```
# !pip install jcopdl==1.1.1
from jcopdl.callback import set_config, Callback
import matplotlib.pyplot as plt
```

```
# !git clone https://github.com/WiraDKP/deep_learning content/deep_learning
```

```
import torch
import torch.nn as nn
import torch.optim as optim

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
device
!ls content/deep_learning/"14 - Convolutional Neural Network"/data.zip
```

```
'content/deep_learning/14 - Convolutional Neural Network/data.zip'
```

```
# import zipfile
# import io

# zip1 = '/content/content/deep_learning/14 - Convolutional Neural Network/data.zip'

# with zipfile.ZipFile(zip1, 'r') as zip_ref:
#     zip_ref.extractall('content/deeplearning') # Ekstrak ke direktori /content/

# print(f"File {zip1} berhasil diekstrak ke /content/")
# !ls /content # Verifikasi
```

```
!ls /content # Verifikasi

```

```
content  data  __MACOSX  models  sample_data
```

```
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

bs = 128
crop_size = 224

train_transform = transforms.Compose([
    transforms.RandomRotation(20),
    transforms.RandomHorizontalFlip(),
    transforms.RandomResizedCrop(crop_size, scale=(0.8, 1.0)),
    transforms.ToTensor()
])

test_transform = transforms.Compose([
    transforms.Resize(235),
    transforms.CenterCrop(crop_size),
    transforms.ToTensor()])

trainset = datasets.ImageFolder('/content/content/deeplearning/data/train', transform=train_transform)
trainloader = DataLoader(trainset, batch_size=bs, shuffle=True, num_workers=0)
testset = datasets.ImageFolder('/content/content/deeplearning/data/test', transform=test_transform)
testloader = DataLoader(testset, batch_size=bs, shuffle=False)
```

```
label2cat, idx_class = trainset.classes, trainset.class_to_idx
```

```
feature, target = next(iter(trainloader))
feature.shape, target.shape
```

```
(torch.Size([128, 3, 224, 224]), torch.Size([128]))
```

arsitektur dan config

```
def conv_block(in_feature, out_feature, padding=1, stride=1,
                   activation="relu", maxpool=True, kernel_size=3,
                   kernel_size_pool=2, pool_stride=2)-> list[nn.Sequential]:
    """
The conv_block function is designed to construct a sequential block of convolutional layers, complete with optional activation and m

Parameters
in_feature (int): The number of input channels (input features) for the convolutional layer.
out_feature (int): The number of output channels (output features) to be produced by the convolutional layer.
padding (int, optional, default=1): The amount of padding applied to the convolutional layer.
```

```
stride (int, optional, default=1): The stride for the convolutional kernel's movement.
activation (str, optional, default="relu"): The type of activation function to use after the convolutional layer. Currently supports
maxpool (bool, optional, default=True): If True, a max pooling layer will be appended after the activation.
kernel_size (int, optional, default=3): The kernel (filter) size for the convolutional layer.
kernel_size_pool (int, optional, default=2): The kernel size for the max pooling layer, if maxpool is enabled.
pool_stride (int, optional, default=2): The stride for the max pooling layer, if maxpool is enabled.
Returns
nn.Sequential: Returns a PyTorch nn.Sequential object containing the following sequence of layers:
A Convolutional layer (nn.Conv2d)
An Activation function (e.g., nn.ReLU), if specified
A Max Pooling layer (nn.MaxPool2d), if maxpool is True
    """
    layers = [nn.Conv2d(in_feature, out_feature, kernel_size=kernel_size, padding=padding, stride=stride)]
    if activation == "relu":
        layers.append(nn.ReLU())
    elif activation == "leakyrelu":
        layers.append(nn.LeakyReLU())
    elif activation == "sigmoid":
        layers.append(nn.Sigmoid())
    elif activation == "tanh":
        layers.append(nn.Tanh())
    if maxpool:
        layers.append(nn.MaxPool2d(kernel_size=kernel_size_pool, stride=pool_stride))
    else:
        layers.append(nn.AvgPool2d(kernel_size=kernel_size_pool, stride=pool_stride))
    return nn.Sequential(*layers)


def linear_block(in_features, out_features, activation='relu', dropout=0.0):
    layers = [nn.Linear(in_features, out_features)]
    # if batch_norm:
    #     layers.append(BatchNorm1d(out_features))
    if activation == 'relu':
        layers.append(nn.ReLU())
    elif activation == 'sigmoid':
        layers.append(nn.Sigmoid())
    elif activation == 'tanh':
        layers.append(nn.Tanh())
    elif activation == 'leakyrelu':
        layers.append(nn.LeakyReLU())
    elif activation == 'softmax':
        layers.append(nn.Softmax(dim=1))
    elif activation == 'elu':
        layers.append(nn.ELU())
    elif activation == 'selu':
        layers.append(nn.SELU())
    elif activation == 'lsoftmax':
        layers.append(nn.LogSoftmax(dim=1))
    if dropout > 0.0:
        layers.append(nn.Dropout(dropout))
    return nn.Sequential(*layers)
```

```
class CNN(nn.Module):
    def __init__(self, dropout=0.0):
        super().__init__()
        self.conv = nn.Sequential(
            conv_block(3, 8),
            conv_block(8, 16),
            conv_block(16, 32),
            conv_block(32, 64),
            nn.Flatten()
        )
        self.fc = nn.Sequential(
            linear_block(64 * 14 * 14, 2**14, activation='relu', dropout=dropout),
            linear_block(2**14, 2**12, activation='relu', dropout=dropout),
            linear_block(2**12, 2**10, activation='relu', dropout=dropout),
            linear_block(2**10, 2**9, activation='relu', dropout=dropout),
            linear_block(2**9, 2**8, activation='relu'),
            linear_block(256, len(label2cat), activation='lsoftmax')

        )
    def forward(self, x):
        x = self.conv(x)
        x = self.fc(x)
        return x
```

```
config = set_config({
    "batch_size": bs,
```

```
    "crop_size": crop_size
    })
```

Model

```
model = CNN(dropout=0.1).to(device)
criterion = nn.NLLLoss()
optimizer = optim.RAdam(model.parameters(), lr=0.005)
callback = Callback(model, config, outdir='models', early_stop_patience= 25)
```

