Cell 0: Install Dependencies

Installs all required Python packages to ensure the notebook runs correctly.

In [1]:
```
# Original shell command
# !pip install torch torchvision pandas scikit-learn matplotlib tqdm
```

Cell 1: Build CSVs and Split Dataset

This cell loads image paths and corresponding PSPI scores, creates a dataframe, and splits it into training (80%), validation (10%), and test (10%) sets. The result is saved into three CSV files.

In [2]:
```
import pandas as pd
from pathlib import Path
from sklearn.model_selection import train_test_split

def build_csv(image_dir, pspi_dir):
    data = []
    for img_file in Path(image_dir).glob("*.png"):
        base_name = img_file.stem.rsplit("_", 1)[0]
        facs_file = Path(pspi_dir) / f"{base_name}_facs.txt"
        if facs_file.exists():
            try:
                with open(facs_file) as f:
                    score = float(f.readline().strip())
                data.append((str(img_file), score))
            except Exception as e:
                print(f"Error reading {facs_file}: {e}")
    return pd.DataFrame(data, columns=["image_path", "pspi_score"])

# Load datasets
df = build_csv("pain-dataset", "pain-dataset/pspi")
print(df.head())
print("Loaded samples:", len(df))

df["image_path"] = df["image_path"].apply(lambda x: str(Path(x)))

train_df, temp_df = train_test_split(df, test_size=0.2, random_state=42)
val_df, test_df = train_test_split(temp_df, test_size=0.5, random_state=42)

# Save all splits
train_df.to_csv("train_split.csv", index=False)
val_df.to_csv("val_split.csv", index=False)
test_df.to_csv("test_split.csv", index=False)
```

```
                          image_path  pspi_score
0  pain-dataset\aa048t2afaff017_a.png        1.0
1  pain-dataset\aa048t2afaff017_b.png        1.0
2  pain-dataset\aa048t2afaff018_a.png        1.0
3  pain-dataset\aa048t2afaff018_b.png        1.0
4  pain-dataset\aa048t2afaff019_a.png        1.0
Loaded samples: 1241
```

Step 3: Dataset Class, Training, and Evaluation Using ResNet18

This section defines the training pipeline for predicting PSPI scores using a pretrained ResNet18 model. It includes dataset loading, data augmentation, model setup, training with early stopping, and evaluation using multiple performance metrics.

Define a Custom PyTorch Dataset

We define a dataset class `PSPIDataset` to load images and corresponding PSPI scores from a dataframe. It also ensures that any PNG images with an alpha channel (RGBA) are converted to standard RGB format.

In [3]:
```python
# --- Imports ---
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, models
import os
import numpy as np
import pandas as pd
from PIL import Image
import matplotlib.pyplot as plt

# -----------------------------
# Custom Dataset Class
# -----------------------------
class PSPIDataset(Dataset):
    """
    PyTorch dataset to load images and PSPI scores from a dataframe.
    Converts RGBA to RGB to match model input.
    Handles missing files and incorrect formats gracefully.
    """
    def __init__(self, dataframe, transform=None):
        self.data = dataframe
        self.transform = transform

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

    def __getitem__(self, idx):
        # Clean up path formatting (handles \ or / automatically)
        img_path = os.path.join(self.data.iloc[idx]['image_path']).replace("\\", "/
        label = torch.tensor(self.data.iloc[idx]['pspi_score'], dtype=torch.float32
```

```
        try:
            image = Image.open(img_path).convert("RGB")  # Ensures 3-channel image
        except Exception as e:
            print(f"[Error] Failed to load image {img_path}: {e}")
            image = Image.new("RGB", (224, 224))  # fallback dummy image

        if self.transform:
            image = self.transform(image)

        return image, label
```

Define Image Transforms

We apply different image transforms for training and evaluation:

- **Training transform**: Includes resizing, random cropping, flipping, and color jittering to help the model generalize and reduce overfitting.
- **Validation/Test transform**: Only resizes and normalizes images for consistent evaluation.

