**

Batching -> DataLoader

```
from torch.utils.data import DataLoader

# Create DataLoaders for each set
train_loader = DataLoader(train_dataset, batch_size=32)
val_loader = DataLoader(val_dataset, batch_size=32)
test_loader = DataLoader(test_dataset, batch_size=32)
```

Batching -> DataLoader

```
from torch.utils.data import DataLoader

# Create DataLoaders for each set
train_loader = DataLoader(train_dataset, batch_size=32)
val_loader = DataLoader(val_dataset, batch_size=32)
test_loader = DataLoader(test_dataset, batch_size=32)
```

Batching -> DataLoader

```
# This loop will go through ALL your data, one batch at a time
for images, labels in train_loader:
    print(f"Batch of images shape: {images.shape}")
    print(f"Batch of labels shape: {labels.shape}")
    break  # Stop after first batch
```

Batching -> DataLoader

```
# This loop will go through ALL your data, one batch at a time
for images, labels in train_loader:
    print(f"Batch of images shape: {images.shape}")
    print(f"Batch of labels shape: {labels.shape}")
    break # Stop after first batch
```

Output:

```
C:>
Batch of images shape: torch.Size([32, 3, 224, 224])
Batch of labels shape: torch.Size([32])
```

Batching -> DataLoader

```
# This loop will go through ALL your data, one batch at a time
for images, labels in train_loader:
    print(f"Batch of images shape: {images.shape}")
    print(f"Batch of labels shape: {labels.shape}")
    break  # Stop after first batch
```

Output:

```
C:>
Batch of images shape: torch.Size([32, 3, 224, 224])
Batch of labels shape: torch.Size([32])
```

Batching -> DataLoader

```
# This loop will go through ALL your data, one batch at a time
for images, labels in train_loader:
    print(f"Batch of images shape: {images.shape}")
    print(f"Batch of labels shape: {labels.shape}")
    break  # Stop after first batch
```

Output:

```
C:>
Batch of images shape: torch.Size([32, 3, 224, 224])
Batch of labels shape: torch.Size([32])
```

Batching -> DataLoader

```
# This loop will go through ALL your data, one batch at a time
for images, labels in train_loader:
    print(f"Batch of images shape: {images.shape}")
    print(f"Batch of labels shape: {labels.shape}")
    break # Stop after first batch
```

Output:

```
C:>
Batch of images shape: torch.Size([32, 3, 224, 224])
Batch of labels shape: torch.Size([32])
```

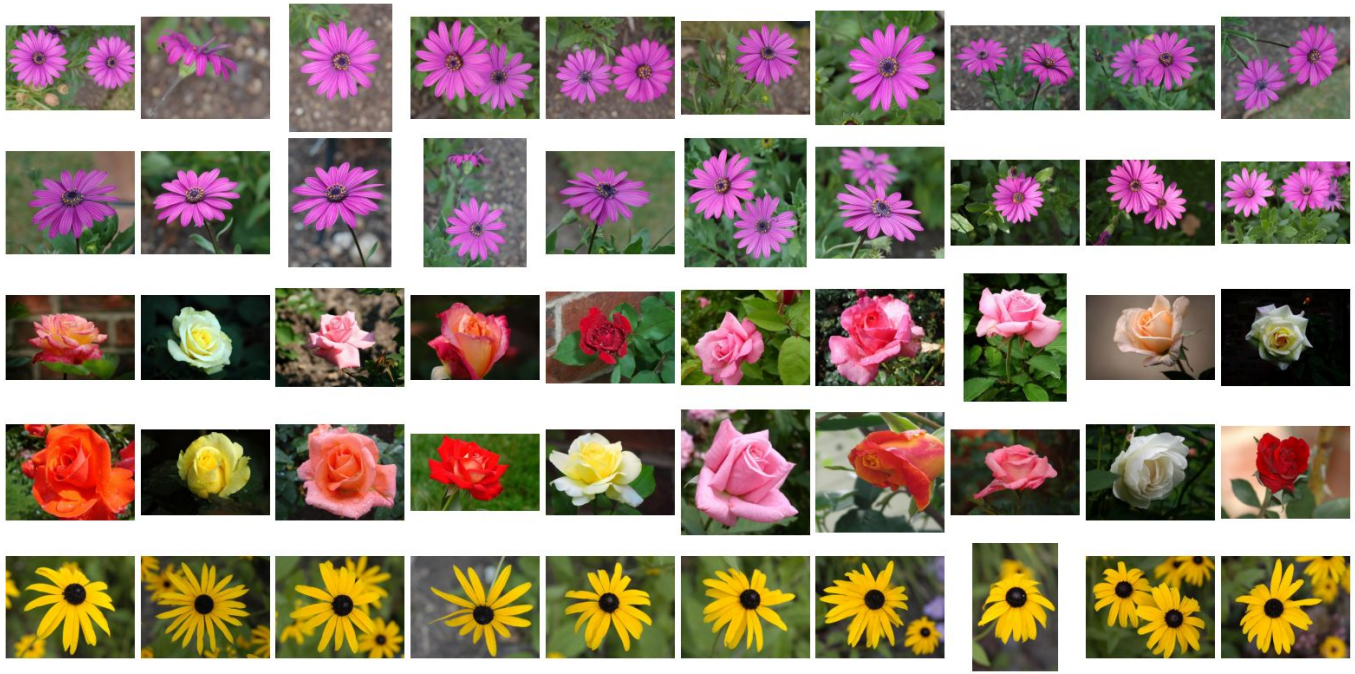
Batching -> DataLoader

```
# Get just the first batch to inspect
images, labels = next(iter(train_loader))
```