Training loop

```
def loop_fn(mode, dataset, dataloader, model, criterion, optimizer, device):
    from tqdm.auto import tqdm
    if mode == 'train':
        model.train()
    elif mode == 'test':
        model.eval()
    cost = correct = 0
    for feature, target in tqdm(dataloader, desc=mode.title()):
        feature, target = feature.to(device), target.to(device)
        output = model(feature)
        loss = criterion(output, target)

        if mode == 'train':
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()

        cost += loss.item() * feature.shape[0]
        correct += (output.argmax(dim=1) == target).sum().item()
    cost = cost / len(dataset)
    accuracy = correct / len(dataset)
    return cost, accuracy
```

```
while True:
    train_cost, train_acc = loop_fn('train', trainset, trainloader, model, criterion, optimizer, device)
    with torch.no_grad():
        test_cost, test_acc = loop_fn('test', testset, testloader, model, criterion, optimizer, device)

    callback.log(train_cost=train_cost, test_cost=test_cost, test_score=test_acc, train_score=train_acc)
    callback.save_checkpoint()
    callback.cost_runtime_plotting()
    callback.score_runtime_plotting()
    if callback.early_stopping(model, monitor='test_score', load_best_when_stop=True):
        print("Early stopping triggered.")
        callback.plot_cost()
        callback.plot_score()
        break




    print(f'Train Cost: {train_cost:.4f}, Test Cost: {test_cost:.4f}')
```

Train: 100% 8/8 [00:04<00:00, 1.81it/s]

Test: 100% 4/4 [00:01<00:00, 2.77it/s]

```
Epoch     1
Train_cost  = 0.6934 | Test_cost  = 0.6928 | Train_score = 0.5000 | Test_score = 0.5100 |
Train Cost: 0.6934, Test Cost: 0.6928
```

Train: 100% 8/8 [00:04<00:00, 1.85it/s]

Test: 100% 4/4 [00:01<00:00, 2.75it/s]

```
Epoch     2
Train_cost  = 0.6941 | Test_cost  = 0.6934 | Train_score = 0.5040 | Test_score = 0.5000 |
==> EarlyStop patience =  1 | Best test_score: 0.5100
Train Cost: 0.6941, Test Cost: 0.6934
```

Train: 100% 8/8 [00:04<00:00, 1.81it/s]

Test: 100% 4/4 [00:01<00:00, 2.78it/s]

```
Epoch     3
Train_cost  = 0.6900 | Test_cost  = 0.6766 | Train_score = 0.5080 | Test_score = 0.7050 |
Train Cost: 0.6900, Test Cost: 0.6766
```

Train: 100% 8/8 [00:04<00:00, 1.85it/s]

Test: 100% 4/4 [00:01<00:00, 2.74it/s]

```
Epoch     4
Train_cost  = 0.6597 | Test_cost  = 0.6467 | Train_score = 0.6420 | Test_score = 0.6525 |
==> EarlyStop patience =  1 | Best test_score: 0.7050
Train Cost: 0.6597, Test Cost: 0.6467
```

Train: 100% 8/8 [00:04<00:00, 1.87it/s]

Test: 100% 4/4 [00:01<00:00, 2.30it/s]

```
Epoch     5
Train_cost  = 0.6235 | Test_cost  = 0.6380 | Train_score = 0.6750 | Test_score = 0.6025 |
==> EarlyStop patience =  2 | Best test_score: 0.7050
Train Cost: 0.6235, Test Cost: 0.6380
```

Train: 100% 8/8 [00:04<00:00, 1.80it/s]

Test: 100% 4/4 [00:01<00:00, 2.77it/s]

```
Epoch     6
Train_cost  = 0.6134 | Test_cost  = 0.5646 | Train_score = 0.6760 | Test_score = 0.7200 |
Train Cost: 0.6134, Test Cost: 0.5646
```

Train: 100% 8/8 [00:04<00:00, 1.87it/s]

Test: 100% 4/4 [00:01<00:00, 2.70it/s]

```
Epoch     7
Train_cost  = 0.5677 | Test_cost  = 0.5574 | Train_score = 0.7300 | Test_score = 0.7725 |
Train Cost: 0.5677, Test Cost: 0.5574
```

Train: 100% 8/8 [00:04<00:00, 1.83it/s]

Test: 100% 4/4 [00:01<00:00, 2.70it/s]

```
Epoch     8
Train_cost  = 0.5098 | Test_cost  = 0.4770 | Train_score = 0.7580 | Test_score = 0.7750 |
Train Cost: 0.5098, Test Cost: 0.4770
```

Train: 100% 8/8 [00:04<00:00, 1.86it/s]

Test: 100% 4/4 [00:01<00:00, 2.71it/s]

```
Epoch     9
Train_cost  = 0.4748 | Test_cost  = 0.4585 | Train_score = 0.7720 | Test_score = 0.8100 |
Train Cost: 0.4748, Test Cost: 0.4585
```

Train: 100% 8/8 [00:04<00:00, 1.87it/s]

Test: 100% 4/4 [00:01<00:00, 1.91it/s]

```
Epoch    10
Train_cost  = 0.5093 | Test_cost  = 0.5203 | Train_score = 0.7650 | Test_score = 0.7925 |
==> EarlyStop patience =  1 | Best test_score: 0.8100
Train Cost: 0.5093, Test Cost: 0.5203
```

Train: 100% 8/8 [00:04<00:00, 1.87it/s]

Test: 100% 4/4 [00:01<00:00, 2.70it/s]

```
Epoch    11
Train_cost  = 0.4822 | Test_cost  = 0.4321 | Train_score = 0.7770 | Test_score = 0.8075 |
==> EarlyStop patience =  2 | Best test_score: 0.8100
Train Cost: 0.4822, Test Cost: 0.4321
```

Train: 100% 8/8 [00:04<00:00, 1.88it/s]

Test: 100% 4/4 [00:01<00:00, 2.18it/s]

```
Epoch    12
Train_cost  = 0.4096 | Test_cost  = 0.6088 | Train_score = 0.8230 | Test_score = 0.7925 |
==> EarlyStop patience =  3 | Best test_score: 0.8100
Train Cost: 0.4096, Test Cost: 0.6088
```

Train: 100% 8/8 [00:04<00:00, 1.85it/s]