In [4]:

```
# ----------------------------
# Image Preprocessing / Augmentation
# ----------------------------

# Strong augmentations for training to reduce overfitting
train_transform = transforms.Compose([
    transforms.Resize((256, 256)),  # Resize first to allow random cropping
    transforms.RandomCrop((224, 224)),  # Crop to input size
    transforms.RandomHorizontalFlip(),  # Augment horizontal orientation
    transforms.ColorJitter(brightness=0.2, contrast=0.2),  # Brightness/contrast sh
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],  # Normalize using ImageNet stats
                         [0.229, 0.224, 0.225])
])

# Consistent transforms for val/test (no augmentation)
eval_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])
```

Load Dataset Splits and Create DataLoaders

We load the preprocessed CSVs (`train_split.csv` , `val_split.csv` , `test_split.csv`) and use them to initialize PyTorch `DataLoader` objects:

- `train_loader` : uses augmented images (with shuffling).
- `val_loader` and `test_loader` : use consistent transforms without augmentation.

- These loaders allow batch-wise training and evaluation with performance optimization.

In [5]:
```
# ----------------------------
# Load train/val/test splits
# ----------------------------
train_df = pd.read_csv("train_split.csv")
val_df = pd.read_csv("val_split.csv")
test_df = pd.read_csv("test_split.csv")

# Create PyTorch DataLoaders
train_loader = DataLoader(PSPIDataset(train_df, train_transform), batch_size=8, shu
val_loader = DataLoader(PSPIDataset(val_df, eval_transform), batch_size=8)
test_loader = DataLoader(PSPIDataset(test_df, eval_transform), batch_size=8)
```

Define ResNet18 Model for Regression

We use a pretrained ResNet18 from `torchvision.models` as our feature extractor.

- The original final layer (for classification) is replaced with a linear layer that outputs a **single value** for regression.
- The model is moved to GPU (`cuda`) if available.
- We use Adam optimizer and Mean Squared Error (MSE) loss for training.

In [6]:
```
# ----------------------------
# Define Model (ResNet18 for Regression)
# ----------------------------
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Load pretrained ResNet18
model = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)

# Replace final layer to output a single value (regression)
model.fc = nn.Linear(model.fc.in_features, 1)
model = model.to(device)

# Optimizer and Loss Function
optimizer = optim.Adam(model.parameters(), lr=0.001)
loss_fn = nn.MSELoss()
```

Training Loop with Early Stopping

This loop performs the training over multiple epochs with validation monitoring:

- We track the best model using validation **MAE** (Mean Absolute Error).
- If validation MAE does not improve for `patience` epochs, training stops early.
- The best model weights are stored and later saved to disk.