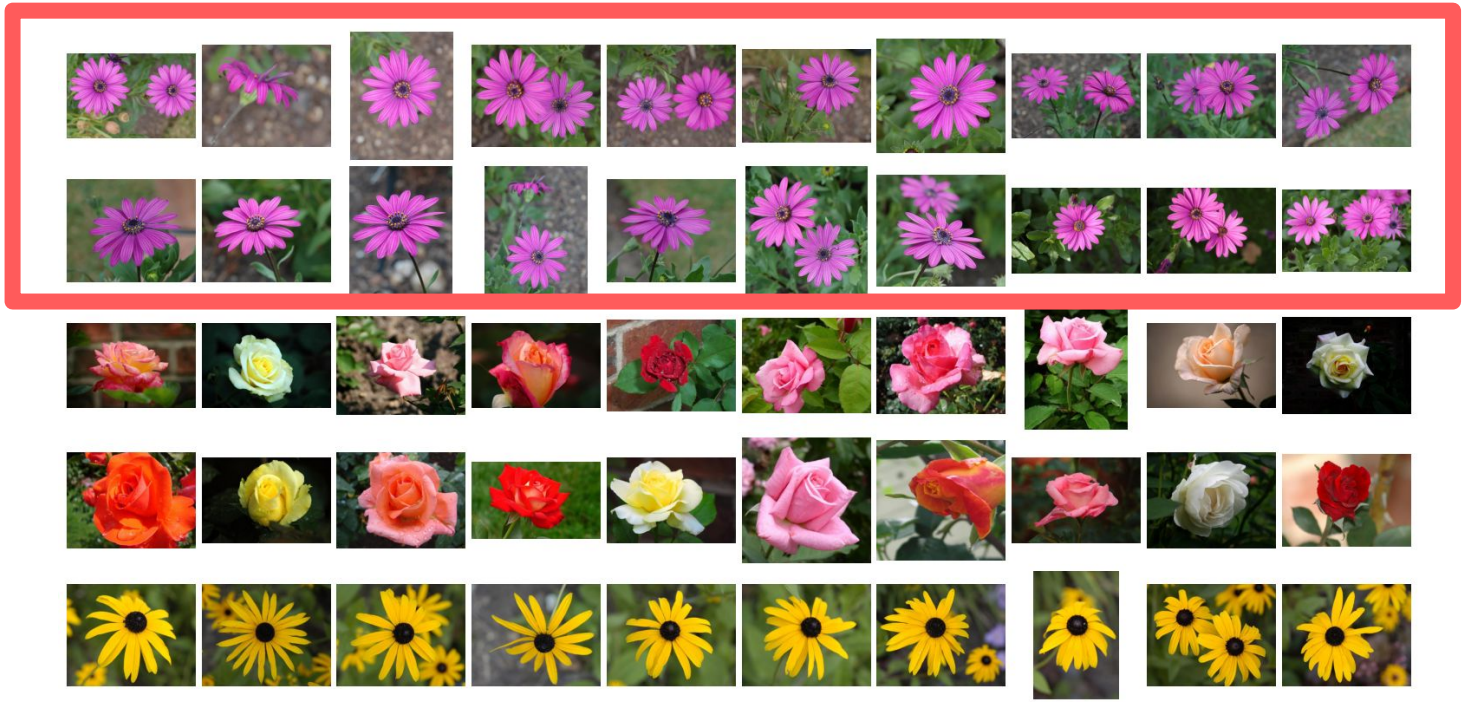
Batching -> DataLoader -> shuffle

```
# For training - shuffle=True
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

# For validation and test - shuffle=False
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

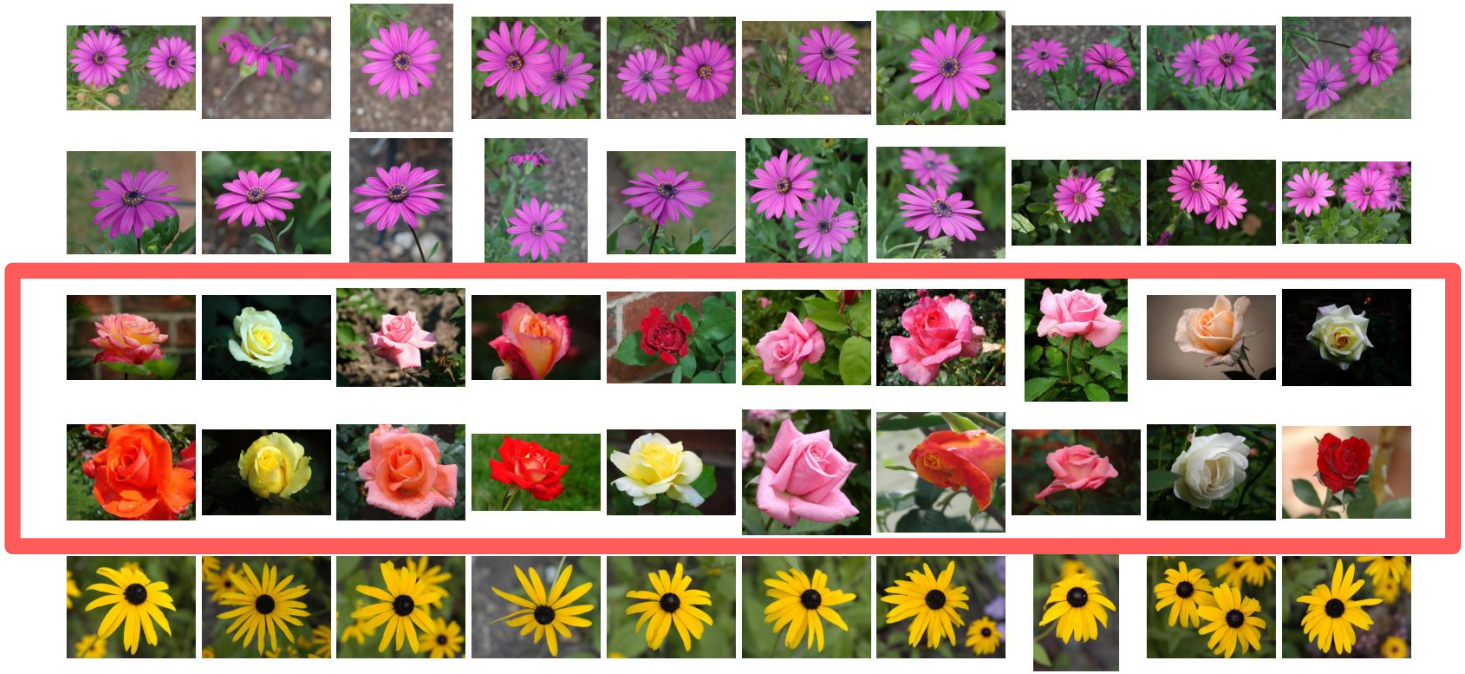
Shuffling Data



Shuffling Data



Shuffling Data



Shuffling Data

Batching -> DataLoader -> shuffle

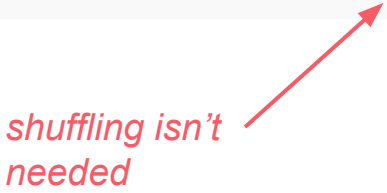
```
# For training - shuffle=True
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