Test: 100% 4/4 [00:01<00:00, 2.74it/s]

```
Epoch    13
Train_cost  = 0.5662 | Test_cost  = 0.5576 | Train_score = 0.7620 | Test_score = 0.6925 |
==> EarlyStop patience =  4 | Best test_score: 0.8100
Train Cost: 0.5662, Test Cost: 0.5576
```

Train: 100% 8/8 [00:04<00:00, 1.87it/s]

Test: 100% 4/4 [00:01<00:00, 2.71it/s]

```
Epoch    14
Train_cost  = 0.5024 | Test_cost  = 0.4352 | Train_score = 0.7460 | Test_score = 0.7975 |
==> EarlyStop patience =  5 | Best test_score: 0.8100
Train Cost: 0.5024, Test Cost: 0.4352
```

Train: 100% 8/8 [00:04<00:00, 1.80it/s]

Test: 100% 4/4 [00:01<00:00, 2.73it/s]

```
Epoch    15
Train_cost  = 0.4772 | Test_cost  = 0.4413 | Train_score = 0.7810 | Test_score = 0.7875 |
==> EarlyStop patience =  6 | Best test_score: 0.8100
Train Cost: 0.4772, Test Cost: 0.4413
```

Train: 100% 8/8 [00:04<00:00, 1.87it/s]

Test: 100% 4/4 [00:01<00:00, 2.77it/s]

```
Epoch    16
Train_cost  = 0.4113 | Test_cost  = 0.4543 | Train_score = 0.8220 | Test_score = 0.7750 |
==> EarlyStop patience =  7 | Best test_score: 0.8100
Train Cost: 0.4113, Test Cost: 0.4543
```

Train: 100% 8/8 [00:04<00:00, 1.77it/s]

Test: 100% 4/4 [00:01<00:00, 2.72it/s]

```
Epoch    17
Train_cost  = 0.3859 | Test_cost  = 0.3725 | Train_score = 0.8340 | Test_score = 0.8450 |
Train Cost: 0.3859, Test Cost: 0.3725
```

Train: 100% 8/8 [00:04<00:00, 1.87it/s]

Test: 100% 4/4 [00:01<00:00, 2.73it/s]

```
Epoch    18
Train_cost  = 0.3475 | Test_cost  = 0.3577 | Train_score = 0.8470 | Test_score = 0.8525 |
Train Cost: 0.3475, Test Cost: 0.3577
```

Train: 100% 8/8 [00:04<00:00, 1.87it/s]

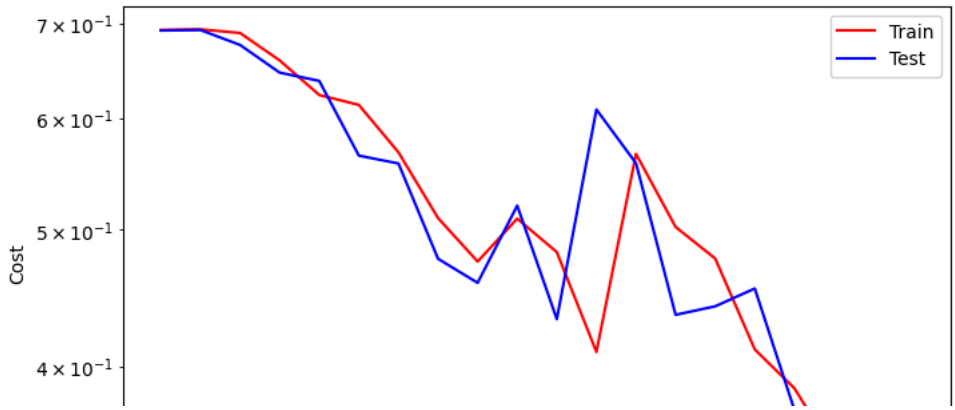
Test: 100% 4/4 [00:01<00:00, 2.73it/s]

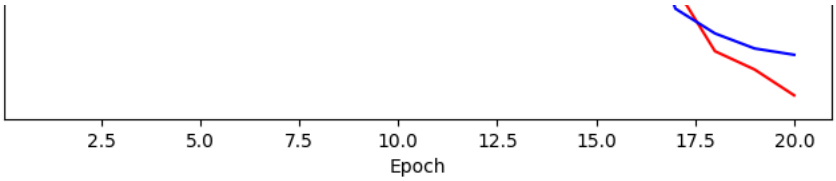
```
Epoch    19
Train_cost  = 0.3373 | Test_cost  = 0.3490 | Train_score = 0.8600 | Test_score = 0.8625 |
Train Cost: 0.3373, Test Cost: 0.3490
```

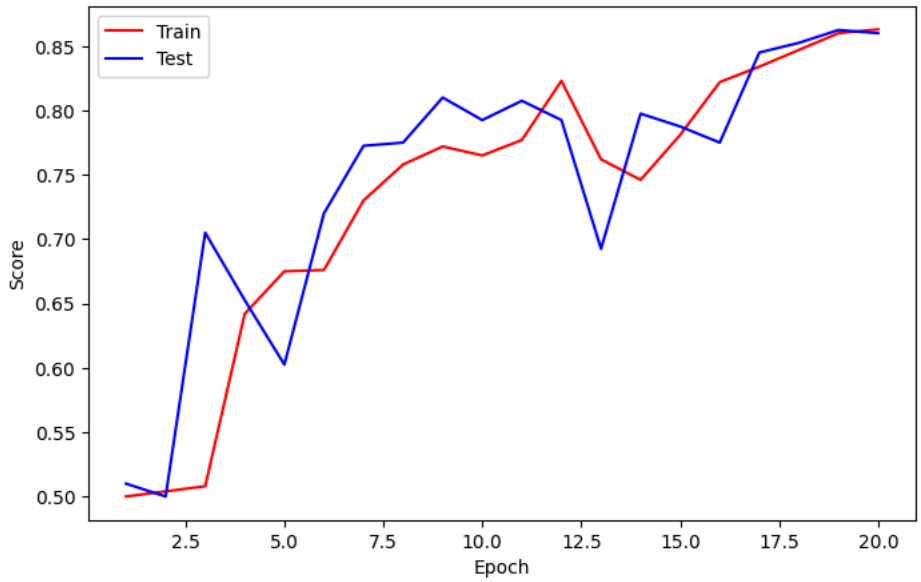
Train: 100% 8/8 [00:04<00:00, 1.86it/s]

