```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torchvision import models
import matplotlib.pyplot as plt
import numpy as np
import time

# Device configuration (Use GPU if available)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

# Hyperparameters
BATCH_SIZE = 64
EPOCHS = 5
LEARNING_RATE = 0.001
```

```
Using device: cuda
```

DATASET LOADING

```
# Image transforms (Augmentation + Normalization)
transform_train = transforms.Compose([
    transforms.Resize((32, 32)), # CIFAR is 32x32, CatsDogs needs resize to 64x64 or 128x128
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

transform_test = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# Load CIFAR-10
print("Loading CIFAR-10 Dataset...")
train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform_train)
test_dataset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform_test)

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)

classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

```
Loading CIFAR-10 Dataset...
100%|██████████| 170M/170M [00:03<00:00, 43.2MB/s]
```

DEFINING THE CONFIGURABLE CNN ARCHITECTURE

```
class ConfigurableCNN(nn.Module):
    def __init__(self, activation_fn_name='relu', num_classes=10):
        super(ConfigurableCNN, self).__init__()

        # Select Activation Function
        if activation_fn_name == 'relu':
            self.activation = nn.ReLU()
        elif activation_fn_name == 'tanh':
            self.activation = nn.Tanh()
        elif activation_fn_name == 'leaky_relu':
            self.activation = nn.LeakyReLU(0.01)


        # Layer 1
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.pool = nn.MaxPool2d(2, 2)

        # Layer 2
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(64)
```

```
        # Layer 3
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(128)

        # Fully Connected Layers
        # 32x32 image -> pool -> 16x16 -> pool -> 8x8 -> pool -> 4x4
        self.flatten_dim = 128 * 4 * 4

        self.fc1 = nn.Linear(self.flatten_dim, 512)
        self.dropout = nn.Dropout(0.5)
        self.fc2 = nn.Linear(512, num_classes)

    def forward(self, x):
        # Block 1
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.activation(x)
        x = self.pool(x)

        # Block 2
        x = self.conv2(x)
        x = self.bn2(x)
        x = self.activation(x)
        x = self.pool(x)

        # Block 3
        x = self.conv3(x)
        x = self.bn3(x)
        x = self.activation(x)
        x = self.pool(x)

        # Classifier
        x = x.view(-1, self.flatten_dim)
        x = self.fc1(x)
        x = self.activation(x)
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

INITIALIZATION TECHNIQUES

```
def apply_weight_init(model, init_type='random'):
    def init_weights(m):
        if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
            if init_type == 'xavier':
                nn.init.xavier_uniform_(m.weight)
            elif init_type == 'kaiming':
                nn.init.kaiming_uniform_(m.weight, nonlinearity='relu')
            elif init_type == 'random':
                nn.init.normal_(m.weight, mean=0.0, std=0.02) # Simple random normal

            if m.bias is not None:
                nn.init.constant_(m.bias, 0)

    model.apply(init_weights)
```

TRAINING UTILITY FUNCTION

```
def train_model(model, train_loader, criterion, optimizer, epochs=5):
    model.train()
    history = {'loss': [], 'acc': []}

    for epoch in range(epochs):
        running_loss = 0.0
        correct = 0
        total = 0

        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
```

```
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        epoch_acc = 100 * correct / total
        epoch_loss = running_loss / len(train_loader)
        history['loss'].append(epoch_loss)
        history['acc'].append(epoch_acc)

        print(f"Epoch [{epoch+1}/{epochs}], Loss: {epoch_loss:.4f}, Acc: {epoch_acc:.2f}%")

    return history
```

EVALUATION UTILITY FUNCTION

```
def evaluate_model(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    acc = 100 * correct / total
    return acc
```

EXPERIMENT CONFIGURATIONS (AROUND 27 DIFFERENT CONFIG)

```
# Experiment Configurations
activations = ['relu', 'tanh', 'leaky_relu']
initializations = ['xavier', 'kaiming', 'random']
optimizers_list = ['sgd', 'adam', 'rmsprop']

results = []
best_acc = 0
best_config = {}
best_model_state = None

print("Starting Experiments...")

# Iterate through all combinations
for act in activations:
    for init in initializations:
        for opt_name in optimizers_list:
            print(f"\n--- Config: Act={act}, Init={init}, Optim={opt_name} ---")

            # 1. Initialize Model
            model = ConfigurableCNN(activation_fn_name=act, num_classes=10).to(device)

            # 2. Apply Weight Init
            apply_weight_init(model, init_type=init)

            # 3. Setup Optimizer
            criterion = nn.CrossEntropyLoss()
            if opt_name == 'sgd':
                optimizer = optim.SGD(model.parameters(), lr=LEARNING_RATE, momentum=0.9)
            elif opt_name == 'adam':
                optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
            elif opt_name == 'rmsprop':
                optimizer = optim.RMSprop(model.parameters(), lr=LEARNING_RATE)

            # 4. Train
            # Using 2 epochs to speed up demonstration (Use 5+ for real lab)
            train_hist = train_model(model, train_loader, criterion, optimizer, epochs=30)

            # 5. Evaluate
            test_acc = evaluate_model(model, test_loader)
```

```
            test_acc = evaluate_model(model, test_loader)
            print(f"Test Accuracy: {test_acc:.2f}%")

            # Store results
            results.append({
                'config': f"{act}_{init}_{opt_name}",
                'acc': test_acc
            })

            # Check for best model
            if test_acc > best_acc:
                best_acc = test_acc
                best_config = {'act': act, 'init': init, 'opt': opt_name}
                best_model_state = model.state_dict()
                torch.save(model.state_dict(), "best_custom_cnn.pth")

print(f"\nBest Custom Configuration: {best_config} with Acc: {best_acc:.2f}%")
```

```
Epoch [9/30], Loss: 0.5387, Acc: 81.27%
Epoch [10/30], Loss: 0.5090, Acc: 82.20%
Epoch [11/30], Loss: 0.4770, Acc: 83.36%
Epoch [12/30], Loss: 0.4428, Acc: 84.71%
Epoch [13/30], Loss: 0.4146, Acc: 85.62%
Epoch [14/30], Loss: 0.3900, Acc: 86.23%
Epoch [15/30], Loss: 0.3591, Acc: 87.62%
Epoch [16/30], Loss: 0.3452, Acc: 87.73%
Epoch [17/30], Loss: 0.3175, Acc: 88.85%
Epoch [18/30], Loss: 0.3022, Acc: 89.48%
Epoch [19/30], Loss: 0.2837, Acc: 89.97%
Epoch [20/30], Loss: 0.2669, Acc: 90.72%
Epoch [21/30], Loss: 0.2502, Acc: 91.07%
Epoch [22/30], Loss: 0.2411, Acc: 91.48%
Epoch [23/30], Loss: 0.2289, Acc: 91.91%
Epoch [24/30], Loss: 0.2161, Acc: 92.48%
Epoch [25/30], Loss: 0.2008, Acc: 92.91%
Epoch [26/30], Loss: 0.1941, Acc: 93.14%
Epoch [27/30], Loss: 0.1853, Acc: 93.40%
Epoch [28/30], Loss: 0.1769, Acc: 93.86%
Epoch [29/30], Loss: 0.1703, Acc: 94.10%
Epoch [30/30], Loss: 0.1621, Acc: 94.31%
Test Accuracy: 82.90%

--- Config: Act=leaky_relu, Init=random, Optim=rmsprop ---
Epoch [1/30], Loss: 1.6577, Acc: 42.30%
Epoch [2/30], Loss: 1.1508, Acc: 58.99%
Epoch [3/30], Loss: 0.9477, Acc: 66.86%
Epoch [4/30], Loss: 0.8321, Acc: 70.96%
Epoch [5/30], Loss: 0.7494, Acc: 74.17%
Epoch [6/30], Loss: 0.6672, Acc: 76.89%
Epoch [7/30], Loss: 0.6149, Acc: 78.52%
Epoch [8/30], Loss: 0.5608, Acc: 80.48%
Epoch [9/30], Loss: 0.5173, Acc: 82.02%
Epoch [10/30], Loss: 0.4752, Acc: 83.46%
Epoch [11/30], Loss: 0.4418, Acc: 84.65%
Epoch [12/30], Loss: 0.4139, Acc: 85.55%
Epoch [13/30], Loss: 0.3834, Acc: 86.70%
Epoch [14/30], Loss: 0.3557, Acc: 87.69%
Epoch [15/30], Loss: 0.3341, Acc: 88.25%
Epoch [16/30], Loss: 0.3163, Acc: 88.97%
Epoch [17/30], Loss: 0.2975, Acc: 89.66%
Epoch [18/30], Loss: 0.2808, Acc: 90.11%
Epoch [19/30], Loss: 0.2696, Acc: 90.69%
Epoch [20/30], Loss: 0.2483, Acc: 91.27%
Epoch [21/30], Loss: 0.2419, Acc: 91.58%
Epoch [22/30], Loss: 0.2282, Acc: 91.96%
Epoch [23/30], Loss: 0.2239, Acc: 92.14%
Epoch [24/30], Loss: 0.2099, Acc: 92.74%
Epoch [25/30], Loss: 0.2054, Acc: 92.95%
Epoch [26/30], Loss: 0.1910, Acc: 93.29%
Epoch [27/30], Loss: 0.1862, Acc: 93.48%
Epoch [28/30], Loss: 0.1820, Acc: 93.69%
Epoch [29/30], Loss: 0.1746, Acc: 94.04%
Epoch [30/30], Loss: 0.1738, Acc: 93.88%
Test Accuracy: 82.50%

Best Custom Configuration: {'act': 'leaky_relu', 'init': 'xavier', 'opt': 'adam'} with Acc: 83.02%
```

```
print("Starting Transfer Learning (ResNet-18)")

# 1. Load Pretrained Model
resnet = models.resnet18(weights=models.ResNet18_Weights.IMAGENET1K_V1)

# 2. Freeze parameters (optional, but standard for fine-tuning feature extractors)
```

```
# For this lab, we might want to unfreeze to fine-tune everything,
# but let's just modify the final layer as is standard.
for param in resnet.parameters():
    param.requires_grad = False

# 3. Modify the final Fully Connected layer
# ResNet18 input to fc is 512
num_ftrs = resnet.fc.in_features
resnet.fc = nn.Linear(num_ftrs, 10) # 10 classes for CIFAR-10

resnet = resnet.to(device)

# 4. Train ResNet
# We only train the final layer, so it converges fast
optimizer_res = optim.Adam(resnet.fc.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

train_model(resnet, train_loader, criterion, optimizer_res, epochs=30)
resnet_acc = evaluate_model(resnet, test_loader)

print(f"ResNet-18 Accuracy: {resnet_acc:.2f}%")
print(f"Custom CNN Best Accuracy: {best_acc:.2f}%")
```

```
Starting Transfer Learning (ResNet-18)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/checkpoints/resnet18-f37072fd
100%|██████████| 44.7M/44.7M [00:00<00:00, 203MB/s]
Epoch [1/30], Loss: 1.7274, Acc: 39.51%
Epoch [2/30], Loss: 1.6023, Acc: 44.17%
Epoch [3/30], Loss: 1.5883, Acc: 44.67%
Epoch [4/30], Loss: 1.5800, Acc: 44.88%
Epoch [5/30], Loss: 1.5759, Acc: 45.36%
Epoch [6/30], Loss: 1.5675, Acc: 45.45%
Epoch [7/30], Loss: 1.5718, Acc: 45.32%
Epoch [8/30], Loss: 1.5699, Acc: 45.63%
Epoch [9/30], Loss: 1.5614, Acc: 45.69%
Epoch [10/30], Loss: 1.5695, Acc: 45.44%
Epoch [11/30], Loss: 1.5709, Acc: 45.25%
Epoch [12/30], Loss: 1.5626, Acc: 45.42%
Epoch [13/30], Loss: 1.5667, Acc: 45.70%
Epoch [14/30], Loss: 1.5726, Acc: 45.26%
Epoch [15/30], Loss: 1.5616, Acc: 45.74%
Epoch [16/30], Loss: 1.5668, Acc: 45.38%
Epoch [17/30], Loss: 1.5668, Acc: 45.62%
Epoch [18/30], Loss: 1.5675, Acc: 45.82%
Epoch [19/30], Loss: 1.5675, Acc: 45.24%
Epoch [20/30], Loss: 1.5683, Acc: 45.51%
Epoch [21/30], Loss: 1.5685, Acc: 45.60%
Epoch [22/30], Loss: 1.5684, Acc: 45.45%
Epoch [23/30], Loss: 1.5649, Acc: 45.62%
Epoch [24/30], Loss: 1.5693, Acc: 45.61%
Epoch [25/30], Loss: 1.5613, Acc: 45.74%
Epoch [26/30], Loss: 1.5597, Acc: 45.93%
Epoch [27/30], Loss: 1.5668, Acc: 45.44%
Epoch [28/30], Loss: 1.5720, Acc: 45.39%
Epoch [29/30], Loss: 1.5646, Acc: 45.67%
Epoch [30/30], Loss: 1.5685, Acc: 45.59%
ResNet-18 Accuracy: 45.80%
Custom CNN Best Accuracy: 83.02%
```