# For validation and test - shuffle=False
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

Batching -> DataLoader -> shuffle

```
# For training - shuffle=True
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

# For validation and test - shuffle=False
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

Batching -> DataLoader -> shuffle

```
# For training - shuffle=True
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

# For validation and test - shuffle=False
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```



Batching -> DataLoader -> shuffle

```
# For training - shuffle=True
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

# For validation and test - shuffle=False
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

The original dataset remains unchanged

Batching Numbers

```
Dataset: 5,732 training images
Batch size: 32

5,732 ÷ 32 = 179.125
```

Batching Numbers

```
Dataset: 5,732 training images
Batch size: 32

5,732 ÷ 32 = 179.125
```

Batching Numbers

```
Dataset: 5,732 training images
Batch size: 32

5,732 ÷ 32 = 179.125
```

- 179 full batches of 32 images
- 1 partial batch with 4 images (the remainder)
- Total: 180 batches
- One epoch -> going through all 180 batches once

Batching Numbers

```
batch_count = 0
total_images = 0

for images, labels in train_loader:
    batch_count += 1
    total_images += len(images)

    # Show the last few batches
    if batch_count >= 178:
        print(f"Batch {batch_count}: {len(images)} images")

print(f"\nTotal batches in one epoch: {batch_count}")
print(f"Total images seen: {total_images}")
```

Output:

```
C:>
Batch 178: 32 images
Batch 179: 32 images
Batch 180: 4 images

Total batches in one epoch: 180
Total images seen: 5732
```

Batching Numbers

```
# For training - shuffle=True
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

# For validation and test - shuffle=False
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

Common Mistakes

```
# Don’t load your data in __getitem__!
class BadDataset(Dataset):
    def __getitem__(self, idx):
        data = pd.read_csv('huge_file.csv')
        return data.iloc[idx]
```

Common Mistakes

```
# Don’t load your data in __getitem__!
class BadDataset(Dataset):
    def __getitem__(self, idx):
        data = pd.read_csv('huge_file.csv')
        return data.iloc[idx]
```