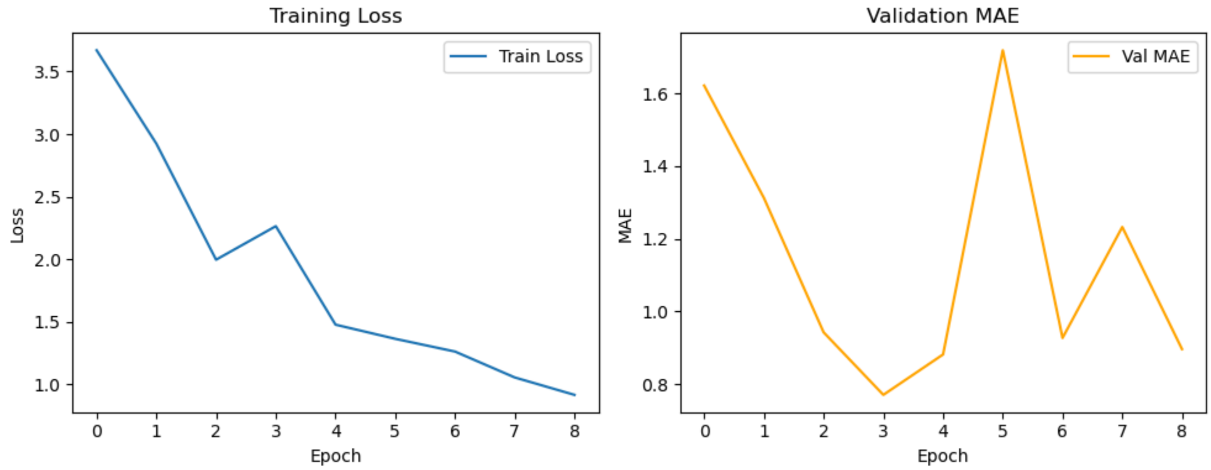
In [8]:
```
import copy
from sklearn.metrics import mean_absolute_error, mean_squared_error
from tqdm import tqdm
import matplotlib.pyplot as plt
```

```python
import torch

# -----------------------------
# Training Loop with Early Stopping + Tracking
# -----------------------------
best_val_mae = float('inf')
best_model_wts = copy.deepcopy(model.state_dict())
patience, patience_counter = 5, 0

train_losses = []
val_maes = []

print(f"Train loader batches: {len(train_loader)}")
print(f"Val loader batches: {len(val_loader)}")

model.to(device)

print("🚀 Starting training...")
for epoch in range(15):
    print(f"\n🟦 Epoch {epoch+1}")
    model.train()
    total_loss = 0

    for x, y in tqdm(train_loader, desc=f"Training Epoch {epoch+1}"):
        try:
            x, y = x.to(device), y.to(device)
            optimizer.zero_grad()
            pred = model(x).squeeze()
            loss = loss_fn(pred, y)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        except Exception as e:
            print(f"🔥 ERROR in training: {e}")
            break

    avg_train_loss = total_loss / len(train_loader)
    train_losses.append(avg_train_loss)
    print(f"✅ Epoch {epoch+1} - Training Loss: {avg_train_loss:.4f}")

    # --- Validation ---
    model.eval()
    y_true, y_pred = [], []

    with torch.no_grad():
        for x, y in tqdm(val_loader, desc="Validating"):
            try:
                x, y = x.to(device), y.to(device)
                preds = model(x).squeeze().detach().cpu().numpy()
                y_true.extend(y.cpu().numpy())
                y_pred.extend(preds)
            except Exception as e:
                print(f"🔥 ERROR in validation: {e}")
                break

    val_mse = mean_squared_error(y_true, y_pred)
```

```python
    val_mae = mean_absolute_error(y_true, y_pred)
    val_maes.append(val_mae)

    print(f"📊 Validation MSE: {val_mse:.4f}, MAE: {val_mae:.4f}")

    # --- Early stopping logic ---
    if val_mae < best_val_mae:
        best_val_mae = val_mae
        best_model_wts = copy.deepcopy(model.state_dict())
        torch.save(best_model_wts, "best_model.pt")
        patience_counter = 0
        print("💾 Model improved and saved to best_model.pt")
    else:
        patience_counter += 1
        print(f"⏸ No improvement. Patience: {patience_counter}/{patience}")
        if patience_counter >= patience:
            print("⛔ Early stopping triggered.")
            break

# ----------------------------
# Plot Training Loss & MAE
# ----------------------------
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.plot(train_losses, label="Train Loss")
plt.title("Training Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(val_maes, label="Val MAE", color='orange')
plt.title("Validation MAE")
plt.xlabel("Epoch")
plt.ylabel("MAE")
plt.legend()

plt.tight_layout()
plt.show()
```

```
Train loader batches: 124
Val loader batches: 16
🚀 Starting training...

🟦 Epoch 1
Training Epoch 1: 100%|██████████| 124/124 [03:42<00:00,  1.79s/it]
✅ Epoch 1 - Training Loss: 3.6696
Validating: 100%|██████████| 16/16 [00:09<00:00,  1.68it/s]
📊 Validation MSE: 7.6577, MAE: 1.6218
💾 Model improved and saved to best_model.pt

🟦 Epoch 2
Training Epoch 2: 100%|██████████| 124/124 [03:37<00:00,  1.75s/it]
✅ Epoch 2 - Training Loss: 2.9234
Validating: 100%|██████████| 16/16 [00:08<00:00,  1.92it/s]
```

```
📊 Validation MSE: 3.1647, MAE: 1.3125
💾 Model improved and saved to best_model.pt

🟦 Epoch 3
Training Epoch 3: 100%|██████████| 124/124 [03:44<00:00,  1.81s/it]
✅ Epoch 3 - Training Loss: 1.9964
Validating: 100%|██████████| 16/16 [00:07<00:00,  2.02it/s]
📊 Validation MSE: 2.7879, MAE: 0.9425
💾 Model improved and saved to best_model.pt

🟦 Epoch 4
Training Epoch 4: 100%|██████████| 124/124 [03:25<00:00,  1.65s/it]
✅ Epoch 4 - Training Loss: 2.2643
Validating: 100%|██████████| 16/16 [00:08<00:00,  1.93it/s]
📊 Validation MSE: 1.2257, MAE: 0.7704
💾 Model improved and saved to best_model.pt

🟦 Epoch 5
Training Epoch 5: 100%|██████████| 124/124 [03:21<00:00,  1.63s/it]
✅ Epoch 5 - Training Loss: 1.4774
Validating: 100%|██████████| 16/16 [00:08<00:00,  1.82it/s]
📊 Validation MSE: 1.4622, MAE: 0.8816
⏸️ No improvement. Patience: 1/5

🟦 Epoch 6
Training Epoch 6: 100%|██████████| 124/124 [03:38<00:00,  1.76s/it]
✅ Epoch 6 - Training Loss: 1.3644
Validating: 100%|██████████| 16/16 [00:09<00:00,  1.63it/s]
📊 Validation MSE: 6.2802, MAE: 1.7190
⏸️ No improvement. Patience: 2/5

🟦 Epoch 7
Training Epoch 7: 100%|██████████| 124/124 [03:44<00:00,  1.81s/it]
✅ Epoch 7 - Training Loss: 1.2627
Validating: 100%|██████████| 16/16 [00:09<00:00,  1.65it/s]
📊 Validation MSE: 2.9111, MAE: 0.9270
⏸️ No improvement. Patience: 3/5

🟦 Epoch 8
Training Epoch 8: 100%|██████████| 124/124 [03:44<00:00,  1.81s/it]
✅ Epoch 8 - Training Loss: 1.0558
Validating: 100%|██████████| 16/16 [00:09<00:00,  1.61it/s]
📊 Validation MSE: 4.1448, MAE: 1.2324
⏸️ No improvement. Patience: 4/5

🟦 Epoch 9
Training Epoch 9: 100%|██████████| 124/124 [03:47<00:00,  1.84s/it]
✅ Epoch 9 - Training Loss: 0.9167
Validating: 100%|██████████| 16/16 [00:09<00:00,  1.69it/s]
📊 Validation MSE: 3.6591, MAE: 0.8963
⏸️ No improvement. Patience: 5/5
⛔ Early stopping triggered.
```



