In [1]:

```python
### ======================================================================= #
###                        BIOLOGICALLY INSPIRED CNN                        #
### ======================================================================= #

### ------------------------------- ###
###           IMPORTS               ###
### ------------------------------- ###
import os
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import Dataset
import numpy as np
import time
from tqdm import tqdm
import random
import matplotlib.pyplot as plt
from torchvision.models import MobileNet_V3_Small_Weights

### ------------------------------- ###
###         CONFIGURATION           ###
### ------------------------------- ###
FAST_MODE = False
USE_AMP = True
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
FAST_EPOCHS = 5
NOISE_LEVELS = [0.01, 0.1, 0.3]
FAST_BATCH_SIZE = 64
STOCHASTIC_MASK_PROB = 0.2

print(f"Using device: {DEVICE}")
print(f"CUDA available: {torch.cuda.is_available()}")

### ------------------------------- ###
###       DATASET PREPARATION        ###
### ------------------------------- ###
class BiologicalVisionDataset(Dataset):
    def __init__(self, base_dataset, transform_fn=None):
        self.base_dataset = base_dataset
        self.transform_fn = transform_fn
        self.indices = list(range(len(base_dataset)))

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

    def __getitem__(self, idx):
        img, label = self.base_dataset[self.indices[idx]]
        if self.transform_fn:
            img = self.transform_fn(img)
        return img, label

def get_data_loaders(noise_std=0.01):
    os.makedirs('./data', exist_ok=True)
    transform = transforms.Compose([
```

```
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ])

    train_set = torchvision.datasets.CIFAR10(root='./data', train=True, downl
    test_set = torchvision.datasets.CIFAR10(root='./data', train=False, downl

    def add_noise(img):
        noise = torch.randn_like(img) * noise_std
        return torch.clamp(img + noise, 0, 1)

    def add_occlusion(img):
        occl_size = 50
        h, w = img.shape[1:]
        x = np.random.randint(0, w-occl_size)
        y = np.random.randint(0, h-occl_size)
        img[:, y:y+occl_size, x:x+occl_size] = 0
        return img

    return (
        torch.utils.data.DataLoader(
            BiologicalVisionDataset(train_set),
            batch_size=FAST_BATCH_SIZE,
            shuffle=True,
            num_workers=2,
            pin_memory=True
        ),
        {
            'clean': torch.utils.data.DataLoader(
                BiologicalVisionDataset(test_set),
                batch_size=FAST_BATCH_SIZE,
                shuffle=False,
                num_workers=2
            ),
            'noisy': torch.utils.data.DataLoader(
                BiologicalVisionDataset(test_set, add_noise),
                batch_size=FAST_BATCH_SIZE,
                shuffle=False,
                num_workers=2
            ),
            'occluded': torch.utils.data.DataLoader(
                BiologicalVisionDataset(test_set, add_occlusion),
                batch_size=FAST_BATCH_SIZE,
                shuffle=False,
                num_workers=2
            )
        }
    )

### ------------------------------- ###
###  BIOLOGICAL MECHANISMS (UPDATED) ###
### ------------------------------- ###
class SelfAttention(nn.Module):
    def __init__(self, in_dim):
        super().__init__()
```

```python
        self.channel_in = in_dim
        self.query_conv = nn.Conv2d(in_dim, in_dim//16, 1)
        self.key_conv = nn.Conv2d(in_dim, in_dim//16, 1)
        self.value_conv = nn.Conv2d(in_dim, in_dim, 1)
        self.gamma = nn.Parameter(torch.ones(1))
        self.softmax = nn.Softmax(dim=-1)

    def forward(self, x):
        batch_size, C, H, W = x.size()
        Q = self.query_conv(x).view(batch_size, -1, H*W)
        K = self.key_conv(x).view(batch_size, -1, H*W)
        V = self.value_conv(x).view(batch_size, -1, H*W)

        energy = torch.bmm(Q.permute(0,2,1), K) / np.sqrt(C)
        attention = self.softmax(energy)

        out = torch.bmm(V, attention.permute(0,2,1))
        out = out.view(batch_size, C, H, W)
        return self.gamma * out + x

class LateralInhibition(nn.Module):
    def __init__(self, channels, kernel_size=7):
        super().__init__()
        self.conv = nn.Conv2d(channels, channels, kernel_size,
                              padding=kernel_size//2, bias=False)
        nn.init.normal_(self.conv.weight, mean=-0.05, std=0.02)
        self.alpha = nn.Parameter(torch.ones(1))

    def forward(self, x):
        self.conv.weight.data.clamp_(max=0)
        return x + self.alpha * self.conv(x)

class BioRegularizer(nn.Module):
    def __init__(self, p=STOCHASTIC_MASK_PROB):
        super().__init__()
        self.p = p

    def forward(self, x):
        if self.training:
            mask = torch.bernoulli((1-self.p)*torch.ones_like(x))
            return x * mask / (1-self.p)
        return x

### ------------------------------- ###
###       MODEL ARCHITECTURE        ###
### ------------------------------- ###
class ExperimentalCNN(nn.Module):
    def __init__(self, use_inhib=True, use_attn=True):
        super().__init__()
        weights = MobileNet_V3_Small_Weights.DEFAULT
        base = torchvision.models.mobilenet_v3_small(weights=weights)
        base.classifier[-1] = nn.Linear(1024, 10)
        self.features = base.features

        with torch.no_grad():
            dummy = torch.randn(1, 3, 224, 224)
            self.actual_channels = self.features(dummy).size(1)
```

```python
        self.use_inhib = use_inhib
        self.use_attn = use_attn

        if use_inhib:
            self.inhibition = LateralInhibition(self.actual_channels)
        if use_attn:
            self.attention = SelfAttention(self.actual_channels)
        if use_inhib and use_attn:
            self.mixing = nn.Parameter(torch.tensor([0.5]))

        self.bio_reg = BioRegularizer()
        self.avgpool = nn.AdaptiveAvgPool2d(1)
        self.classifier = base.classifier

    def forward(self, x):
        x = self.features(x)

        if self.use_inhib and self.use_attn:
            attn = self.attention(x)
            inhib = self.inhibition(x)
            gamma = torch.sigmoid(self.mixing)
            x = gamma * attn + (1 - gamma) * inhib
        elif self.use_inhib:
            x = self.inhibition(x)
        elif self.use_attn:
            x = self.attention(x)

        x = self.bio_reg(x)
        x = self.avgpool(x)
        return self.classifier(x.flatten(1))

class ControlCNN(nn.Module):
    def __init__(self):
        super().__init__()
        weights = MobileNet_V3_Small_Weights.DEFAULT
        base = torchvision.models.mobilenet_v3_small(weights=weights)
        base.classifier[-1] = nn.Linear(1024, 10)
        self.features = base.features
        self.avgpool = nn.AdaptiveAvgPool2d(1)
        self.classifier = base.classifier

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

### ------------------------------- ###
###    TRAINING & EVALUATION        ###
### ------------------------------- ###
def train_model(model, train_loader, test_loaders, model_type=None):
    model.to(DEVICE)
    optimizer = torch.optim.AdamW(model.parameters(), lr=1e-4, weight_decay=1
    criterion = nn.CrossEntropyLoss()
    scaler = torch.amp.GradScaler('cuda', enabled=USE_AMP)

    train_accuracies = []
```