5,732 samples

Loads full dataset 5,732 times

10 epochs = Loads all the data 57,320 times

Common Mistakes

```
# RIGHT - load once, access many times
class GoodDataset(Dataset):
    def __init__(self):
        self.data = pd.read_csv('huge_file.csv')  # Load once

    def __getitem__(self, idx):
        return self.data.iloc[idx]  # Just access
```

Common Errors

```
# For training - shuffle=True
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
```

```
C:>
RuntimeError: CUDA out of memory
```

Common Errors

```
# For training - shuffle=True
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
```

```
C:>
RuntimeError: CUDA out of memory
```

Common Errors

```
# For training - shuffle=True
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
```

```
C:>
RuntimeError: CUDA out of memory
```

Complete setup

```
# Complete setup for the botanical garden app
from torch.utils.data import DataLoader, random_split

# Our dataset with transforms from the previous video
dataset = OxfordFlowersDataset(r'C:\Users\SolidK\Desktop\DLAI\Projects\PyTorch\Oxford 102 flowers', transform=transform)

# Split into train/val/test: 70/15/15
train_size = int(0.7 * len(dataset))
val_size = int(0.15 * len(dataset))
test_size = len(dataset) - train_size - val_size

train_dataset, val_dataset, test_dataset = random_split(
    dataset, [train_size, val_size, test_size]
)

# Create DataLoaders with appropriate settings
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

# Verify everything works
print(f"Train: {len(train_loader)} batches")
print(f"Val: {len(val_loader)} batches")
print(f"Test: {len(test_loader)} batches")

# Quick test - get one batch from each
for name, loader in [("Train", train_loader), ("Val", val_loader), ("Test", test_loader)]:
    images, labels = next(iter(loader))
    print(f"{name} batch: {images.shape}")
```

Batching Numbers

Output:

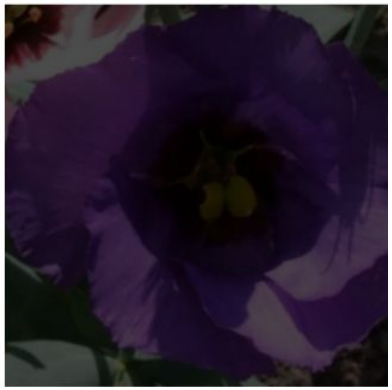
```
C:>
Train: 180 batches
Val: 39 batches
Test: 39 batches
Train batch: torch.Size([32, 3, 224, 224])
Val batch: torch.Size([32, 3, 224, 224])
Test batch: torch.Size([32, 3, 224, 224])
```