Saving the Best Model

Once training is complete (or early stopping is triggered), we save the best-performing model's weights to disk. This model will later be used for evaluation and inference.

In [9]:
```
# ----------------------------
# Save Best Model to Disk
# ----------------------------
model.load_state_dict(best_model_wts)  # Load best weights before saving
torch.save(model.state_dict(), "pain-model.pth")
print("✅ Best model saved as 'pain-model.pth'")
```

```
✅ Best model saved as 'pain-model.pth'
```

Final Evaluation on the Test Set

We now evaluate the saved best model on the test set to see how well it generalizes to unseen data.
We compute three key regression metrics:

- **Mean Squared Error (MSE)**
- **Mean Absolute Error (MAE)**
- **R^2 Score**

In [11]:
```
from sklearn.metrics import r2_score

# ----------------------------
# Final Evaluation on Test Set
# ----------------------------
model.eval()
y_true, y_pred = [], []
with torch.no_grad():
    for x, y in test_loader:
        x = x.to(device)
        preds = model(x).squeeze().cpu().numpy()
        y_true.extend(y.numpy())
        y_pred.extend(preds)
```

```
# Calculate and print metrics
print("📊 Final Test MSE:", mean_squared_error(y_true, y_pred))
print("📊 Final Test MAE:", mean_absolute_error(y_true, y_pred))
print("📊 Final R² Score:", r2_score(y_true, y_pred))
```