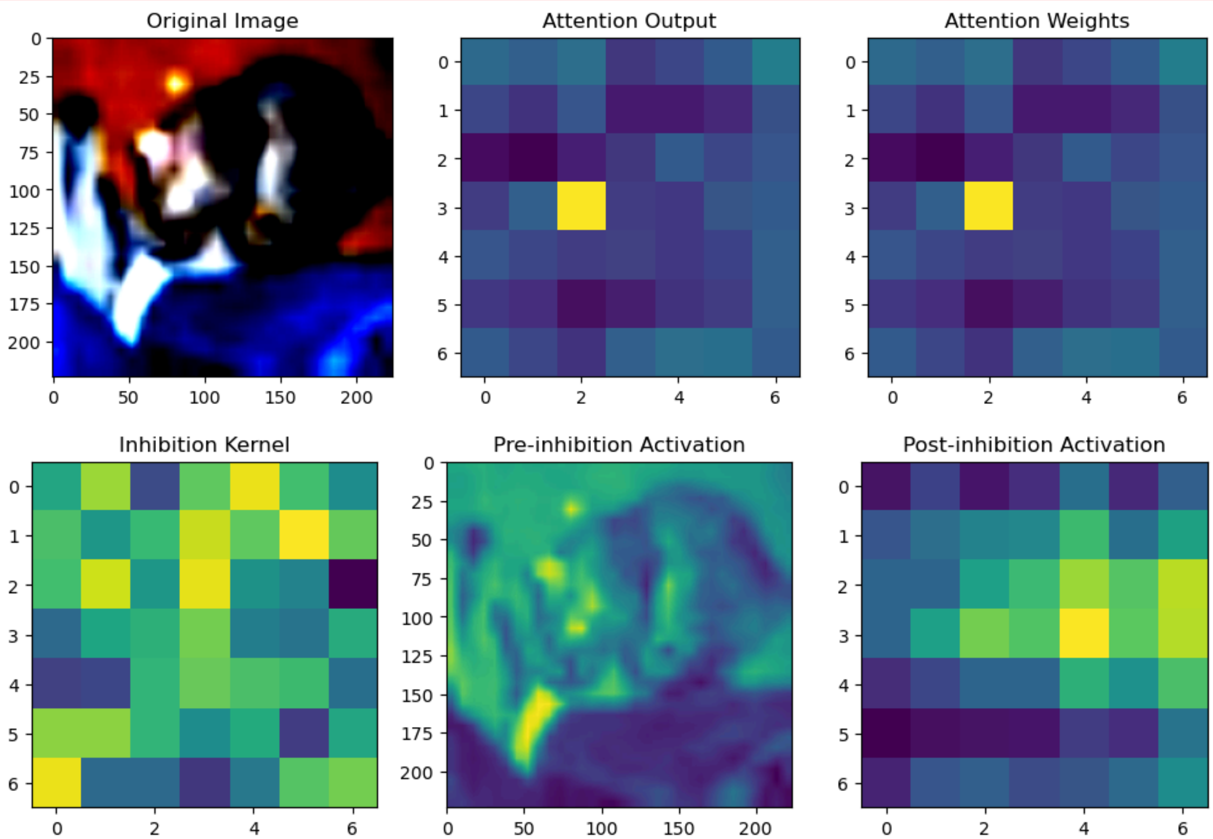
```python
    val_accuracies = []

    for epoch in range(FAST_EPOCHS):
        model.train()
        epoch_loss = 0
        correct = 0
        total = 0
        start_time = time.time()

        pbar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{FAST_EPOCHS}")
        for inputs, labels in pbar:
            inputs = inputs.to(DEVICE, non_blocking=True)
            labels = labels.to(DEVICE, non_blocking=True)

            optimizer.zero_grad(set_to_none=True)

            with torch.amp.autocast('cuda', enabled=USE_AMP):
                outputs = model(inputs)
                loss = criterion(outputs, labels)

            scaler.scale(loss).backward()
            torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
            scaler.step(optimizer)
            scaler.update()

            epoch_loss += loss.item()
            _, predicted = outputs.max(1)
            total += labels.size(0)
            correct += predicted.eq(labels).sum().item()
            pbar.set_postfix({'loss': f"{loss.item():.3f}", 'acc': f"{100*cor

        # Validation
        model.eval()
        val_acc = evaluate_model(model, test_loaders['clean'])
        val_accuracies.append(val_acc)
        train_accuracies.append(100 * correct / total)

        # Print adaptive mixing parameter
        if hasattr(model, 'use_inhib') and hasattr(model, 'use_attn'):
            if model.use_inhib and model.use_attn:
                print(f"Attention weight: {torch.sigmoid(model.mixing).item()

        print(f"\nEpoch {epoch+1} Summary:")
        print(f"  Train Acc: {train_accuracies[-1]:.2f}%")
        print(f"  Val Acc: {val_acc:.2f}%")

    return train_accuracies, val_accuracies

def evaluate_model(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs = inputs.to(DEVICE)
            labels = labels.to(DEVICE)
            outputs = model(inputs)
```

```
            _, predicted = outputs.max(1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    return 100 * correct / total

### -------------------------------- ###
###      MECHANISM VISUALIZATION     ###
### -------------------------------- ###
def visualize_biological_mechanisms(model, sample_input):
    activations = {}

    def hook_fn(name):
        def hook(model, input, output):
            activations[name] = output.detach().cpu()
        return hook

    hooks = []
    if hasattr(model, 'attention'):
        hooks.append(model.attention.register_forward_hook(hook_fn('attention
    if hasattr(model, 'inhibition'):
        hooks.append(model.inhibition.register_forward_hook(hook_fn('inhibiti

    with torch.no_grad():
        model(sample_input.to(DEVICE))

    if 'attention' in activations:
        plt.figure(figsize=(12, 4))
        plt.subplot(131)
        plt.title("Original Image")
        plt.imshow(sample_input[0].permute(1,2,0).cpu().numpy())
        plt.subplot(132)
        plt.title("Attention Output")
        plt.imshow(activations['attention'][0,0].cpu().numpy())
        plt.subplot(133)
        plt.title("Attention Weights")
        plt.imshow(model.attention.gamma.item() * activations['attention'][0,
        plt.show()

    if 'inhibition' in activations:
        plt.figure(figsize=(12, 4))
        plt.subplot(131)
        plt.title("Inhibition Kernel")
        plt.imshow(model.inhibition.conv.weight[0,0].detach().cpu().numpy())
        plt.subplot(132)
        plt.title("Pre-inhibition Activation")
        plt.imshow(sample_input[0,0].cpu().numpy())
        plt.subplot(133)
        plt.title("Post-inhibition Activation")
        plt.imshow(activations['inhibition'][0,0].cpu().numpy())
        plt.show()

    for hook in hooks:
        hook.remove()

### -------------------------------- ###
###       EXPERIMENTAL ANALYSIS      ###
### -------------------------------- ###
```

```
if __name__ == "__main__":
    all_results = {}
    configs = [
        ('Control', ControlCNN, {}),
        ('Attention Only', ExperimentalCNN, {'use_attn': True}),
        ('Inhibition Only', ExperimentalCNN, {'use_inhib': True}),
        ('Adaptive Combined', ExperimentalCNN, {'use_attn': True, 'use_inhib'
    ]

    for noise in NOISE_LEVELS:
        print(f"\n{'='*40}")
        print(f"Training at Noise Level: {noise}")
        print(f"{'='*40}")

        train_loader, test_loaders = get_data_loaders(noise)
        noise_results = {}

        for model_name, model_class, model_kwargs in configs:
            print(f"\nTraining {model_name}")
            model = model_class(**model_kwargs)
            train_acc, val_acc = train_model(model, train_loader, test_loader

            # Full evaluation
            final_results = {
                'train_acc': train_acc,
                'val_acc': val_acc,
                'test_clean': evaluate_model(model, test_loaders['clean']),
                'test_noisy': evaluate_model(model, test_loaders['noisy']),
                'test_occluded': evaluate_model(model, test_loaders['occludec
            }
            noise_results[model_name] = final_results

            # Visualization
            sample_input, _ = next(iter(test_loaders['clean']))
            visualize_biological_mechanisms(model, sample_input)

        all_results[noise] = noise_results

    # Generate comparative plots
    plt.figure(figsize=(15,5))
    for i, noise in enumerate(NOISE_LEVELS):
        plt.subplot(1,3,i+1)
        for (model_name, _, _) in configs:
            accs = [
                all_results[noise][model_name]['test_clean'],
                all_results[noise][model_name]['test_noisy'],
                all_results[noise][model_name]['test_occluded']
            ]
            plt.plot(accs, 'o-', label=model_name)
        plt.title(f"σ={noise}")
        plt.xticks([0,1,2], ['Clean','Noisy','Occluded'])
        plt.ylim(0,100)
    plt.legend()
    plt.tight_layout()
    plt.show()
```