DeepLearning.AI

Bugproof Pipelines

Data Management in PyTorch

Oxford 102 Flowers Dataset

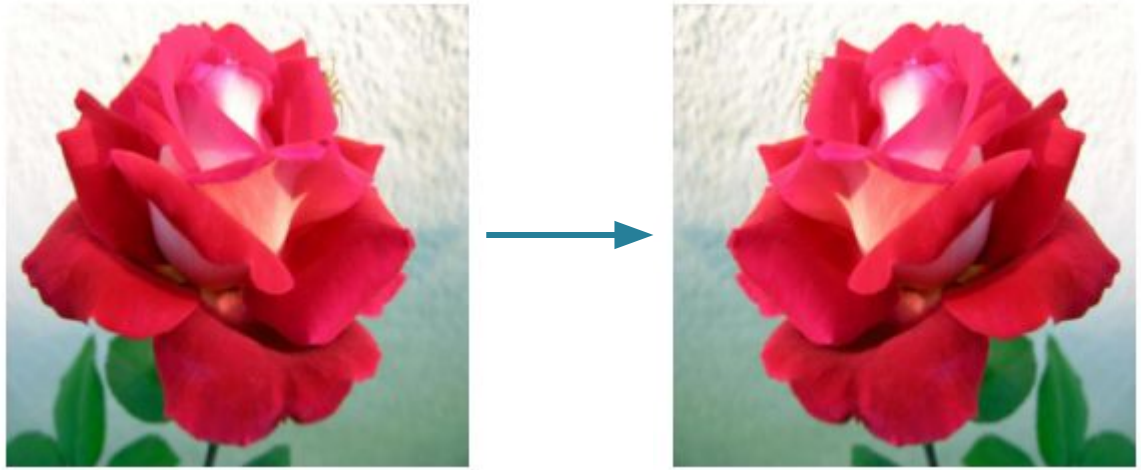


Augmentation



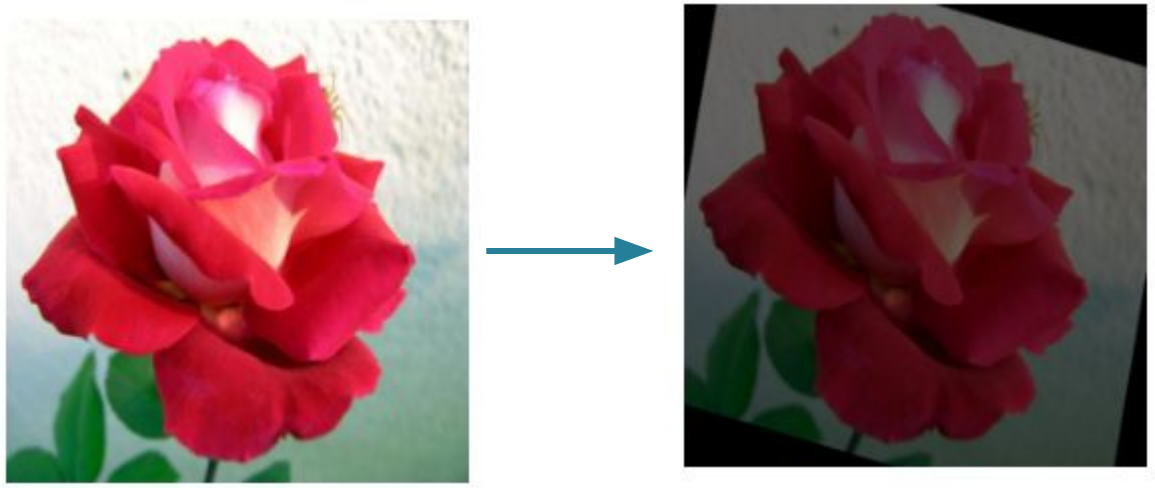

Augmentation

Augmentation



Epoch 1

Augmentation



Epoch 2

Augmentation Transformations

```
# Training transforms - with random augmentation
train_transform = transforms.Compose([
    # Random augmentations (different each time!)
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=10),
    transforms.ColorJitter(brightness=0.2),

    # Standard preprocessing (same as before)
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

```
# Validation transforms - NO augmentation
val_transform = transforms.Compose([
    # Only standard preprocessing
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

Augmentation Transformations

```
# Training transforms - with random augmentation
train_transform = transforms.Compose([
    # Random augmentations (different each time!)
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=10),
    transforms.ColorJitter(brightness=0.2),

    # Standard preprocessing (same as before)
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

```
# Validation transforms - NO augmentation
val_transform = transforms.Compose([
    # Only standard preprocessing
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

Augmentation Transformations

```
# Training transforms - with random augmentation
train_transform = transforms.Compose([
    # Random augmentations (different each time!)
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=10),
    transforms.ColorJitter(brightness=0.2),

    # Standard preprocessing (same as before)
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

```
# Validation transforms - NO augmentation
val_transform = transforms.Compose([
    # Only standard preprocessing
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

Augmentation Transformations

```
# Training transforms - with random augmentation
train_transform = transforms.Compose([
    # Random augmentations (different each time!)
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=10),
    transforms.ColorJitter(brightness=0.2),

    # Standard preprocessing (same as before)
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

```
# Validation transforms - NO augmentation
val_transform = transforms.Compose([
    # Only standard preprocessing
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

Augmentation Transformations

```python
# Training transforms - with random augmentation
train_transform = transforms.Compose([
    # Random augmentations (different each time!)
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=10),
    transforms.ColorJitter(brightness=0.2),

    # Standard preprocessing (same as before)
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

```python
# Validation transforms - NO augmentation
val_transform = transforms.Compose([
    # Only standard preprocessing
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

Augmentation Transformations

```
# Training transforms - with random augmentation
train_transform = transforms.Compose([
    # Random augmentations (different each time!)
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=10),
    transforms.ColorJitter(brightness=0.2),

    # Standard preprocessing (same as before)
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

```
# Validation transforms - NO augmentation
val_transform = transforms.Compose([
    # Only standard preprocessing
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

Image Problems

Training…

Image Problems

CRASH

Corrupted image!

Image Problems

CRASH

Image too small!

Tracking Errors

```
class RobustFlowerDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        # Same setup as before
        self.root_dir = root_dir
        self.img_dir = os.path.join(root_dir, 'jpg')
        self.transform = transform

        # Load labels
        labels_mat = scipy.io.loadmat(os.path.join(root_dir, 'imagelabels.mat'))
        self.labels = labels_mat['labels'][0] - 1

        # NEW: Keep track of any errors we encounter
        self.error_log = []
```

Tracking Errors

```
class RobustFlowerDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        # Same setup as before
        self.root_dir = root_dir
        self.img_dir = os.path.join(root_dir, 'jpg')
        self.transform = transform

        # Load labels
        labels_mat = scipy.io.loadmat(os.path.join(root_dir, 'imagelabels.mat'))
        self.labels = labels_mat['labels'][0] - 1

        # NEW: Keep track of any errors we encounter
        self.error_log = []
```

Tracking Errors

```
def __getitem__(self, idx):
    """Load image with error handling"""
    try:
        # Normal loading code
        img_name = f'image_{idx+1:05d}.jpg'
        img_path = os.path.join(self.img_dir, img_name)
        image = Image.open(img_path)

        # Check for corruption
        image.verify()  # verify() closes the file
        image = Image.open(img_path)  # Reopen for transform use

        # Skip tiny images
        if image.size[0] < 32 or image.size[1] < 32:
            raise ValueError(f"Image too small: {image.size}")

        # Convert grayscale to RGB
        if image.mode != 'RGB':
            image = image.convert('RGB')
```

