Imports

```
import os
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tqdm import tqdm
import joblib

warnings.filterwarnings("ignore")

```

Config & Reproducibility

```
SEED = 42
BASE_PATH = "/kaggle/input/csiro-biomass"
IMG_SIZE = 224
BATCH_SIZE = 16
NUM_EPOCHS = 50
LEARNING_RATE = 1e-3
WEIGHT_DECAY = 1e-4

DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")

np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed_all(SEED)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False

```

Load & Pivot Dataset

```
df = pd.read_csv(os.path.join(BASE_PATH, "train.csv"))

df = df.pivot_table(
    index="image_path",
    columns="target_name",
    values="target"
).reset_index()

df.fillna(0, inplace=True)

TARGET_COLUMNS = [
    'Dry_Green_g',
    'Dry_Dead_g',
    'Dry_Clover_g',
    'GDM_g',
    'Dry_Total_g'
]

print("Dataset shape:", df.shape)

```

```
Dataset shape: (357, 6)
```

Train / Validation Split & Scaling

```
train_df, val_df = train_test_split(
    df,
    test_size=0.2,
    random_state=SEED,
    shuffle=True
)

target_scaler = StandardScaler()
target_scaler.fit(train_df[TARGET_COLUMNS].values)

```

```
▾ StandardScaler
StandardScaler()
```

Dataset Class

```
class BiomassDataset(Dataset):
    def __init__(self, df, image_dir, transform=None, scaler=None):
        self.df = df.copy()
        self.df["image_path"] = self.df["image_path"].apply(os.path.basename)
        self.samples = list(self.df.itertuples(index=False))
        self.image_dir = image_dir
        self.transform = transform
        self.scaler = scaler

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

    def __getitem__(self, idx):
        row = self.samples[idx]
        image = Image.open(
            os.path.join(self.image_dir, row.image_path)
        ).convert("RGB")

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

        target = np.array(
            [getattr(row, t) for t in TARGET_COLUMNS],
            dtype=np.float32
        )

```

```
        if self.scaler:
            target = self.scaler.transform(target.reshape(1, -1))[0]

        return image, torch.tensor(target)

```

Transforms (Medium Augmentation)

```
train_transforms = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(10),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
])

val_transforms = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
])

```

DataLoaders

```
train_dataset = BiomassDataset(
    train_df,
    os.path.join(BASE_PATH, "train"),
    train_transforms,
    target_scaler
)

val_dataset = BiomassDataset(
    val_df,
    os.path.join(BASE_PATH, "train"),
    val_transforms,
    target_scaler
)

train_loader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True,
    num_workers=2
)

val_loader = DataLoader(
    val_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=2
)

```

Weighted Loss & Metric

```
TARGET_WEIGHTS = torch.tensor([0.1,0.1,0.1,0.2,0.5], device=DEVICE)

class WeightedMSELoss(nn.Module):
    def __init__(self, weights):
        super().__init__()
        self.register_buffer("weights", weights)

    def forward(self, preds, targets):
```

```
        return ((preds - targets) ** 2 * self.weights).mean()

criterion = WeightedMSELoss(TARGET_WEIGHTS)

def weighted_r2_score(y_true, y_pred, weights):
    mean_y = y_true.mean(0, keepdim=True)
    ss_res = ((y_true - y_pred) ** 2).sum(0)
    ss_tot = ((y_true - mean_y) ** 2).sum(0)
    return (1 - (weights * ss_res).sum() / ((weights * ss_tot).sum() + 1e-8)).item()

