Imports

```
from dataset_wrapper import *
from trainer import *
from vit_wrapper import *
```

ViT Base

Leave-One-Domain-Out (LODO) Training

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=(0.5,), std=(0.5,))
])

results_base = {}

for test_domain in DOMAINS:
    print(f"\nTesting on domain: {test_domain}")
    train_domains = [d for d in DOMAINS if d != test_domain]

    # Load datasets
    dataset = PACSDataset(DATA_ROOT, DOMAINS, transform)
    train_loaders = [dataset.get_dataloader(d, train=True) for d in
train_domains]
    val_loaders = [dataset.get_dataloader(d, train=False) for d in
train_domains]
    test_loader = dataset.get_dataloader(test_domain, train=False)

    # Concatenate datasets
    train_ds = ConcatDataset([dl.dataset for dl in train_loaders])
    val_ds = ConcatDataset([dl.dataset for dl in val_loaders])

    train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE,
shuffle=True)
    val_loader = DataLoader(val_ds, batch_size=BATCH_SIZE,
shuffle=False)

    # Initialize model, optimizer, and criterion
    model_base = ViTModel(NUM_CLASSES, model_size="base")
    optimizer = optim.Adam(model_base.parameters(), lr=1e-4)
    criterion = nn.CrossEntropyLoss()
    trainer = Trainer(model_base, optimizer, criterion)

    # Train
```

```
    for epoch in range(NUM_EPOCHS):
        print(f"Epoch {epoch + 1}/{NUM_EPOCHS}")
        train_loss_base = trainer.train(train_loader)
        val_acc_base = trainer.evaluate(val_loader)
        print(f"Train Loss: {train_loss_base:.4f} | Val Acc:
{val_acc_base:.4f}")

    # Test
    test_acc_base = trainer.evaluate(test_loader)
    results_base[test_domain] = test_acc_base
    print(f"Test Accuracy on {test_domain}: {test_acc_base:.4f}")
```

Basline

```
print("\Baseline: training on all domains and testing on mixed
domains")
# Load full train and test sets via leave-all-in loaders
dataset_all = PACSDataset(DATA_ROOT, DOMAINS, transform)
all_train_loaders = [dataset_all.get_dataloader(d, train=True) for d
in DOMAINS]
all_test_loaders = [dataset_all.get_dataloader(d, train=False) for d
in DOMAINS]

# Concatenate
full_train_ds = ConcatDataset([dl.dataset for dl in
all_train_loaders])
full_test_ds = ConcatDataset([dl.dataset for dl in all_test_loaders])
full_train_loader = DataLoader(full_train_ds, batch_size=BATCH_SIZE,
shuffle=True)
full_test_loader = DataLoader(full_test_ds, batch_size=BATCH_SIZE,
shuffle=False)

# Initialize baseline model
baseline_model_base = ViTModel(NUM_CLASSES, model_size="base")
baseline_optimizer = optim.Adam(baseline_model_base.parameters(),
lr=1e-4)
baseline_criterion = nn.CrossEntropyLoss()
baseline_trainer = Trainer(baseline_model_base, baseline_optimizer,
baseline_criterion)

# Train baseline
for epoch in range(NUM_EPOCHS):
    print(f"Baseline Epoch {epoch + 1}/{NUM_EPOCHS}")
    baseline_loss_base = baseline_trainer.train(full_train_loader)
    baseline_val_acc_base =
baseline_trainer.evaluate(full_test_loader)
    print(f"Baseline Loss: {baseline_loss_base:.4f} | Baseline Acc:
{baseline_val_acc_base:.4f}")
```

```
# Test baseline
baseline_test_acc_base = baseline_trainer.evaluate(full_test_loader)
results_base['baseline_all_domains'] = baseline_test_acc_base
print(f"Baseline Test Accuracy: {baseline_test_acc_base:.4f}")
```

Visual Comparison

```
domains = list(results_base.keys())
accuracies = [results_base[d] for d in domains]

plt.figure()
plt.bar(domains, accuracies)
plt.xticks(rotation=45, ha='right')
plt.ylabel('Accuracy')
plt.title('Leave-One-Domain-Out vs. Baseline Accuracy (ViT Base)')
plt.tight_layout()
plt.show()
```

Final Results

```
print("\Final Results (LODO Accuracy):")
for domain, acc in results_base.items():
    print(f"{domain}: {acc:.4f}")

avg_acc = sum(results_base.values()) / len(results_base)
print(f"\nAverage Accuracy: {avg_acc:.4f}")
```

WinKawaks/ViT Small

Leave-One-Domain-Out (LODO) Training

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=(0.5,), std=(0.5,))
])

results_small = {}

for test_domain in DOMAINS:
    print(f"\Testing on domain: {test_domain}")
    train_domains = [d for d in DOMAINS if d != test_domain]

    # Load datasets
    dataset = PACSDataset(DATA_ROOT, DOMAINS, transform)
```

```
    train_loaders = [dataset.get_dataloader(d, train=True) for d in
train_domains]
    val_loaders = [dataset.get_dataloader(d, train=False) for d in
train_domains]
    test_loader = dataset.get_dataloader(test_domain, train=False)

    # Concatenate datasets
    train_ds = ConcatDataset([dl.dataset for dl in train_loaders])
    val_ds = ConcatDataset([dl.dataset for dl in val_loaders])

    train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE,
shuffle=True)
    val_loader = DataLoader(val_ds, batch_size=BATCH_SIZE,
shuffle=False)

    # Initialize model, optimizer, and criterion
    model_small = ViTModel(NUM_CLASSES, model_size="small")
    optimizer = optim.Adam(model_small.parameters(), lr=1e-4)
    criterion = nn.CrossEntropyLoss()
    trainer = Trainer(model_small, optimizer, criterion)

    # Train
    for epoch in range(NUM_EPOCHS):
        print(f"Epoch {epoch + 1}/{NUM_EPOCHS}")
        train_loss_small = trainer.train(train_loader)
        val_acc_small = trainer.evaluate(val_loader)
        print(f"Train Loss: {train_loss_small:.4f} | Val Acc:
{val_acc_small:.4f}")

    # Test
    test_acc_small = trainer.evaluate(test_loader)
    results_small[test_domain] = test_acc_small
    print(f"Test Accuracy on {test_domain}: {test_acc_small:.4f}")
```

Baseline

```
print("\nBaseline: training on all domains and testing on mixed
domains")
# Load full train and test sets via leave-all-in loaders
dataset_all = PACSDataset(DATA_ROOT, DOMAINS, transform)
all_train_loaders = [dataset_all.get_dataloader(d, train=True) for d
in DOMAINS]
all_test_loaders = [dataset_all.get_dataloader(d, train=False) for d
in DOMAINS]

# Concatenate
full_train_ds = ConcatDataset([dl.dataset for dl in
all_train_loaders])
full_test_ds = ConcatDataset([dl.dataset for dl in all_test_loaders])
full_train_loader = DataLoader(full_train_ds, batch_size=BATCH_SIZE,
```

```
shuffle=True)
full_test_loader = DataLoader(full_test_ds, batch_size=BATCH_SIZE,
shuffle=False)

# Initialize baseline model
# CORRECTED: Added model_size="small" to ensure the correct model is
loaded.
baseline_model_small = ViTModel(NUM_CLASSES, model_size="small")
baseline_optimizer = optim.Adam(baseline_model_small.parameters(),
lr=1e-4)
baseline_criterion = nn.CrossEntropyLoss()
baseline_trainer = Trainer(baseline_model_small, baseline_optimizer,
baseline_criterion)

# Train baseline
for epoch in range(NUM_EPOCHS):
    print(f"Baseline Epoch {epoch + 1}/{NUM_EPOCHS}")
    baseline_loss_small = baseline_trainer.train(full_train_loader)
    baseline_val_acc_small =
baseline_trainer.evaluate(full_test_loader)
    print(f"Baseline Loss: {baseline_loss_small:.4f} | Baseline Acc:
{baseline_val_acc_small:.4f}")

# Test baseline
baseline_test_acc_small = baseline_trainer.evaluate(full_test_loader)
results_small['baseline_all_domains'] = baseline_test_acc_small
print(f"Baseline Test Accuracy: {baseline_test_acc_small:.4f}")
```

Visual Comparison

```
domains = list(results_small.keys())
accuracies = [results_small[d] for d in domains]

plt.figure()
plt.bar(domains, accuracies)
plt.xticks(rotation=45, ha='right')
plt.ylabel('Accuracy')
plt.title('Leave-One-Domain-Out vs. Baseline Accuracy (ViT Small)')
plt.tight_layout()
plt.show()
```

Final Results

```
print("\Final Results (LODO Accuracy):")
for domain, acc in results_small.items():
    print(f"{domain}: {acc:.4f}")

avg_acc = sum(results_small.values()) / len(results_small)
print(f"\nAverage Accuracy: {avg_acc:.4f}")
```

WinKawaks/ViT Tiny

Leave-One-Domain-Out (LODO) Training

```
# WinKawaks/ViT Tiny
# Leave-One-Domain-Out (LODO) Training

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=(0.5,), std=(0.5,))
])

results_tiny = {}

for test_domain in DOMAINS:
    print(f"\nTesting on domain: {test_domain}")
    train_domains = [d for d in DOMAINS if d != test_domain]

    # Load datasets
    dataset = PACSDataset(DATA_ROOT, DOMAINS, transform)
    train_loaders = [dataset.get_dataloader(d, train=True) for d in
train_domains]
    val_loaders = [dataset.get_dataloader(d, train=False) for d in
train_domains]
    test_loader = dataset.get_dataloader(test_domain, train=False)

    # Concatenate datasets
    train_ds = ConcatDataset([dl.dataset for dl in train_loaders])
    val_ds = ConcatDataset([dl.dataset for dl in val_loaders])

    train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE,
shuffle=True)
    val_loader = DataLoader(val_ds, batch_size=BATCH_SIZE,
shuffle=False)

    # Initialize model, optimizer, and criterion
    model_tiny = ViTModel(NUM_CLASSES, model_size="tiny")
    # CORRECTED: Optimizer now uses parameters from model_tiny, not
model_base.
    optimizer = optim.Adam(model_tiny.parameters(), lr=1e-4)
    criterion = nn.CrossEntropyLoss()
    trainer = Trainer(model_tiny, optimizer, criterion)

    # Train
    for epoch in range(NUM_EPOCHS):
        print(f"Epoch {epoch + 1}/{NUM_EPOCHS}")
```

```
        train_loss_tiny = trainer.train(train_loader)
        val_acc_tiny = trainer.evaluate(val_loader)
        print(f"Train Loss: {train_loss_tiny:.4f} | Val Acc:
{val_acc_tiny:.4f}")

    # Test
    test_acc_tiny = trainer.evaluate(test_loader)
    results_tiny[test_domain] = test_acc_tiny
    print(f"Test Accuracy on {test_domain}: {test_acc_tiny:.4f}")
```

Baseline

```
print("\nBaseline: training on all domains and testing on mixed
domains")
# Load full train and test sets via leave-all-in loaders
dataset_all = PACSDataset(DATA_ROOT, DOMAINS, transform)
all_train_loaders = [dataset_all.get_dataloader(d, train=True) for d
in DOMAINS]
all_test_loaders = [dataset_all.get_dataloader(d, train=False) for d
in DOMAINS]

# Concatenate
full_train_ds = ConcatDataset([dl.dataset for dl in
all_train_loaders])
full_test_ds = ConcatDataset([dl.dataset for dl in all_test_loaders])
full_train_loader = DataLoader(full_train_ds, batch_size=BATCH_SIZE,
shuffle=True)
full_test_loader = DataLoader(full_test_ds, batch_size=BATCH_SIZE,
shuffle=False)

# Initialize baseline model
# CORRECTED: Added model_size="tiny" to ensure the correct model is
loaded.
baseline_model_tiny = ViTModel(NUM_CLASSES, model_size="tiny")
baseline_optimizer = optim.Adam(baseline_model_tiny.parameters(),
lr=1e-4)
baseline_criterion = nn.CrossEntropyLoss()
baseline_trainer = Trainer(baseline_model_tiny, baseline_optimizer,
baseline_criterion)

# Train baseline
for epoch in range(NUM_EPOCHS):
    print(f"Baseline Epoch {epoch + 1}/{NUM_EPOCHS}")
    baseline_loss_tiny = baseline_trainer.train(full_train_loader)
    baseline_val_acc_tiny =
baseline_trainer.evaluate(full_test_loader)
    print(f"Baseline Loss: {baseline_loss_tiny:.4f} | Baseline Acc:
{baseline_val_acc_tiny:.4f}")
```

```
# Test baseline
baseline_test_acc_tiny = baseline_trainer.evaluate(full_test_loader)
results_tiny['baseline_all_domains'] = baseline_test_acc_tiny
print(f"Baseline Test Accuracy: {baseline_test_acc_tiny:.4f}")
```

Visual Comparison

```
domains = list(results_tiny.keys())
accuracies = [results_tiny[d] for d in domains]

plt.figure()
plt.bar(domains, accuracies)
plt.xticks(rotation=45, ha='right')
plt.ylabel('Accuracy')
plt.title('Leave-One-Domain-Out vs. Baseline Accuracy (ViT Tiny)')
plt.tight_layout()
plt.show()
```

Final Results

```
print("\Final Results (LODO Accuracy):")
for domain, acc in results_tiny.items():
    print(f"{domain}: {acc:.4f}")

avg_acc = sum(results_tiny.values()) / len(results_tiny)
print(f"\nAverage Accuracy: {avg_acc:.4f}")
```

Performance Comparison

```
plt.figure(figsize=(20, 12))

for idx, domain in enumerate(DOMAINS):
    plt.subplot(2, 3, idx + 1)

    domain_accuracies = [
        results_base[domain],
        results_small[domain],
        results_tiny[domain]
    ]

    bars = plt.bar(MODELS.keys(), domain_accuracies, color=["skyblue",
"orange", "green"])
    plt.ylim(0, 1)
    plt.ylabel("Test Accuracy")
    plt.title(f"Model Comparison - {domain} Domain")
    plt.grid(axis="y", linestyle="--", alpha=0.5)
```

```
    # Add value labels on top of bars
    for i, acc in enumerate(domain_accuracies):
        plt.text(i, acc + 0.01, f"{acc:.2%}", ha="center")

# Create the baseline comparison subplot
plt.subplot(2, 3, 5)
baseline_accuracies = [
    results_base["baseline_all_domains"],
    results_small["baseline_all_domains"],
    results_tiny["baseline_all_domains"]
]
plt.bar(MODELS.keys(), baseline_accuracies, color=["skyblue",
"orange", "green"])
plt.ylim(0, 1)
plt.ylabel("Test Accuracy")
plt.title("Model Comparison - Baseline (All Domains)")
plt.grid(axis="y", linestyle="--", alpha=0.5)

for i, acc in enumerate(baseline_accuracies):
    plt.text(i, acc + 0.01, f"{acc:.2%}", ha="center")

plt.tight_layout()
plt.show()

print("\nDetailed Performance Comparison:")
print("-" * 60)
print(f"{'Domain':<15} {'Base':>10} {'Small':>10} {'Tiny':>10}")
print("-" * 60)

for domain in DOMAINS:
    print(f"{domain:<15} {results_base[domain]:>10.2%}
{results_small[domain]:>10.2%} {results_tiny[domain]:>10.2%}")

print("-" * 60)
print(f"{'Baseline':<15} {results_base['baseline_all_domains']:>10.2%}
{results_small['baseline_all_domains']:>10.2%}
{results_tiny['baseline_all_domains']:>10.2%}")
print("-" * 60)
```