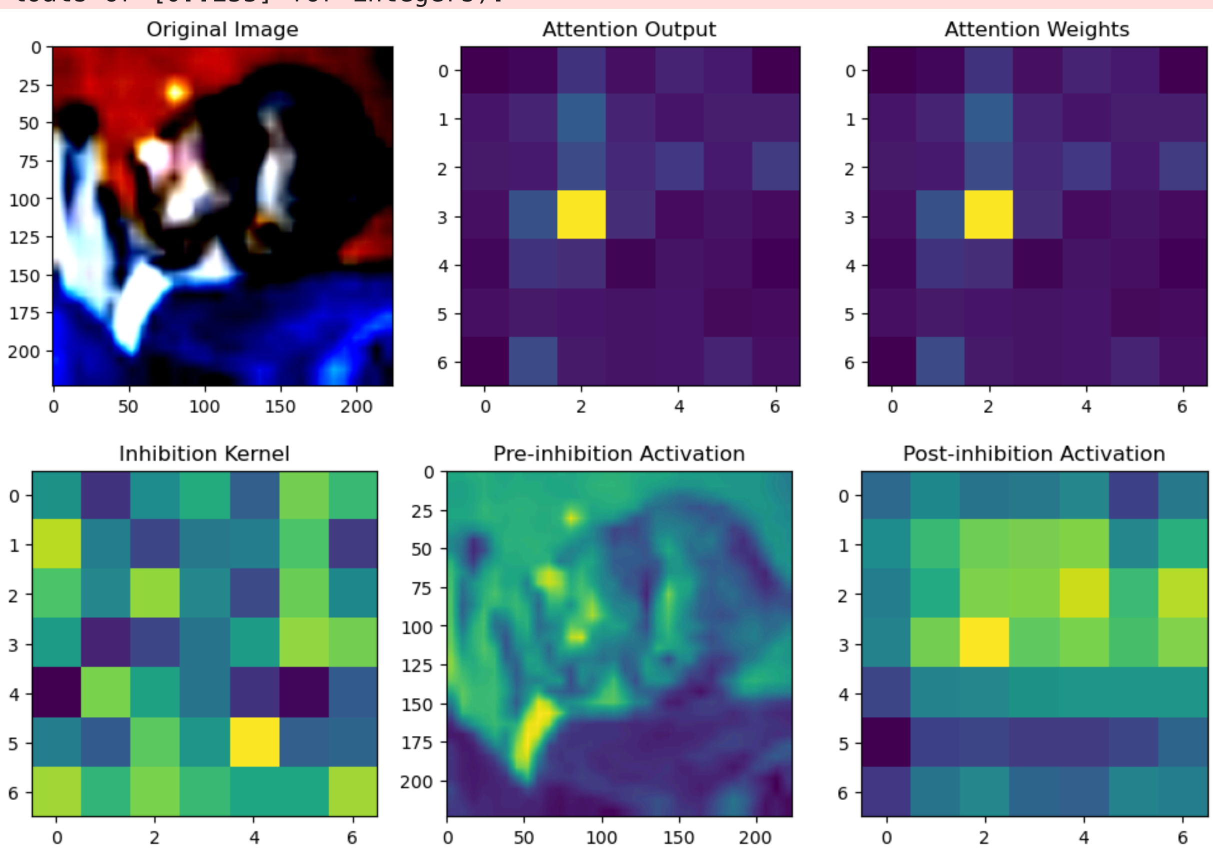
```
Using device: cuda
CUDA available: True

========================================
Training at Noise Level: 0.01
========================================

Training Control
Epoch 1/5: 100%|██████████| 782/782 [00:29<00:00, 26.12it/s, loss=0.242, acc=
81.71%]
Epoch 1 Summary:
  Train Acc: 81.71%
  Val Acc: 91.51%
Epoch 2/5: 100%|██████████| 782/782 [00:28<00:00, 27.90it/s, loss=1.013, acc=
92.93%]
Epoch 2 Summary:
  Train Acc: 92.93%
  Val Acc: 92.71%
Epoch 3/5: 100%|██████████| 782/782 [00:28<00:00, 27.82it/s, loss=0.437, acc=
95.10%]
Epoch 3 Summary:
  Train Acc: 95.10%
  Val Acc: 93.72%
Epoch 4/5: 100%|██████████| 782/782 [00:28<00:00, 27.52it/s, loss=0.430, acc=
95.95%]
Epoch 4 Summary:
  Train Acc: 95.95%
  Val Acc: 93.80%
Epoch 5/5: 100%|██████████| 782/782 [00:27<00:00, 28.21it/s, loss=0.011, acc=
97.38%]
Epoch 5 Summary:
  Train Acc: 97.38%
  Val Acc: 93.96%

Training Attention Only
Epoch 1/5: 100%|██████████| 782/782 [00:30<00:00, 25.43it/s, loss=0.363, acc=
67.19%]
Attention weight: 0.63

Epoch 1 Summary:
  Train Acc: 67.19%
  Val Acc: 86.08%
Epoch 2/5: 100%|██████████| 782/782 [00:30<00:00, 25.23it/s, loss=0.804, acc=
88.60%]
Attention weight: 0.63

Epoch 2 Summary:
  Train Acc: 88.60%
  Val Acc: 91.08%
Epoch 3/5: 100%|██████████| 782/782 [00:30<00:00, 25.46it/s, loss=0.236, acc=
92.58%]
```

```
Attention weight: 0.64

Epoch 3 Summary:
  Train Acc: 92.58%
  Val Acc: 92.68%
Epoch 4/5: 100%|██████████| 782/782 [00:30<00:00, 25.36it/s, loss=0.397, acc=
94.55%]
Attention weight: 0.64

Epoch 4 Summary:
  Train Acc: 94.55%
  Val Acc: 92.89%
Epoch 5/5: 100%|██████████| 782/782 [00:30<00:00, 25.38it/s, loss=0.022, acc=
95.61%]
Attention weight: 0.64

Epoch 5 Summary:
  Train Acc: 95.61%
  Val Acc: 92.81%
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```

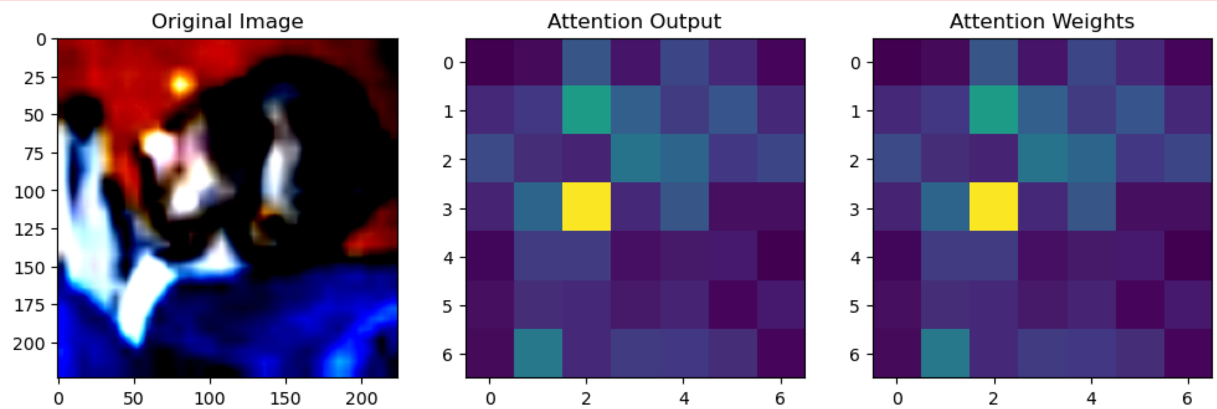


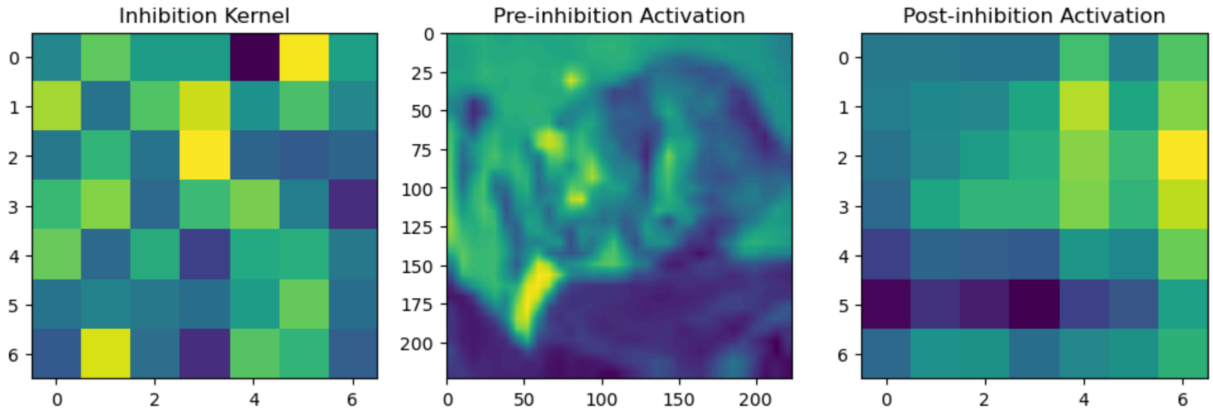

```
Training Inhibition Only
Epoch 1/5: 100%|██████████| 782/782 [00:31<00:00, 24.91it/s, loss=0.796, acc=
65.97%]
Attention weight: 0.63

Epoch 1 Summary:
  Train Acc: 65.97%
  Val Acc: 86.81%
```

```
Epoch 2/5: 100%|██████████| 782/782 [00:31<00:00, 24.98it/s, loss=0.695, acc=
88.41%]
Attention weight: 0.63

Epoch 2 Summary:
  Train Acc: 88.41%
  Val Acc: 90.44%
Epoch 3/5: 100%|██████████| 782/782 [00:31<00:00, 25.08it/s, loss=0.067, acc=
92.72%]
Attention weight: 0.64

Epoch 3 Summary:
  Train Acc: 92.72%
  Val Acc: 92.44%
Epoch 4/5: 100%|██████████| 782/782 [00:31<00:00, 25.09it/s, loss=0.222, acc=
94.72%]
Attention weight: 0.64

Epoch 4 Summary:
  Train Acc: 94.72%
  Val Acc: 92.00%
Epoch 5/5: 100%|██████████| 782/782 [00:31<00:00, 25.09it/s, loss=0.433, acc=
95.65%]
Attention weight: 0.64

Epoch 5 Summary:
  Train Acc: 95.65%
  Val Acc: 92.87%
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```



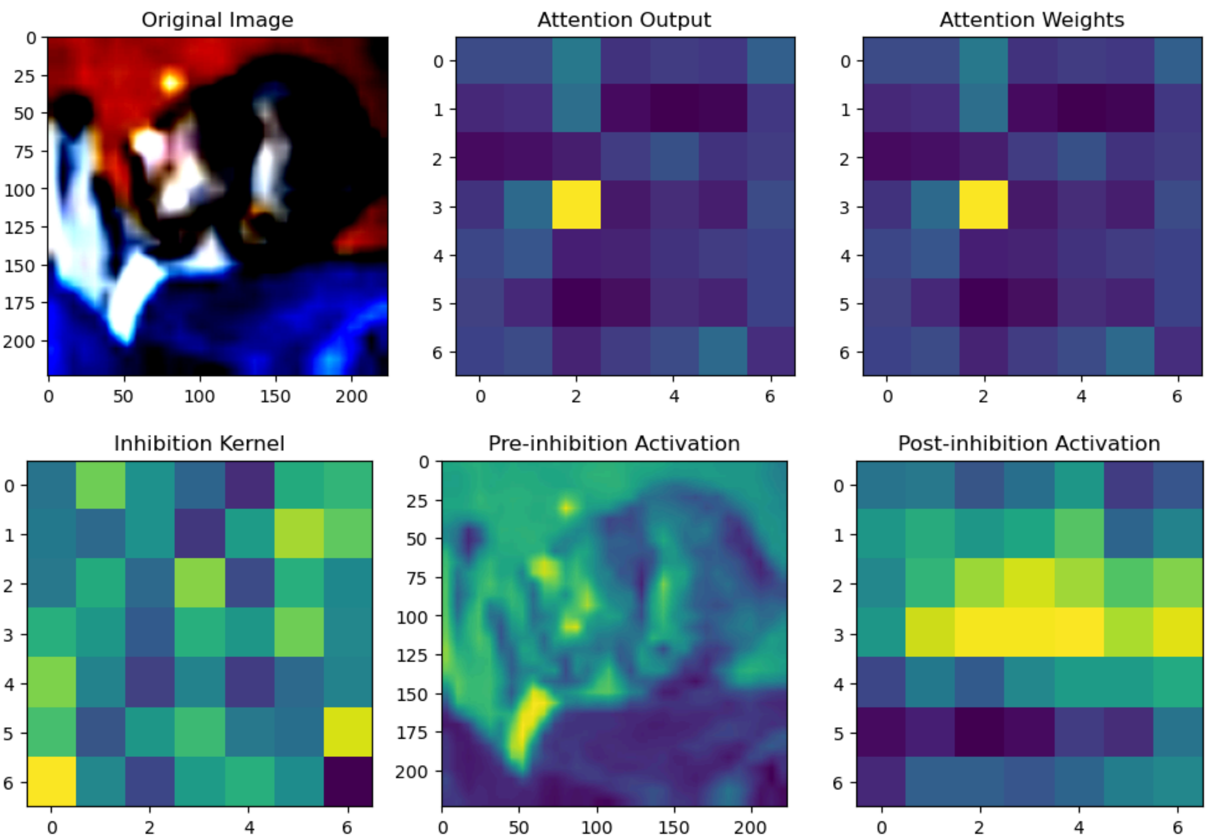
```
Training Adaptive Combined
Epoch 1/5: 100%|██████████| 782/782 [00:31<00:00, 25.04it/s, loss=0.667, acc=
67.93%]
Attention weight: 0.63

Epoch 1 Summary:
  Train Acc: 67.93%
  Val Acc: 87.25%
Epoch 2/5: 100%|██████████| 782/782 [00:31<00:00, 25.11it/s, loss=0.844, acc=
89.01%]
Attention weight: 0.63

Epoch 2 Summary:
  Train Acc: 89.01%
  Val Acc: 91.24%
Epoch 3/5: 100%|██████████| 782/782 [00:31<00:00, 25.20it/s, loss=0.401, acc=
92.77%]
Attention weight: 0.64

Epoch 3 Summary:
  Train Acc: 92.77%
  Val Acc: 92.38%
Epoch 4/5: 100%|██████████| 782/782 [00:31<00:00, 25.10it/s, loss=0.190, acc=
94.85%]
Attention weight: 0.64

Epoch 4 Summary:
  Train Acc: 94.85%
  Val Acc: 92.81%
Epoch 5/5: 100%|██████████| 782/782 [00:31<00:00, 25.18it/s, loss=0.257, acc=
95.84%]
Attention weight: 0.64

Epoch 5 Summary:
  Train Acc: 95.84%
  Val Acc: 92.49%
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```





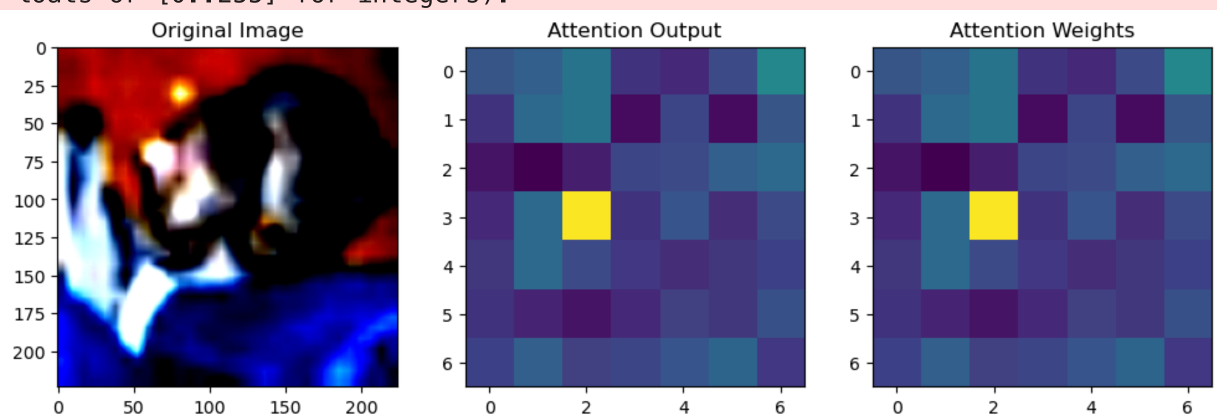

```
========================================
Training at Noise Level: 0.1
========================================

Training Control
Epoch 1/5: 100%|██████████| 782/782 [00:28<00:00, 27.23it/s, loss=0.082, acc=
82.37%]
Epoch 1 Summary:
  Train Acc: 82.37%
  Val Acc: 91.84%
Epoch 2/5: 100%|██████████| 782/782 [00:28<00:00, 27.04it/s, loss=0.031, acc=
93.07%]
Epoch 2 Summary:
  Train Acc: 93.07%
  Val Acc: 92.90%
Epoch 3/5: 100%|██████████| 782/782 [00:28<00:00, 27.29it/s, loss=0.238, acc=
95.20%]
Epoch 3 Summary:
  Train Acc: 95.20%
  Val Acc: 93.98%
Epoch 4/5: 100%|██████████| 782/782 [00:28<00:00, 27.27it/s, loss=0.004, acc=
96.77%]
Epoch 4 Summary:
  Train Acc: 96.77%
  Val Acc: 94.02%
Epoch 5/5: 100%|██████████| 782/782 [00:29<00:00, 26.89it/s, loss=0.877, acc=
97.08%]
Epoch 5 Summary:
  Train Acc: 97.08%
  Val Acc: 93.31%

Training Attention Only
Epoch 1/5: 100%|██████████| 782/782 [00:31<00:00, 24.61it/s, loss=0.565, acc=
66.98%]
Attention weight: 0.63

Epoch 1 Summary:
  Train Acc: 66.98%
  Val Acc: 86.58%
Epoch 2/5: 100%|██████████| 782/782 [00:31<00:00, 24.65it/s, loss=0.636, acc=
88.81%]
```

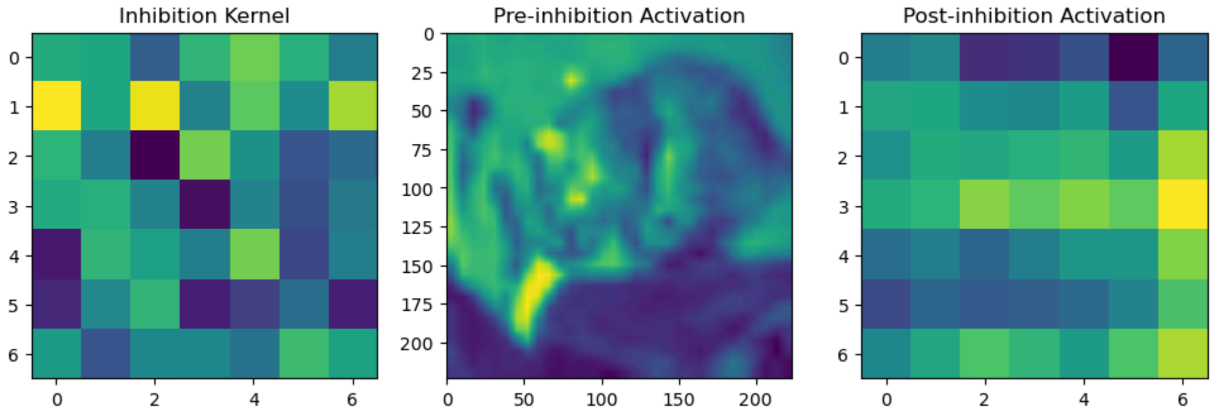
```
Attention weight: 0.63

Epoch 2 Summary:
  Train Acc: 88.81%
  Val Acc: 91.17%
Epoch 3/5: 100%|██████████| 782/782 [00:31<00:00, 24.68it/s, loss=0.161, acc=
92.84%]
Attention weight: 0.64

Epoch 3 Summary:
  Train Acc: 92.84%
  Val Acc: 92.41%
Epoch 4/5: 100%|██████████| 782/782 [00:31<00:00, 24.94it/s, loss=0.158, acc=
94.88%]
Attention weight: 0.64

Epoch 4 Summary:
  Train Acc: 94.88%
  Val Acc: 93.17%
Epoch 5/5: 100%|██████████| 782/782 [00:31<00:00, 24.76it/s, loss=0.719, acc=
95.62%]
Attention weight: 0.64

Epoch 5 Summary:
  Train Acc: 95.62%
  Val Acc: 93.17%
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```