Test: 100% 4/4 [00:01<00:00, 1.88it/s]

```
Epoch    20
Train_cost  = 0.3233 | Test_cost  = 0.3455 | Train_score = 0.8630 | Test_score = 0.8600 |
```








```
==> EarlyStop patience =  1 | Best test_score: 0.8625
Train Cost: 0.3233, Test Cost: 0.3455
```

Train: 100% 8/8 [00:04<00:00, 1.89it/s]

Test: 100% 4/4 [00:01<00:00, 2.72it/s]

```
Epoch    21
Train_cost  = 0.2922 | Test_cost  = 0.3441 | Train_score = 0.8750 | Test_score = 0.8625 |
==> EarlyStop patience =  2 | Best test_score: 0.8625
Train Cost: 0.2922, Test Cost: 0.3441
```

Train: 100% 8/8 [00:04<00:00, 1.86it/s]

Test: 100% 4/4 [00:01<00:00, 1.90it/s]

```
Epoch    22
Train_cost  = 0.3324 | Test_cost  = 0.4004 | Train_score = 0.8790 | Test_score = 0.7925 |
==> EarlyStop patience =  3 | Best test_score: 0.8625
Train Cost: 0.3324, Test Cost: 0.4004
```

Train: 100% 8/8 [00:04<00:00, 1.85it/s]

Test: 100% 4/4 [00:01<00:00, 2.68it/s]

```
Epoch    23
Train_cost  = 0.3436 | Test_cost  = 0.3069 | Train_score = 0.8500 | Test_score = 0.8825 |
Train Cost: 0.3436, Test Cost: 0.3069
```

Train: 100% 8/8 [00:04<00:00, 1.83it/s]

Test: 100% 4/4 [00:01<00:00, 2.62it/s]

```
Epoch    24
Train_cost  = 0.2771 | Test_cost  = 0.2885 | Train_score = 0.8870 | Test_score = 0.8875 |
Train Cost: 0.2771, Test Cost: 0.2885
```

Train: 100% 8/8 [00:04<00:00, 1.84it/s]

Test: 100% 4/4 [00:01<00:00, 2.08it/s]

```
Epoch    25
Train_cost  = 0.3203 | Test_cost  = 0.2938 | Train_score = 0.8740 | Test_score = 0.8725 |
==> EarlyStop patience =  1 | Best test_score: 0.8875
Train Cost: 0.3203, Test Cost: 0.2938
```

Train: 100% 8/8 [00:04<00:00, 1.79it/s]

Test: 100% 4/4 [00:01<00:00, 2.61it/s]

```
Epoch    26
Train_cost  = 0.2375 | Test_cost  = 0.2739 | Train_score = 0.9090 | Test_score = 0.9025 |
Train Cost: 0.2375, Test Cost: 0.2739
```

Train: 100% 8/8 [00:04<00:00, 1.80it/s]

Test: 100% 4/4 [00:01<00:00, 2.60it/s]

```
Epoch    27
```

```
Train_cost  = 0.2922 | Test_cost  = 0.3937 | Train_score = 0.8810 | Test_score = 0.8200 |
==> EarlyStop patience =  1 | Best test_score: 0.9025
Train Cost: 0.2922, Test Cost: 0.3937
```

Train: 100% 8/8 [00:05<00:00, 1.71it/s]

Test: 100% 4/4 [00:01<00:00, 2.61it/s]

```
Epoch    28
Train_cost  = 0.2844 | Test_cost  = 0.2689 | Train_score = 0.8790 | Test_score = 0.8900 |
==> EarlyStop patience =  2 | Best test_score: 0.9025
Train Cost: 0.2844, Test Cost: 0.2689
```

Train: 100% 8/8 [00:04<00:00, 1.77it/s]

Test: 100% 4/4 [00:01<00:00, 2.58it/s]

```
Epoch    29
Train_cost  = 0.2191 | Test_cost  = 0.2496 | Train_score = 0.9120 | Test_score = 0.9225 |
Train Cost: 0.2191, Test Cost: 0.2496
```

Train: 100% 8/8 [00:04<00:00, 1.68it/s]

Test: 100% 4/4 [00:01<00:00, 2.57it/s]

```
Epoch    30
Train_cost  = 0.2231 | Test_cost  = 0.3311 | Train_score = 0.9110 | Test_score = 0.8625 |
==> EarlyStop patience =  1 | Best test_score: 0.9225
Train Cost: 0.2231, Test Cost: 0.3311
```

Train: 100% 8/8 [00:04<00:00, 1.86it/s]

Test: 100% 4/4 [00:01<00:00, 2.63it/s]

```
Epoch    31
Train_cost  = 0.2274 | Test_cost  = 0.3297 | Train_score = 0.9010 | Test_score = 0.8800 |
==> EarlyStop patience =  2 | Best test_score: 0.9225
Train Cost: 0.2274, Test Cost: 0.3297
```

Train: 100% 8/8 [00:04<00:00, 1.63it/s]

Test: 100% 4/4 [00:01<00:00, 2.67it/s]

```
Epoch    32
Train_cost  = 0.2370 | Test_cost  = 0.3033 | Train_score = 0.9010 | Test_score = 0.8875 |
==> EarlyStop patience =  3 | Best test_score: 0.9225
Train Cost: 0.2370, Test Cost: 0.3033
```

Train: 100% 8/8 [00:04<00:00, 1.86it/s]

Test: 100% 4/4 [00:01<00:00, 2.63it/s]

```
Epoch    33
Train_cost  = 0.2011 | Test_cost  = 0.2998 | Train_score = 0.9220 | Test_score = 0.9150 |
==> EarlyStop patience =  4 | Best test_score: 0.9225
Train Cost: 0.2011, Test Cost: 0.2998
```

Train: 100% 8/8 [00:04<00:00, 1.61it/s]

Test: 100% 4/4 [00:01<00:00, 2.66it/s]

```
Epoch    34
Train_cost  = 0.2240 | Test_cost  = 0.2707 | Train_score = 0.9230 | Test_score = 0.9175 |
==> EarlyStop patience =  5 | Best test_score: 0.9225
Train Cost: 0.2240, Test Cost: 0.2707
```