```

Optimized CNN Architecture

```
class OptimizedCNN(nn.Module):
    def __init__(self, num_outputs=5):
        super().__init__()
        layers = []
        in_channels = 3
        num_filters = 256

        # EXPERIMENT WINNER: Kernel Size 3x3 (Best Val R2: 0.238753)
        kernel_size = 3
        padding = kernel_size // 2

        # EXPERIMENT WINNER: Dropout 0.6 (Best Val R2: 0.260552 with wd 0.0)
        dropout_p = 0.6

        # EXPERIMENT WINNER: 3 Blocks (Strong performance, balancing depth and loss)
        for _ in range(3):
            layers += [
                nn.Conv2d(in_channels, num_filters, kernel_size, padding=padding),
                nn.BatchNorm2d(num_filters),
                nn.ReLU(inplace=True),
                nn.MaxPool2d(2),
                nn.Dropout(dropout_p)
            ]
            in_channels = num_filters

        layers.append(nn.AdaptiveAvgPool2d(1))
        self.features = nn.Sequential(*layers)
        self.regressor = nn.Linear(num_filters, num_outputs)

    def forward(self, x):
        x = self.features(x)
        return self.regressor(x.flatten(1))

# --- TRAINING CONFIGURATION ---
# EXPERIMENT WINNER: Weight Decay 0.0 (per the dropout_0.6_wd_0.0 config)
model = OptimizedCNN(num_outputs=5)
optimizer = torch.optim.Adam(
    model.parameters(),
    lr=0.001,
    weight_decay=0.0  # Set to 0.0 based on your best R2 result
)
```

Epoch Function

```
def run_epoch(model, loader, optimizer=None):
    training = optimizer is not None
    model.train() if training else model.eval()

    total_loss, total_r2, n = 0.0, 0.0, 0

    with torch.set_grad_enabled(training):
        for images, targets in loader:
            images, targets = images.to(DEVICE), targets.to(DEVICE)
            preds = model(images)
            loss = criterion(preds, targets)
```

```
            if training:
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()

            b = images.size(0)
            total_loss += loss.item() * b
            total_r2 += weighted_r2_score(targets, preds, TARGET_WEIGHTS) * b
            n += b

    return total_loss / n, total_r2 / n

```

Training Loop + Metrics + Best Model Saving

```
model = OptimizedCNN(num_outputs=len(TARGET_COLUMNS)).to(DEVICE)
optimizer = optim.Adam(
    model.parameters(),
    lr=LEARNING_RATE,
    weight_decay=WEIGHT_DECAY
)

train_losses, val_losses = [], []
train_r2s, val_r2s = [], []

best_val_r2 = -1e9

for epoch in range(NUM_EPOCHS):
    tr_l, tr_r2 = run_epoch(model, train_loader, optimizer)
    va_l, va_r2 = run_epoch(model, val_loader)

    train_losses.append(tr_l)
    val_losses.append(va_l)
    train_r2s.append(tr_r2)
    val_r2s.append(va_r2)

    if va_r2 > best_val_r2:
        best_val_r2 = va_r2
        torch.save(model.state_dict(), "best_model_weights.pth")
        joblib.dump(target_scaler, "target_scaler.pkl")

    print(
        f"Epoch {epoch+1}/{NUM_EPOCHS} | "
        f"Train R²: {tr_r2:.4f} | Val R²: {va_r2:.4f}"
    )