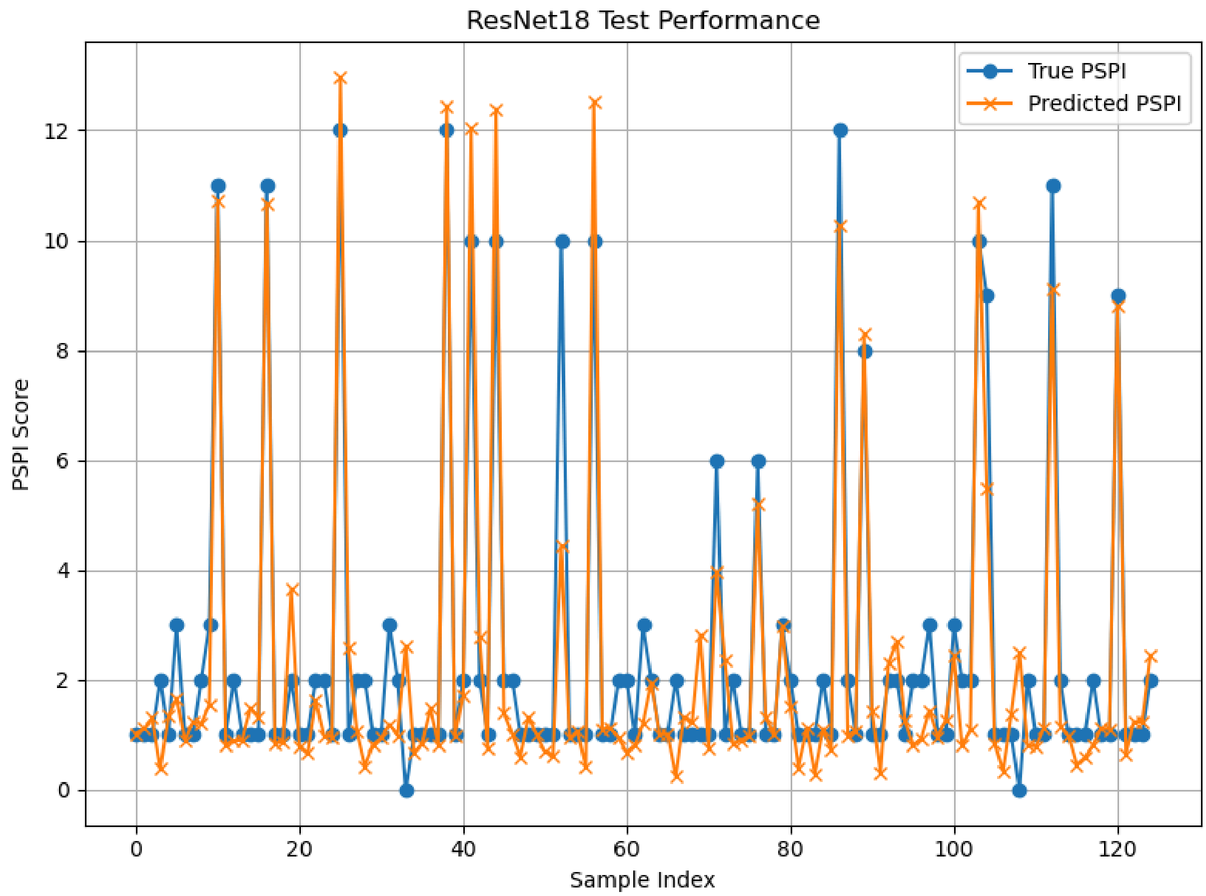
```
📊 Final Test MSE: 1.1681167
📊 Final Test MAE: 0.71402264
📊 Final R² Score: 0.8654431807964688
```

Visualizing Predictions: True vs. Predicted PSPI (Line Plot)

This line plot shows how closely the model's predictions match the actual PSPI values for each test sample.
A good model will produce predictions that follow the same pattern as the true values.

In [12]:

```
# ----------------------------
# Line plot: True vs Predicted over sample index
# ----------------------------
plt.figure(figsize=(8, 6))
plt.plot(y_true, label='True PSPI', marker='o')
plt.plot(y_pred, label='Predicted PSPI', marker='x')
plt.title('ResNet18 Test Performance')
plt.xlabel('Sample Index')
plt.ylabel('PSPI Score')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```


ResNet18 Test Performance
True PSPI
Predicted PSPI
PSPI Score
Sample Index
0
2
4
6
8
10
12
0
20
40
60
80
100
120

In [ ]: