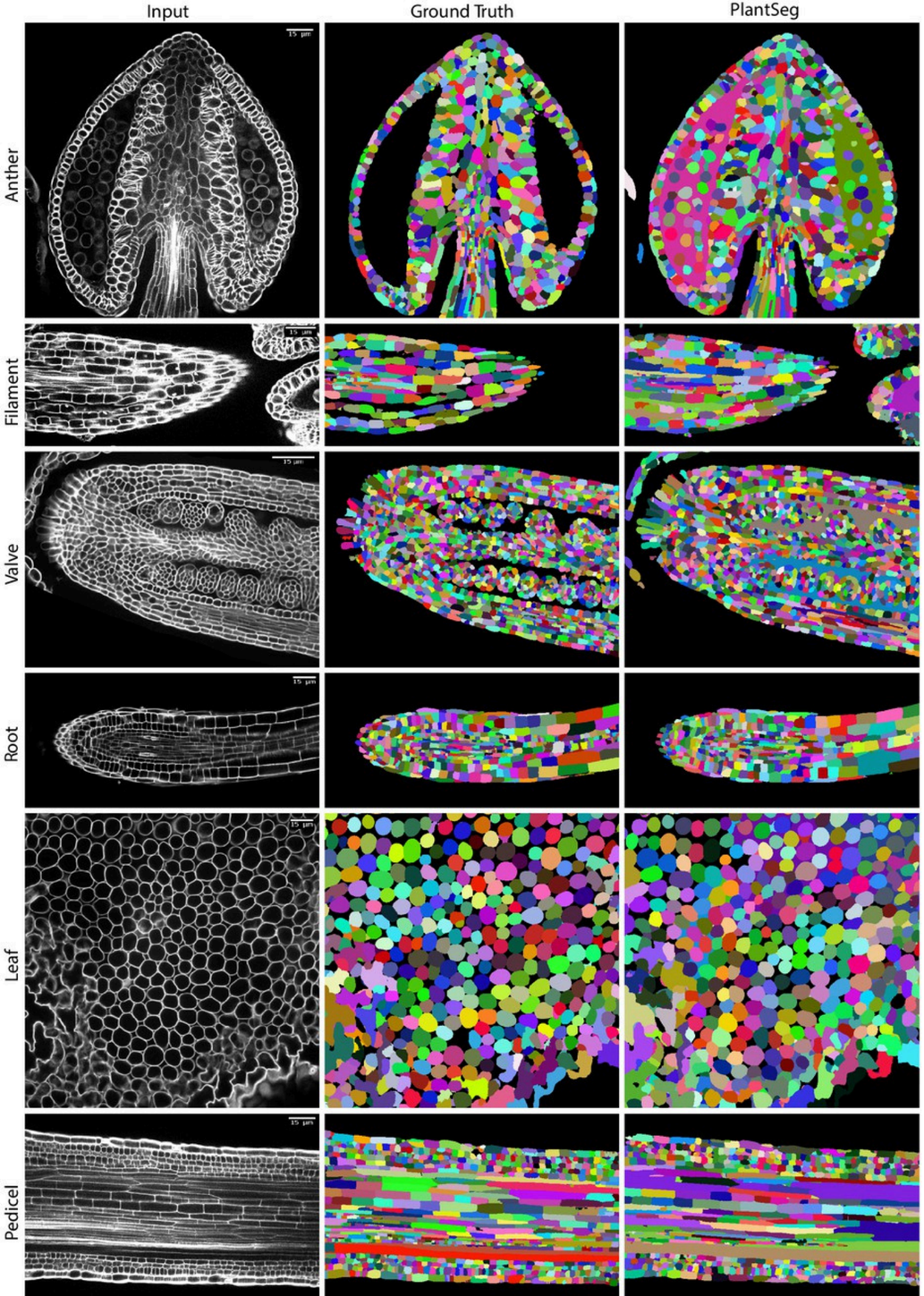
# Session // 05
# TRANSFER LEARNING

## FACULTY OF
## SCIENCE AND ENGINEERING
## +++

Diego Corona Lopez – AI Technical Specialist




# Agenda

**Introduction to Transfer Learning**

- Definition and core concepts

**Dataset Preparation**

- Creating PyTorch dataset classes
- Computing dataset statistics
- Data augmentation for medical images