```

```
Epoch 1/50 | Train R²: -0.5740 | Val R²: -0.0984
Epoch 2/50 | Train R²: 0.0401 | Val R²: 0.0997
Epoch 3/50 | Train R²: 0.0532 | Val R²: 0.1116
Epoch 4/50 | Train R²: 0.0264 | Val R²: 0.1226
Epoch 5/50 | Train R²: 0.0703 | Val R²: 0.1093
Epoch 6/50 | Train R²: 0.0615 | Val R²: 0.0556
Epoch 7/50 | Train R²: 0.0878 | Val R²: 0.1356
Epoch 8/50 | Train R²: 0.0453 | Val R²: 0.1924
Epoch 9/50 | Train R²: 0.1055 | Val R²: 0.1803
Epoch 10/50 | Train R²: 0.0880 | Val R²: 0.1738
Epoch 11/50 | Train R²: 0.0980 | Val R²: 0.0649
Epoch 12/50 | Train R²: 0.1660 | Val R²: 0.1816
Epoch 13/50 | Train R²: 0.1701 | Val R²: 0.1947
Epoch 14/50 | Train R²: 0.1961 | Val R²: 0.1153
Epoch 15/50 | Train R²: 0.1477 | Val R²: -0.3880
Epoch 16/50 | Train R²: 0.1605 | Val R²: -0.0753
Epoch 17/50 | Train R²: 0.0156 | Val R²: 0.2601
Epoch 18/50 | Train R²: 0.1537 | Val R²: -0.0886
Epoch 19/50 | Train R²: 0.2025 | Val R²: -0.0312
Epoch 20/50 | Train R²: 0.1616 | Val R²: -0.3568
Epoch 21/50 | Train R²: 0.1774 | Val R²: 0.2065
Epoch 22/50 | Train R²: 0.1473 | Val R²: -0.0164
Epoch 23/50 | Train R²: 0.0551 | Val R²: 0.0299
Epoch 24/50 | Train R²: 0.1657 | Val R²: 0.2932
Epoch 25/50 | Train R²: 0.2500 | Val R²: 0.1842
Epoch 26/50 | Train R²: 0.1892 | Val R²: 0.2224
```

```
Epoch 27/50 | Train R²: 0.2550 | Val R²: -0.4257
Epoch 28/50 | Train R²: 0.1907 | Val R²: 0.1485
Epoch 29/50 | Train R²: 0.1430 | Val R²: 0.0554
Epoch 30/50 | Train R²: 0.2137 | Val R²: 0.2602
Epoch 31/50 | Train R²: 0.2197 | Val R²: 0.0840
Epoch 32/50 | Train R²: 0.1808 | Val R²: 0.0855
Epoch 33/50 | Train R²: 0.2244 | Val R²: 0.2070
Epoch 34/50 | Train R²: 0.2928 | Val R²: 0.0582
Epoch 35/50 | Train R²: 0.1811 | Val R²: -0.2494
Epoch 36/50 | Train R²: 0.2347 | Val R²: 0.2028
Epoch 37/50 | Train R²: 0.1022 | Val R²: 0.2441
Epoch 38/50 | Train R²: 0.1132 | Val R²: 0.0808
Epoch 39/50 | Train R²: 0.2269 | Val R²: -0.6006
Epoch 40/50 | Train R²: 0.1876 | Val R²: -0.4622
Epoch 41/50 | Train R²: 0.2043 | Val R²: 0.0961
Epoch 42/50 | Train R²: 0.2406 | Val R²: 0.1442
Epoch 43/50 | Train R²: 0.1684 | Val R²: -0.0167
Epoch 44/50 | Train R²: 0.2829 | Val R²: 0.2773
Epoch 45/50 | Train R²: 0.2372 | Val R²: -0.0062
Epoch 46/50 | Train R²: 0.1654 | Val R²: -0.0181
Epoch 47/50 | Train R²: 0.2651 | Val R²: -0.3135
Epoch 48/50 | Train R²: 0.3143 | Val R²: -0.4012
Epoch 49/50 | Train R²: 0.2001 | Val R²: 0.3201
Epoch 50/50 | Train R²: 0.2963 | Val R²: 0.2955
```