Train: 100% 8/8 [00:04<00:00, 1.84it/s]

Test: 100% 4/4 [00:01<00:00, 2.66it/s]

```
Epoch    35
Train_cost  = 0.2094 | Test_cost  = 0.2985 | Train_score = 0.9100 | Test_score = 0.8900 |
==> EarlyStop patience =  6 | Best test_score: 0.9225
Train Cost: 0.2094, Test Cost: 0.2985
```

Train: 100% 8/8 [00:04<00:00, 1.55it/s]

Test: 100% 4/4 [00:01<00:00, 2.63it/s]

```
Epoch    36
Train_cost  = 0.2056 | Test_cost  = 0.2425 | Train_score = 0.9100 | Test_score = 0.9150 |
==> EarlyStop patience =  7 | Best test_score: 0.9225
Train Cost: 0.2056, Test Cost: 0.2425
```

Train: 100% 8/8 [00:04<00:00, 1.83it/s]

Test: 100% 4/4 [00:01<00:00, 2.67it/s]

```
Epoch    37
Train_cost  = 0.1748 | Test_cost  = 0.2273 | Train_score = 0.9290 | Test_score = 0.9075 |
==> EarlyStop patience =  8 | Best test_score: 0.9225
Train Cost: 0.1748, Test Cost: 0.2273
```

Train: 100% 8/8 [00:04<00:00, 1.56it/s]

Test: 100% 4/4 [00:01<00:00, 2.46it/s]

```
Epoch    38
Train_cost  = 0.1563 | Test_cost  = 0.2893 | Train_score = 0.9390 | Test_score = 0.9025 |
==> EarlyStop patience =  9 | Best test_score: 0.9225
Train Cost: 0.1563, Test Cost: 0.2893
```

Train: 100% 8/8 [00:04<00:00, 1.83it/s]

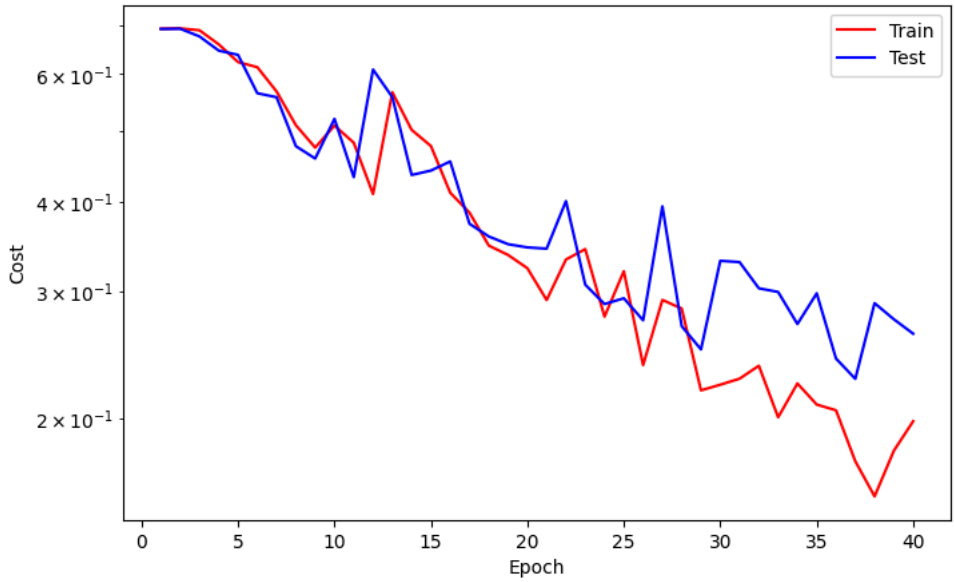
Test: 100% 4/4 [00:01<00:00, 2.58it/s]

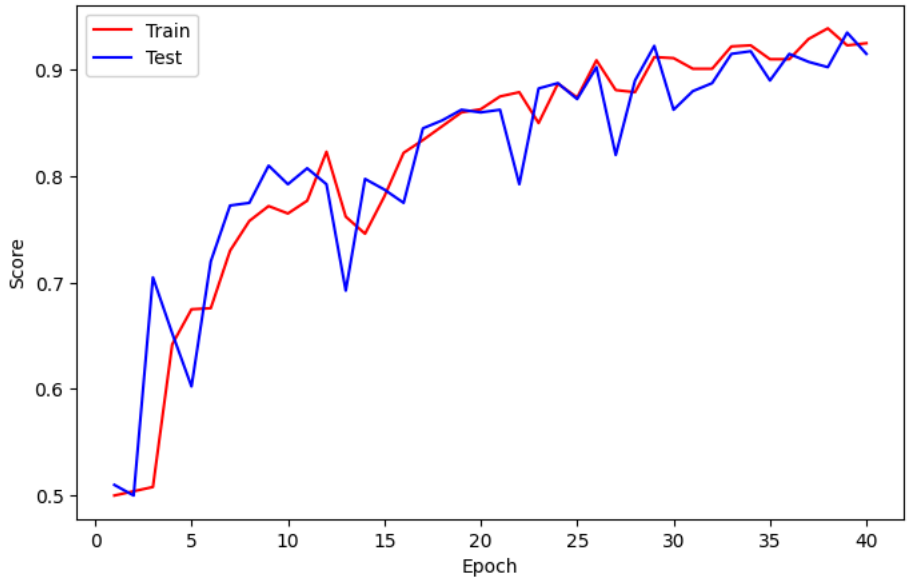
```
Epoch    39
Train_cost  = 0.1808 | Test_cost  = 0.2748 | Train_score = 0.9230 | Test_score = 0.9350 |
Train Cost: 0.1808, Test Cost: 0.2748
```

Train: 100% 8/8 [00:04<00:00, 1.61it/s]

Test: 100% 4/4 [00:01<00:00, 2.18it/s]

```
Epoch    40
Train_cost  = 0.1986 | Test_cost  = 0.2625 | Train_score = 0.9250 | Test_score = 0.9150 |
```






```
==> EarlyStop patience =  1 | Best test_score: 0.9350
Train Cost: 0.1986, Test Cost: 0.2625
```

Train: 100% 8/8 [00:04<00:00, 1.83it/s]

Test: 100% 4/4 [00:01<00:00, 2.61it/s]

```
Epoch    41
Train_cost  = 0.1998 | Test_cost  = 0.3652 | Train_score = 0.9180 | Test_score = 0.8975 |
==> EarlyStop patience =  2 | Best test_score: 0.9350
Train Cost: 0.1998, Test Cost: 0.3652
```

Train: 100% 8/8 [00:04<00:00, 1.61it/s]

Test: 100% 4/4 [00:01<00:00, 2.24it/s]

```
Epoch    42
Train_cost  = 0.2077 | Test_cost  = 0.2636 | Train_score = 0.9240 | Test_score = 0.8975 |
```

```
==> EarlyStop patience =  3 | Best test_score: 0.9350
Train Cost: 0.2077, Test Cost: 0.2636
```

Train: 100% 8/8 [00:04<00:00, 1.84it/s]

Test: 100% 4/4 [00:01<00:00, 2.61it/s]

```
Epoch    43
Train_cost  = 0.1741 | Test_cost  = 0.2049 | Train_score = 0.9240 | Test_score = 0.9175 |
==> EarlyStop patience =  4 | Best test_score: 0.9350
Train Cost: 0.1741, Test Cost: 0.2049
```

Train: 100% 8/8 [00:04<00:00, 1.67it/s]

Test: 100% 4/4 [00:01<00:00, 2.03it/s]

```
Epoch    44
Train_cost  = 0.1568 | Test_cost  = 0.2559 | Train_score = 0.9290 | Test_score = 0.9175 |
==> EarlyStop patience =  5 | Best test_score: 0.9350
Train Cost: 0.1568, Test Cost: 0.2559
```

Train: 100% 8/8 [00:04<00:00, 1.85it/s]

Test: 100% 4/4 [00:01<00:00, 2.66it/s]

```
Epoch    45
Train_cost  = 0.1361 | Test_cost  = 0.2421 | Train_score = 0.9400 | Test_score = 0.9050 |
==> EarlyStop patience =  6 | Best test_score: 0.9350
Train Cost: 0.1361, Test Cost: 0.2421
```

Train: 100% 8/8 [00:04<00:00, 1.72it/s]

Test: 100% 4/4 [00:01<00:00, 1.85it/s]

```
Epoch    46
Train_cost  = 0.1697 | Test_cost  = 0.2667 | Train_score = 0.9380 | Test_score = 0.9275 |
==> EarlyStop patience =  7 | Best test_score: 0.9350
Train Cost: 0.1697, Test Cost: 0.2667
```

Train: 100% 8/8 [00:04<00:00, 1.85it/s]

Test: 100% 4/4 [00:01<00:00, 2.65it/s]

```
Epoch    47
Train_cost  = 0.1547 | Test_cost  = 0.2194 | Train_score = 0.9380 | Test_score = 0.9250 |
==> EarlyStop patience =  8 | Best test_score: 0.9350
Train Cost: 0.1547, Test Cost: 0.2194
```

Train: 100% 8/8 [00:04<00:00, 1.86it/s]

Test: 100% 4/4 [00:01<00:00, 1.84it/s]

```
Epoch    48
Train_cost  = 0.1858 | Test_cost  = 0.2425 | Train_score = 0.9290 | Test_score = 0.9100 |
==> EarlyStop patience =  9 | Best test_score: 0.9350
Train Cost: 0.1858, Test Cost: 0.2425
```

Train: 100% 8/8 [00:04<00:00, 1.84it/s]

Test: 100% 4/4 [00:01<00:00, 2.64it/s]

```
Epoch    49
Train_cost  = 0.1689 | Test_cost  = 0.2298 | Train_score = 0.9330 | Test_score = 0.9075 |
==> EarlyStop patience = 10 | Best test_score: 0.9350
Train Cost: 0.1689, Test Cost: 0.2298
```

Train: 100% 8/8 [00:04<00:00, 1.86it/s]

Test: 100% 4/4 [00:01<00:00, 1.89it/s]

```
Epoch    50
Train_cost  = 0.1789 | Test_cost  = 0.2538 | Train_score = 0.9360 | Test_score = 0.9100 |
==> EarlyStop patience = 11 | Best test_score: 0.9350
Train Cost: 0.1789, Test Cost: 0.2538
```

Train: 100% 8/8 [00:04<00:00, 1.83it/s]

Test: 100% 4/4 [00:01<00:00, 2.64it/s]

```
Epoch    51
Train_cost  = 0.1445 | Test_cost  = 0.2886 | Train_score = 0.9420 | Test_score = 0.9225 |
==> EarlyStop patience = 12 | Best test_score: 0.9350
Train Cost: 0.1445, Test Cost: 0.2886
```

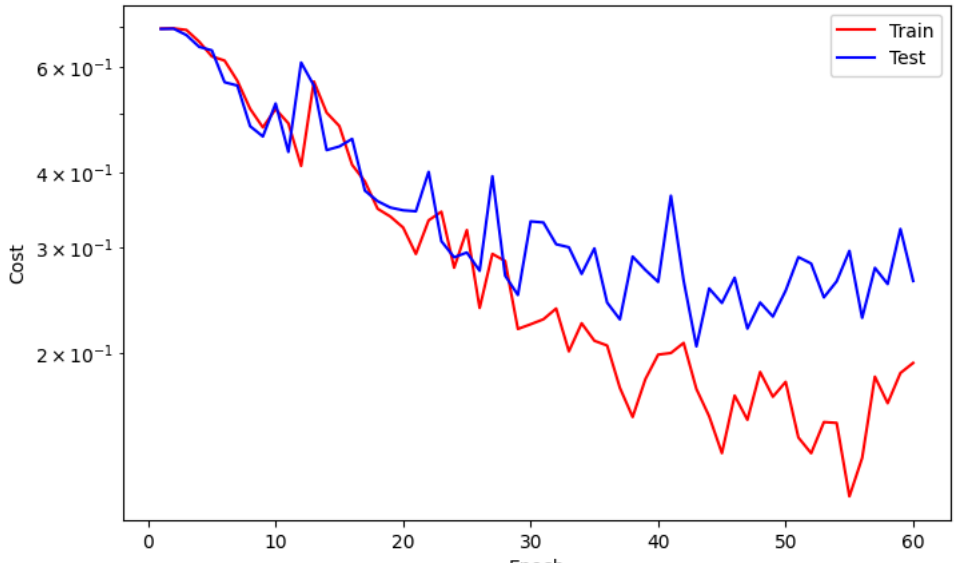
Train: 100% 8/8 [00:04<00:00, 1.72it/s]

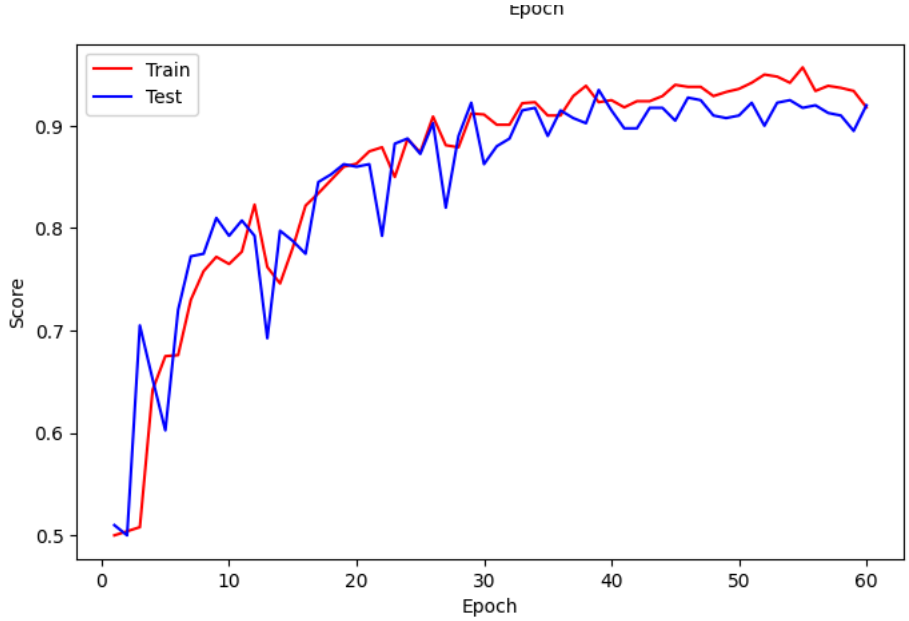
Test: 100% 4/4 [00:01<00:00, 2.01it/s]

```
Epoch    52
Train_cost  = 0.1360 | Test_cost  = 0.2816 | Train_score = 0.9500 | Test_score = 0.9000 |
==> EarlyStop patience = 13 | Best test_score: 0.9350
Train Cost: 0.1360, Test Cost: 0.2816
```

Train: 100% 8/8 [00:04<00:00, 1.84it/s]

```
Test: 100%                                          4/4 [00:01<00:00,  2.59it/s]

Epoch    53
Train_cost  = 0.1533 | Test_cost  = 0.2473 | Train_score = 0.9480 | Test_score = 0.9225 |
==> EarlyStop patience = 14 | Best test_score: 0.9350
Train Cost: 0.1533, Test Cost: 0.2473
Train: 100%                                         8/8 [00:04<00:00,  1.71it/s]

Test: 100%                                          4/4 [00:01<00:00,  1.84it/s]

Epoch    54
Train_cost  = 0.1528 | Test_cost  = 0.2630 | Train_score = 0.9420 | Test_score = 0.9250 |
==> EarlyStop patience = 15 | Best test_score: 0.9350
Train Cost: 0.1528, Test Cost: 0.2630
Train: 100%                                         8/8 [00:04<00:00,  1.85it/s]

Test: 100%                                          4/4 [00:01<00:00,  2.61it/s]

Epoch    55
Train_cost  = 0.1153 | Test_cost  = 0.2957 | Train_score = 0.9570 | Test_score = 0.9175 |
==> EarlyStop patience = 16 | Best test_score: 0.9350
Train Cost: 0.1153, Test Cost: 0.2957
Train: 100%                                         8/8 [00:04<00:00,  1.87it/s]

Test: 100%                                          4/4 [00:01<00:00,  1.86it/s]

Epoch    56
Train_cost  = 0.1336 | Test_cost  = 0.2287 | Train_score = 0.9340 | Test_score = 0.9200 |
==> EarlyStop patience = 17 | Best test_score: 0.9350
Train Cost: 0.1336, Test Cost: 0.2287
Train: 100%                                         8/8 [00:04<00:00,  1.84it/s]

Test: 100%                                          4/4 [00:01<00:00,  2.62it/s]

Epoch    57
Train_cost  = 0.1825 | Test_cost  = 0.2770 | Train_score = 0.9390 | Test_score = 0.9125 |
==> EarlyStop patience = 18 | Best test_score: 0.9350
Train Cost: 0.1825, Test Cost: 0.2770
Train: 100%                                         8/8 [00:04<00:00,  1.86it/s]

Test: 100%                                          4/4 [00:01<00:00,  1.99it/s]

Epoch    58
Train_cost  = 0.1648 | Test_cost  = 0.2603 | Train_score = 0.9370 | Test_score = 0.9100 |
==> EarlyStop patience = 19 | Best test_score: 0.9350
Train Cost: 0.1648, Test Cost: 0.2603
Train: 100%                                         8/8 [00:04<00:00,  1.83it/s]

Test: 100%                                          4/4 [00:01<00:00,  2.60it/s]

Epoch    59
Train_cost  = 0.1850 | Test_cost  = 0.3216 | Train_score = 0.9340 | Test_score = 0.8950 |
==> EarlyStop patience = 20 | Best test_score: 0.9350
Train Cost: 0.1850, Test Cost: 0.3216
Train: 100%                                         8/8 [00:04<00:00,  1.86it/s]

Test: 100%                                          4/4 [00:01<00:00,  2.21it/s]

Epoch    60
Train_cost  = 0.1923 | Test_cost  = 0.2635 | Train_score = 0.9180 | Test_score = 0.9200 |
```






```
==> EarlyStop patience = 21 | Best test_score: 0.9350
Train Cost: 0.1923, Test Cost: 0.2635
```

Train: 100% 8/8 [00:04<00:00, 1.79it/s]

Test: 100% 4/4 [00:01<00:00, 2.62it/s]

```
Epoch    61
Train_cost  = 0.1426 | Test_cost  = 0.2110 | Train_score = 0.9360 | Test_score = 0.9225 |
==> EarlyStop patience = 22 | Best test_score: 0.9350
Train Cost: 0.1426, Test Cost: 0.2110
```

Train: 100% 8/8 [00:04<00:00, 1.85it/s]

Test: 100% 4/4 [00:01<00:00, 2.20it/s]

```
Epoch    62
Train_cost  = 0.1586 | Test_cost  = 0.2626 | Train_score = 0.9370 | Test_score = 0.9100 |
==> EarlyStop patience = 23 | Best test_score: 0.9350
Train Cost: 0.1586, Test Cost: 0.2626
```

Train: 100% 8/8 [00:05<00:00, 1.77it/s]

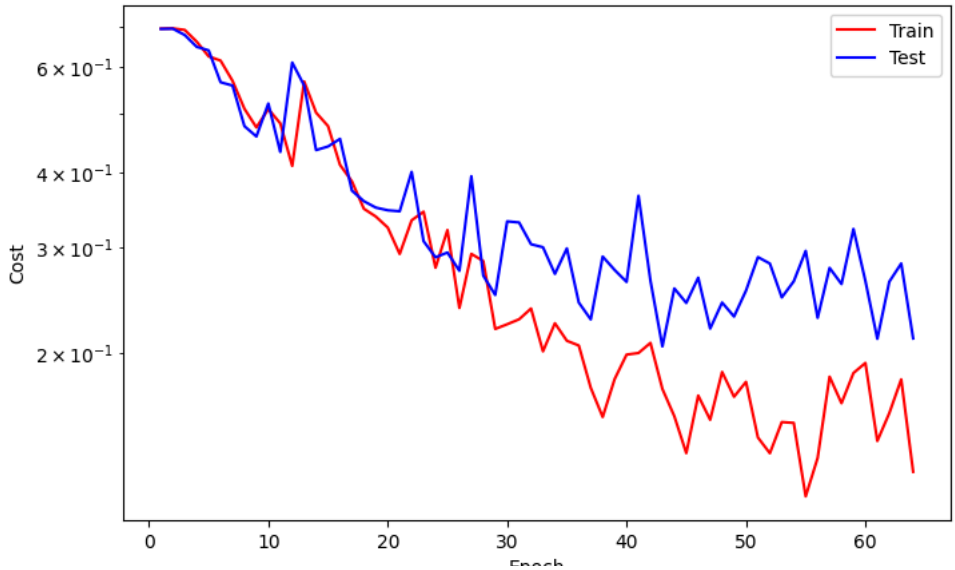
Test: 100% 4/4 [00:01<00:00, 2.59it/s]

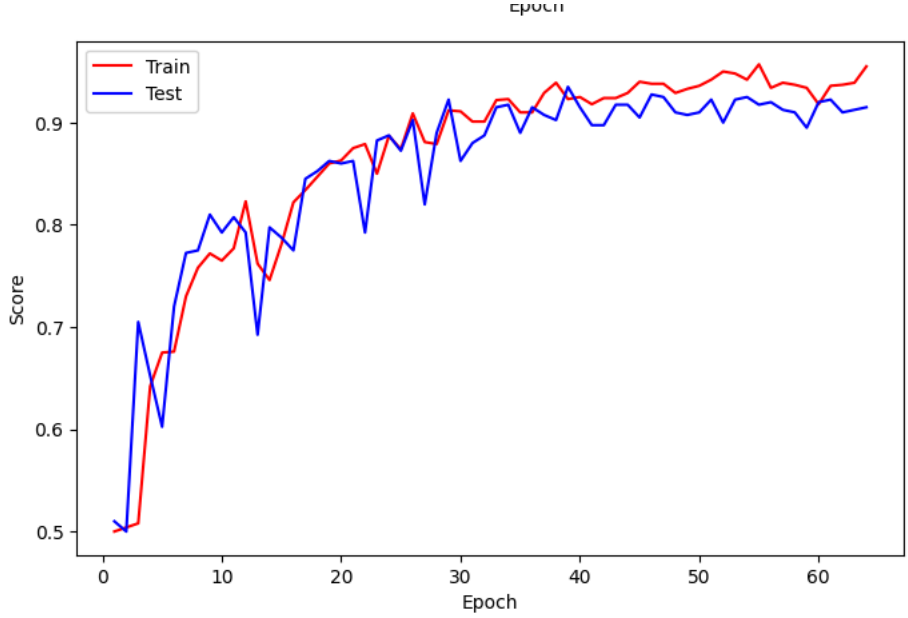
```
Epoch    63
Train_cost  = 0.1806 | Test_cost  = 0.2817 | Train_score = 0.9390 | Test_score = 0.9125 |
==> EarlyStop patience = 24 | Best test_score: 0.9350
Train Cost: 0.1806, Test Cost: 0.2817
```

Train: 100% 8/8 [00:04<00:00, 1.87it/s]

Test: 100% 4/4 [00:01<00:00, 2.62it/s]

```
Epoch    64
Train_cost  = 0.1267 | Test_cost  = 0.2113 | Train_score = 0.9550 | Test_score = 0.9150 |
==> EarlyStop patience = 25 | Best test_score: 0.9350
==> Execute Early Stopping at epoch: 64 | Best test_score: 0.9350
==> Best model is saved at models
Early stopping triggered.
```




Epoch
Train
Test
Score
0.9
0.8
0.7
0.6
0.5
0
10
20
30
40
50
60
Epoch

predict

```
feature, target = next(iter(testloader))
feature, target = feature.to(device), target.to(device)
```

```
with torch.no_grad():
    model.eval()
    output = model(feature)
    preds = output.argmax(dim=1)
```

```
from PIL import Image
import matplotlib.pyplot as plt
fig, axes = plt.subplots(6,6, figsize=(24,24))
for image, label, pred, ax in zip(feature, target, preds, axes.flatten()):
    # Reshape and transpose the image for plotting
    ax.imshow(image.cpu().reshape(3, 224, 224).permute(1, 2, 0))
    font = {'color': 'r'} if label != pred else {'color':'g'}
    label_text, pred_text = label2cat[label.item()], label2cat[pred.item()]
    ax.set_title(f'Label: {label_text}\nPred: {pred_text}', fontdict=font)
    ax.axis('off');
```