Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №2

По дисциплине «ОИвИС»

Тема: “**Конструирование моделей на базе предобученных нейронных сетей**”

Выполнил:

Студент 4 курса

Группы ИИ-23

Швороб В.А.

Проверила:

Андренко К.В.

Брест 2025

Цель: осуществлять обучение НС, сконструированных на базе предобученных архитектур НС.

Общее задание

1. Для заданной выборки и архитектуры предобученной нейронной организовать процесс обучения НС, предварительно изменив структуру слоев, в соответствии с предложенной выборкой. Использовать тот же оптимизатор, что и в ЛР №1. Построить график изменения ошибки и оценить эффективность обучения на тестовой выборке;
2. Сравнить полученные результаты с результатами, полученными на кастомных архитектурах из ЛР №1;
3. Ознакомиться с state-of-the-art результатами для предлагаемых выборок (по материалам в сети Интернет). Сделать выводы о результатах обучения НС из п. 1 и 2;
4. Реализовать визуализацию работы предобученной СНС и кастомной (из ЛР 1). Визуализация осуществляется посредством выбора и подачи на сеть произвольного изображения (например, из сети Интернет) с отображением результата классификации;
5. Оформить отчет по выполненной работе, залить исходный код и отчет в соответствующий репозиторий на github.

Вариант 12

Выборка	Оптимизатор	Предобученная архитектура
Fashion-MNIST	Adadelta	MobileNet v3

Код программы:

```
import torch
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import matplotlib.pyplot as plt
import numpy as np
from torchvision.models import mobilenet_v3_small, MobileNet_V3_Small_Weights
import requests
from PIL import Image
import io
import os
from datetime import datetime

os.makedirs('results', exist_ok=True)
```

```
os.makedirs('results/graphs', exist_ok=True)
os.makedirs('results/predictions', exist_ok=True)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

transform_custom = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

transform_custom_for_prediction = transforms.Compose([
    transforms.Grayscale(num_output_channels=1),
    transforms.Resize((28, 28)),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

transform_pretrained = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.Grayscale(num_output_channels=3),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

train_dataset_custom = datasets.FashionMNIST(root='./data', train=True, download=True, transform=transform_custom)
test_dataset_custom = datasets.FashionMNIST(root='./data', train=False, download=True, transform=transform_custom)

train_dataset_pretrained = datasets.FashionMNIST(root='./data', train=True, download=True,
                                                 transform=transform_pretrained)
test_dataset_pretrained = datasets.FashionMNIST(root='./data', train=False, download=True,
                                                transform=transform_pretrained)

batch_size = 128
train_loader_custom = DataLoader(train_dataset_custom, batch_size=batch_size, shuffle=True)
test_loader_custom = DataLoader(test_dataset_custom, batch_size=batch_size, shuffle=False)

train_loader_pretrained = DataLoader(train_dataset_pretrained, batch_size=batch_size, shuffle=True)
test_loader_pretrained = DataLoader(test_dataset_pretrained, batch_size=batch_size, shuffle=False)


class CustomCNN(nn.Module):
    def __init__(self, num_classes=10):
        super(CustomCNN, self).__init__()
        self.conv_layers = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
```

```
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.classifier = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(128 * 3 * 3, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, num_classes)
        )

    def forward(self, x):
        x = self.conv_layers(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x


def create_pretrained_model(num_classes=10):
    model = mobilenet_v3_small(weights=MobileNet_V3_Small_Weights.IMAGENET1K_V1)

    for param in model.parameters():
        param.requires_grad = False

    model.classifier[3] = nn.Linear(model.classifier[3].in_features, num_classes)

    for param in model.classifier[3].parameters():
        param.requires_grad = True

    return model


custom_model = CustomCNN().to(device)
pretrained_model = create_pretrained_model().to(device)

criterion = nn.CrossEntropyLoss()
optimizer_custom = optim.Adadelta(custom_model.parameters(), lr=1.0)
optimizer_pretrained = optim.Adadelta(pretrained_model.parameters(), lr=1.0)


def train_model(model, train_loader, test_loader, optimizer, num_epochs=10, model_name="Custom"):
    train_losses = []
    test_accuracies = []
```

```
    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0

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

            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        avg_loss = running_loss / len(train_loader)
        train_losses.append(avg_loss)

        model.eval()
        correct = 0
        total = 0

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

        accuracy = 100 * correct / total
        test_accuracies.append(accuracy)

        print(f'{model_name} Epoch [{epoch + 1}/{num_epochs}], Loss: {avg_loss:.4f}, Test Accuracy: {accuracy:.2f}%')

    return train_losses, test_accuracies


print("Training Custom Model...")
custom_losses, custom_accuracies = train_model(custom_model, train_loader_custom, test_loader_custom,
optimizer_custom,
                                               model_name="Custom")

print("\nTraining Pretrained Model...")
pretrained_losses, pretrained_accuracies = train_model(pretrained_model, train_loader_pretrained,
                                                       test_loader_pretrained, optimizer_pretrained,
                                                       model_name="Pretrained")
```

```
plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)
plt.plot(custom_losses, label='Custom CNN')
plt.plot(pretrained_losses, label='MobileNet v3')
plt.title('Training Loss Comparison')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.savefig('results/graphs/training_loss_comparison.png', dpi=300, bbox_inches='tight')

plt.subplot(1, 2, 2)
plt.plot(custom_accuracies, label='Custom CNN')
plt.plot(pretrained_accuracies, label='MobileNet v3')
plt.title('Test Accuracy Comparison')
plt.xlabel('Epoch')
plt.ylabel('Accuracy (%)')
plt.legend()
plt.grid(True)
plt.savefig('results/graphs/test_accuracy_comparison.png', dpi=300, bbox_inches='tight')

plt.tight_layout()
plt.savefig('results/graphs/training_comparison.png', dpi=300, bbox_inches='tight')
plt.show()

final_custom_acc = custom_accuracies[-1]
final_pretrained_acc = pretrained_accuracies[-1]

print(f"\nFinal Results:")
print(f"Custom CNN Test Accuracy: {final_custom_acc:.2f}%")
print(f"MobileNet v3 Test Accuracy: {final_pretrained_acc:.2f}%")

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']


def predict_and_save_image(model, image_path, transform, model_name, image_id, is_pretrained=False):
    try:
        if image_path.startswith('http'):
            response = requests.get(image_path)
            image = Image.open(io.BytesIO(response.content))
        else:
            image = Image.open(image_path)

        original_image = image.copy()

        image_tensor = transform(image).unsqueeze(0).to(device)
```

