

Imports

```
1 import os
2 import warnings
3 import numpy as np
4 import pandas as pd
5 import matplotlib.pyplot as plt
6 from PIL import Image
7
8 import torch
9 import torch.nn as nn
10 import torch.optim as optim
11 from torch.utils.data import Dataset, DataLoader
12 from torchvision import transforms
13
14 from sklearn.model_selection import train_test_split
15 from sklearn.preprocessing import StandardScaler
16 from tqdm import tqdm
17
18 warnings.filterwarnings("ignore")
19
```

Configuration & Reproducibility

```
1 SEED = 42
2 BASE_PATH = "/kaggle/input/csiro-biomass"
3 IMG_SIZE = 224
4 BATCH_SIZE = 16
5 NUM_EPOCHS = 20
6 LEARNING_RATE = 1e-3
7 WEIGHT_DECAY = 1e-4
8
9 DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
10
11 np.random.seed(SEED)
12 torch.manual_seed(SEED)
13 torch.cuda.manual_seed_all(SEED)
14 torch.backends.cudnn.deterministic = True
15 torch.backends.cudnn.benchmark = False
16
```

Load & Pivot Dataset

```
1 df = pd.read_csv(os.path.join(BASE_PATH, "train.csv"))
2
3 df = df.pivot_table(
4     index="image_path",
5     columns="target_name",
6     values="target"
7 ).reset_index()
8
9 df.fillna(0, inplace=True)
10
11 TARGET_COLUMNS = [
12     'Dry_Green_g',
13     'Dry_Dead_g',
14     'Dry_Clover_g',
15     'GDM_g',
16     'Dry_Total_g'
17 ]
18
19 print("Dataset shape:", df.shape)
20
```

Dataset shape: (357, 6)

Train / Validation Split & Target Scaling

```

1 train_df, val_df = train_test_split(
2     df,
3     test_size=0.2,
4     random_state=SEED,
5     shuffle=True
6 )
7
8 scaler = StandardScaler()
9 scaler.fit(train_df[TARGET_COLUMNS].values)
10

```

StandardScaler ⓘ ?

StandardScaler()

Dataset Class

```

1 class BiomassDataset(Dataset):
2     def __init__(self, df, image_dir, transform=None, scaler=None):
3         self.df = df.copy()
4         self.df["image_path"] = self.df["image_path"].apply(os.path.basename)
5         self.samples = list(self.df.itertuples(index=False))
6         self.image_dir = image_dir
7         self.transform = transform
8         self.scaler = scaler
9
10    def __len__(self):
11        return len(self.samples)
12
13    def __getitem__(self, idx):
14        row = self.samples[idx]
15        img_path = os.path.join(self.image_dir, row.image_path)
16        image = Image.open(img_path).convert("RGB")
17
18        if self.transform:
19            image = self.transform(image)
20
21        target = np.array(
22            [getattr(row, t) for t in TARGET_COLUMNS],
23            dtype=np.float32
24        )
25
26        if self.scaler:
27            target = self.scaler.transform(target.reshape(1, -1))[0]
28
29        return image, torch.tensor(target)
30

```

DataLoaders

```

1 transform = transforms.Compose([
2     transforms.Resize((IMG_SIZE, IMG_SIZE)),
3     transforms.ToTensor(),
4     transforms.Normalize(
5         mean=[0.485, 0.456, 0.406],
6         std=[0.229, 0.224, 0.225]
7     )
8 ])
9
10 train_dataset = BiomassDataset(
11     train_df,

```

```

12     os.path.join(BASE_PATH, "train"),
13     transform,
14     scaler
15 )
16
17 val_dataset = BiomassDataset(
18     val_df,
19     os.path.join(BASE_PATH, "train"),
20     transform,
21     scaler
22 )
23
24 train_loader = DataLoader(
25     train_dataset,
26     batch_size=BATCH_SIZE,
27     shuffle=True,
28     num_workers=2
29 )
30
31 val_loader = DataLoader(
32     val_dataset,
33     batch_size=BATCH_SIZE,
34     shuffle=False,
35     num_workers=2
36 )
37