Tracking Errors

```
def __getitem__(self, idx):
    """Load image with error handling"""
    try:
        # Normal loading code
        img_name = f'image_{idx+1:05d}.jpg'
        img_path = os.path.join(self.img_dir, img_name)
        image = Image.open(img_path)

        # Check for corruption
        image.verify()  # verify() closes the file
        image = Image.open(img_path)  # Reopen for transform use

        # Skip tiny images
        if image.size[0] < 32 or image.size[1] < 32:
            raise ValueError(f"Image too small: {image.size}")

        # Convert grayscale to RGB
        if image.mode != 'RGB':
            image = image.convert('RGB')
```

Tracking Errors

```
def __getitem__(self, idx):
    """Load image with error handling"""
    try:
        # Normal loading code
        img_name = f'image_{idx+1:05d}.jpg'
        img_path = os.path.join(self.img_dir, img_name)
        image = Image.open(img_path)

        # Check for corruption
        image.verify()  # verify() closes the file
        image = Image.open(img_path)  # Reopen for transform use

        # Skip tiny images
        if image.size[0] < 32 or image.size[1] < 32:
            raise ValueError(f"Image too small: {image.size}")

        # Convert grayscale to RGB
        if image.mode != 'RGB':
            image = image.convert('RGB')
```

Tracking Errors

```
def __getitem__(self, idx):
    """Load image with error handling"""
    try:
        # Normal loading code
        img_name = f'image_{idx+1:05d}.jpg'
        img_path = os.path.join(self.img_dir, img_name)
        image = Image.open(img_path)

        # Check for corruption
        image.verify()  # verify() closes the file
        image = Image.open(img_path)  # Reopen for transform use

        # Skip tiny images
        if image.size[0] < 32 or image.size[1] < 32:
            raise ValueError(f"Image too small: {image.size}")

        # Convert grayscale to RGB
        if image.mode != 'RGB':
            image = image.convert('RGB')
```

Tracking Errors

```
def __getitem__(self, idx):
    """Load image with error handling"""
    try:
        # Normal loading code
        img_name = f'image_{idx+1:05d}.jpg'
        img_path = os.path.join(self.img_dir, img_name)
        image = Image.open(img_path)

        # Check for corruption
        image.verify()  # verify() closes the file
        image = Image.open(img_path)  # Reopen for transform use

        # Skip tiny images
        if image.size[0] < 32 or image.size[1] < 32:
            raise ValueError(f"Image too small: {image.size}")

        # Convert grayscale to RGB
        if image.mode != 'RGB':
            image = image.convert('RGB')
```

Tracking Errors

```
def __getitem__(self, idx):
    """Load image with error handling"""
    try:
        # . . . Previous Checks . . .

    except Exception as e:
        # Log the issue instead of crashing
        self.error_log.append({

            'index': idx,
            'error': str(e),
            'path': img_path if 'img_path' in locals() else 'unknown'})

        print(f"Warning: Skipping corrupted image {idx}: {e}")
        # Try the next image (wrap around if needed)
        next_idx = (idx + 1) % len(self)
        return self.__getitem__(next_idx)
```

Tracking Errors

```
def __getitem__(self, idx):
    """Load image with error handling"""
    try:
        # . . . Previous Checks . . .

    except Exception as e:
        # Log the issue instead of crashing
        self.error_log.append({

            'index': idx,
            'error': str(e),
            'path': img_path if 'img_path' in locals() else 'unknown'})

        print(f"Warning: Skipping corrupted image {idx}: {e}")
        # Try the next image (wrap around if needed)
        next_idx = (idx + 1) % len(self)
        return self.__getitem__(next_idx)
```

Tracking Errors

```
def __getitem__(self, idx):
    """Load image with error handling"""
    try:
        # . . . Previous Checks . . .

    except Exception as e:
        # Log the issue instead of crashing
        self.error_log.append({

            'index': idx,
            'error': str(e),
            'path': img_path if 'img_path' in locals() else 'unknown'})

        print(f"Warning: Skipping corrupted image {idx}: {e}")
        # Try the next image (wrap around if needed)
        next_idx = (idx + 1) % len(self)
        return self.__getitem__(next_idx)
```

