AI VIETNAM
All-in-One Course

Exercise Class

Pretrained Models
Image Retrieval using Vector DB

Nguyen Quoc Thai

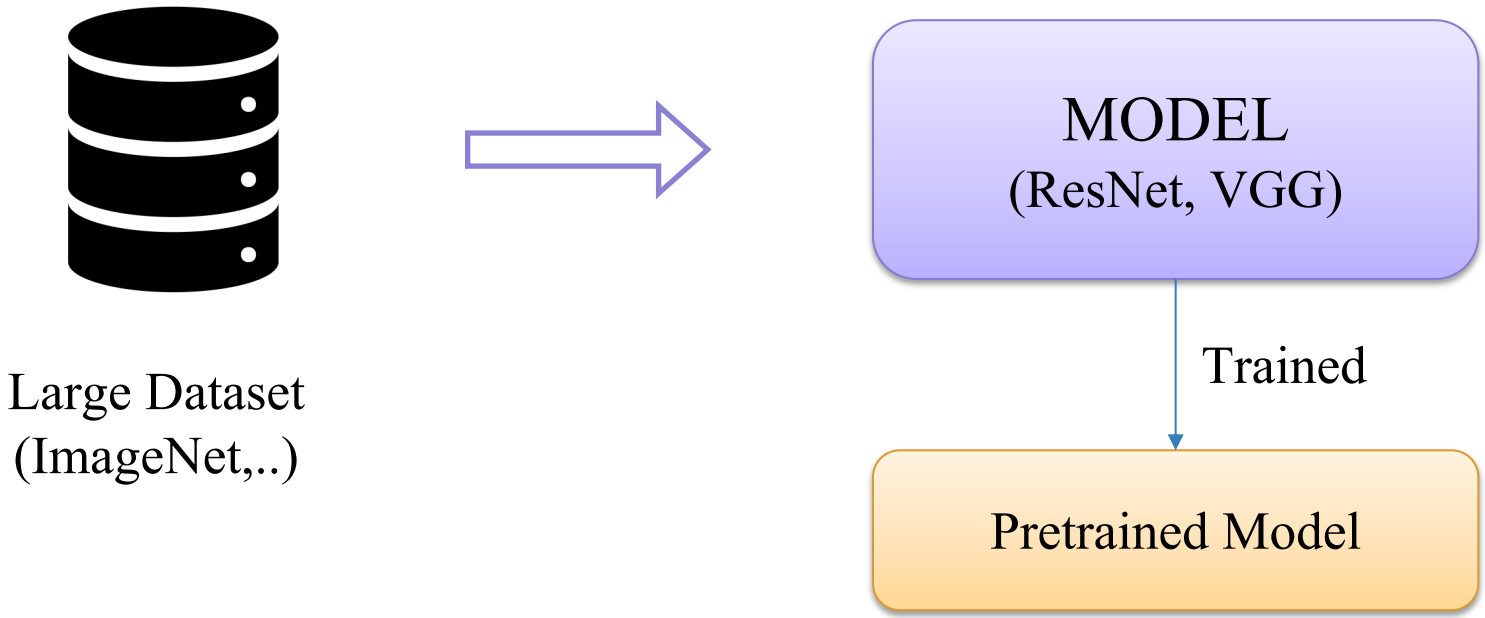
Year 2023

CONTENT

1 – Pretrained Models

Pretrained Models

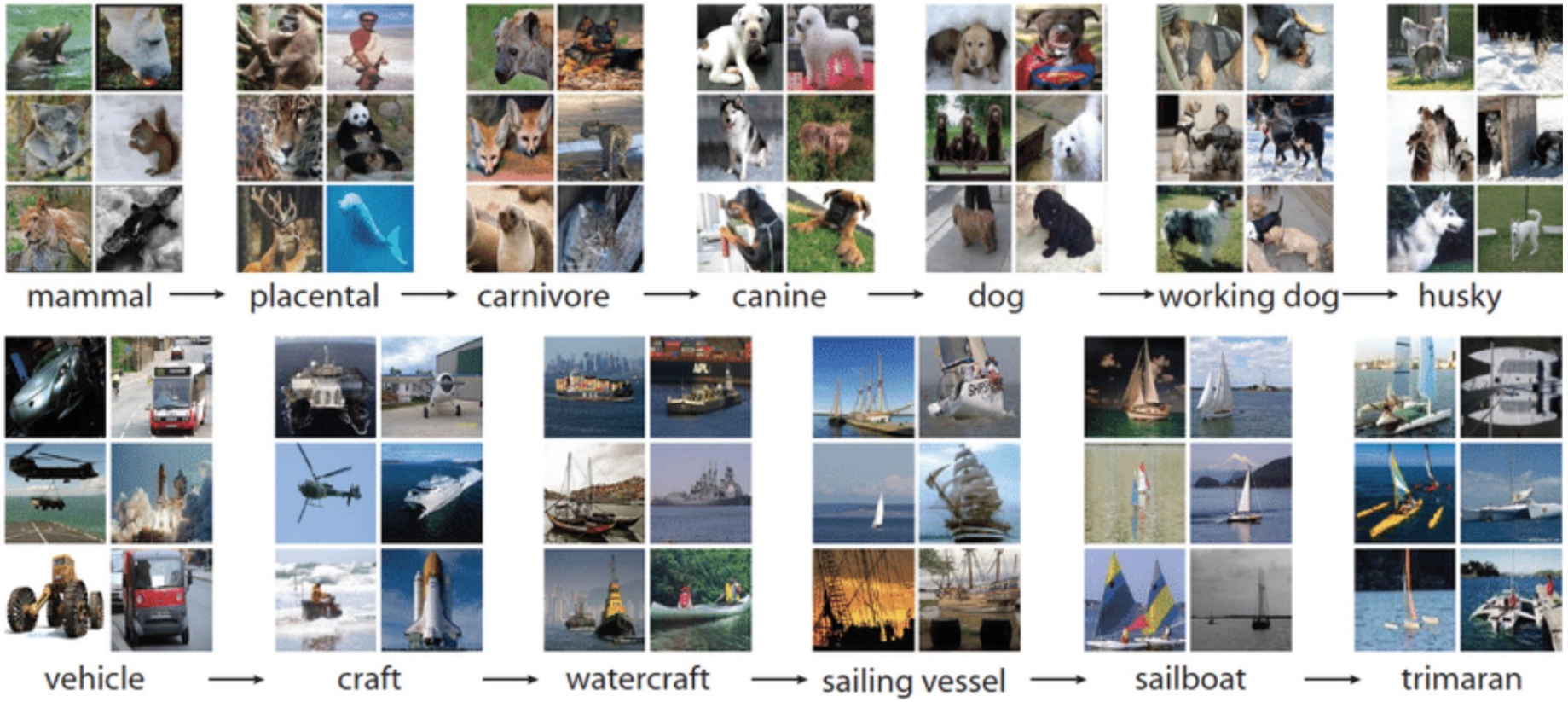
- A model created by some one else to solve a similar problem



1 – Pretrained Models

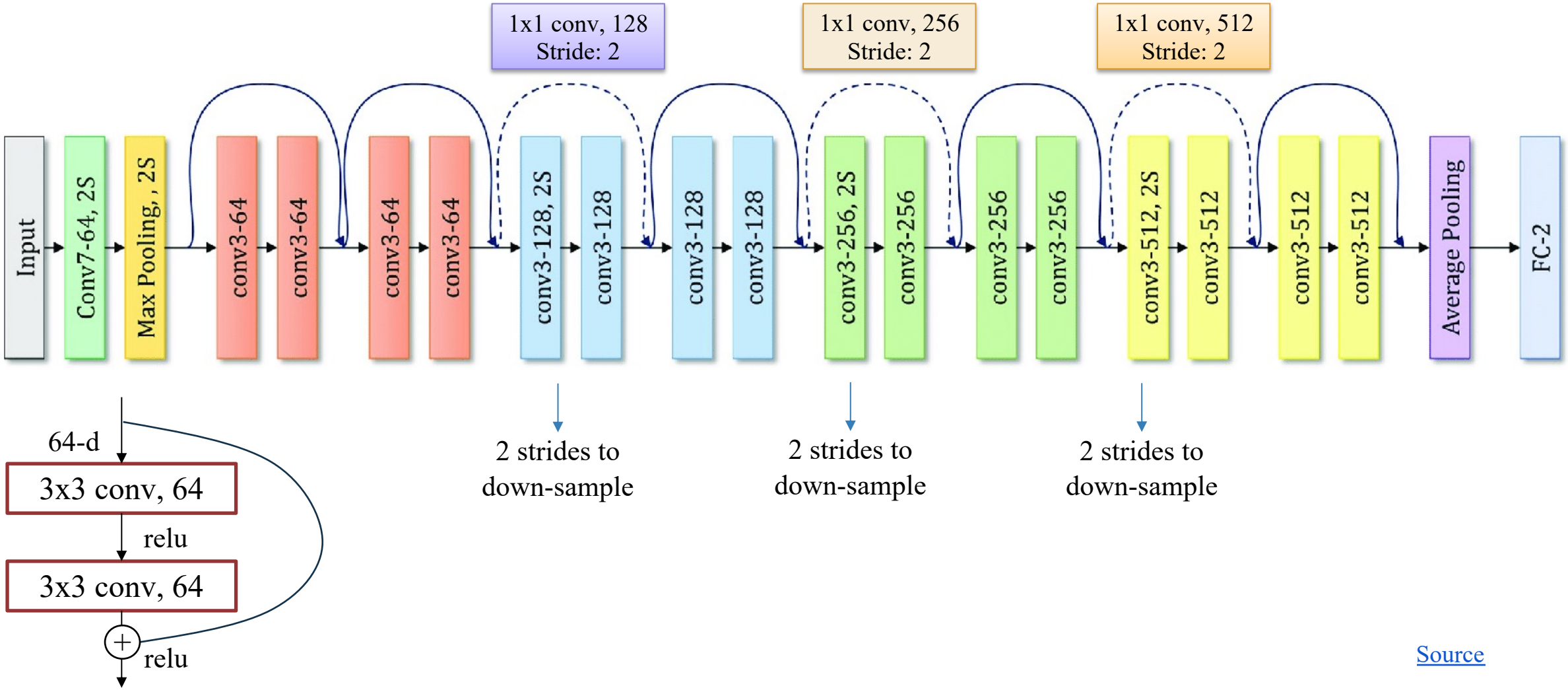
Pretrained Models

- Training: 1,281,167 images. Validation: 50,000 images. Testing: 100,000 images
- Object classes: 1,000



1 – Pretrained Models

ResNet18



Source

1 – Pretrained Models

ResNet

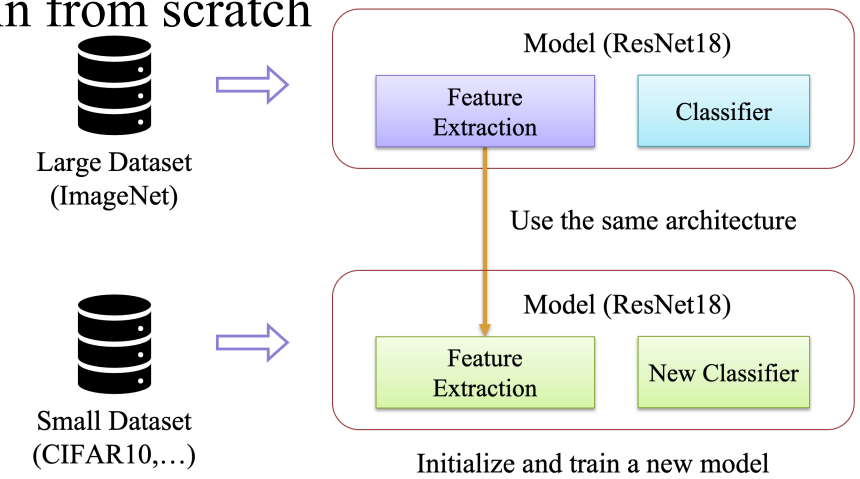
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				