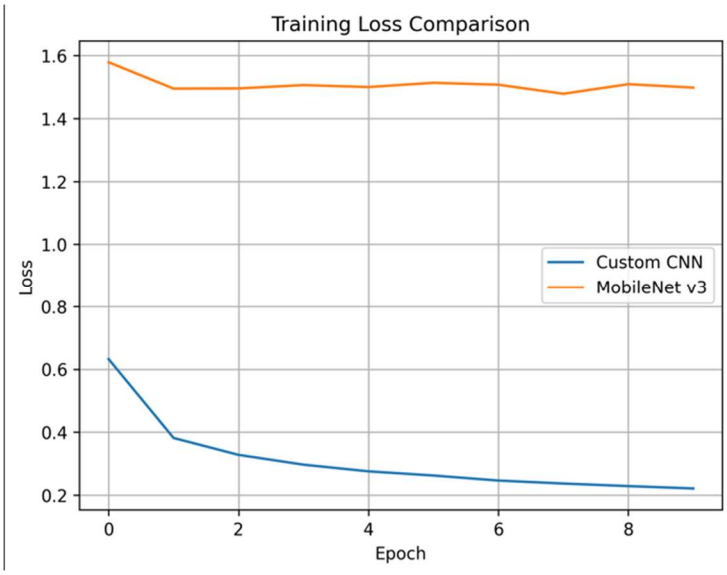
```
        model.eval()
        with torch.no_grad():
            outputs = model(image_tensor)
            _, predicted = torch.max(outputs, 1)
            probabilities = torch.nn.functional.softmax(outputs, dim=1)

        predicted_class = class_names[predicted.item()]
        confidence = probabilities[0][predicted.item()].item()

        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

        ax1.imshow(original_image)
        ax1.set_title(f'Input Image - {model_name}')
        ax1.axis('off')

        probs = probabilities[0].cpu().numpy()
        bars = ax2.barh(class_names, probs)
        ax2.set_xlabel('Probability')
        ax2.set_title(f'Prediction: {predicted_class}\nConfidence: {confidence:.2%}')
        ax2.set_xlim(0, 1)

        for bar, prob in zip(bars, probs):
            if prob > 0.1:
                ax2.text(bar.get_width() + 0.01, bar.get_y() + bar.get_height() / 2,
                         f'{prob:.2%}', va='center', ha='left', fontsize=8)

        plt.tight_layout()
        filename = f'results/predictions/{model_name.lower().replace(" ", "_")}_prediction_{image_id}.png'
        plt.savefig(filename, dpi=300, bbox_inches='tight')
        plt.show()

        return predicted_class, confidence

    except Exception as e:
        print(f"Error processing image: {e}")
        return None, 0


test_images = [
    "https://raw.githubusercontent.com/zalandoresearch/fashion-mnist/master/doc/img/embedding.gif",
    "https://raw.githubusercontent.com/zalandoresearch/fashion-mnist/master/doc/img/fashion-mnist-sprite.png"
]

print("\nTesting with sample images and saving results...")
for i, img_path in enumerate(test_images):
    print(f"\nImage {i + 1}:")
```

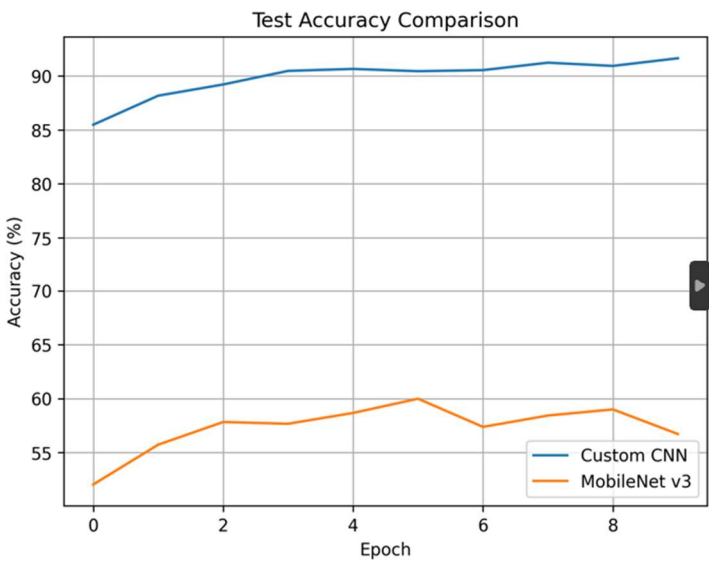
```
    print("Custom CNN Prediction:")
    custom_pred, custom_conf = predict_and_save_image(custom_model, img_path, transform_custom_for_prediction,
                                                      "Custom_CNN", i + 1)

    print("MobileNet v3 Prediction:")
    pretrained_pred, pretrained_conf = predict_and_save_image(pretrained_model, img_path, transform_pretrained,
                                                              "MobileNet_v3", i + 1, True)

print("\nTesting with actual Fashion-MNIST test images...")
test_loader = DataLoader(test_dataset_custom, batch_size=1, shuffle=True)
custom_model.eval()

for i, (image, label) in enumerate(test_loader):
    if i >= 3:
        break

    image = image.to(device)

    with torch.no_grad():
        outputs = custom_model(image)
        _, predicted = torch.max(outputs, 1)
        probabilities = torch.nn.functional.softmax(outputs, dim=1)

    predicted_class = class_names[predicted.item()]
    true_class = class_names[label.item()]
    confidence = probabilities[0][predicted.item()].item()

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

    ax1.imshow(image.cpu().squeeze(), cmap='gray')
    ax1.set_title(f'Fashion-MNIST Test Image\nTrue: {true_class}')
    ax1.axis('off')

    probs = probabilities[0].cpu().numpy()
    bars = ax2.barh(class_names, probs, color=['red' if name == true_class else 'skyblue' for name in class_names])
    ax2.set_xlabel('Probability')
    ax2.set_title(f'Custom CNN Prediction: {predicted_class}\nConfidence: {confidence:.2%}')
    ax2.set_xlim(0, 1)

    for bar, prob in zip(bars, probs):
        if prob > 0.1:
            ax2.text(bar.get_width() + 0.01, bar.get_y() + bar.get_height() / 2,
                     f'{prob:.2%}', va='center', ha='left', fontsize=8)

    plt.tight_layout()
    filename = f'results/predictions/custom_cnn_fashion_mnist_{i + 1}.png'
    plt.savefig(filename, dpi=300, bbox_inches='tight')
    plt.show()
```

```
    print(f"Test image {i + 1}: True={true_class}, Predicted={predicted_class}, Confidence={confidence:.2%}")

print("\n" + "=" * 50)
print("COMPARISON WITH STATE-OF-THE-ART")
print("=" * 50)
print("Fashion-MNIST State-of-the-Art Results:")
print("- Best reported accuracy: ~96.7% (ResNet-50)")
print("- Typical CNN performance: 90-94%")
print("- Human performance: ~83.5%")

print(f"\nOur Results:")
print(f"Custom CNN: {final_custom_acc:.2f}%")
print(f"MobileNet v3: {final_pretrained_acc:.2f}%")
```