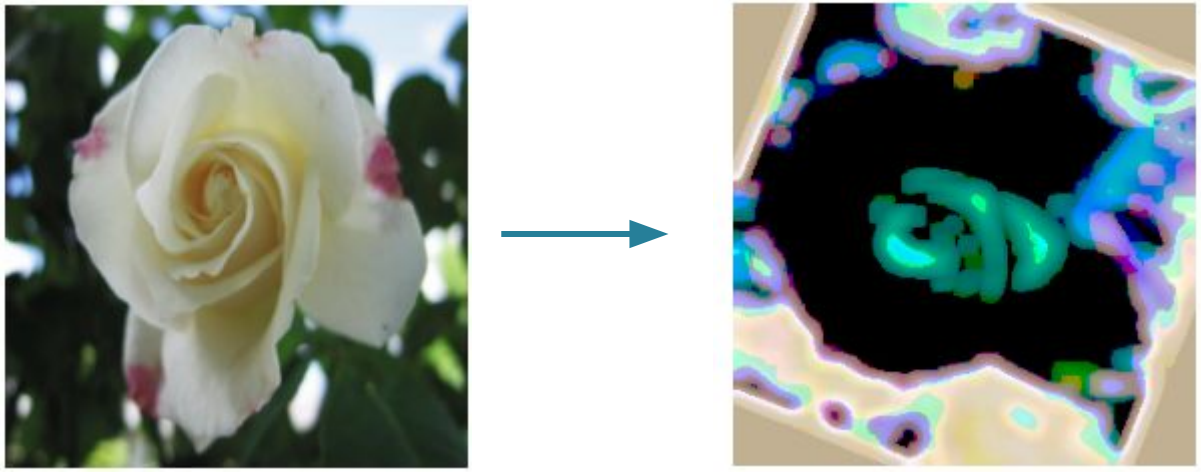
Tracking Errors

```
def __getitem__(self, idx):
    """Load image with error handling"""
    try:
        # . . . Previous Checks . . .

    except Exception as e:
         # Log the issue instead of crashing
         self.error_log.append({

             'index': idx,
             'error': str(e),
             'path': img_path if 'img_path' in locals() else 'unknown'})

         print(f"Warning: Skipping corrupted image {idx}: {e}")
         # Try the next image (wrap around if needed)
         next_idx = (idx + 1) % len(self)
         return self.__getitem__(next_idx)
```

Error Summary

```
    def get_error_summary(self):
        """Review what went wrong after training"""
        if not self.error_log:
            print("No errors encountered - dataset is clean!")
        else:
            print(f"\nEncountered {len(self.error_log)} problematic images:")
            for error in self.error_log[:5]:  # Show first 5
                print(f"  Index {error['index']}: {error['error']}")
            if len(self.error_log) > 5:
                print(f"  ... and {len(self.error_log) - 5} more")
```

Aggressive Augmentation



Visualization Method

```python
def visualize_augmentations(dataset, idx=0, num_versions=8):
    """See what augmentation actually does to your images"""
    fig, axes = plt.subplots(2, 4, figsize=(12, 6))
    axes = axes.flatten()

    for i in range(num_versions):
        img, label = dataset[idx]  # Get augmented version

        # Denormalize for display
        img = denormalize(img)

        axes[i].imshow(img.permute(1, 2, 0))  # CHW -> HWC
        axes[i].set_title(f'Version {i+1}')
        axes[i].axis('off')

    plt.suptitle(f'Same flower (index {idx}), 8 different augmentations')
    plt.tight_layout()
    plt.show()
```

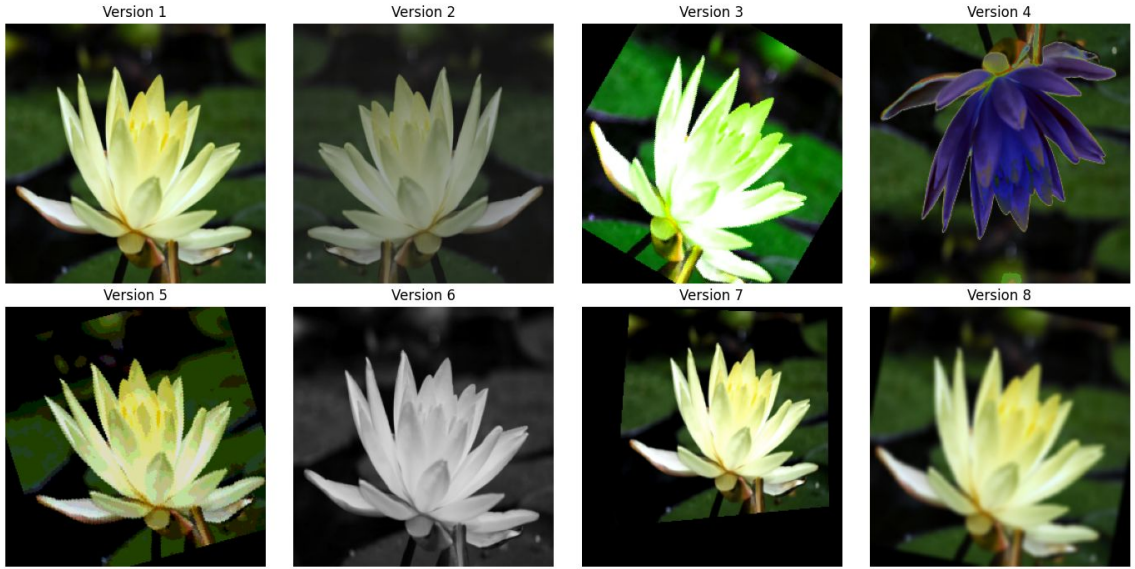
Visualization Method

```
def visualize_augmentations(dataset, idx=0, num_versions=8):
    """See what augmentation actually does to your images"""
    fig, axes = plt.subplots(2, 4, figsize=(12, 6))
    axes = axes.flatten()

    for i in range(num_versions):
        img, label = dataset[idx]  # Get augmented version

        # Denormalize for display
        img = denormalize(img)

        axes[i].imshow(img.permute(1, 2, 0))  # CHW -> HWC
        axes[i].set_title(f'Version {i+1}')
        axes[i].axis('off')

    plt.suptitle(f'Same flower (index {idx}), 8 different augmentations')
    plt.tight_layout()
    plt.show()
```