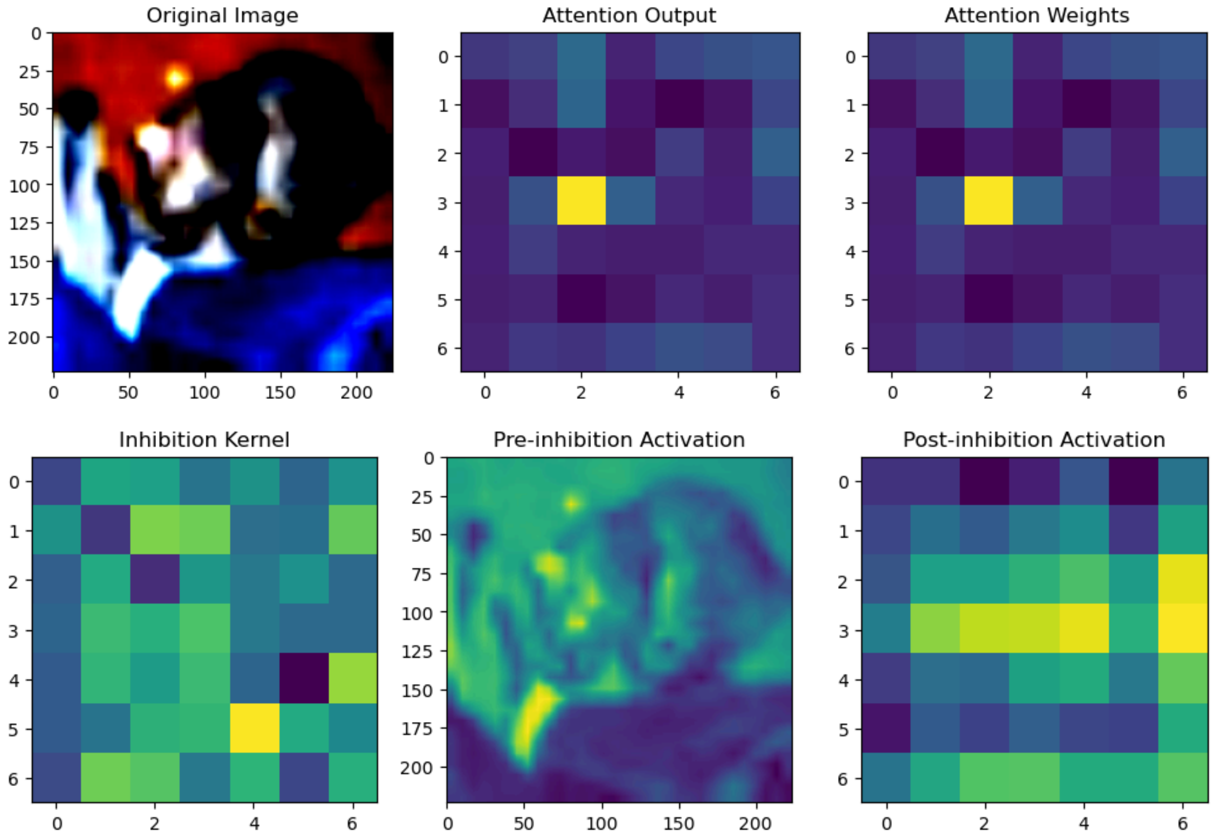

```
Training Inhibition Only
```

```
Epoch 1/5: 100%|██████████| 782/782 [00:31<00:00, 24.70it/s, loss=0.183, acc=
65.01%]
Attention weight: 0.63

Epoch 1 Summary:
  Train Acc: 65.01%
  Val Acc: 86.40%
Epoch 2/5: 100%|██████████| 782/782 [00:31<00:00, 24.59it/s, loss=0.978, acc=
87.91%]
Attention weight: 0.63

Epoch 2 Summary:
  Train Acc: 87.91%
  Val Acc: 89.75%
Epoch 3/5: 100%|██████████| 782/782 [00:31<00:00, 25.01it/s, loss=0.067, acc=
92.37%]
Attention weight: 0.64

Epoch 3 Summary:
  Train Acc: 92.37%
  Val Acc: 92.24%
Epoch 4/5: 100%|██████████| 782/782 [00:31<00:00, 24.79it/s, loss=0.311, acc=
94.48%]
Attention weight: 0.64

Epoch 4 Summary:
  Train Acc: 94.48%
  Val Acc: 91.95%
Epoch 5/5: 100%|██████████| 782/782 [00:31<00:00, 24.80it/s, loss=0.020, acc=
95.45%]
Attention weight: 0.64

Epoch 5 Summary:
  Train Acc: 95.45%
  Val Acc: 92.24%
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```



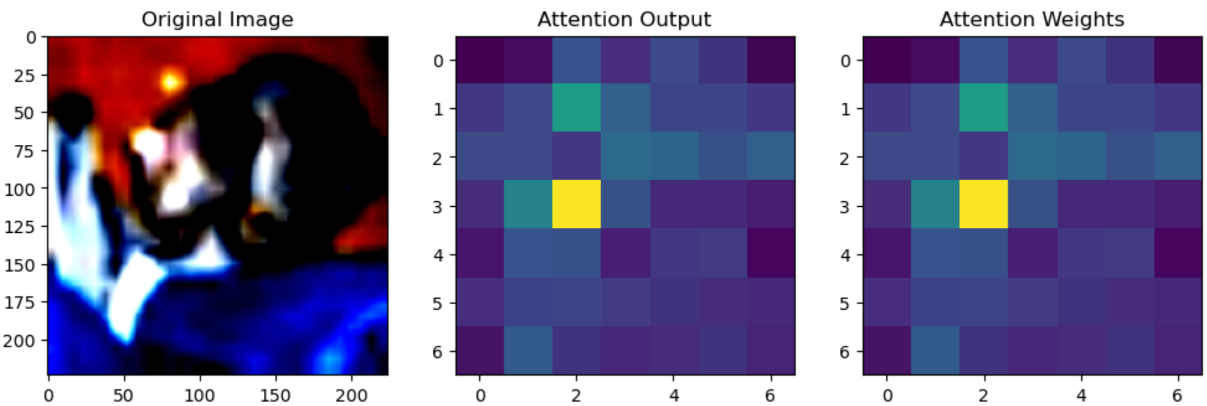


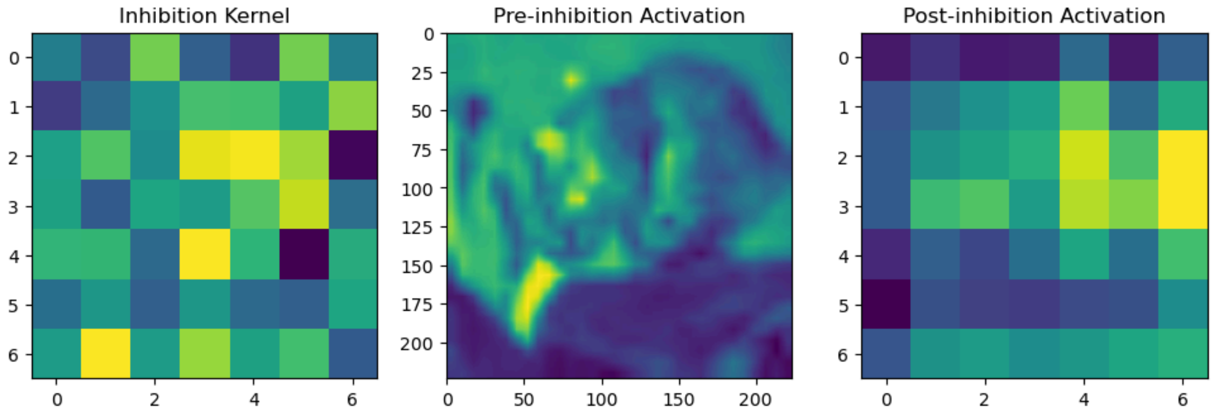

```
Training Adaptive Combined
Epoch 1/5: 100%|██████████| 782/782 [00:31<00:00, 24.82it/s, loss=0.343, acc=
64.47%]
Attention weight: 0.63

Epoch 1 Summary:
  Train Acc: 64.47%
  Val Acc: 85.34%
Epoch 2/5: 100%|██████████| 782/782 [00:31<00:00, 24.82it/s, loss=0.791, acc=
87.78%]
Attention weight: 0.63

Epoch 2 Summary:
  Train Acc: 87.78%
  Val Acc: 89.70%
Epoch 3/5: 100%|██████████| 782/782 [00:31<00:00, 24.80it/s, loss=0.267, acc=
92.10%]
Attention weight: 0.64

Epoch 3 Summary:
  Train Acc: 92.10%
  Val Acc: 92.35%
Epoch 4/5: 100%|██████████| 782/782 [00:30<00:00, 25.23it/s, loss=0.152, acc=
94.39%]
Attention weight: 0.64

Epoch 4 Summary:
  Train Acc: 94.39%
  Val Acc: 92.24%
Epoch 5/5: 100%|██████████| 782/782 [00:30<00:00, 25.27it/s, loss=0.344, acc=
95.48%]
Attention weight: 0.64

Epoch 5 Summary:
  Train Acc: 95.48%
  Val Acc: 92.79%
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```



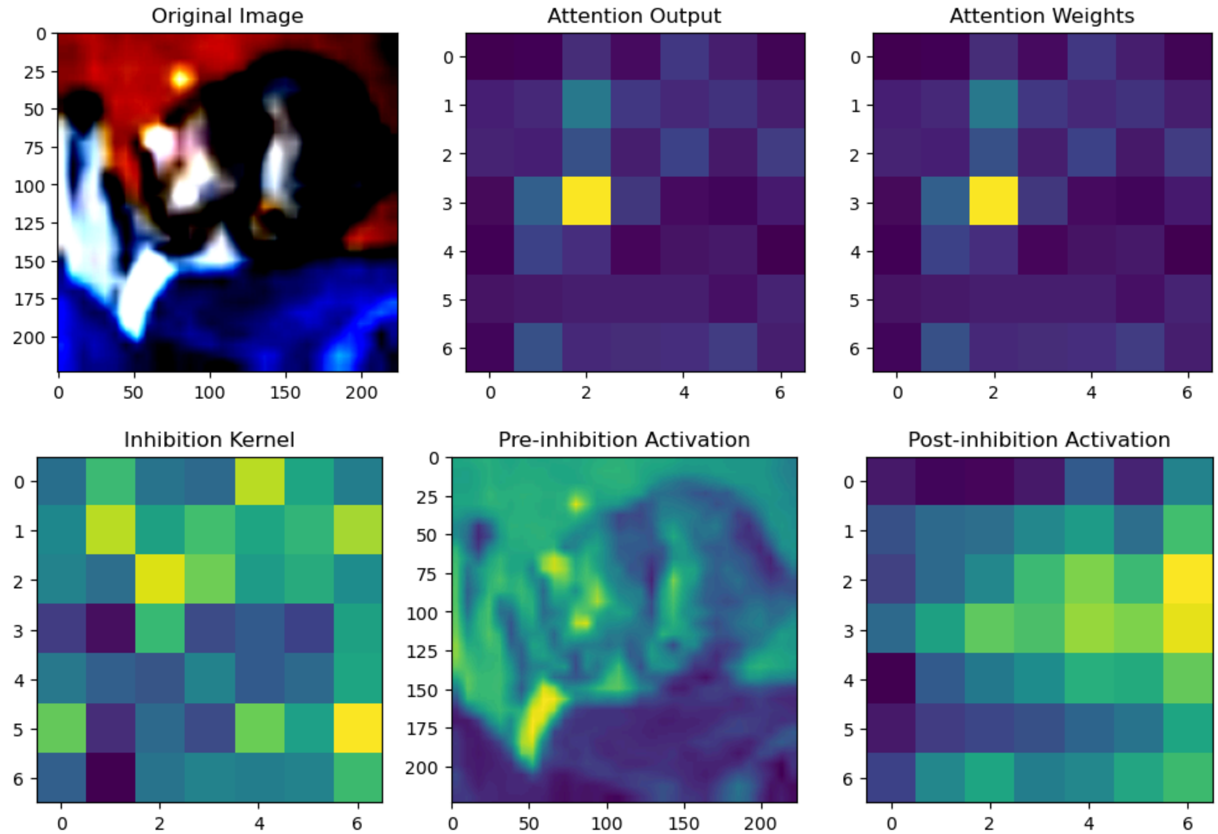

```
========================================
Training at Noise Level: 0.3
========================================

Training Control
Epoch 1/5: 100%|██████████| 782/782 [00:28<00:00, 27.74it/s, loss=0.277, acc=
82.24%]
Epoch 1 Summary:
  Train Acc: 82.24%
  Val Acc: 91.44%
Epoch 2/5: 100%|██████████| 782/782 [00:28<00:00, 27.65it/s, loss=0.114, acc=
92.83%]
Epoch 2 Summary:
  Train Acc: 92.83%
  Val Acc: 93.22%
Epoch 3/5: 100%|██████████| 782/782 [00:28<00:00, 27.44it/s, loss=0.288, acc=
95.02%]
Epoch 3 Summary:
  Train Acc: 95.02%
  Val Acc: 93.73%
Epoch 4/5: 100%|██████████| 782/782 [00:29<00:00, 26.91it/s, loss=0.516, acc=
96.48%]
Epoch 4 Summary:
  Train Acc: 96.48%
  Val Acc: 93.43%
Epoch 5/5: 100%|██████████| 782/782 [00:28<00:00, 27.09it/s, loss=0.010, acc=
96.89%]
```

```
Epoch 5 Summary:
  Train Acc: 96.89%
  Val Acc: 93.21%

Training Attention Only
Epoch 1/5: 100%|██████████| 782/782 [00:31<00:00, 25.18it/s, loss=0.673, acc=
66.55%]
Attention weight: 0.63

Epoch 1 Summary:
  Train Acc: 66.55%
  Val Acc: 85.70%
Epoch 2/5: 100%|██████████| 782/782 [00:31<00:00, 24.98it/s, loss=0.076, acc=
88.10%]
Attention weight: 0.63

Epoch 2 Summary:
  Train Acc: 88.10%
  Val Acc: 90.82%
Epoch 3/5: 100%|██████████| 782/782 [00:31<00:00, 25.09it/s, loss=0.303, acc=
92.54%]
Attention weight: 0.64

Epoch 3 Summary:
  Train Acc: 92.54%
  Val Acc: 92.15%
Epoch 4/5: 100%|██████████| 782/782 [00:31<00:00, 24.96it/s, loss=0.227, acc=
94.67%]
Attention weight: 0.64

Epoch 4 Summary:
  Train Acc: 94.67%
  Val Acc: 92.44%
Epoch 5/5: 100%|██████████| 782/782 [00:31<00:00, 25.11it/s, loss=0.930, acc=
95.57%]
Attention weight: 0.64

Epoch 5 Summary:
  Train Acc: 95.57%
  Val Acc: 92.50%
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```



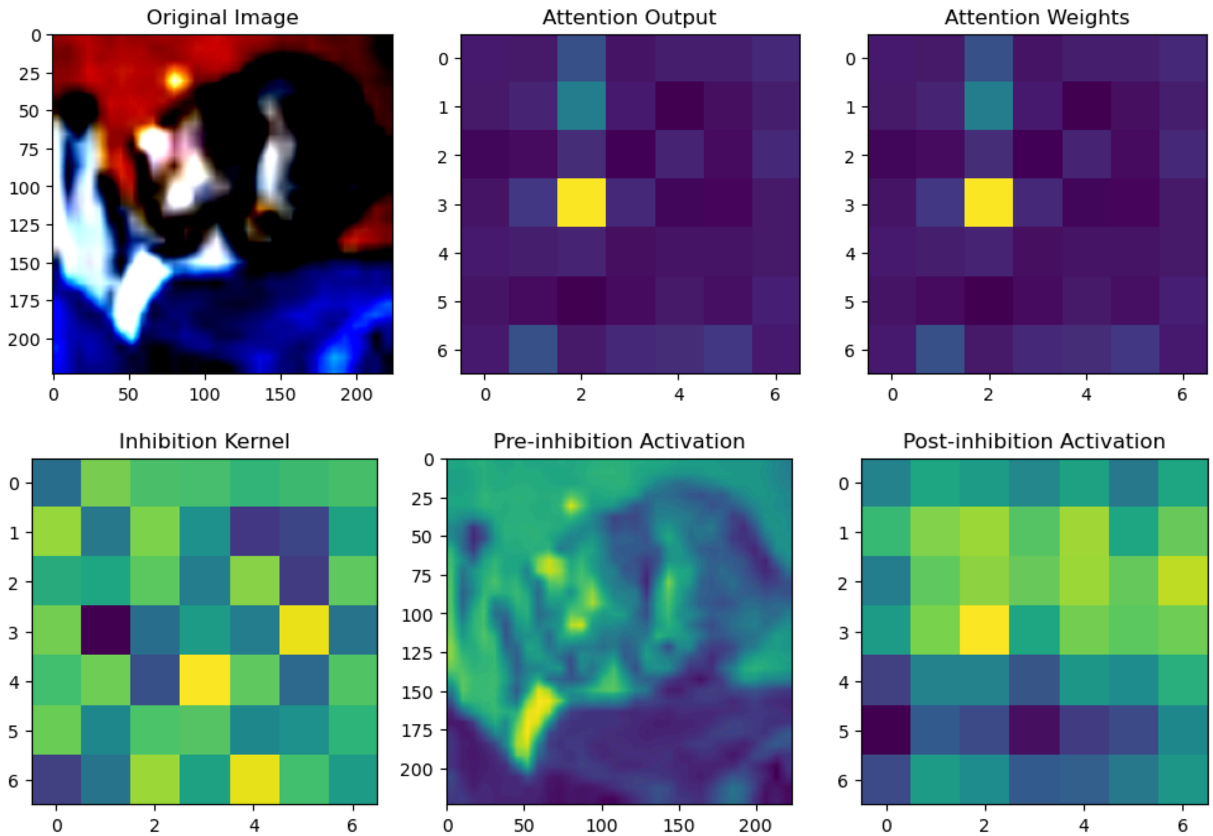


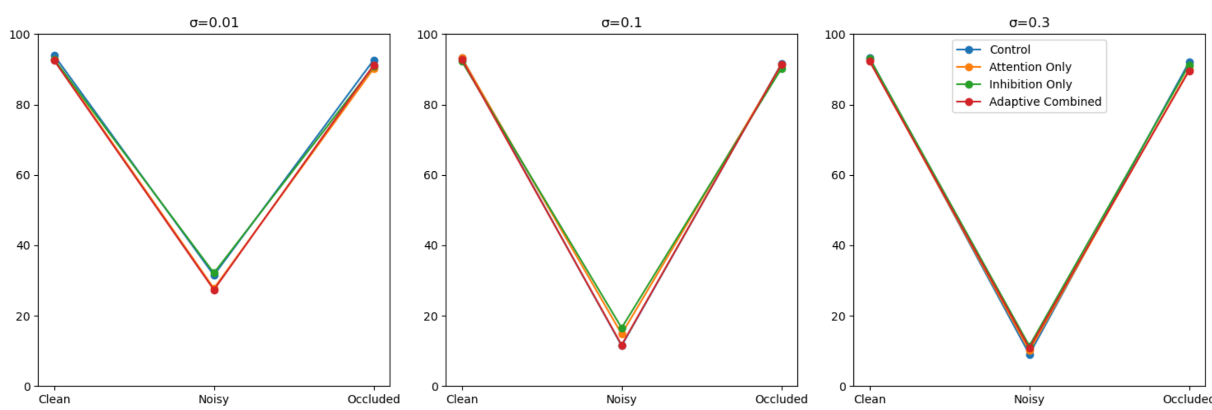

```
Training Inhibition Only
Epoch 1/5: 100%|██████████| 782/782 [00:31<00:00, 24.85it/s, loss=0.924, acc=68.13%]
Attention weight: 0.63

Epoch 1 Summary:
  Train Acc: 68.13%
  Val Acc: 86.66%
Epoch 2/5: 100%|██████████| 782/782 [00:31<00:00, 24.73it/s, loss=0.148, acc=88.64%]
Attention weight: 0.63

Epoch 2 Summary:
  Train Acc: 88.64%
  Val Acc: 89.69%
Epoch 3/5: 100%|██████████| 782/782 [00:31<00:00, 24.99it/s, loss=0.320, acc=92.60%]
Attention weight: 0.64

Epoch 3 Summary:
  Train Acc: 92.60%
  Val Acc: 92.47%
Epoch 4/5: 100%|██████████| 782/782 [00:31<00:00, 24.99it/s, loss=0.069, acc=94.83%]
Attention weight: 0.64

Epoch 4 Summary:
  Train Acc: 94.83%
  Val Acc: 92.85%
Epoch 5/5: 100%|██████████| 782/782 [00:31<00:00, 24.98it/s, loss=0.381, acc=95.69%]
Attention weight: 0.64

Epoch 5 Summary:
  Train Acc: 95.69%
  Val Acc: 93.13%
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```




```
Training Adaptive Combined
Epoch 1/5: 100%|██████████| 782/782 [00:31<00:00, 24.85it/s, loss=0.613, acc=67.98%]
Attention weight: 0.63

Epoch 1 Summary:
  Train Acc: 67.98%
  Val Acc: 86.44%
Epoch 2/5: 100%|██████████| 782/782 [00:31<00:00, 25.12it/s, loss=0.336, acc=88.46%]
Attention weight: 0.63

Epoch 2 Summary:
  Train Acc: 88.46%
  Val Acc: 90.56%
Epoch 3/5: 100%|██████████| 782/782 [00:31<00:00, 25.06it/s, loss=0.194, acc=92.77%]
Attention weight: 0.64

Epoch 3 Summary:
  Train Acc: 92.77%
  Val Acc: 92.51%
Epoch 4/5: 100%|██████████| 782/782 [00:31<00:00, 24.85it/s, loss=0.018, acc=94.75%]
Attention weight: 0.64

Epoch 4 Summary:
  Train Acc: 94.75%
  Val Acc: 92.18%
```

```
Epoch 5/5: 100%|██████████| 782/782 [00:31<00:00, 24.80it/s, loss=0.264, acc=
95.79%]
Attention weight: 0.64

Epoch 5 Summary:
  Train Acc: 95.79%
  Val Acc: 92.26%
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```

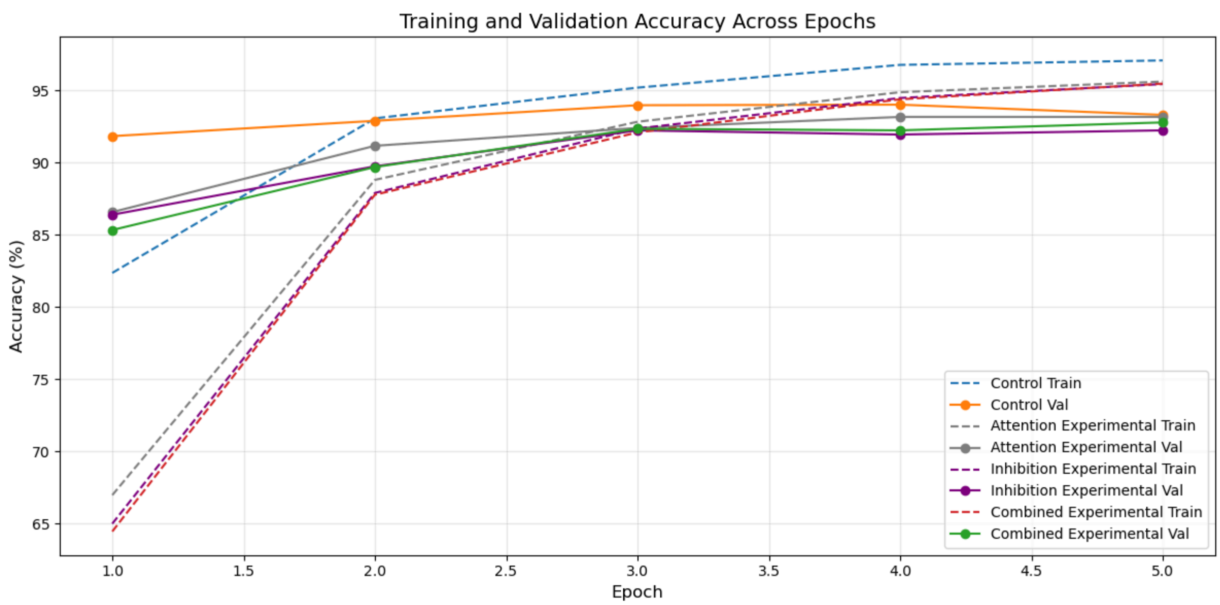


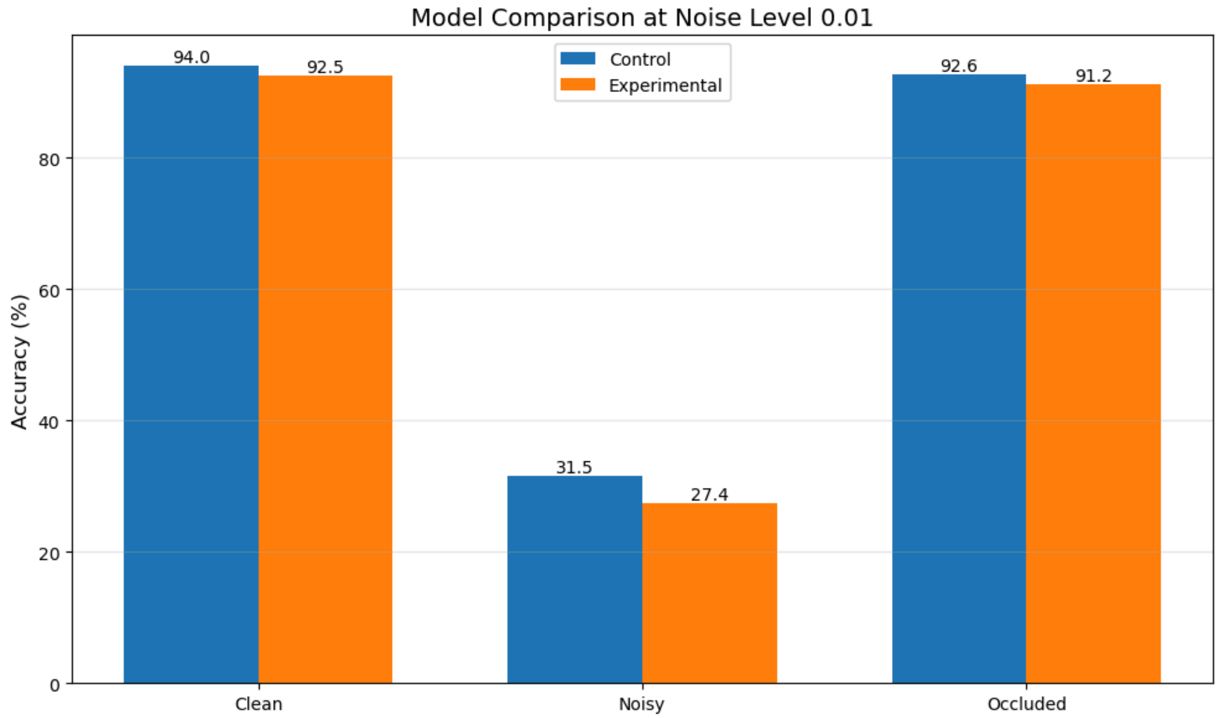


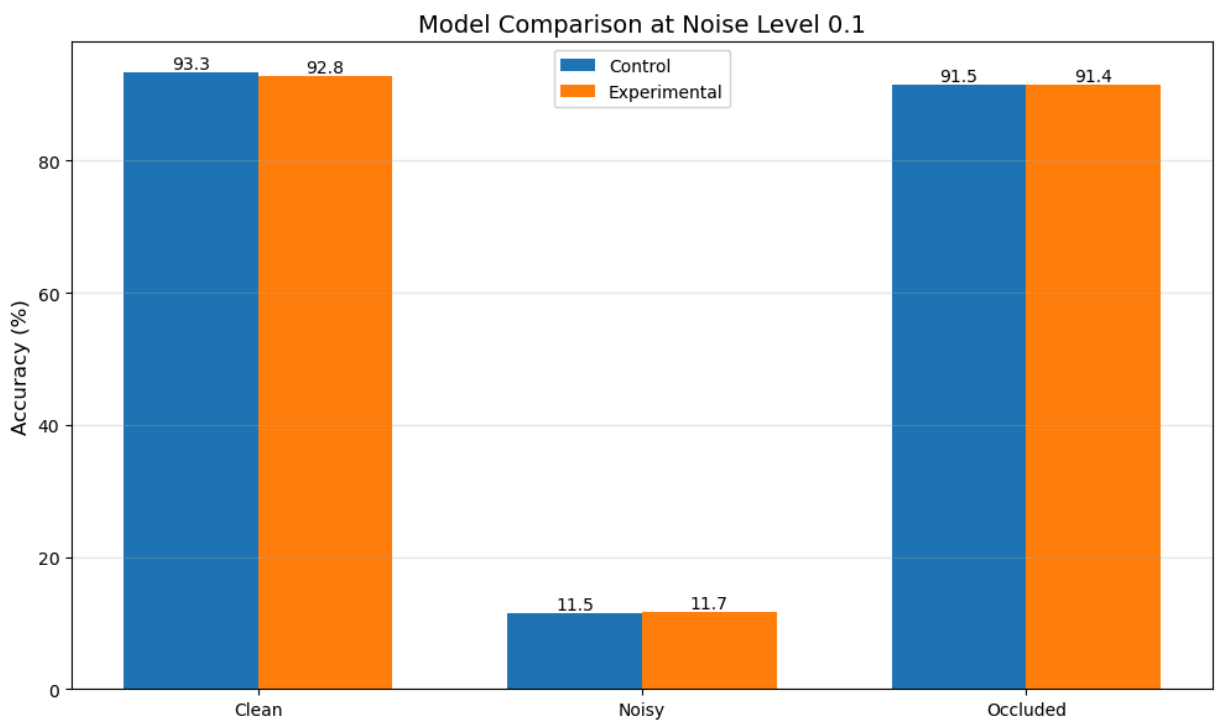
In [6]:
```python
### ------------------------------- ###
###     ADDITIONAL VISUALIZATIONS   ###
### ------------------------------- ###