```

✓ Weighted Loss

```

1 TARGET_WEIGHTS = torch.tensor(
2     [0.1, 0.1, 0.1, 0.2, 0.5],
3     device=DEVICE
4 )
5
6 class WeightedMSELoss(nn.Module):
7     def __init__(self, weights):
8         super().__init__()
9         self.register_buffer("weights", weights)
10
11     def forward(self, preds, targets):
12         return ((preds - targets) ** 2 * self.weights).mean()
13
14 criterion = WeightedMSELoss(TARGET_WEIGHTS)
15

```

✓ Weighted R² Metric

```

1 def weighted_r2_score(y_true, y_pred, weights):
2     mean_y = y_true.mean(0, keepdim=True)
3     ss_res = ((y_true - y_pred) ** 2).sum(0)
4     ss_tot = ((y_true - mean_y) ** 2).sum(0)
5     return (1 - (weights * ss_res).sum() /
6             ((weights * ss_tot).sum() + 1e-8)).item()
7

```

✓ CNN with Variable Kernel Size

```

1 class CNNWithKernelSize(nn.Module):
2     def __init__(self, kernel_size=3, num_filters=256, num_blocks=3, num_outputs=5):
3         super().__init__()
4
5         layers = []
6         in_channels = 3
7         padding = kernel_size // 2
8

```

```

9         for _ in range(num_blocks):
10             layers.append(
11                 nn.Conv2d(
12                     in_channels,
13                     num_filters,
14                     kernel_size=kernel_size,
15                     padding=padding
16                 )
17             )
18             layers.append(nn.BatchNorm2d(num_filters))
19             layers.append(nn.ReLU(inplace=True))
20             layers.append(nn.MaxPool2d(2))
21             in_channels = num_filters
22
23         layers.append(nn.AdaptiveAvgPool2d(1))
24
25         self.features = nn.Sequential(*layers)
26         self.regressor = nn.Linear(num_filters, num_outputs)
27
28     def forward(self, x):
29         x = self.features(x)
30         return self.regressor(x.flatten(1))
31

```

▼ One-Epoch Train / Eval Function

```

1 def run_epoch(model, loader, optimizer=None):
2     training = optimizer is not None
3     model.train() if training else model.eval()
4
5     total_loss, total_r2, n = 0.0, 0.0, 0
6
7     with torch.set_grad_enabled(training):
8         for images, targets in loader:
9             images = images.to(DEVICE)
10            targets = targets.to(DEVICE)
11
12            preds = model(images)
13            loss = criterion(preds, targets)
14
15            if training:
16                optimizer.zero_grad()
17                loss.backward()
18                optimizer.step()
19
20            bsz = images.size(0)
21            total_loss += loss.item() * bsz
22            total_r2 += weighted_r2_score(
23                targets, preds, TARGET_WEIGHTS
24            ) * bsz
25            n += bsz
26
27     return total_loss / n, total_r2 / n
28

```

▼ Kernel Size Experiment [3, 5, 7]

```

1 kernel_options = [3, 5, 7]
2 results = {}
3
4 for k in kernel_options:
5     print(f"\n=====")
6     print(f" Training CNN with kernel {k}x{k}")
7     print(f"=====")
8
9     model = CNNWithKernelSize(
10         kernel_size=k,
11         num_filters=256,