Average Metrics

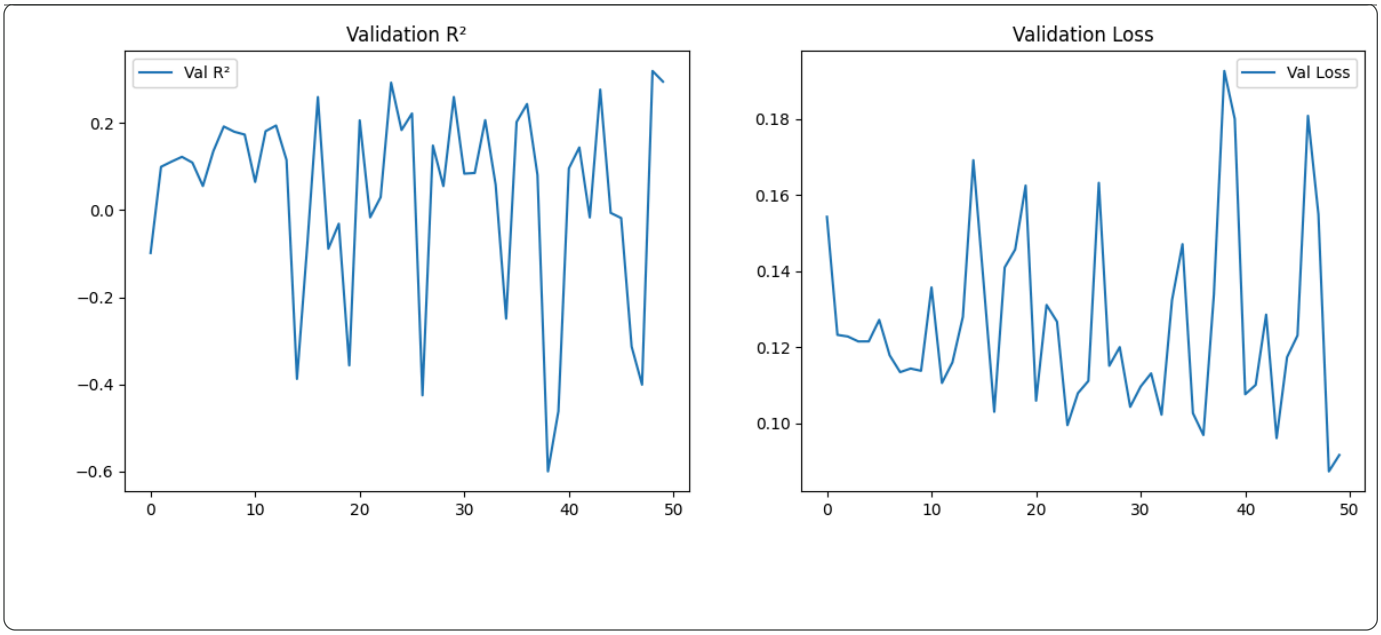
```
print("\n===== FINAL AVERAGES =====")
print("Avg Train Loss:", np.mean(train_losses))
print("Avg Val Loss:", np.mean(val_losses))
print("Avg Train R²:", np.mean(train_r2s))
print("Avg Val R²:", np.mean(val_r2s))
print("Best Val R²:", best_val_r2)
```

```
===== FINAL AVERAGES =====
Avg Train Loss: 0.15376009347381298
Avg Val Loss: 0.12544329347709815
Avg Train R²: 0.1520189860787308
Avg Val R²: 0.03890925923983255
Best Val R²: 0.320112943649292
```

Plots

```
plt.figure(figsize=(14,5))
plt.subplot(1,2,1)
plt.plot(val_r2s, label="Val R²")
plt.title("Validation R²")
plt.legend()

plt.subplot(1,2,2)
plt.plot(val_losses, label="Val Loss")
plt.title("Validation Loss")
plt.legend()
plt.show()
```



Inference

Device & Paths

```
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
BASE_PATH = "/kaggle/input/csiro-biomass"
TEST_DIR = f"{BASE_PATH}/test"
MODEL_PATH = "best_model_weights.pth"
SCALER_PATH = "target_scaler.pkl"
```

Target Columns (Must Match Training)

```
TARGET_COLUMNS = [
    'Dry_Green_g',
    'Dry_Dead_g',
    'Dry_Clover_g',
    'GDM_g',
    'Dry_Total_g'
]
```

Load Model Weights & Scaler

```
best_model = OptimizedCNN(num_outputs=len(TARGET_COLUMNS)).to(DEVICE)
best_model.load_state_dict(
    torch.load(MODEL_PATH, map_location=DEVICE)
)
best_model.eval()

target_scaler = joblib.load(SCALER_PATH)

```

Image Transforms (Validation / Test Only)

```
image_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
```

```
        std=[0.229, 0.224, 0.225]
    )
])

```

Test-Time Inference

```
submission_rows = []

with torch.no_grad():
    for file in tqdm(sorted(os.listdir(TEST_DIR))):
        image = Image.open(os.path.join(TEST_DIR, file)).convert("RGB")
        tensor = image_transforms(image).unsqueeze(0).to(DEVICE)

        preds_scaled = best_model(tensor).cpu().numpy()
        preds = target_scaler.inverse_transform(preds_scaled)[0]

        image_id = file.replace(".jpg", "")

        for name, value in zip(TARGET_COLUMNS, preds):
            submission_rows.append({
                "sample_id": f"{image_id}__{name}",
                "target": float(max(0.0, value))  # clamp negatives
            })

```

```
100%|██████████| 1/1 [00:00<00:00,  5.59it/s]
```