**Baseline Architecture**

- U-Net architecture overview
- Implementation of encoder-decoder structure

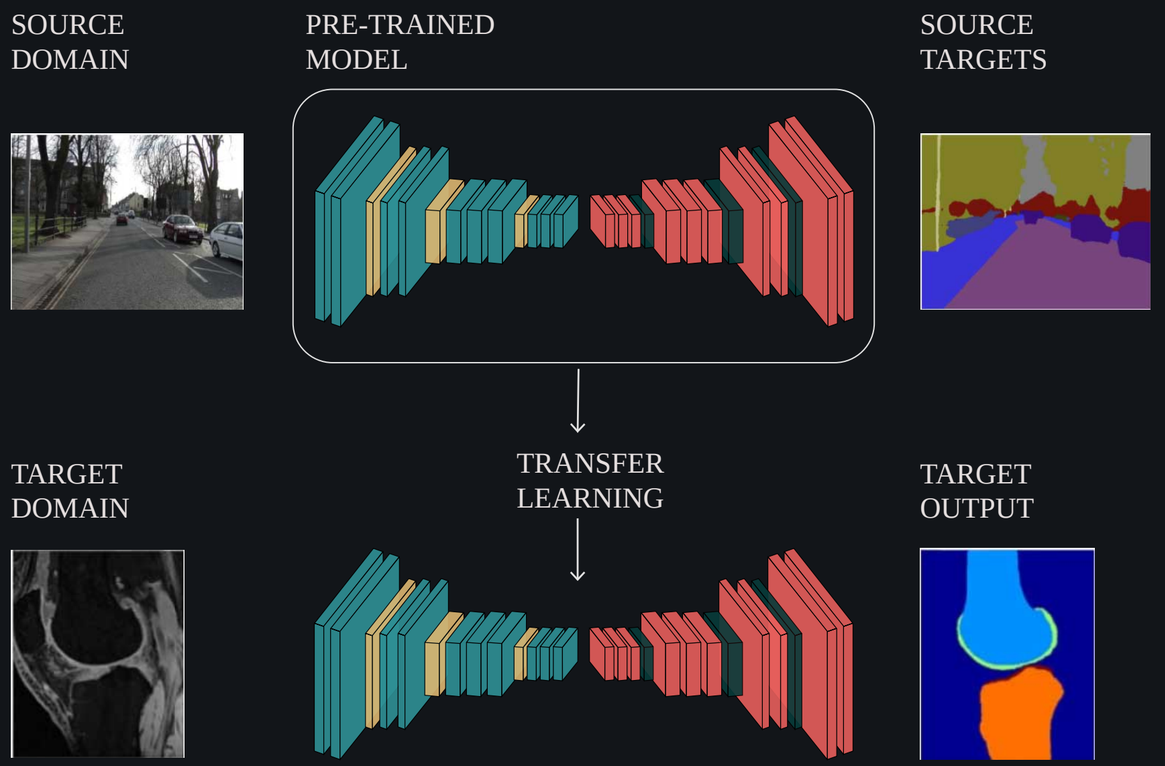
**Loss Functions for Segmentation**

- Dice Loss implementation

**Transfer Learning with Pre-trained Models**

- Introducing EfficientNet architecture
- Adapting pre-trained models for segmentation

**Advantages of transfer learning approach**




# WHAT IS TRANSFER LEARNING?

*A technique where a model developed for one task is reused as a starting point for a model on a second task. It leverages knowledge from pre-*

# When should YOU use transfer learning?

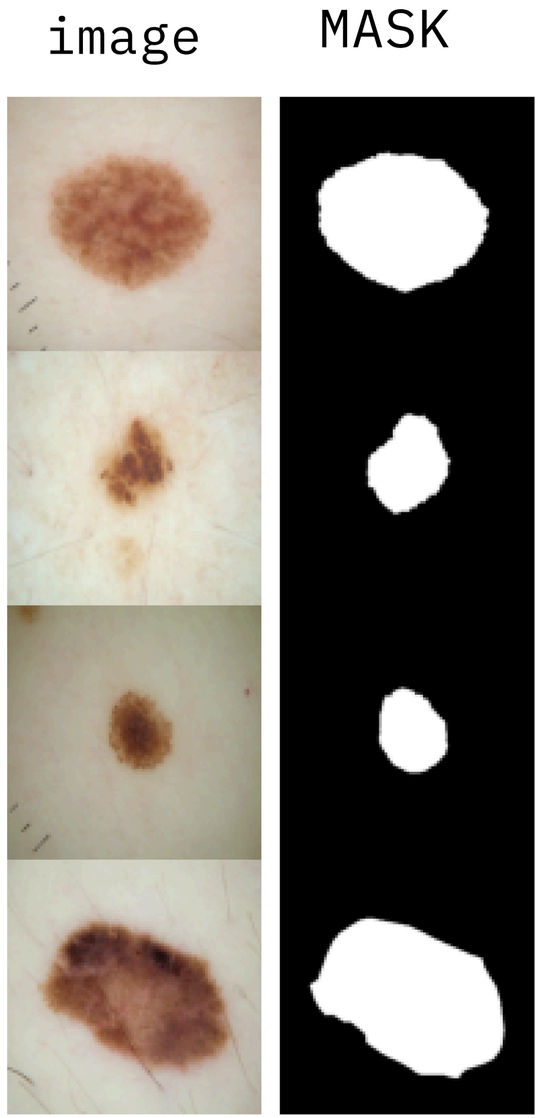
The **effectiveness** depends on similarity between source and target domains

| SCENARIO | EXAMPLE | BENEFIT |
|---|---|---|
| **Limited training data** | Medical imaging with few samples | Pre-trained features compensate for data scarcity |
| **Similar domains** | From natural images to satellite imagery | Underlying features (edges, textures) transfer well |
| **Time constraints** | Rapid prototyping needs | Accelerates model development cycle |
| **Hardware limitations** | Training with limited GPU access | Reduces computational requirements |
| **Preventing overfitting** | Small dataset applications | Regularization effect from pre-trained weights |

# ISIC 2016 Skin Lesion Dataset

Medical datasets like ISIC are typically smaller than general computer vision datasets, making transfer learning particularly valuable

- Contains dermoscopic images of skin lesions with expert-annotated segmentation masks
- 900 training images and 379 test images with corresponding binary masks
- Critical for developing automated diagnostic tools for early melanoma detection
- Challenging due to varying lesion sizes, shapes, colours, and skin types



# PYTORCH DATASET

```python
from torch.utils.data import Dataset, DataLoader

class ISICDataset(Dataset):
    def __init__(self, image_dir, mask_dir, img_transform=None, mask_transform=None):
        self.image_dir = Path(image_dir)
        self.mask_dir = Path(mask_dir)
        self.img_transform = img_transform
        self.mask_transform = mask_transform
        self.images = sorted(self.image_dir.glob("*.jpg"))

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        # Load image and mask
        img_name = self.images[idx]
        image = Image.open(self.image_dir / img_name).convert("RGB")
        mask = Image.open(self.mask_dir / f"{img_name.stem}_segmentation.png").convert("L")

        # Apply transformations if provided
        if self.img_transform:
            image = self.img_transform(image)
        if self.mask_transform:
            mask = self.mask_transform(mask)

        return image, mask
```

PyTorch's Dataset class is the foundation for data loading

**Three key methods:**

- __init__
- __len__
- __getitem__

Enables efficient data handling and batch processing

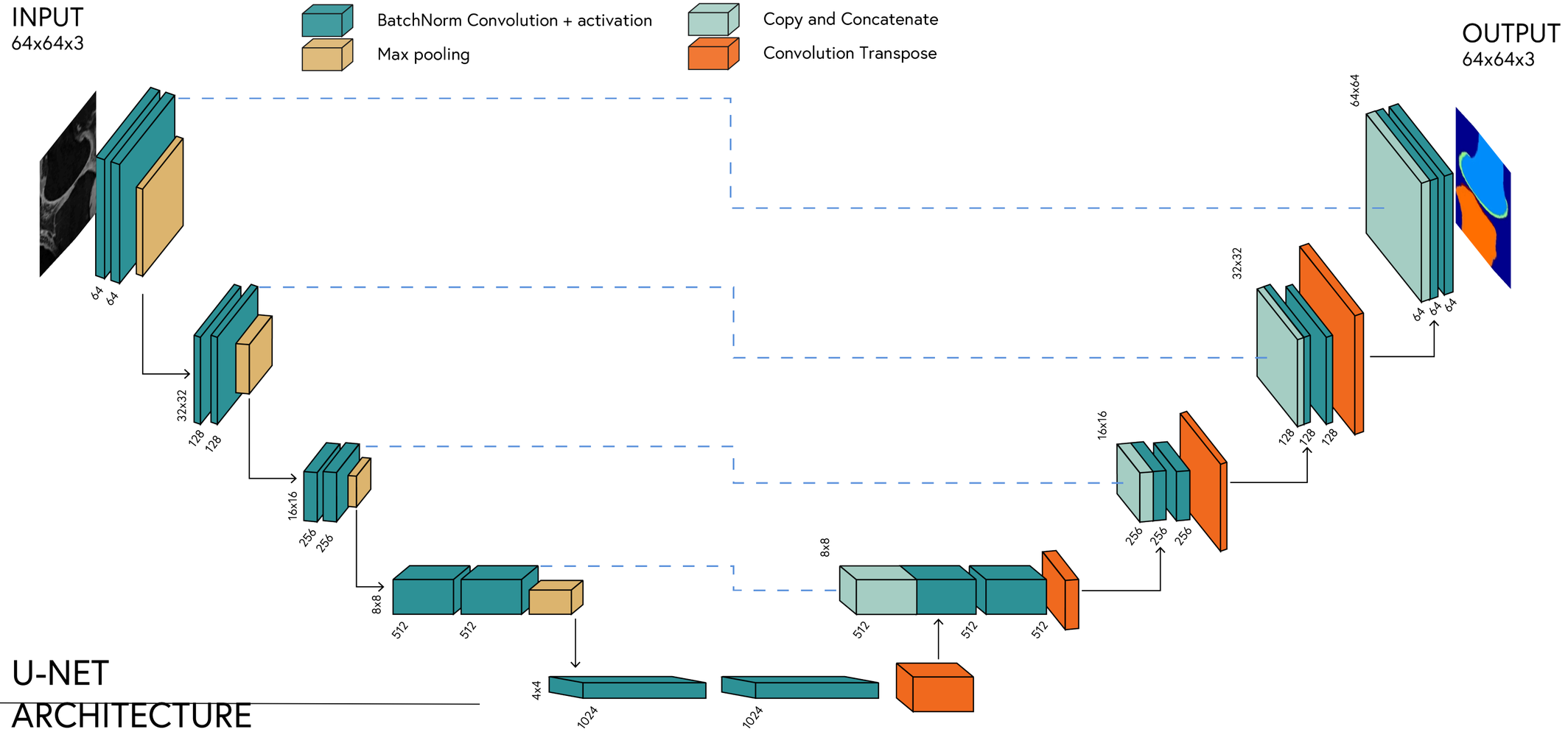
# DATA AUGMENTATION FOR SEGMENTATION

- **Synchronized** augmentation for images and masks
- **Careful selection** of transformations to preserve diagnostic features
- **Geometric transforms:** rotation, flipping, resizing
- **Color adjustments:** brightness, contrast

```python
import albumentations as A
from albumentations.pytorch import ToTensorV2

train_img_ts = A.Compose([
    A.Resize(64, 64),
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.5),
    A.Rotate(limit=10, p=0.5),
    A.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1, p=0.5),
    A.Normalize(mean=mean, std=std, p=1.0),
    ToTensorV2()
])
```

# U-NET

- **Encoder-decoder** architecture with skip connections
- Captures both **context and localisation** information
- Widely used for biomedical **image segmentation**



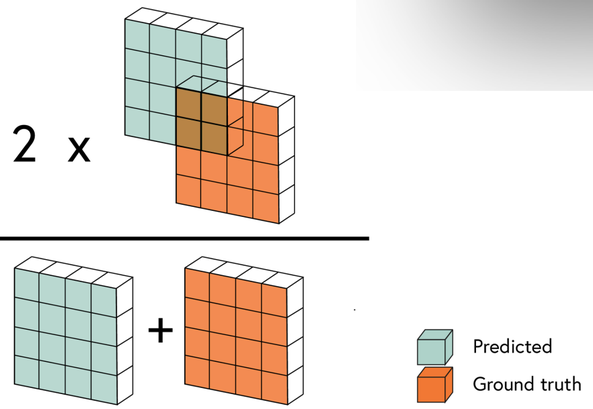

## U-NET ARCHITECTURE

# DICE LOSS

- Measures **overlap** between predicted and ground truth masks
- Handles **class imbalance** better than binary cross-entropy

```python
class DiceLoss(torch.nn.Module):
    def __init__(self, smooth=1.0):
        super().__init__()
        self.smooth = smooth

    def forward(self, y_pred, y_true):
        y_pred = y_pred.view(-1)
        y_true = y_true.view(-1).float()
        intersection = (y_pred * y_true).sum()
        union = y_pred.sum() + y_true.sum()
        dice = (2. * intersection + self.smooth) / (union + self.smooth)
        return torch.clamp(1 - dice, 0.0, 1.0)
```

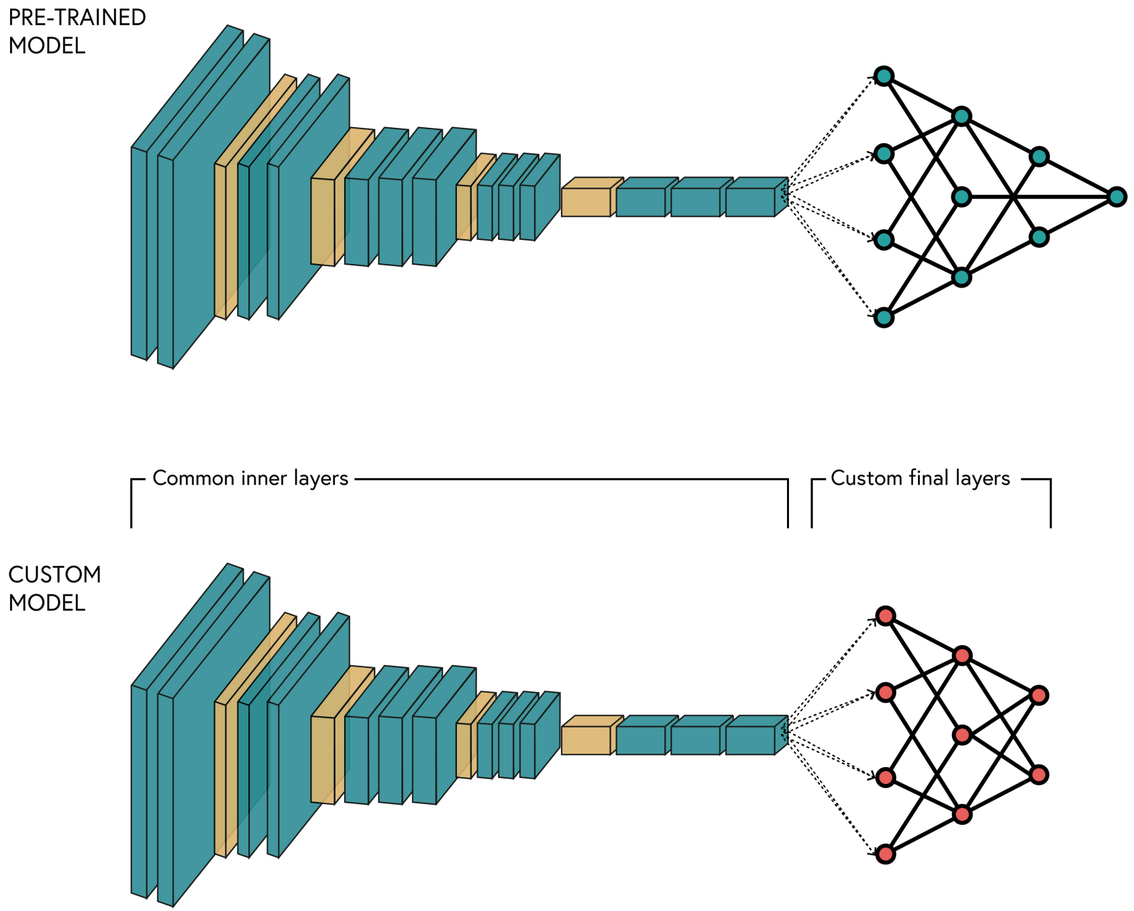
$$\text{DICE} = \frac{2 \times \text{OVERLAP}}{\text{TOTAL AREA}} = \frac{2 \times}{+}$$



# Transfer learning steps

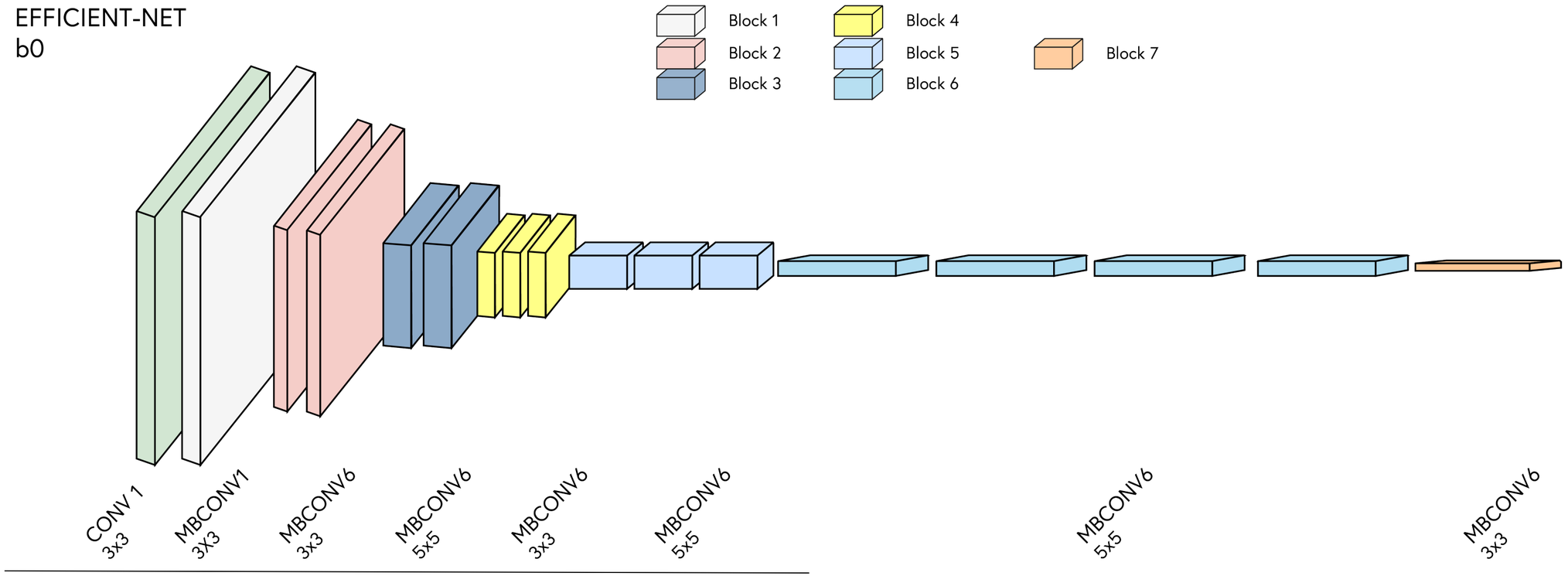
Steps:

1. Select source model
2. Feature extraction
3. Fine-tuning
4. Model adaptation



# PRE-TRAINED MODELS

- ResNet (11.7M-60M parameters)
- VGG (138M-144M parameters)
- Inception (6.8M-54M parameters)
- EfficientNet (5.3M-66M parameters)
- MobileNet (4.2M-6.9M parameters)



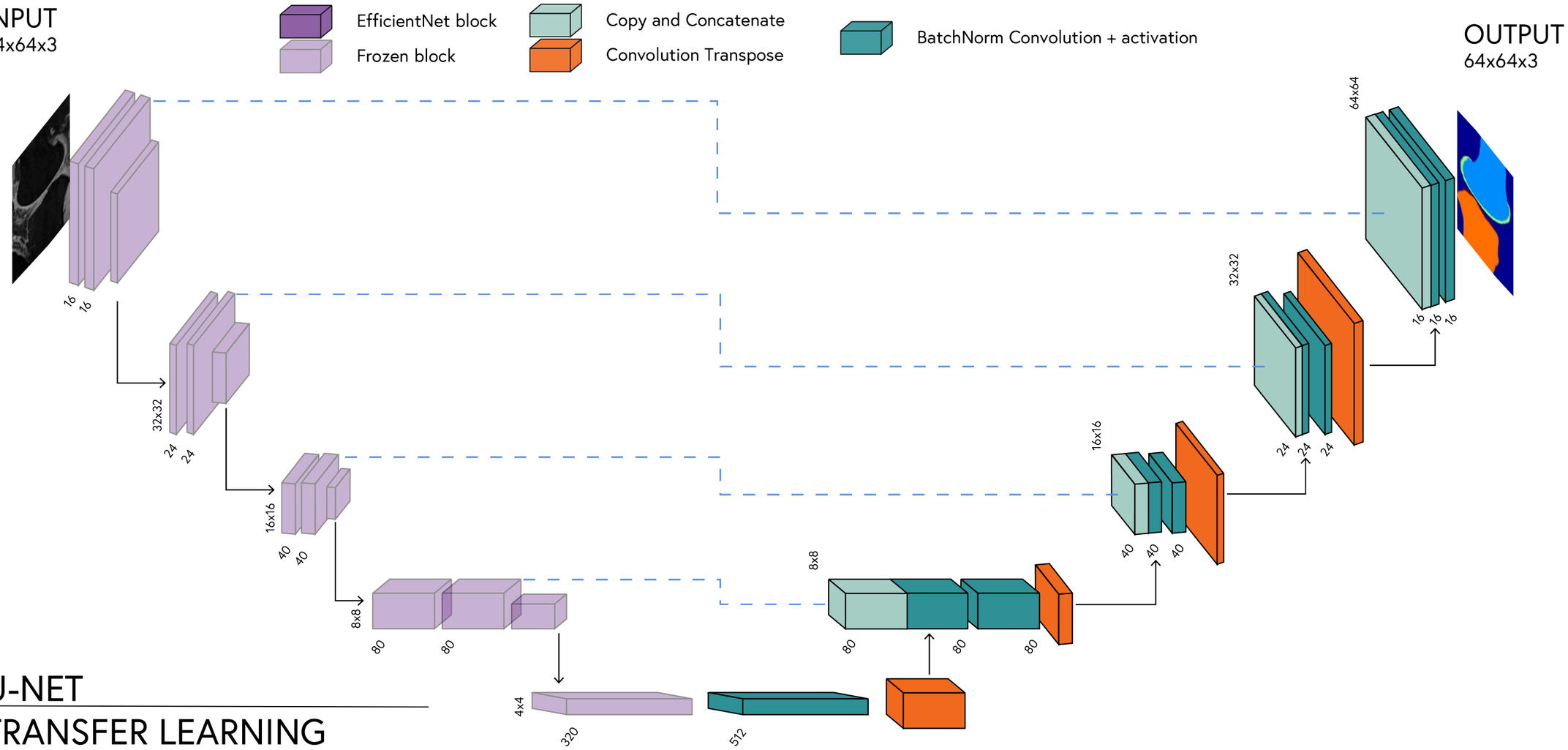
# Loading pre-trained models

- **Torchvision** provides easy access to state-of-the-art pre-trained models
- Models include weights trained on **ImageNet** (1.2M+ images, 1000 classes)
- Simple API for loading models **with or without** pre-trained weights
- Supports **many architectures**: ResNet, EfficientNet, VGG, MobileNet, etc.

```python
import torchvision.models as models

# Load with pre-trained weights (default)
resnet50 = models.resnet50(weights='IMAGENET1K_V1')

# Load with latest weights
efficientnet = models.efficientnet_b0(weights='IMAGENET1K_V2')

# Load without pre-trained weights
vgg16 = models.vgg16(weights=None)

# Access only feature extractor portion
features = efficientnet.features

# See available architectures
print(dir(models))

# Access weights info
print(models.EfficientNet_B0_Weights.IMAGENET1K_V1.meta)
```

# EFFICIENT U-NET

- **Replace** standard encoder with pre-trained EfficientNet
- **Freeze** pre-trained weights to preserve learned features
- Train **onlv decoder** lavers initiallv

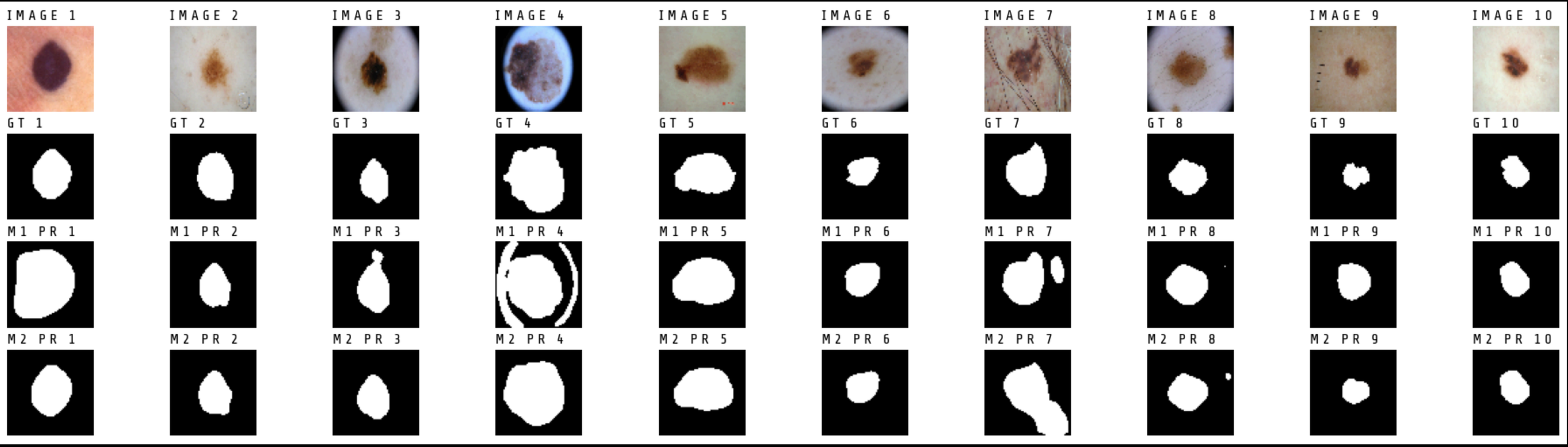



## U-NET
## TRANSFER LEARNING

# SMART UP BLOCK

- Handles feature map size mismatches between encoder and decoder
- Employs bilinear interpolation to resolve size discrepancies when needed
- Bilinear interpolation creates smoother transitions between pixels compared to nearest-neighbour
- Allows flexibility when working with arbitrary encoder architectures

```python
class SmartUp(torch.nn.Module):
    def __init__(self, in_channels, skip_channels, out_channels):
        super().__init__()
        self.up = torch.nn.ConvTranspose2d(in_channels,
                                           skip_channels,
                                           kernel_size=2, stride=2)
        self.conv = DoubleConv(skip_channels * 2, out_channels)

    def forward(self, x1, x2):
        x1 = self.up(x1)  # Initial upsampling via transposed convolution

        # Handle size mismatches with interpolation
        if x1.size()[2:] != x2.size()[2:]:
            # Bilinear interpolation - weighted average of 4 nearest pixels
            # align_corners=False prevents edge artifacts
            x1 = torch.nn.functional.interpolate(
                x1,
                size=x2.size()[2:],  # Match skip connection dimensions
                mode='bilinear',     # Smoother than nearest neighbor
                align_corners=False  # Consistent behavior with odd dimensions
            )

        # Concatenate skip connection along channel dimension
        x = torch.cat([x2, x1], dim=1)
        return self.conv(x)  # Apply convolution to fused features
```




# Advantages of t-learning

- Reduced training time (5-10x faster)
- Works with limited medical imaging data
- Better performance with challenging cases
- Faster convergence
- Lower computational cost
- Knowledge retention from source domain