```

```

12     num_blocks=3,
13     num_outputs=len(TARGET_COLUMNS)
14 ).to(DEVICE)
15
16 optimizer = optim.Adam(
17     model.parameters(),
18     lr=LEARNING_RATE,
19     weight_decay=WEIGHT_DECAY
20 )
21
22 train_losses, val_losses = [], []
23 train_r2s, val_r2s = [], []
24 best_val_r2 = -1e9
25
26 # ----- Training epochs -----
27 for epoch in range(NUM_EPOCHS):
28     train_loss, train_r2 = run_epoch(
29         model, train_loader, optimizer
30     )
31     val_loss, val_r2 = run_epoch(
32         model, val_loader
33     )
34
35     train_losses.append(train_loss)
36     val_losses.append(val_loss)
37     train_r2s.append(train_r2)
38     val_r2s.append(val_r2)
39
40     best_val_r2 = max(best_val_r2, val_r2)
41
42     print(
43         f"Epoch [{epoch+1}/{NUM_EPOCHS}] | "
44         f"Train Loss: {train_loss:.4f} | "
45         f"Val Loss: {val_loss:.4f} | "
46         f"Train R²: {train_r2:.4f} | "
47         f"Val R²: {val_r2:.4f}"
48     )
49
50 # ----- Store averaged metrics -----
51 results[k] = {
52     "best_val_r2": best_val_r2,
53     "avg_train_loss": float(np.mean(train_losses)),
54     "avg_val_loss": float(np.mean(val_losses)),
55     "avg_train_r2": float(np.mean(train_r2s)),
56     "avg_val_r2": float(np.mean(val_r2s)),
57     "train_losses": train_losses,
58     "val_losses": val_losses,
59     "train_r2s": train_r2s,
60     "val_r2s": val_r2s
61 }
62

```

```

=====
Training CNN with kernel 3x3
=====

```

```

Epoch [1/20] | Train Loss: 0.2101 | Val Loss: 0.4448 | Train R²: -0.1831 | Val R²: -2.8216
Epoch [2/20] | Train Loss: 0.1656 | Val Loss: 0.1320 | Train R²: 0.0664 | Val R²: -0.0146
Epoch [3/20] | Train Loss: 0.1607 | Val Loss: 0.1238 | Train R²: 0.1576 | Val R²: 0.0411
Epoch [4/20] | Train Loss: 0.1579 | Val Loss: 0.1390 | Train R²: 0.1791 | Val R²: -0.1129
Epoch [5/20] | Train Loss: 0.1513 | Val Loss: 0.1151 | Train R²: 0.1808 | Val R²: 0.1225
Epoch [6/20] | Train Loss: 0.1531 | Val Loss: 0.1187 | Train R²: 0.0082 | Val R²: 0.0704
Epoch [7/20] | Train Loss: 0.1486 | Val Loss: 0.1054 | Train R²: 0.2041 | Val R²: 0.2345
Epoch [8/20] | Train Loss: 0.1483 | Val Loss: 0.1077 | Train R²: 0.1555 | Val R²: 0.1686
Epoch [9/20] | Train Loss: 0.1477 | Val Loss: 0.1173 | Train R²: 0.2101 | Val R²: 0.0803
Epoch [10/20] | Train Loss: 0.1419 | Val Loss: 0.1167 | Train R²: 0.1847 | Val R²: 0.0706
Epoch [11/20] | Train Loss: 0.1472 | Val Loss: 0.1306 | Train R²: 0.2255 | Val R²: -0.0399
Epoch [12/20] | Train Loss: 0.1466 | Val Loss: 0.1095 | Train R²: 0.1173 | Val R²: 0.0882
Epoch [13/20] | Train Loss: 0.1304 | Val Loss: 0.1016 | Train R²: 0.3186 | Val R²: 0.2098
Epoch [14/20] | Train Loss: 0.1351 | Val Loss: 0.2300 | Train R²: 0.2740 | Val R²: -0.9007
Epoch [15/20] | Train Loss: 0.1486 | Val Loss: 0.1421 | Train R²: 0.1864 | Val R²: -0.1540
Epoch [16/20] | Train Loss: 0.1351 | Val Loss: 0.1069 | Train R²: 0.2818 | Val R²: 0.1656
Epoch [17/20] | Train Loss: 0.1301 | Val Loss: 0.1265 | Train R²: 0.2715 | Val R²: 0.0951
Epoch [18/20] | Train Loss: 0.1365 | Val Loss: 0.1025 | Train R²: 0.2243 | Val R²: 0.2388
Epoch [19/20] | Train Loss: 0.1299 | Val Loss: 0.1325 | Train R²: 0.2616 | Val R²: 0.0219
Epoch [20/20] | Train Loss: 0.1345 | Val Loss: 0.1029 | Train R²: 0.2633 | Val R²: 0.2370

```

```

=====
Training CNN with kernel 5x5
=====
Epoch [1/20] | Train Loss: 0.2483 | Val Loss: 0.3673 | Train R²: -0.3994 | Val R²: -2.2683
Epoch [2/20] | Train Loss: 0.1737 | Val Loss: 0.1458 | Train R²: 0.0604 | Val R²: -0.1626
Epoch [3/20] | Train Loss: 0.1671 | Val Loss: 0.1212 | Train R²: 0.0852 | Val R²: 0.0894
Epoch [4/20] | Train Loss: 0.1623 | Val Loss: 0.1199 | Train R²: 0.0603 | Val R²: 0.1038
Epoch [5/20] | Train Loss: 0.1637 | Val Loss: 0.1226 | Train R²: 0.0232 | Val R²: 0.0904
Epoch [6/20] | Train Loss: 0.1560 | Val Loss: 0.1179 | Train R²: 0.1562 | Val R²: 0.1146
Epoch [7/20] | Train Loss: 0.1571 | Val Loss: 0.1182 | Train R²: 0.0949 | Val R²: 0.1080
Epoch [8/20] | Train Loss: 0.1503 | Val Loss: 0.1176 | Train R²: 0.2015 | Val R²: 0.1372
Epoch [9/20] | Train Loss: 0.1550 | Val Loss: 0.1184 | Train R²: 0.1731 | Val R²: 0.1094
Epoch [10/20] | Train Loss: 0.1456 | Val Loss: 0.1145 | Train R²: 0.1376 | Val R²: 0.1521
Epoch [11/20] | Train Loss: 0.1418 | Val Loss: 0.1124 | Train R²: 0.2059 | Val R²: 0.1705
Epoch [12/20] | Train Loss: 0.1437 | Val Loss: 0.1347 | Train R²: 0.2206 | Val R²: -0.0410
Epoch [13/20] | Train Loss: 0.1459 | Val Loss: 0.1238 | Train R²: 0.2327 | Val R²: 0.0490
Epoch [14/20] | Train Loss: 0.1452 | Val Loss: 0.1315 | Train R²: 0.1621 | Val R²: -0.0170
Epoch [15/20] | Train Loss: 0.1446 | Val Loss: 0.1267 | Train R²: 0.2143 | Val R²: 0.0255
Epoch [16/20] | Train Loss: 0.1449 | Val Loss: 0.1162 | Train R²: 0.1985 | Val R²: 0.1355
Epoch [17/20] | Train Loss: 0.1424 | Val Loss: 0.1172 | Train R²: 0.2245 | Val R²: 0.1222
Epoch [18/20] | Train Loss: 0.1430 | Val Loss: 0.1057 | Train R²: 0.1684 | Val R²: 0.2099
Epoch [19/20] | Train Loss: 0.1363 | Val Loss: 0.1095 | Train R²: 0.2831 | Val R²: 0.1893
Epoch [20/20] | Train Loss: 0.1413 | Val Loss: 0.1055 | Train R²: 0.1942 | Val R²: 0.1732

=====
Training CNN with kernel 7x7
=====
Epoch [1/20] | Train Loss: 0.2769 | Val Loss: 0.4540 | Train R²: -0.7848 | Val R²: -2.9162
Epoch [2/20] | Train Loss: 0.1641 | Val Loss: 0.1508 | Train R²: 0.1170 | Val R²: -0.2112
Epoch [3/20] | Train Loss: 0.1594 | Val Loss: 0.1208 | Train R²: 0.1437 | Val R²: 0.0800
Epoch [4/20] | Train Loss: 0.1583 | Val Loss: 0.1217 | Train R²: 0.1754 | Val R²: 0.0893
Epoch [5/20] | Train Loss: 0.1568 | Val Loss: 0.1216 | Train R²: 0.1400 | Val R²: 0.0879

```

Summary Table

```

1 summary_df = pd.DataFrame([
2     {
3         "Kernel": f"{k}x{k}",
4         "Avg Train Loss": results[k]["avg_train_loss"],
5         "Avg Val Loss": results[k]["avg_val_loss"],
6         "Avg Train R2": results[k]["avg_train_r2"],
7         "Avg Val R2": results[k]["avg_val_r2"],
8         "Best Val R2": results[k]["best_val_r2"]
9     }
10    for k in kernel_options
11 ]).sort_values("Avg Val R2", ascending=False)
12
13 display(summary_df)
14

```

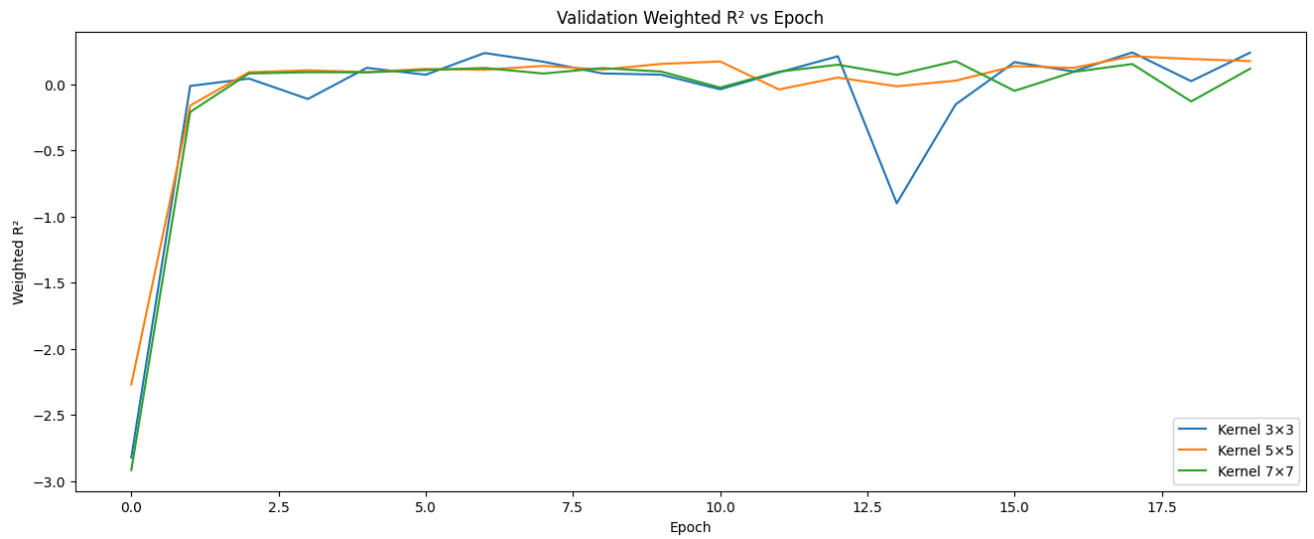
	Kernel	Avg Train Loss	Avg Val Loss	Avg Train R2	Avg Val R2	Best Val R2
1	5x5	0.155397	0.132328	0.13486	-0.025459	0.209893
2	7x7	0.155242	0.137965	0.14376	-0.085953	0.173202
0	3x3	0.147953	0.140283	0.17938	-0.109971	0.238753

Plot Validation R²

```

1 plt.figure(figsize=(16, 6))
2 for k in kernel_options:
3     plt.plot(results[k]["val_r2s"], label=f"Kernel {k}x{k}")
4 plt.title("Validation Weighted R² vs Epoch")
5 plt.xlabel("Epoch")
6 plt.ylabel("Weighted R²")
7 plt.legend()
8 plt.show()
9

```



Plot Validation Loss

```

1 plt.figure(figsize=(16, 6))
2 for k in kernel_options:
3     plt.plot(results[k]["val_losses"], label=f"Kernel {k}x{k}")
4 plt.title("Validation Loss vs Epoch")
5 plt.xlabel("Epoch")
6 plt.ylabel("Loss")
7 plt.legend()
8 plt.show()
9

```