Source

1 – Pretrained Models

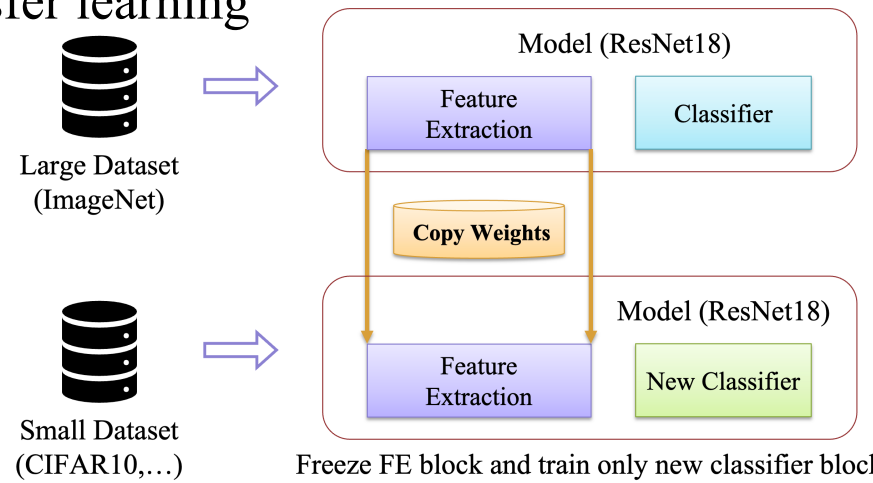
Training with Pretrained Models

Train from scratch



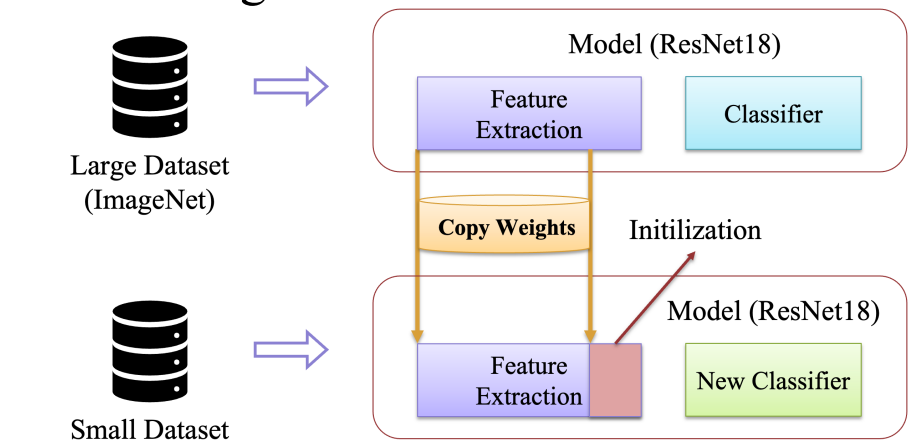
Initialize and train a new model

Transfer learning



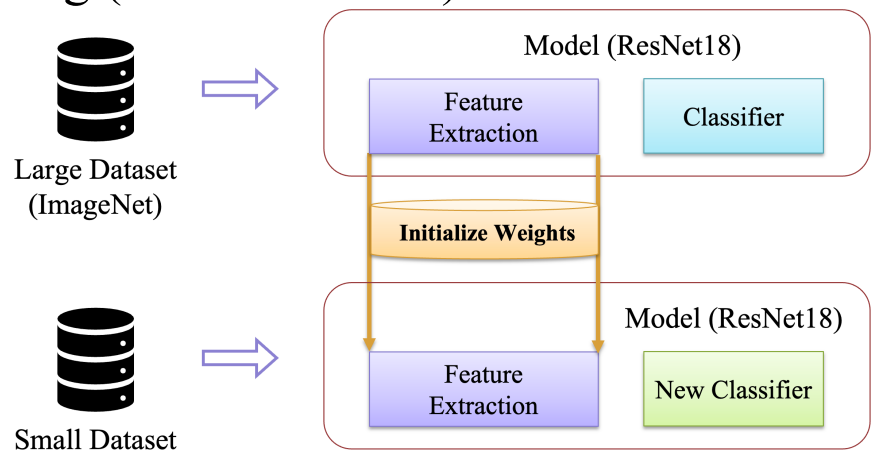
Freeze FE block and train only new classifier block

Fine Tuning



Freeze a part of FE and train the rest + new classifier block

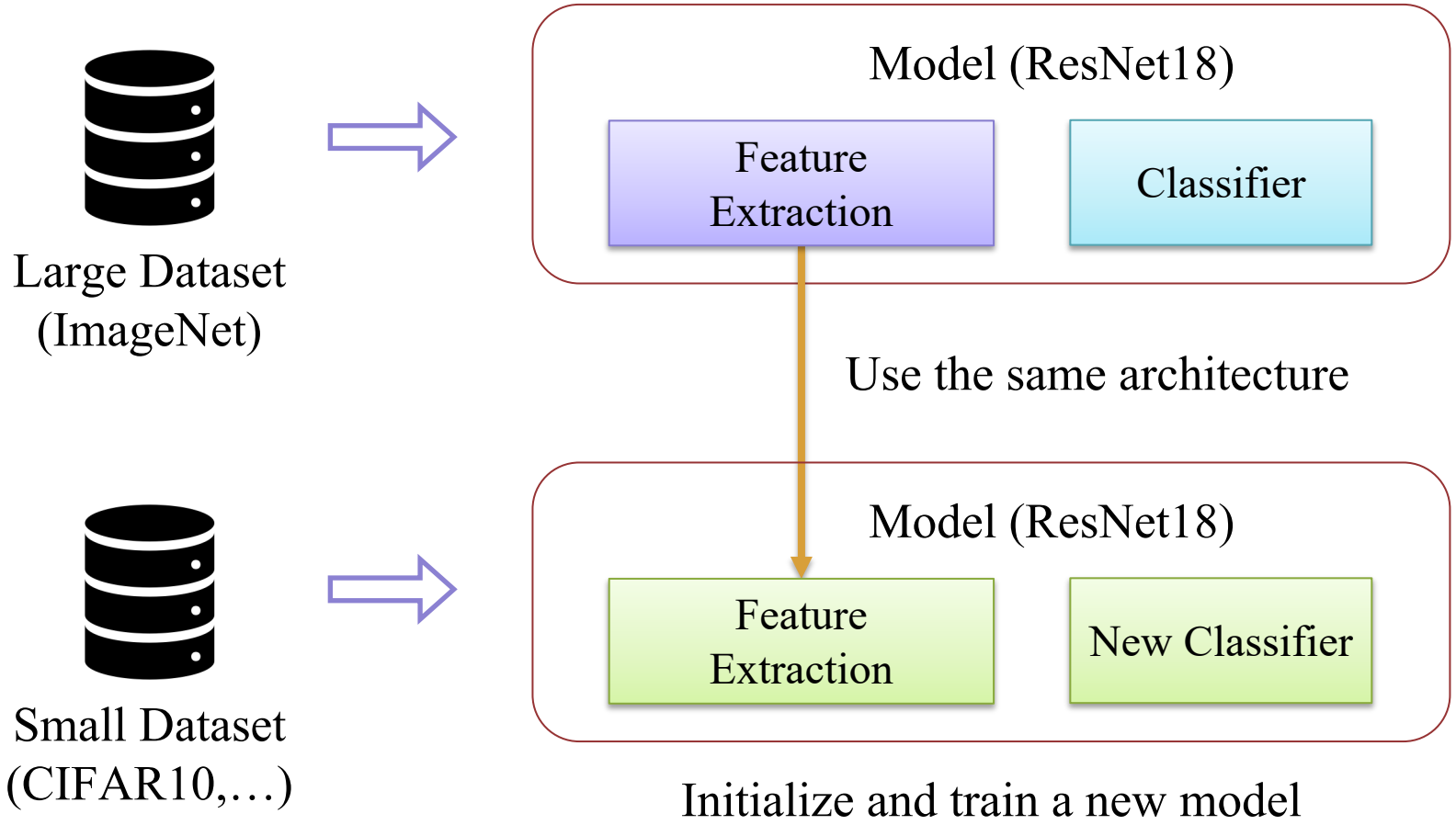
Fine Tuning (As Initilization)



Train a new model

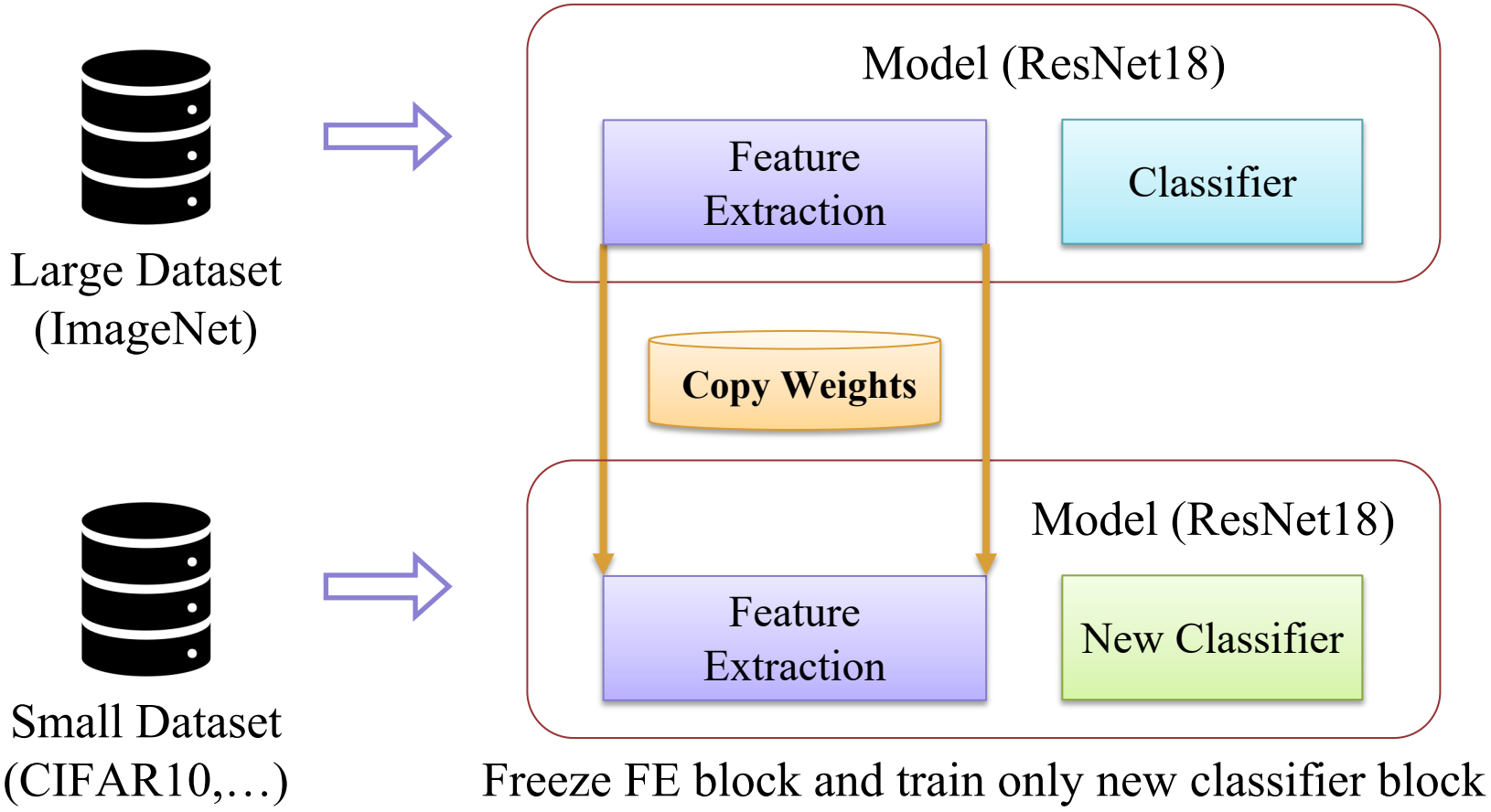
1 – Pretrained Models

Train from Scratch



1 – Pretrained Models

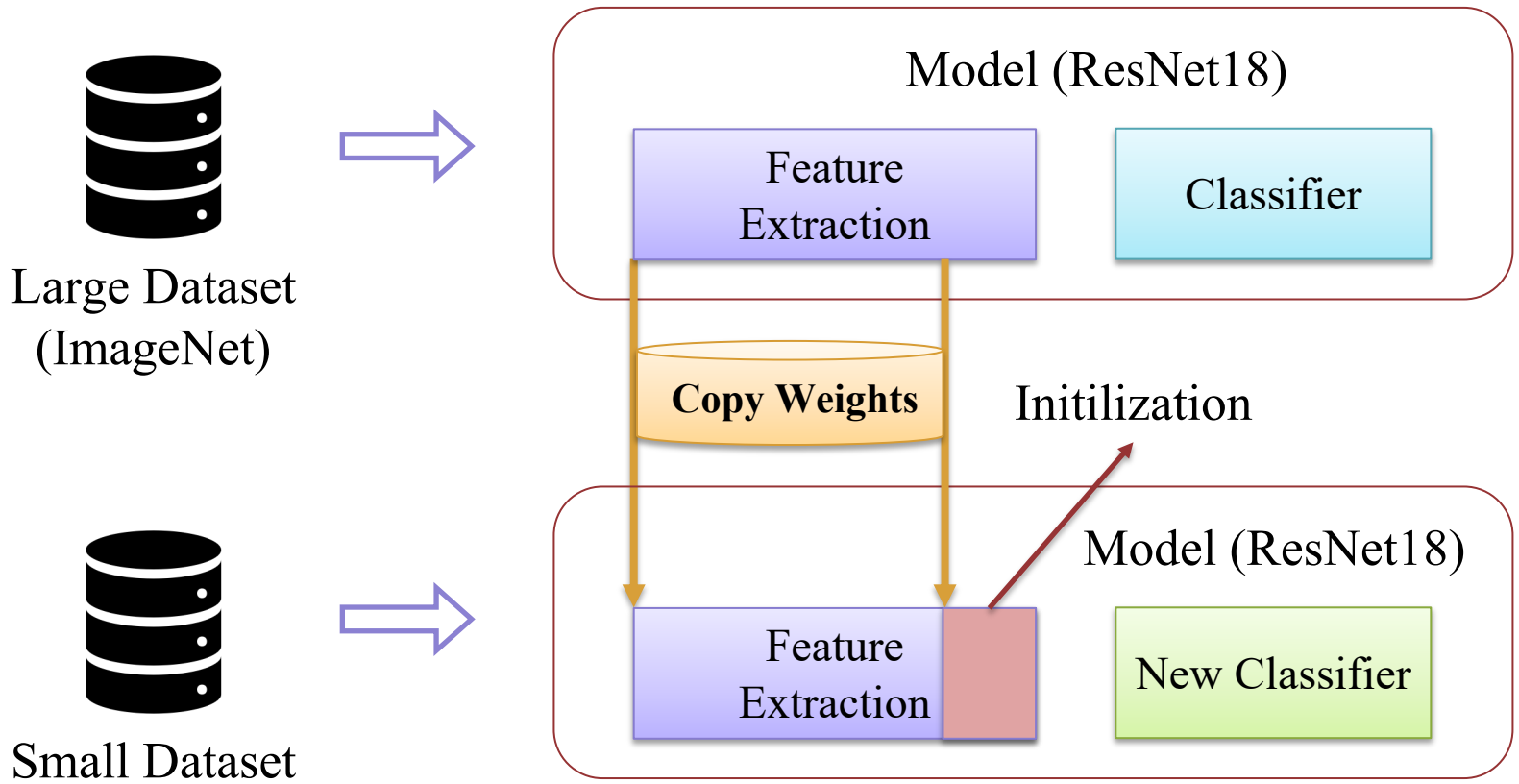
Transfer Learning (Feature Extractor)



Freeze FE block and train only new classifier block

1 – Pretrained Models

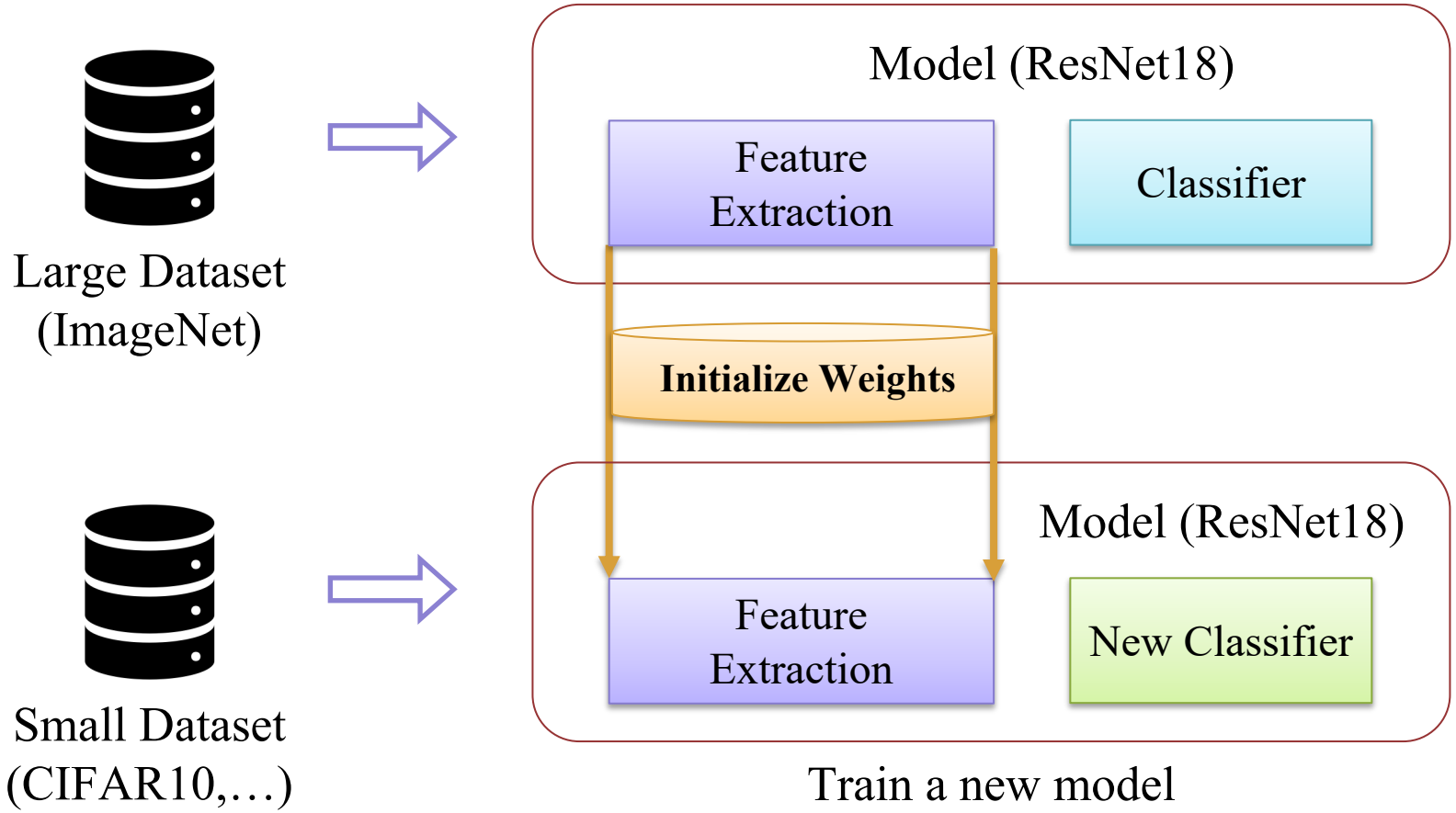
Fine Tuning



Freeze a part of FE and train the rest + new classifier block

1 – Pretrained Models

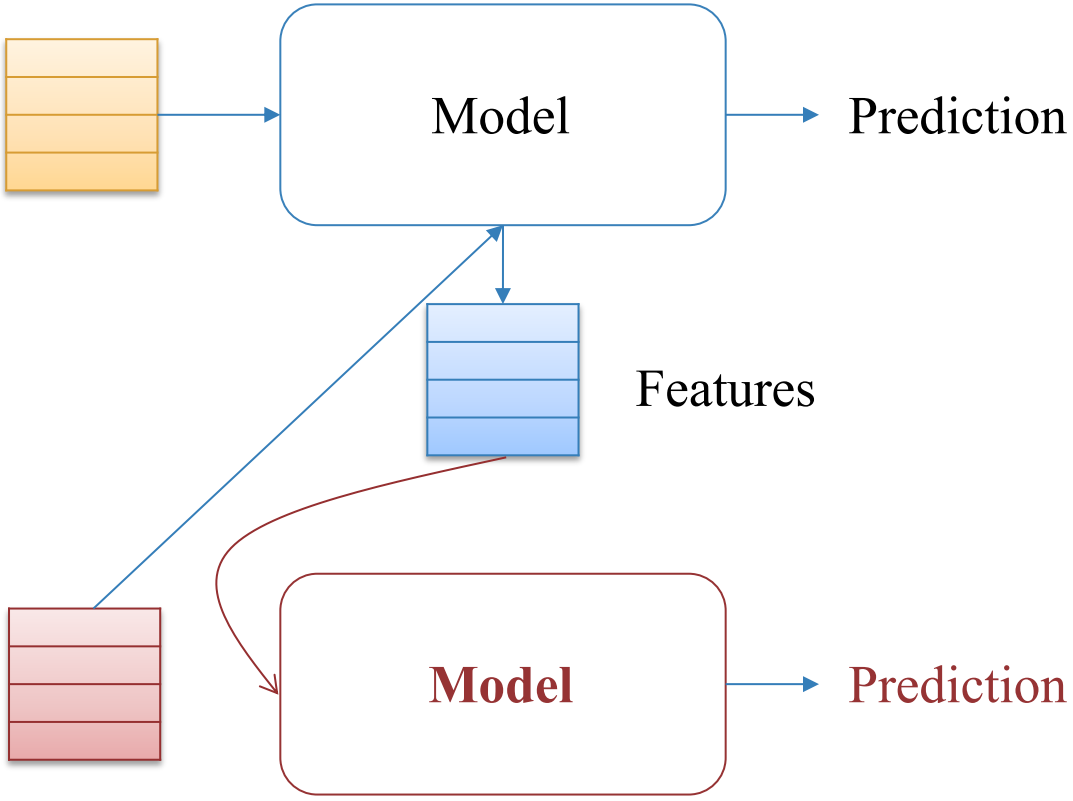
Fine Tuning (As Initialization)



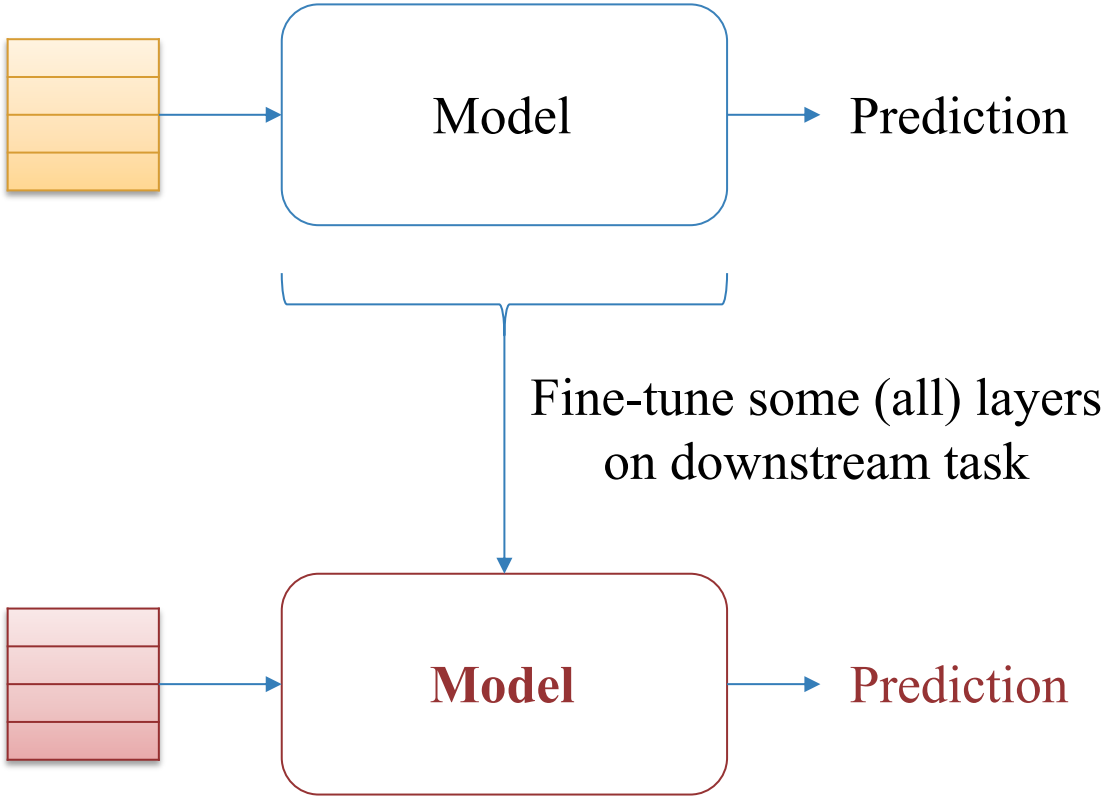
1 – Pretrained Models

Transfer Learning and Fine Tuning using Pretrained Models

❖ Transfer Learning

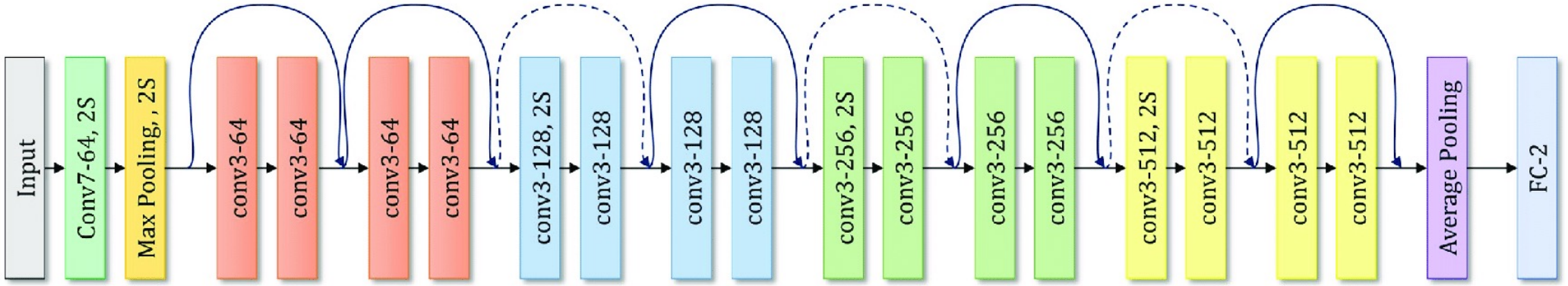


❖ Fine Tuning



1 – Pretrained Models

Transfer Learning and Fine Tuning using Pretrained Models



General Features: Edges…

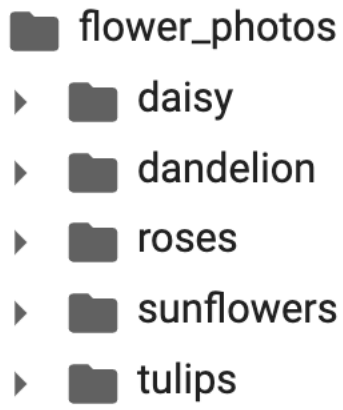
Specific Features

	Similar Dataset	Different Dataset
Small Dataset	Transfer Learning Highest Level Features	Transfer Learning Lower Level features
Large Dataset	Fine Tuning	Fine Tuning

2 – Flower Classification

Flower Dataset - Demo

- Folder Structure




```
[ ] data_path = "./dataset/flower_photos"
```

```
[ ] # load image from path

    dataset = datasets.ImageFolder(root=data_path)
```

```
[ ] dataset
```

```
Dataset ImageFolder
    Number of datapoints: 3670
    Root location: ./dataset/flower_photos
```

```
[ ] num_samples = len(dataset)
    num_samples
```

```
3670
```

```
[ ] dataset.classes
```

```
['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
```

```
[ ] num_classes = len(dataset.classes)
```

2 – Flower Classification

Preprocessing - Demo

- Training : Validation : Testing = 0.9 : 0.1 : 0.1

```
TRAIN_RATIO, VALID_RATIO = 0.8, 0.1

n_train_examples = int(num_samples * TRAIN_RATIO)
n_valid_examples = int(num_samples * VALID_RATIO)
n_test_examples = num_samples - n_train_examples - n_valid_examples
```

```
n_train_examples, n_valid_examples, n_test_examples
```

```
(2936, 367, 367)
```

```
train_dataset, valid_dataset, test_dataset = data.random_split(
    dataset,
    [n_train_examples, n_valid_examples, n_test_examples]
)
```

2 – Flower Classification

Preprocessing - Demo

- Transform

```
# resize + convert to tensor
IMG_SIZE = 224

train_transforms = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(0.2),
    transforms.ToTensor(),
    transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
])

test_transforms = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
])
```

```
train_dataset.dataset.transform = train_transforms
valid_dataset.dataset.transform = test_transforms
test_dataset.dataset.transform = test_transforms
```

2 – Flower Classification

Dataloader - Demo

- Batch size: 256

```
BATCH_SIZE = 256

train_dataloader = data.DataLoader(
    train_dataset,
    shuffle=True,
    batch_size=BATCH_SIZE
)

valid_dataloader = data.DataLoader(
    valid_dataset,
    batch_size=BATCH_SIZE
)

test_dataloader = data.DataLoader(
    test_dataset,
    batch_size=BATCH_SIZE
)
```

2 – Flower Classification

Model – Train from Scratch - Demo

```
model = models.resnet18(weights=None)
```

```
in_features = model.fc.in_features
model.fc = nn.Linear(in_features, num_classes)
```

```
(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
(fc): Linear(in_features=512, out_features=1000, bias=True)
```

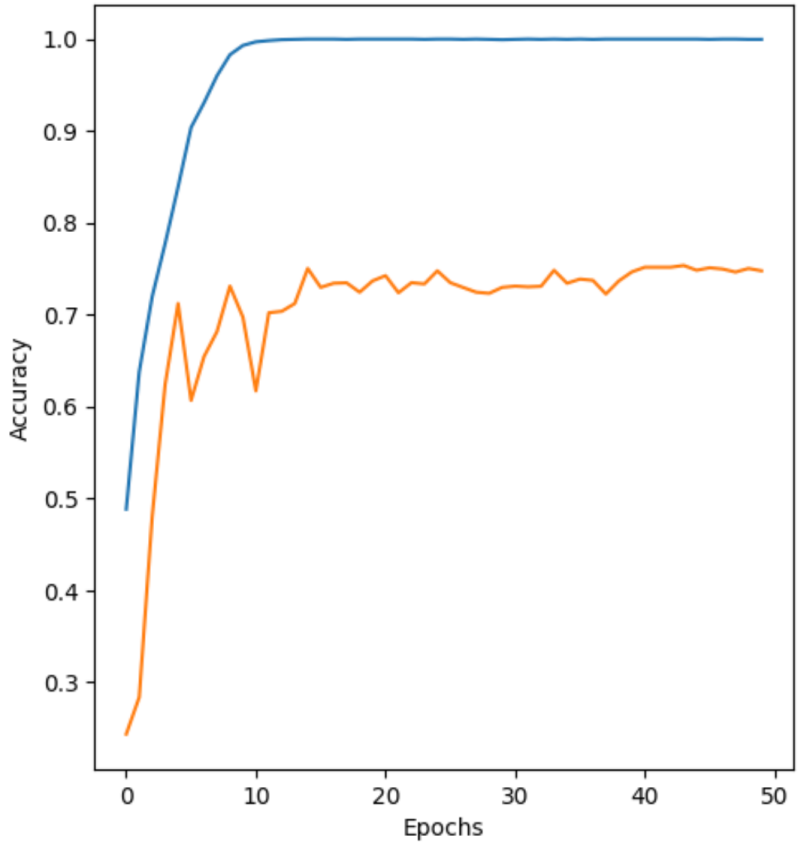
```
(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
(fc): Linear(in_features=512, out_features=5, bias=True)
```

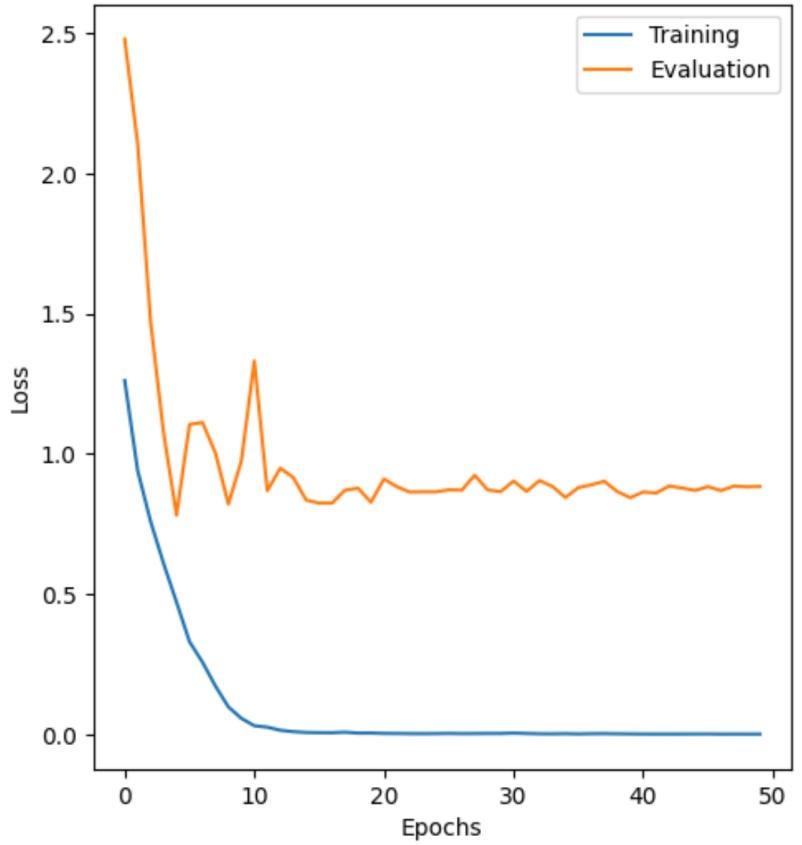
```
Total params: 11,179,077
Trainable params: 11,179,077
Non-trainable params: 0
----------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 62.79
Params size (MB): 42.64
Estimated Total Size (MB): 106.00
----------------------------------------
```

2 – Flower Classification

Model – Train from Scratch - Demo

- Testing: 73.11%





2 – Flower Classification

Model – Transfer Learning - Demo

- Weight: ImageNet1K

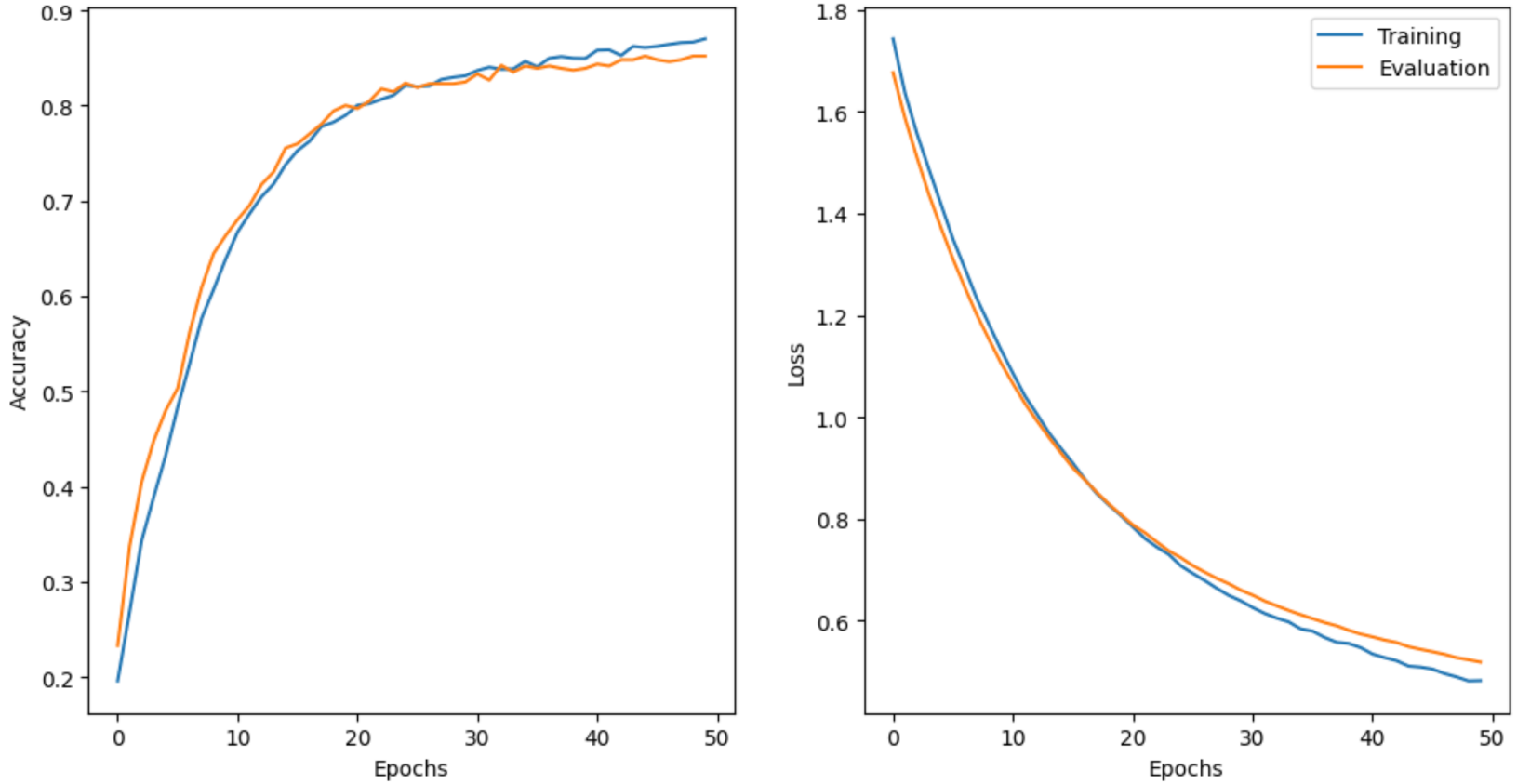
```
transfer_model = models.resnet18(
    weights=models.ResNet18_Weights.IMAGENET1K_V1
)
# freeze FE
for param in transfer_model.parameters():
   param.requires_grad = False

in_features = transfer_model.fc.in_features
transfer_model.fc = nn.Linear(in_features, num_classes)
```

```
AdaptiveAvgPool2d-67            [-1, 512, 1, 1]               0
           Linear-68                     [-1, 5]           2,565
================================================================
Total params: 11,179,077
Trainable params: 2,565
Non-trainable params: 11,176,512
```

2 – Flower Classification

Model – Transfer Learning - Demo

- Testing: 86.98%



2 – Flower Classification

Model – Fine Tuning - Demo

- Freeze the first 50 layers

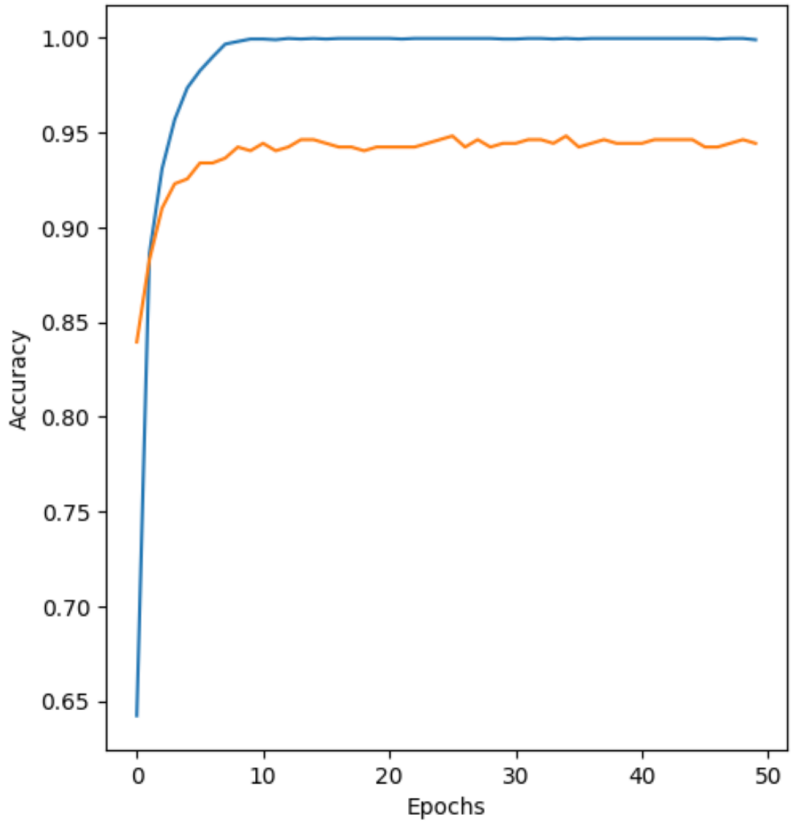
```
fine_tuning_model = models.resnet18(
    weights=models.ResNet18_Weights.IMAGENET1K_V1
)

# freeze the first 50 layers
num_layers = 50
for index, param in enumerate(fine_tuning_model.parameters()):
    if index < num_layers:
        param.requires_grad = False
```

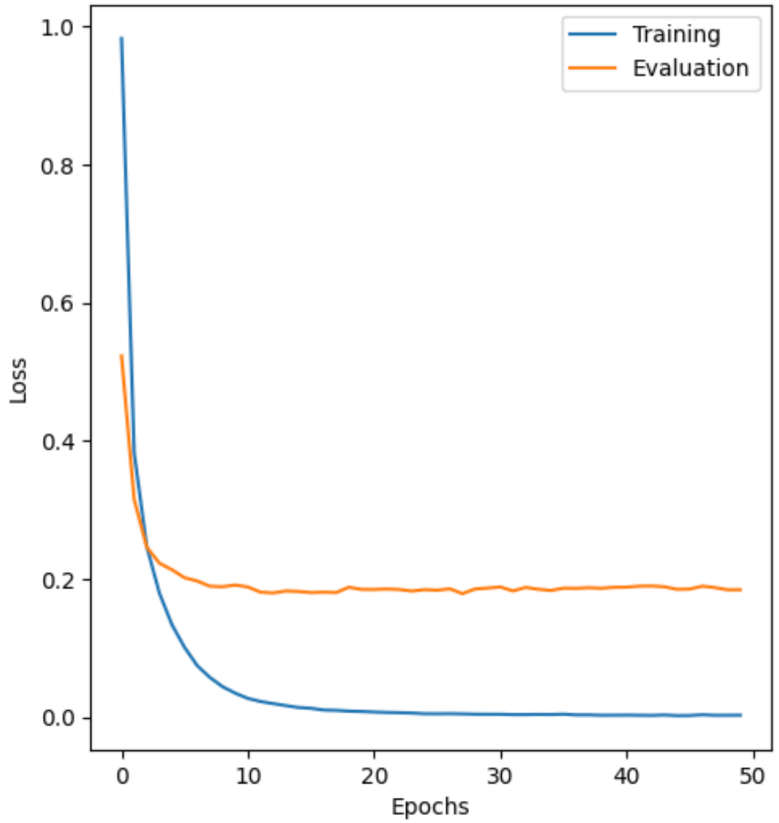
```
AdaptiveAvgPool2d-67            [-1, 512, 1, 1]               0
           Linear-68                     [-1, 5]           2,565
================================================================
Total params: 11,179,077
Trainable params: 4,855,301
Non-trainable params: 6,323,776
----------------------------------------------------------------
```

2 – Flower Classification

Model – Fine Tuning - Demo

- Testing: 93.13%





2 – Flower Classification

Model – Fine Tuning (As Initilization) - Demo

- Unfreeze all layers

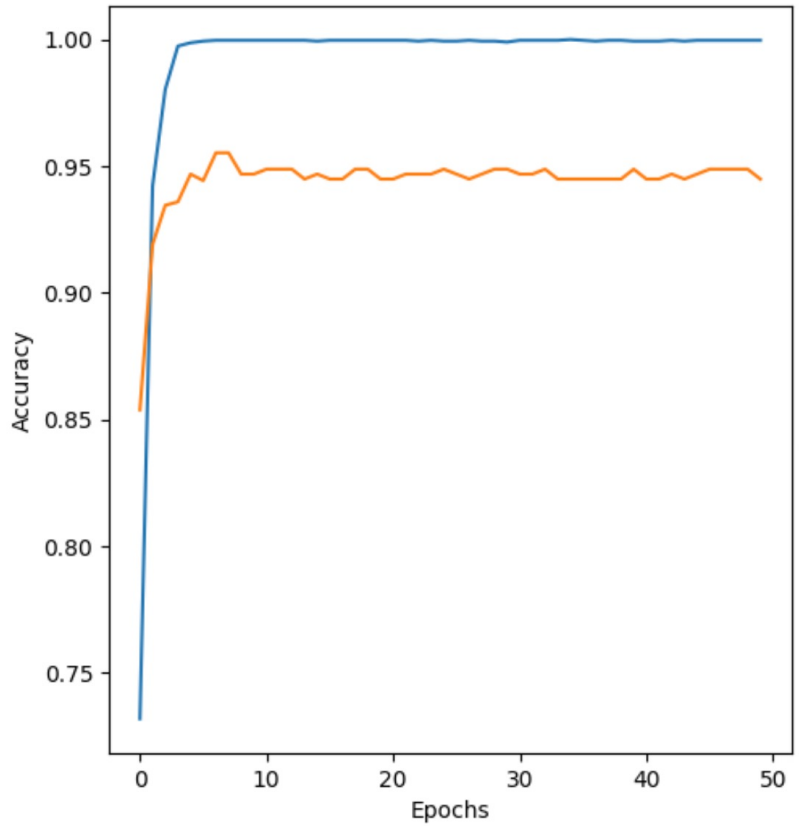
```
initilization_model = models.resnet18(
    weights=models.ResNet18_Weights.IMAGENET1K_V1
    )

in_features = initilization_model.fc.in_features
initilization_model.fc = nn.Linear(in_features, num_classes)
```

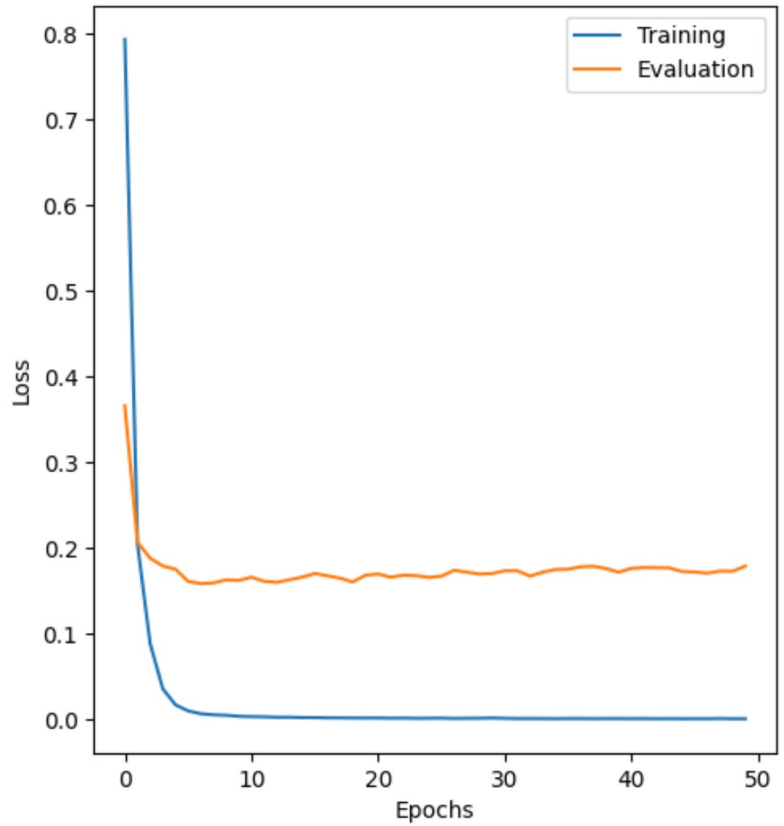
```
AdaptiveAvgPool2d-67            [-1, 512, 1, 1]                0
           Linear-68                     [-1, 5]            2,565
================================================================
Total params: 11,179,077
Trainable params: 11,179,077
Non-trainable params: 0
----------------------------------------------------------------
```

2 – Flower Classification

Model – Fine Tuning (As Initilization) - Demo

- Testing: 95.27%

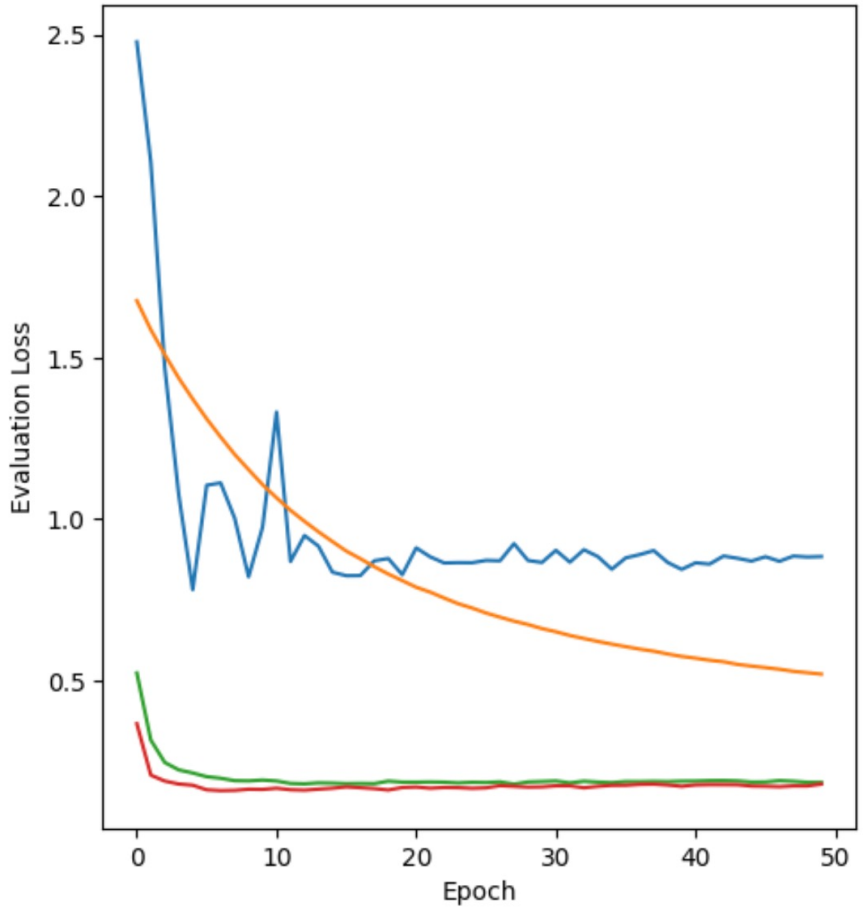


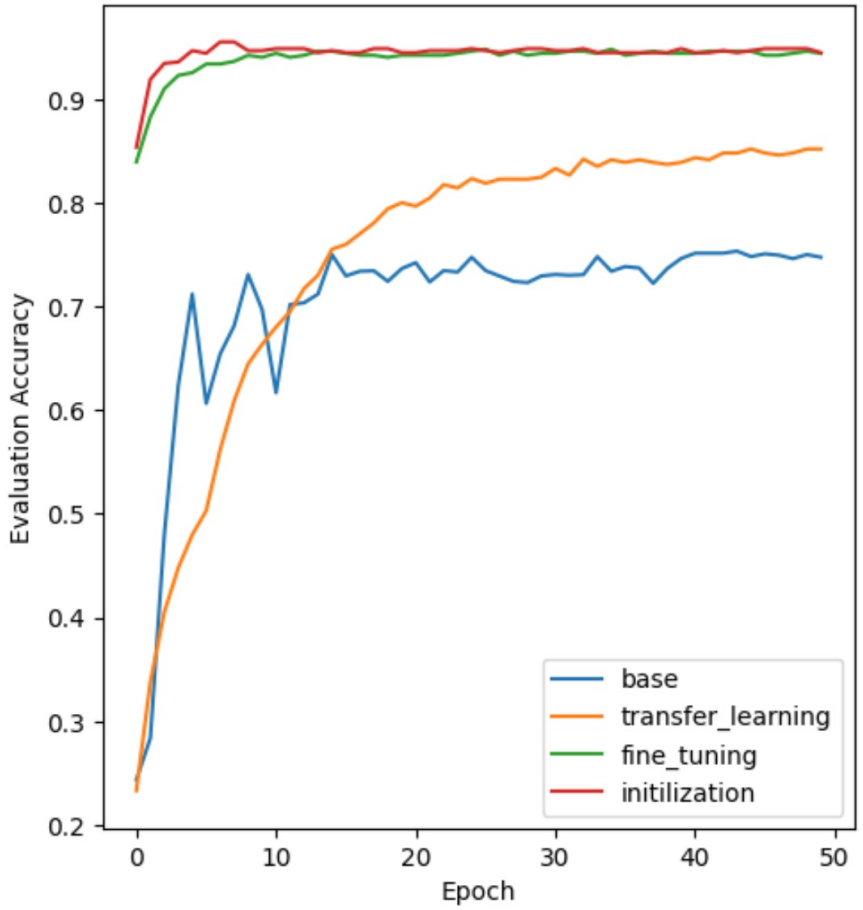


2 – Flower Classification

Comparing

Model	Accuracy
Base	73.11
Transfer	86.98
Fine-Tune	93.11
Initilization	95.27





3 – Image Retrieval

Database

Image Database

Query Image

Search for

Similar images

3 – Image Retrieval

Image Similarity

Similarity/
Distance Function

MSE
L1 – Distance
L2 – Distance
Cosine Similarity
Normalized Cross-Correlation

Similarity Score /
Distance

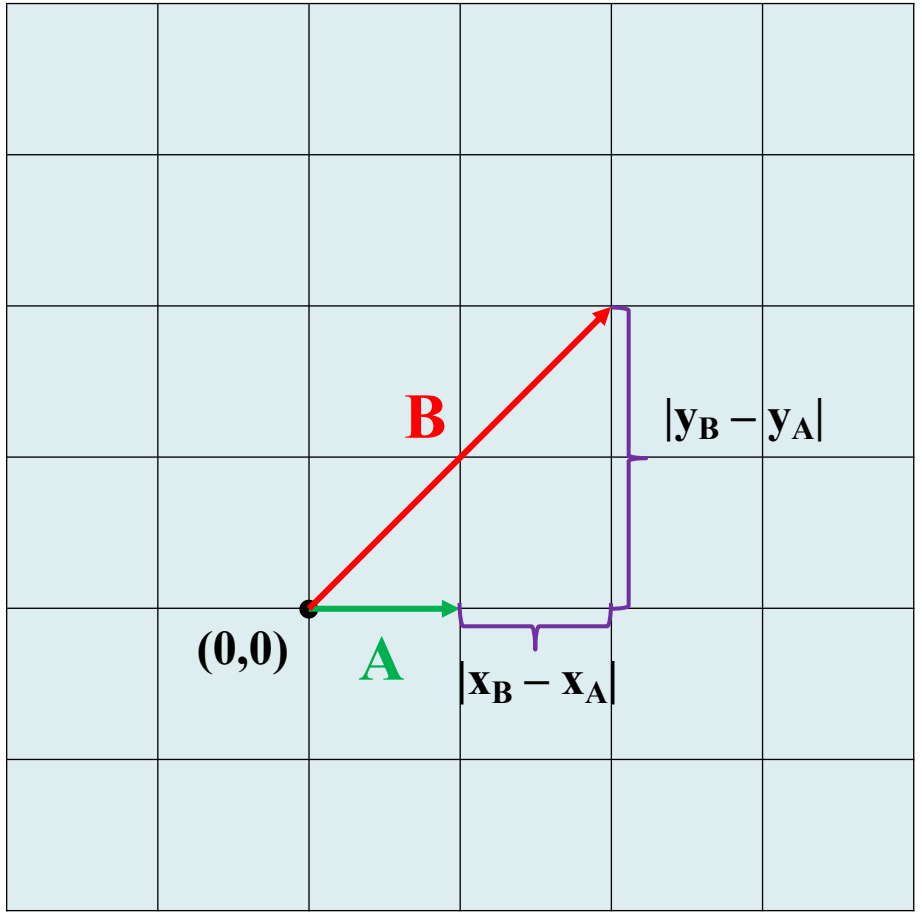
3 – Image Retrieval

L1 Distance

❖ **Compute L1 distance of 2 vectors**

$$L1 = \sum_{i=1}^{n} |x_i - y_i|$$

A	1	0
B	2	2

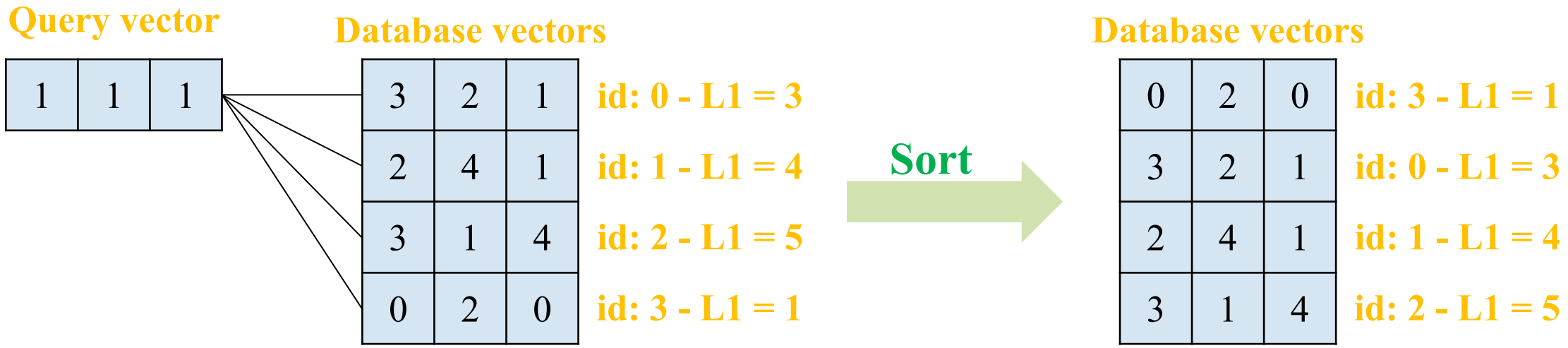


3 – Image Retrieval

L1 Distance

❖ **Compute L1 distance of 2 vectors**

$$L1 = \sum_{i=1}^{n} |x_i - y_i|$$



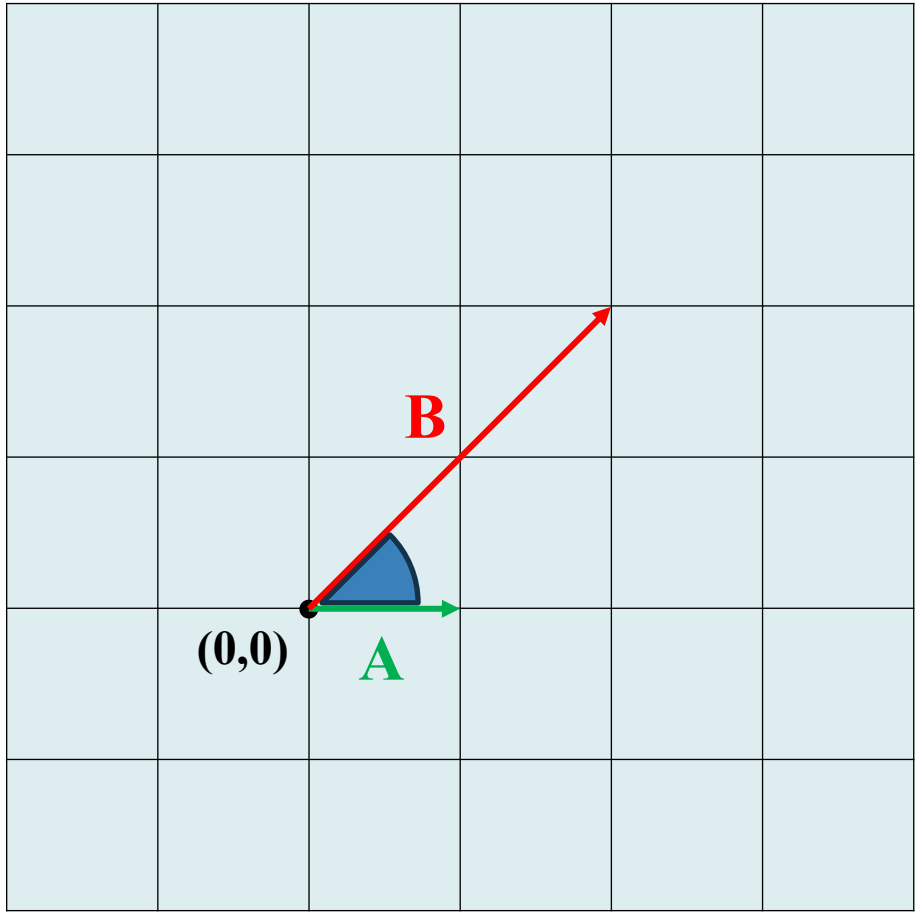
3 – Image Retrieval

Cosine Similarity

❖ **Compute Cosine Similarity of 2 vectors**

$$Cos(A, B) = \frac{A \cdot B}{\|A\|\|B\|}$$

A	1	0
B	2	2



3 – Image Retrieval

Cosine Similarity

❖ **Compute Cosine Similarity of 2 vectors**

$$Cos(A, B) = \frac{A \cdot B}{\|A\|\|B\|}$$

Query vector

1	1	1

Database vectors

3	2	1	id: 0 - Cs = 0.93
2	4	1	id: 1 - Cs = 0.88
3	1	4	id: 2 - Cs = 0.91
0	2	0	id: 3 - Cs = 0.58

Sort

Database vectors

3	2	1	id: 0 - Cs = 0.93
3	1	4	id: 2 - Cs = 0.91
2	4	1	id: 1 - Cs = 0.88
0	2	0	id: 3 - Cs = 0.58

3 – Image Retrieval

Naïve Approach

3 – Image Retrieval

Naïve Approach

3	2	1	0	0	3
0	3	3	1	1	0
3	1	4	1	1	0
2	4	1	1	0	4
1	0	3	0	3	0
3	4	4	3	3	4

Example: size 9 x 9 image

Flatten into a vector with 9 x 9 = 81 dimension

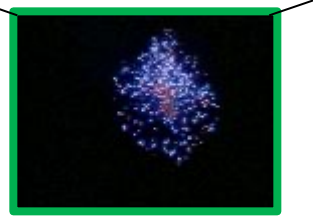
3	2	1	0	0	3	...	...	...	...	3	4	4	3	3	4

dim = 81

Similarity / Distance Function

Similar Vector

3	2	...	...	...	3	4

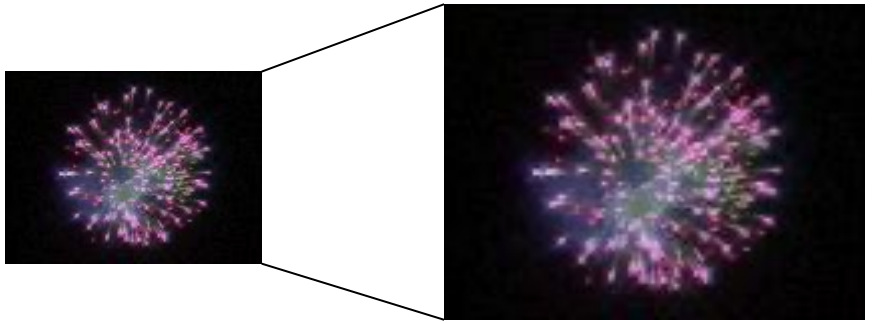


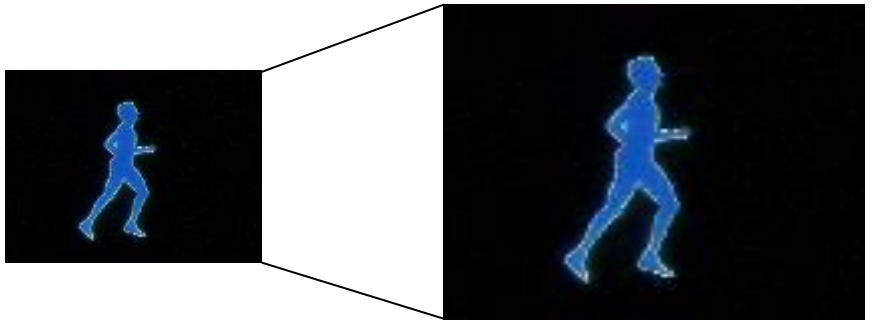
Different Vector

0	3	...	...	...	1	0

3 – Image Retrieval

Naïve Approach – Data Preparation – Demo






```python
# Image transformation: resize and convert to tensor
transform = transforms.Compose([
    transforms.Resize((86, 128)),
    transforms.ToTensor(),
])

# Function to load and preprocess the image
def load_image(path, transform):
    img = Image.open(path).convert('RGB')
    img_tensor = transform(img)
    return img_tensor
```

3 – Image Retrieval

Naïve Approach – L1 Distance – Demo

Query image

```
# Load query image
query_img_path = '/content/drive/MyDrive/img/q2.jpg'
query_tensor = load_image(query_img_path, transform)
print(f'Img tensor shape: {query_tensor.shape}')

# Display query image
plt.figure(figsize=(4, 4))
plt.imshow(transforms.ToPILImage()(query_tensor))
plt.title("Query Image")
plt.axis('off')
plt.show()
```

3 – Image Retrieval

Naïve Approach – L1 Distance – Demo

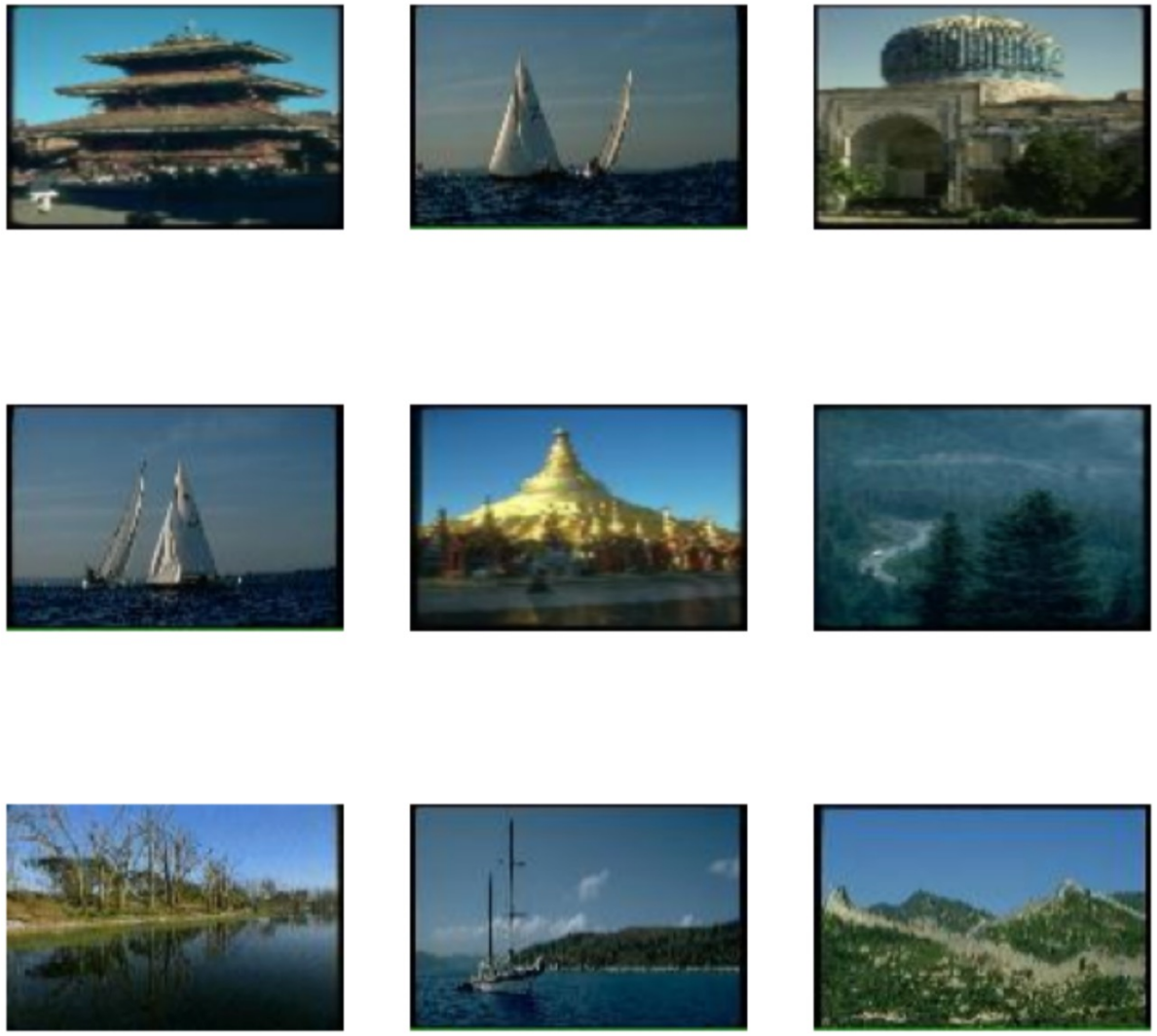
```
# Calculate L1 distance (absolute difference)
distances = torch.abs(data_tensors_flat - query_tensor_flat)
distances = torch.sum(distances, dim=1)

# Sort the distances and get indices
sorted_indices = torch.argsort(distances, descending=False)

# Print top 8 values and their indices
for i in range(8):
    index = sorted_indices[i].item()
    distance = distances[index].item()
    print(f'Index: {index}, Distance: {distance}')
```

```
Index: 3425, Distance: 0.0
Index: 3077, Distance: 5221.431640625
Index: 2611, Distance: 5247.2900390625
Index: 3109, Distance: 5289.16845703125
Index: 3445, Distance: 5358.61962890625
Index: 960, Distance: 5378.94873046875
Index: 4788, Distance: 5395.0078125
Index: 3094, Distance: 5435.6904296875
```

3 – Image Retrieval

Naïve Approach – L1 Distance – Demo

```
# Display top 8 similar images
fig = plt.figure(figsize=(8, 8))
columns = 3
rows = 3
for i in range(columns * rows):
    index = sorted_indices[i].item()
    img = data_tensors[index]

    ax = fig.add_subplot(rows, columns, i + 1)
    ax.axis('off')
    ax.imshow(to_pil_image(img))

plt.show()
```



3 – Image Retrieval

Naïve Approach – Cosine Similarity– Demo

Query image

```
# Load query image
query_img_path = '/content/drive/MyDrive/img/q2.jpg'
query_tensor = load_image(query_img_path, transform)
print(f'Img tensor shape: {query_tensor.shape}')

# Display query image
plt.figure(figsize=(4, 4))
plt.imshow(transforms.ToPILImage()(query_tensor))
plt.title("Query Image")
plt.axis('off')
plt.show()
```

3 – Image Retrieval

Naïve Approach – Cosine Similarity– Demo

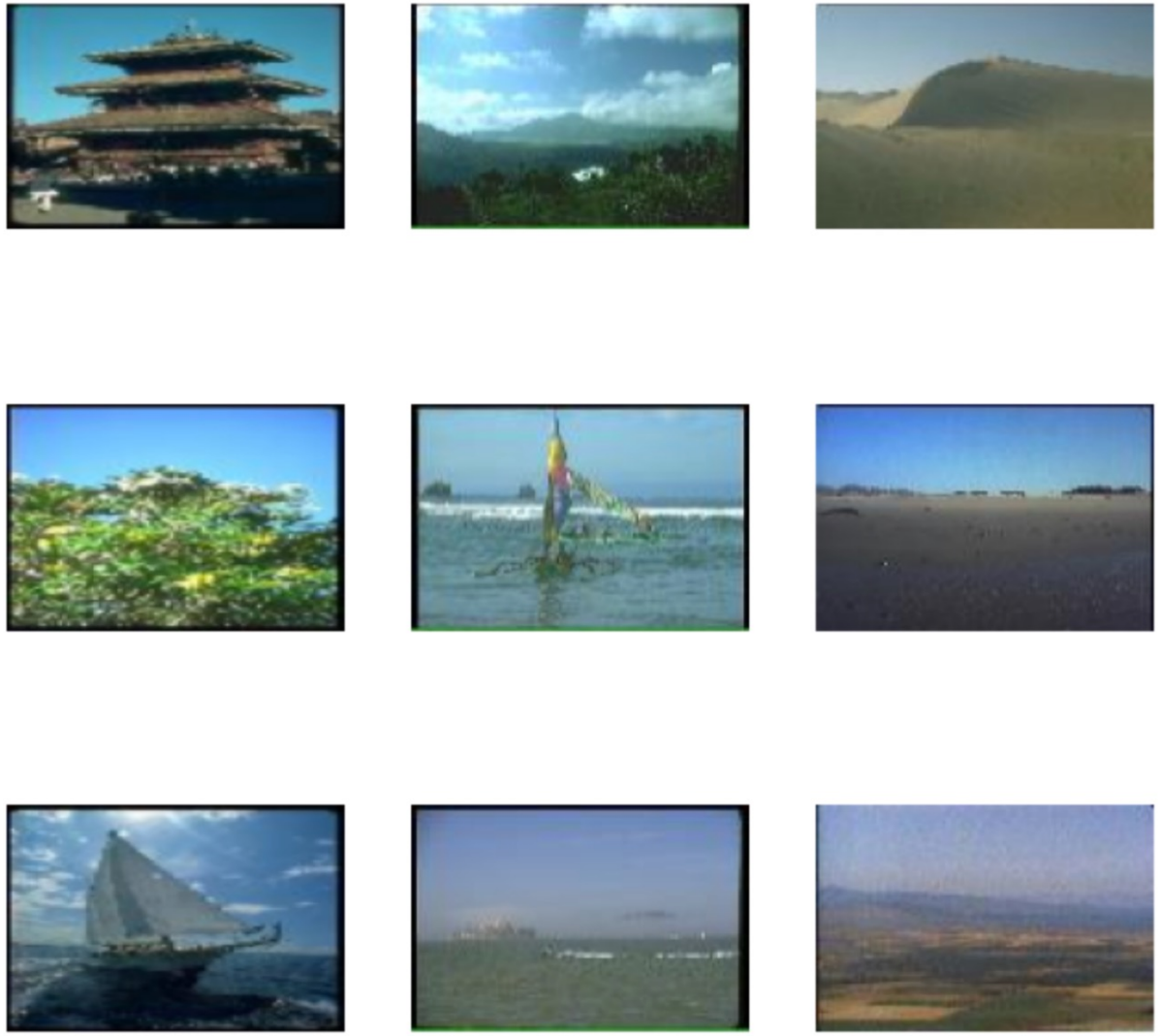
```python
# Calculate cosine similarity
sims = cosine_similarity(query_tensor_flat, data_tensors_flat)

# Sort the similarities and get indices
sorted_indices = torch.argsort(sims, descending=True)

# Print top 8 values and their indices
for i in range(8):
    index = sorted_indices[i].item()
    similarity = sims[index].item()
    print(f'Index: {index}, Similarity: {similarity}')
```

```
Index: 3425, Similarity: 0.999997615814209
Index: 3306, Similarity: 0.8691879510879517
Index: 4645, Similarity: 0.8683653473854065
Index: 3330, Similarity: 0.8676211833953857
Index: 3374, Similarity: 0.867255449295044
Index: 4317, Similarity: 0.8664509057998657
Index: 3053, Similarity: 0.8658316135406494
Index: 3780, Similarity: 0.8647940158843994
```

3 – Image Retrieval

Naïve Approach – Cosine Similarity– Demo

```
# Display top 8 similar images
fig = plt.figure(figsize=(8, 8))
columns = 3
rows = 3
for i in range(columns * rows):
    index = sorted_indices[i].item()
    img = data_tensors[index]

    ax = fig.add_subplot(rows, columns, i + 1)
    ax.axis('off')
    ax.imshow(to_pil_image(img))

plt.show()
```



3 – Image Retrieval

Using Pretrained Model

Feature Extractor (ResNet18)

Feature Extractor (ResNet18)

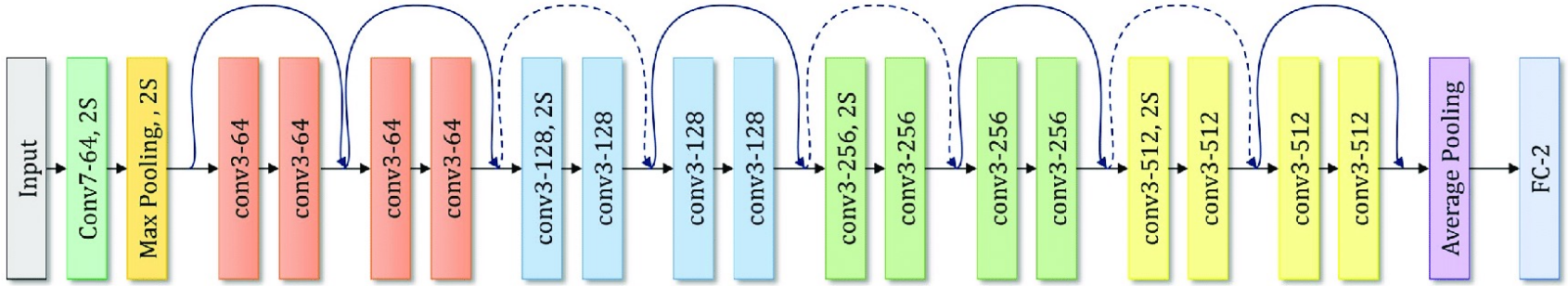
Similarity/ Distance Function

MSE
L1 – Distance
L2 – Distance
Cosine Similarity
Normalized Cross-Correlation

Similarity Score / Distance

3 – Image Retrieval

Using Pretrained Model



Feature Extractor

Pretraied Model (ResNet18)

Vector
Matrix

3 – Image Retrieval

Using Pretrained Model – Demo

- Load pretrained ResNet18 model

```
# Load the pretrained ResNet18 model
resnet18 = models.resnet18(
    weights=models.ResNet18_Weights.IMAGENET1K_V1
)
# Remove the last layer to use as a feature extractor
modules = list(resnet18.children())[:-1]
resnet18 = torch.nn.Sequential(*modules)

resnet18.eval()  # Set the model to evaluation mode
```

3 – Image Retrieval

Using Pretrained Model – Demo

- Feature Extraction

```python
# Function to preprocess and extract features
def extract_features(tensor, model):
    with torch.no_grad():
        # Add batch dimension and get features
        features = model(tensor.unsqueeze(0))
        # Flatten the features
        features = features.view(features.size(0), -1)
        return features

# Extract features for each image and store them
feature_list = []
for tensor in data_tensors:
    # print(tensor.unsqueeze(0).shape)
    features = extract_features(tensor, resnet18)
    feature_list.append(features)

# Stack all features into a single tensor
feature_tensor = torch.stack(feature_list).squeeze()
print(feature_tensor.shape)
torch.save(feature_tensor, '/content/drive/MyDrive/img/images_mr_features.pt')
```

3 – Image Retrieval

Using Pretrained Model – Demo

❖ Result

```
# Load the query image and extract features
query_tensor = load_image(query_img_path, transform)
query_features = extract_features(query_tensor, resnet18)

# Load the feature tensor
feature_tensor = torch.load('/content/drive/MyDrive/img/images_mr_features.pt')

# Calculate cosine similarity
sims = cosine_similarity(query_features, feature_tensor)

# Sort the similarities and get indices
sorted_indices = torch.argsort(sims, descending=True)

# Print top 8 values and their indices
for i in range(8):
    index = sorted_indices[i].item()
    similarity = sims[index].item()
    print(f'Index: {index}, Similarity: {similarity}')
```

```
Index: 3425, Similarity: 1.0000001192092896
Index: 3407, Similarity: 0.6934022903442383
Index: 370, Similarity: 0.687757670879364
Index: 3214, Similarity: 0.6866506338119507
Index: 3667, Similarity: 0.6857972741127014
Index: 3509, Similarity: 0.6846740245819092
Index: 248, Similarity: 0.6844337582588196
Index: 3793, Similarity: 0.6839574575424194
```

4 – IR with Vector Databasse

Database

Data

- Refers to a collection of information
- Forms: numbers, text, or other types of media

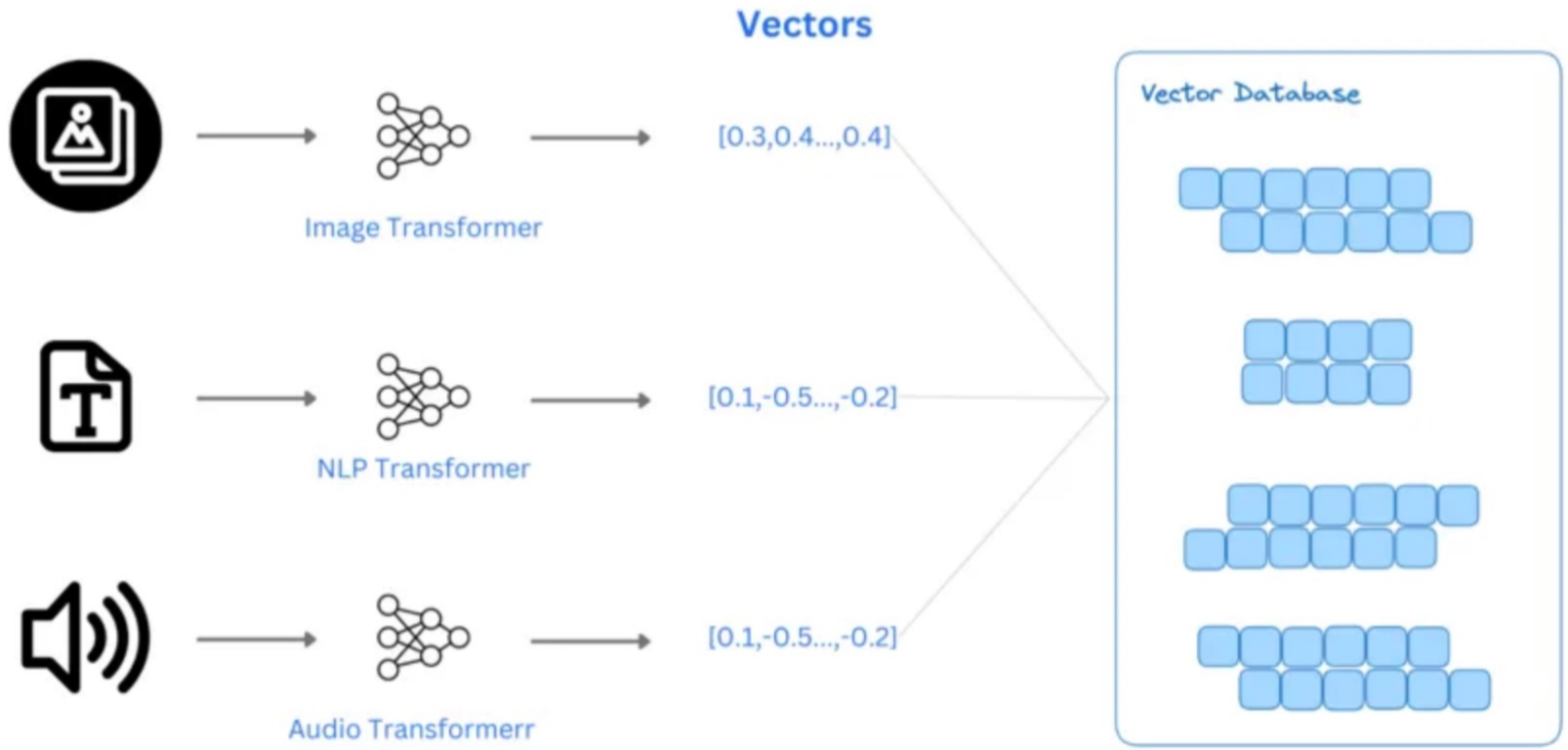
Name: Thai
Age: 20
Image: link

Database

- An organized collection of structured information, or data
- Can be easily accessed, restored, administered and updated

4 – IR with Vector Databasse

Vector Database



4 – IR with Vector Databasse

Indexing

Feature Extractor (ResNet18)

Feature Extractor (ResNet18)

Feature Extractor (ResNet18)

Clustering

Indexing

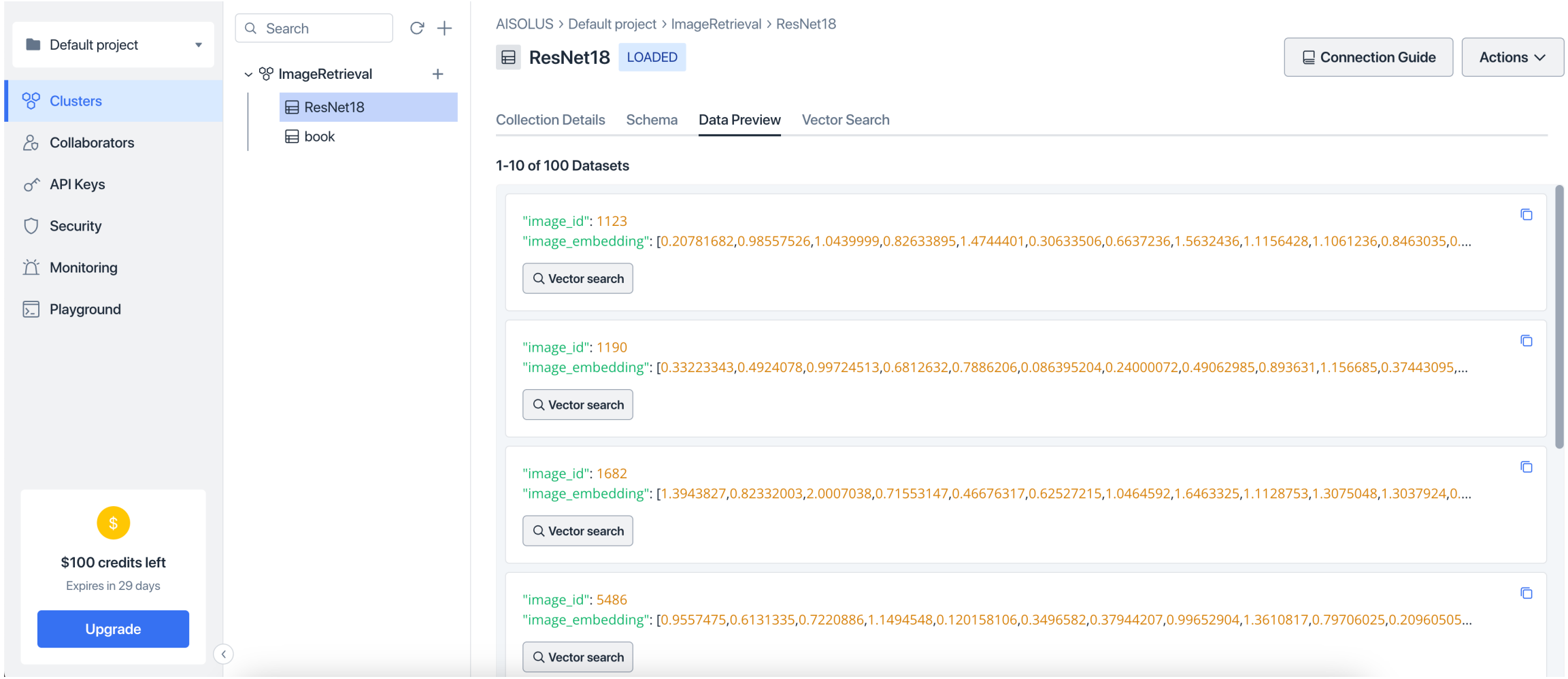
4 – IR with Vector Databasse

! **PyMilvus**

AI VIET NAM
@aivietnam.edu.vn

4 – IR with Vector Databasse

Zilliz Cloud



4 – IR with Vector Databasse

Connect a Database – Demo

```
# connect database
from pymilvus import MilvusClient, connections

CLUSTER_ENDPOINT = "###"
TOKEN = "###"

connections.connect(
    "default",
    uri=CLUSTER_ENDPOINT,
    token=TOKEN
)

client = MilvusClient(
    uri=CLUSTER_ENDPOINT,
    token=TOKEN
)
```

4 – IR with Vector Databasse

Create a Collection – Demo

```python
from pymilvus import Collection, DataType, FieldSchema, CollectionSchema, utility

# create a collection
COLLECTION_NAME = "ResNet18"
check_collection = utility.has_collection(COLLECTION_NAME)
if check_collection:
    drop_result = utility.drop_collection(COLLECTION_NAME)

EMBEDDING_DIM=512
image_id = FieldSchema(
    name="image_id",
    dtype=DataType.INT64,
    is_primary=True,
    description="Image ID"
)

image_embedding = FieldSchema(
    name="image_embedding",
    dtype=DataType.FLOAT_VECTOR,
    dim=EMBEDDING_DIM
)

schema = CollectionSchema(
    fields=[image_id, image_embedding],
    auto_id=False,
    description="Image Retrieval Using ResNet18")

collection = Collection(
    name=COLLECTION_NAME,
    schema=schema
)
```

4 – IR with Vector Databasse

Insert Entities – Demo

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

def preprocess_image(image_path):
    img = Image.open(image_path).convert('RGB')
    processed_img = transform(img)
    return processed_img

def extract_feature(processed_image):
    input = processed_image.unsqueeze(0).to(device)
    with torch.no_grad():
        prediction = resnet18_model(input)
    return prediction.squeeze().cpu().tolist()

# extract feature
image_ids = sorted([
    int(image_name.split(".")[0]) for image_name in os.listdir(image_folder)
])
image_embeddings = []
for file_name in tqdm(image_ids):
    file_name = str(file_name) + ".jpg"
    image_path = os.path.join(image_folder, file_name)
    processed_image = preprocess_image(image_path)
    processed_image = extract_feature(processed_image)
    image_embeddings.append(processed_image)
```

4 – IR with Vector Databasse

Indexing – Demo

```
collection.num_entities
```

```
9908
```

```
index_params = {
    "index_type": "IVF_FLAT",
    "metric_type": "L2",
    "params": {}
}
collection.create_index(
    field_name=image_embedding.name,
    index_params=index_params
)
```

```
Status(code=0, message=)
```

```
collection.drop_index()
```

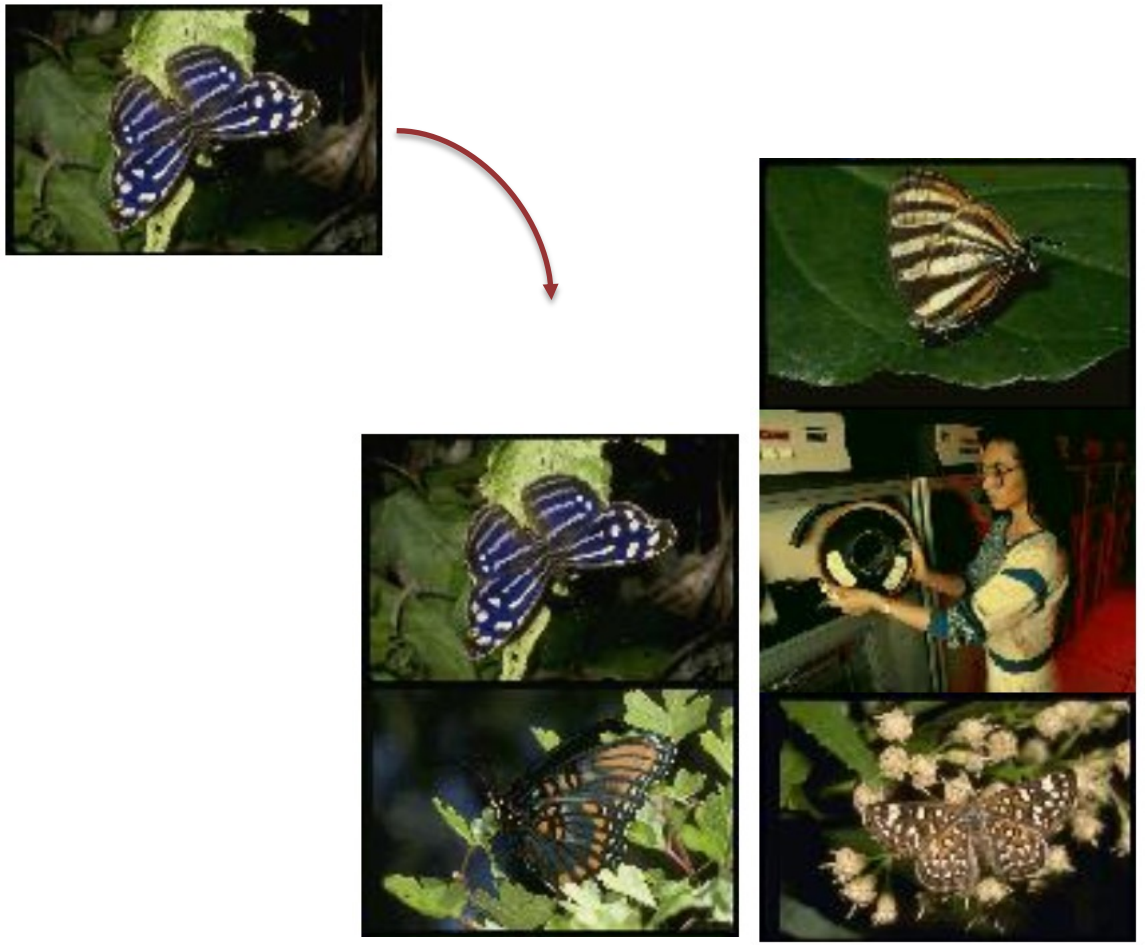
```
collection.has_index()
```

```
True
```

4 – IR with Vector Databasse

Searching – Demo

```
# search
search_params = {
    "metric_type": "L2",
    "params": {"level": 2}
}
def search_images(image_path, topk=5):
    processed_image = preprocess_image(image_path)
    processed_image = extract_feature(processed_image)
    results = collection.search(
        [processed_image],
        anns_field=image_embedding.name,
        param=search_params,
        limit=topk,
        guarantee_timestamp=1
    )
    return results[0]
```



AI VIET NAM

@aivietnam.edu.vn

Thanks!

Any questions?