Visualization Method

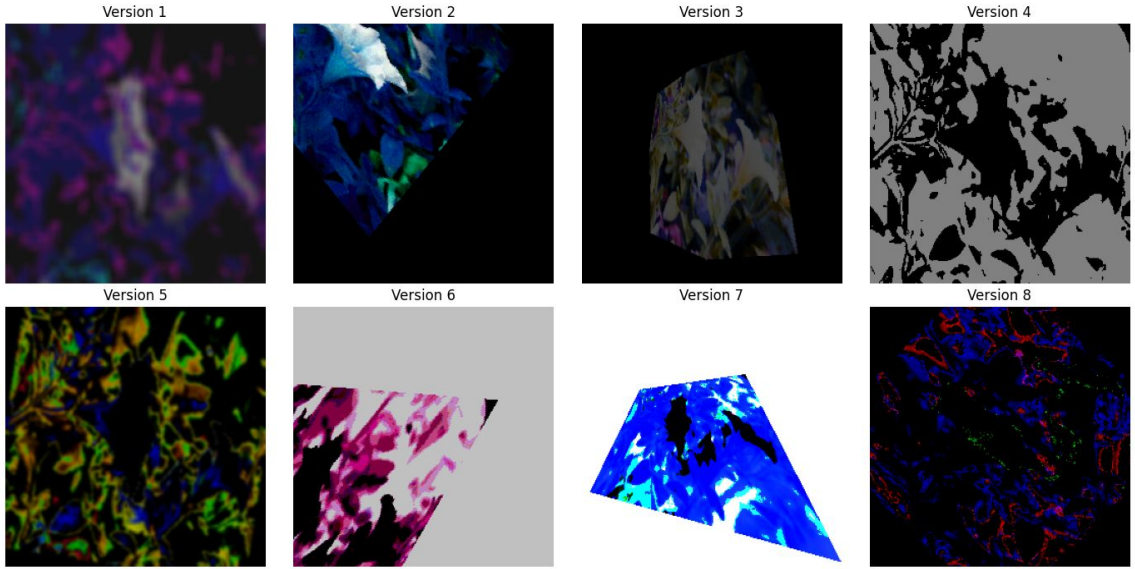
```
def visualize_augmentations(dataset, idx=0, num_versions=8):
    """See what augmentation actually does to your images"""
    fig, axes = plt.subplots(2, 4, figsize=(12, 6))
    axes = axes.flatten()

    for i in range(num_versions):
        img, label = dataset[idx]  # Get augmented version

        # Denormalize for display
        img = denormalize(img)

        axes[i].imshow(img.permute(1, 2, 0))  # CHW -> HWC
        axes[i].set_title(f'Version {i+1}')
        axes[i].axis('off')

    plt.suptitle(f'Same flower (index {idx}), 8 different augmentations')
    plt.tight_layout()
    plt.show()
```

Augmentation Test



Augmentation Test

Augmentation Test



Augmentation Test

Data Tracking

```python
class MonitoredDataset(RobustFlowerDataset):
    # __init__ . . .
    def __getitem__(self, idx):
        import time
        start_time = time.time()

        # Track how often each image is accessed
        self.access_counts[idx] = self.access_counts.get(idx, 0) + 1

        # Load image using parent class (with error handling)
        result = super().__getitem__(idx)

        # Track how long it took
        load_time = time.time() - start_time
        self.load_times.append(load_time)

        # Warn if slow
        if load_time > 1.0:
            print(f"Slow load: Image {idx} took {load_time:.2f}s")

        return result
```

Data Tracking

```python
class MonitoredDataset(RobustFlowerDataset):
    # __init__ . . .
    def __getitem__(self, idx):
        import time
        start_time = time.time()

        # Track how often each image is accessed
        self.access_counts[idx] = self.access_counts.get(idx, 0) + 1

        # Load image using parent class (with error handling)
        result = super().__getitem__(idx)

        # Track how long it took
        load_time = time.time() - start_time
        self.load_times.append(load_time)

        # Warn if slow
        if load_time > 1.0:
            print(f"Slow load: Image {idx} took {load_time:.2f}s")

        return result
```

Data Tracking

```
# Track how often each image is accessed
dataset.print_stats()
```

Tracking Errors

- Shuffling bugs
- Performance issues
- Data imbalance

Oxford 102 Flowers Dataset