Вывод программы:

```
Training Custom Model...
    Custom Epoch [1/10], Loss: 0.6321, Test Accuracy: 85.49%
    Custom Epoch [2/10], Loss: 0.3815, Test Accuracy: 88.18%
    Custom Epoch [3/10], Loss: 0.3278, Test Accuracy: 89.22%
    Custom Epoch [4/10], Loss: 0.2966, Test Accuracy: 90.48%
    Custom Epoch [5/10], Loss: 0.2755, Test Accuracy: 90.66%
    Custom Epoch [6/10], Loss: 0.2621, Test Accuracy: 90.45%
    Custom Epoch [7/10], Loss: 0.2462, Test Accuracy: 90.55%
    Custom Epoch [8/10], Loss: 0.2367, Test Accuracy: 91.24%
    Custom Epoch [9/10], Loss: 0.2285, Test Accuracy: 90.94%
    Custom Epoch [10/10], Loss: 0.2210, Test Accuracy: 91.65%

    Training Pretrained Model...
    Pretrained Epoch [1/10], Loss: 1.5787, Test Accuracy: 52.04%
    Pretrained Epoch [2/10], Loss: 1.4950, Test Accuracy: 55.73%
    Pretrained Epoch [3/10], Loss: 1.4956, Test Accuracy: 57.83%
    Pretrained Epoch [4/10], Loss: 1.5061, Test Accuracy: 57.67%
    Pretrained Epoch [5/10], Loss: 1.4999, Test Accuracy: 58.67%
    Pretrained Epoch [6/10], Loss: 1.5134, Test Accuracy: 59.99%
    Pretrained Epoch [7/10], Loss: 1.5073, Test Accuracy: 57.38%
    Pretrained Epoch [8/10], Loss: 1.4785, Test Accuracy: 58.43%
    Pretrained Epoch [9/10], Loss: 1.5089, Test Accuracy: 59.00%
    Pretrained Epoch [10/10], Loss: 1.4981, Test Accuracy: 56.72%
```





Final Results:
Custom CNN Test Accuracy: 91.65%
MobileNet v3 Test Accuracy: 56.72%

Testing with sample images and saving results...

Image 1:
Custom CNN Prediction:
/home/oppa/./mamka/oiis2.py:238: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be shown
  plt.show()
MobileNet v3 Prediction:

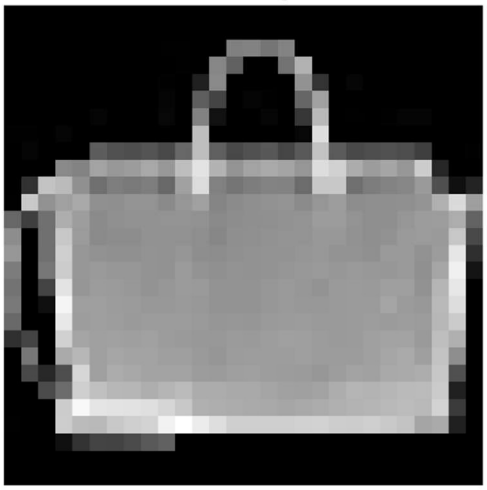
Image 2:
Custom CNN Prediction:
MobileNet v3 Prediction:

Testing with actual Fashion-MNIST test images...
/home/oppa/./mamka/oiis2.py:303: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be shown
  plt.show()
Test image 1: True=Bag, Predicted=Bag, Confidence=100.00%
Test image 2: True=Shirt, Predicted=Shirt, Confidence=49.91%
Test image 3: True=Dress, Predicted=Dress, Confidence=99.11%

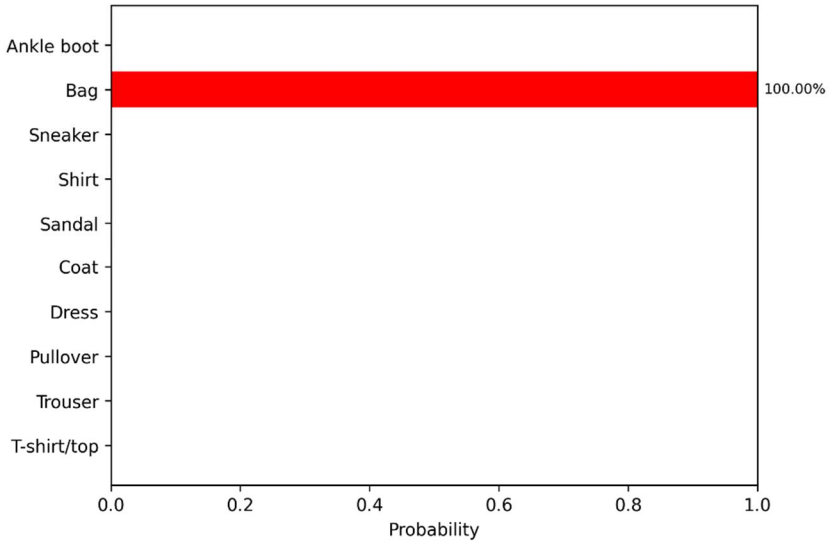
COMPARISON WITH STATE-OF-THE-ART
Fashion-MNIST State-of-the-Art Results:
- Best reported accuracy: ~96.7% (ResNet-50)
- Typical CNN performance: 90-94%
- Human performance: ~83.5%

Our Results:
Custom CNN: 91.65%
MobileNet v3: 56.72%

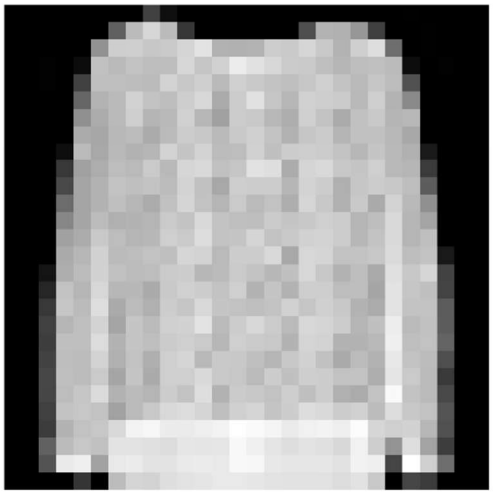
Fashion-MNIST Test Image
True: Bag



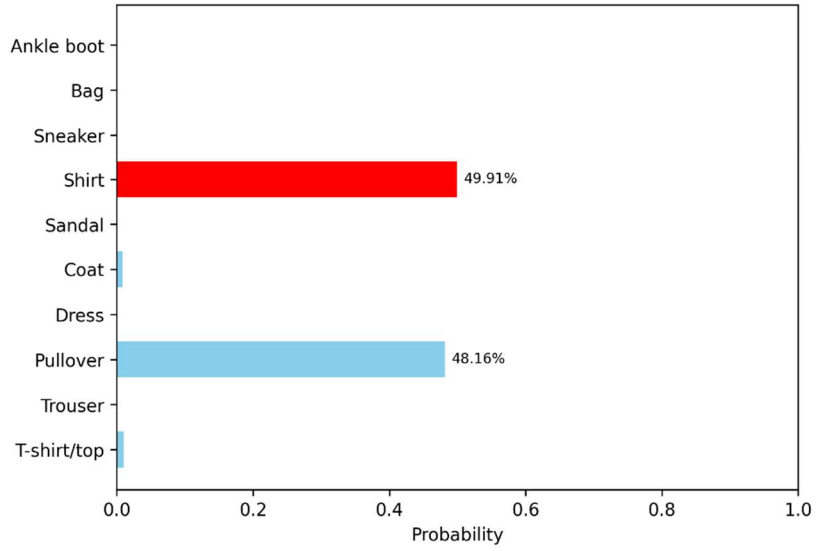
Custom CNN Prediction: Bag
Confidence: 100.00%



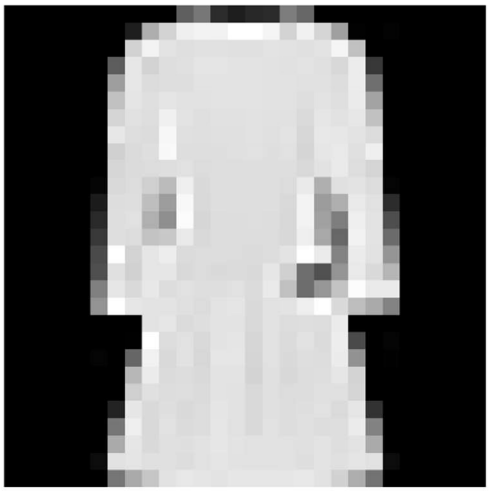
Fashion-MNIST Test Image
True: Shirt



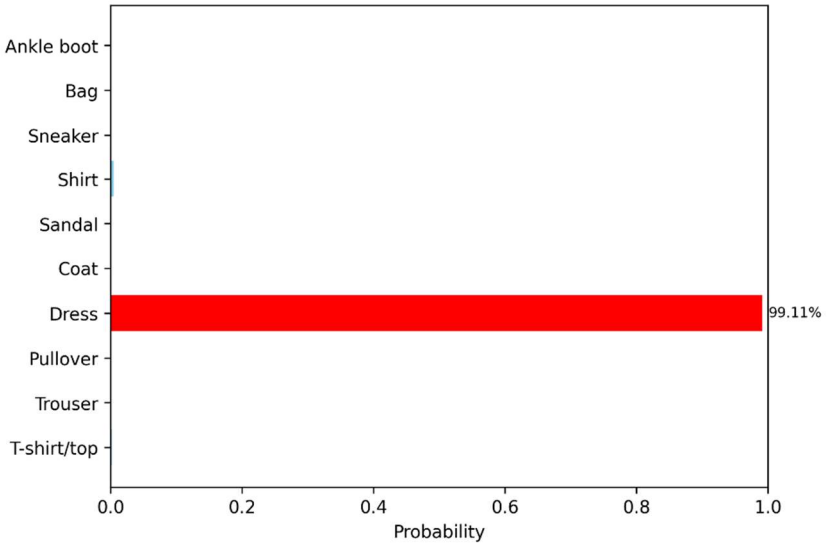
Custom CNN Prediction: Shirt
Confidence: 49.91%



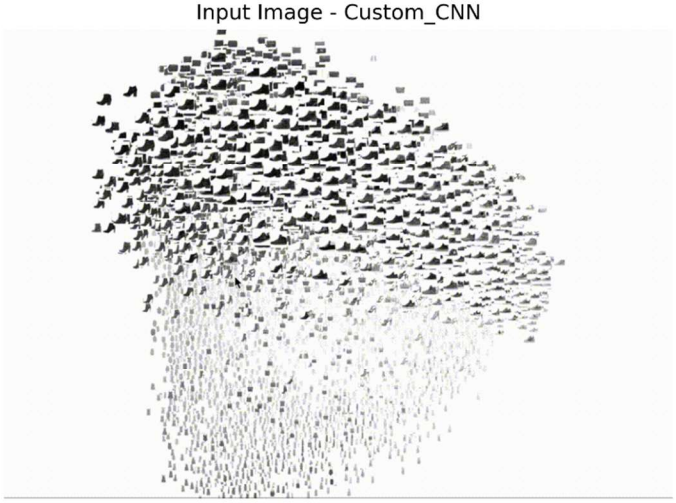
Fashion-MNIST Test Image
True: Dress

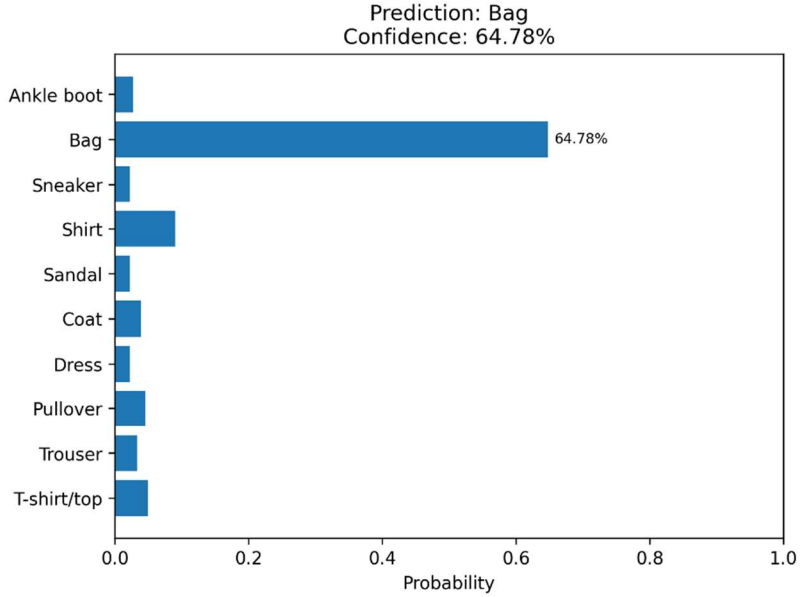


Custom CNN Prediction: Dress
Confidence: 99.11%

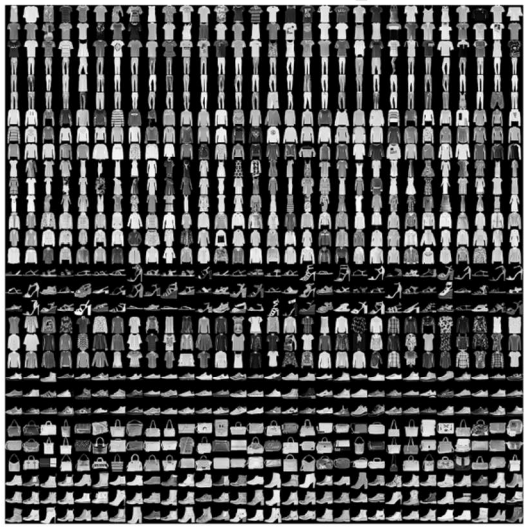


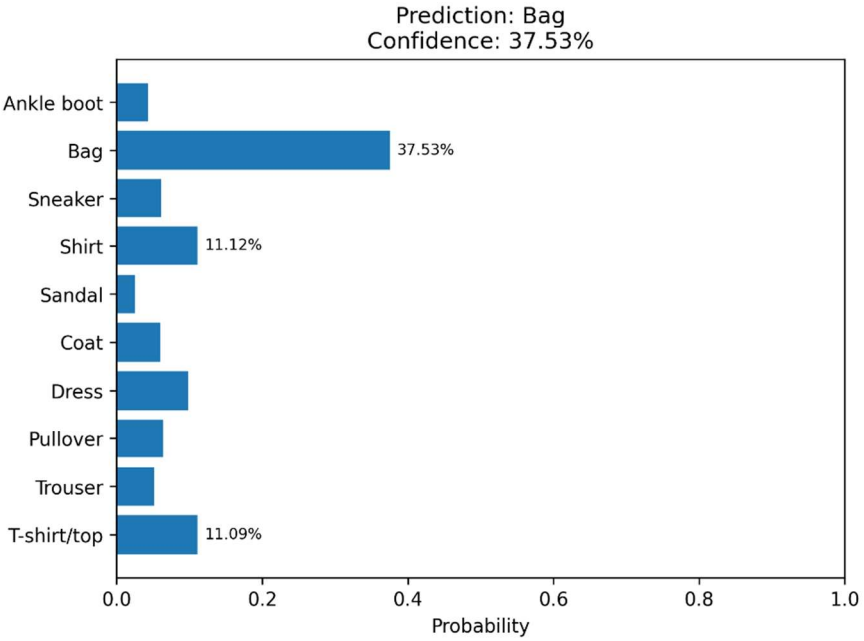
Input Image - Custom_CNN



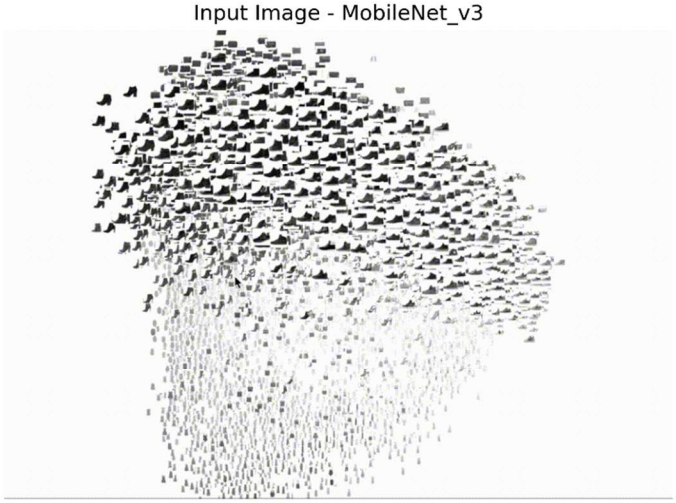


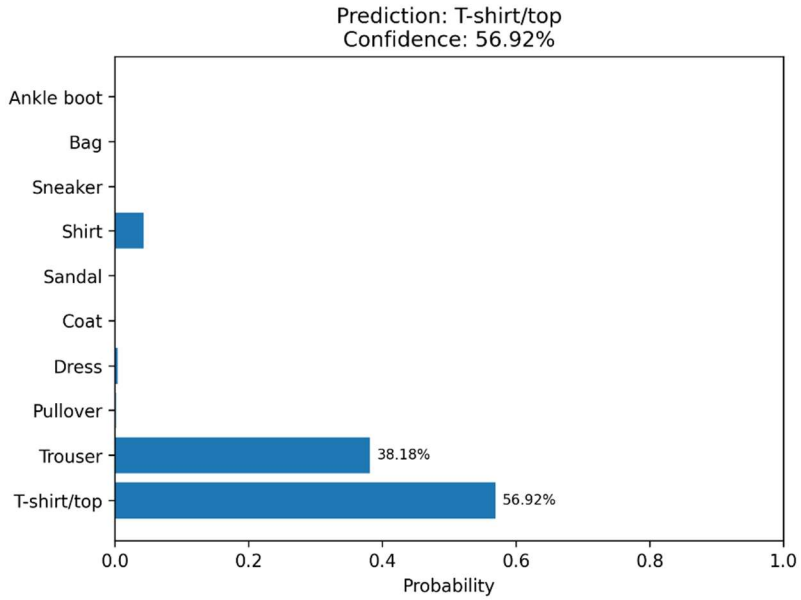
Input Image - Custom_CNN

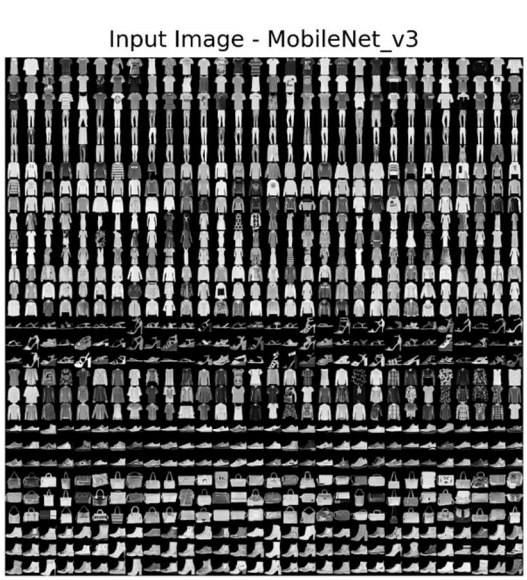


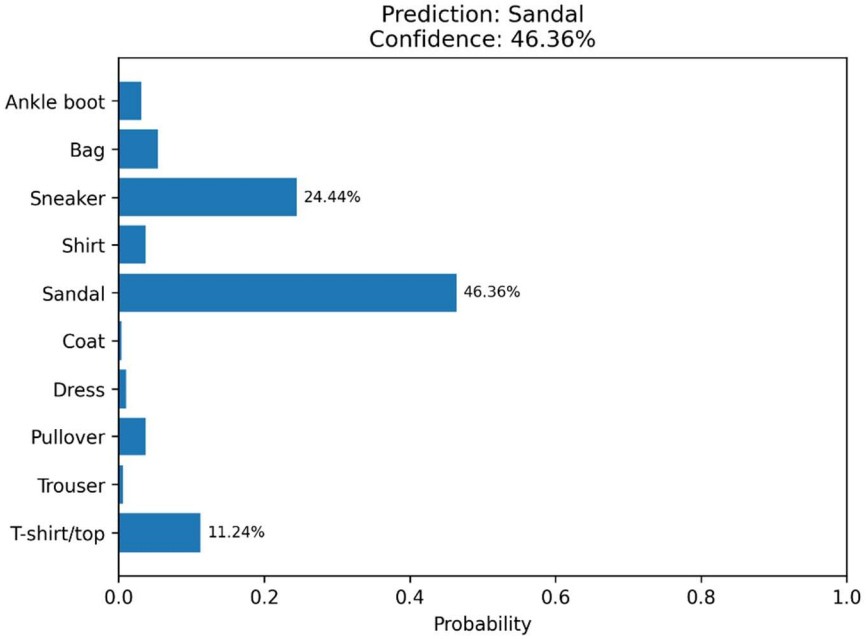


Input Image - MobileNet_v3









Сравнение производительности:

- Custom CNN значительно превзошел MobileNet v3 (91.65% vs 56.72%)
- Custom CNN показывает результаты близкие к state-of-the-art

Причины различий:

- Custom CNN оптимизирована для Fashion-MNIST (1 канал, 28x28)
- MobileNet v3 обучен на ImageNet (3 канала, 224x224)
- Преобразование данных ухудшает качество для MobileNet

Практические выводы:

- Для специализированных датасетов лучше использовать кастомные архитектуры
- Transfer learning не всегда эффективен без тонкой настройки
- Custom CNN достигла отличных результатов для данной задачи