# 1. Plot training progress across epochs
def plot_training_curves(noise_level=0.01):
    """Plot training and validation accuracy across epochs for each model"""
    plt.figure(figsize=(12, 6))
```

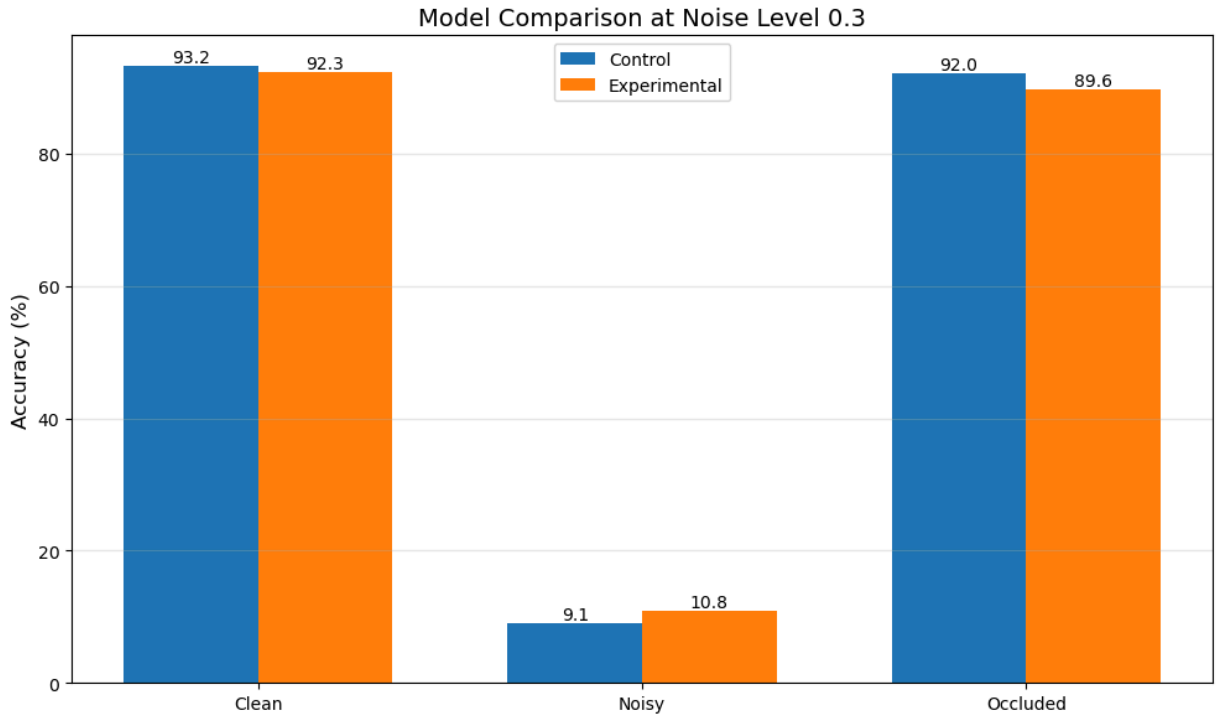
```python
    # Get results for the specified noise level
    noise_results = all_results[noise_level]

    # Plot each model's training curve
    for model_name, results in noise_results.items():
        train_acc = results['train_acc']
        val_acc = results['val_acc']
        epochs = range(1, len(train_acc) + 1)

        if model_name == 'Control':
            plt.plot(epochs, train_acc, '--', color='#1f77b4', label=f'Contro
            plt.plot(epochs, val_acc, 'o-', color='#ff7f0e', label=f'Control 
        elif model_name == 'Adaptive Combined':
            plt.plot(epochs, train_acc, '--', color='#d62728', label=f'Combin
            plt.plot(epochs, val_acc, 'o-', color='#2ca02c', label=f'Combined
        elif model_name == 'Attention Only':
            plt.plot(epochs, train_acc, '--', color='gray', label=f'Attention
            plt.plot(epochs, val_acc, 'o-', color='gray', label=f'Attention E
        elif model_name == 'Inhibition Only':
            plt.plot(epochs, train_acc, '--', color='purple', label=f'Inhibit
            plt.plot(epochs, val_acc, 'o-', color='purple', label=f'Inhibitic

    plt.title('Training and Validation Accuracy Across Epochs', fontsize=14)
    plt.xlabel('Epoch', fontsize=12)
    plt.ylabel('Accuracy (%)', fontsize=12)
    plt.grid(True, alpha=0.3)
    plt.legend()
    plt.tight_layout()
    plt.show()

# 2. Comparative bar chart of model performance
def plot_comparative_bars(noise_level=0.01, models=None):
    """Create a bar chart comparing performance on different test datasets"""
    if models is None:
        models = ['Control', 'Adaptive Combined']

    datasets = ['test_clean', 'test_noisy', 'test_occluded']
    x = np.arange(len(datasets))
    width = 0.35

    fig, ax = plt.subplots(figsize=(10, 6))

    # Extract data for the specified models and noise level
    noise_results = all_results[noise_level]

    # Plot bars for each model
    for i, model_name in enumerate(models):
        if model_name in noise_results:
            results = noise_results[model_name]
            model_label = 'Control' if model_name == 'Control' else 'Experime
            offset = width/2 * (-1 if i == 0 else 1)

            accs = [results[dataset] for dataset in datasets]
            ax.bar(x + offset, accs, width,
                   label=model_label,
                   color='#1f77b4' if i == 0 else '#ff7f0e')
```

```
    ax.set_ylabel('Accuracy (%)', fontsize=12)
    ax.set_title(f'Model Comparison at Noise Level {noise_level}', fontsize=1
    ax.set_xticks(x)
    ax.set_xticklabels([d.replace('test_', '').capitalize() for d in datasets
    ax.legend()
    ax.grid(axis='y', alpha=0.3)

    # Add value labels on bars
    for container in ax.containers:
        ax.bar_label(container, fmt='%.1f')

    plt.tight_layout()
    plt.show()

# Call the visualization functions (you can customize noise level)
plot_training_curves(noise_level=0.1)
plot_comparative_bars(noise_level=0.01)
plot_comparative_bars(noise_level=0.1)
plot_comparative_bars(noise_level=0.3)
```




Model Comparison at Noise Level 0.01
Control
Experimental
Accuracy (%)
94.0
92.5
31.5
27.4
92.6
91.2
0
20
40
60
80
Clean
Noisy
Occluded


Model Comparison at Noise Level 0.1
Control
Experimental
Accuracy (%)
93.3
92.8
11.5
11.7
91.5
91.4
0
20
40
60
80
Clean
Noisy
Occluded



Model Comparison at Noise Level 0.3
Control
Experimental
93.2
92.3
9.1
10.8
92.0
89.6
Accuracy (%)
0
20
40
60
80
Clean
Noisy
Occluded

In [ ]: