

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  
Копач А. В.  
Проверила:  
Андренко К.В.

Брест 2025

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

### Вариант 7.

Выборка	Оптимизатор	Предобученная архитектура
Fashion-MNIST	Adam	ResNet34

Код программы:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
from torchvision.models import resnet34, ResNet34_Weights
import requests
from PIL import Image
import io
import time
from tqdm import tqdm

# 1. НАСТРОЙКА УСТРОЙСТВА И ПАРАМЕТРОВ
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Параметры из ЛР №1 (Adam оптимизатор)
batch_size = 64
learning_rate = 0.001
num_epochs = 10

# 2. ПРЕОБРАЗОВАНИЯ ДАННЫХ ДЛЯ Fashion-MNIST
# Для предобученной ResNet34 (3 канала)
transform_resnet = transforms.Compose([
    transforms.Resize((224, 224)), # Изменение размера для ResNet
    transforms.Grayscale(num_output_channels=3), # Преобразование в 3 канала
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]) # ImageNet нормализация
])

# Для кастомной CNN (1 канал)
transform_custom = transforms.Compose([
    transforms.Resize((32, 32)), # Меньший размер для кастомной CNN
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5], std=[0.5]) # Нормализация для 1 канала
])
```

```

# 3. ЗАГРУЗКА ДАННЫХ
print("Загрузка Fashion-MNIST...")
train_dataset_resnet = torchvision.datasets.FashionMNIST(
    root='./data', train=True, download=True, transform=transform_resnet)
test_dataset_resnet = torchvision.datasets.FashionMNIST(
    root='./data', train=False, download=True, transform=transform_resnet)

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

train_loader_resnet = DataLoader(train_dataset_resnet, batch_size=batch_size, shuffle=True)
test_loader_resnet = DataLoader(test_dataset_resnet, batch_size=batch_size, shuffle=False)

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)

# Классы Fashion-MNIST
classes = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
           'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

```

## # 4. АРХИТЕКТУРЫ МОДЕЛЕЙ

### # 4.1 ПРЕДОБУЧЕННАЯ RESNET34 (адаптированная для Fashion-MNIST)

```

def create_resnet34():
    print("Загрузка предобученной ResNet34...")
    model = resnet34(weights=ResNet34_Weights.IMAGENET1K_V1)

    # Замена последнего слоя для 10 классов Fashion-MNIST
    num_features = model.fc.in_features
    model.fc = nn.Linear(num_features, 10)

    return model

```

### # 4.2 КАСТОМНАЯ CNN (архитектура из ЛР №1)

```

class CustomCNN(nn.Module):
    def __init__(self):
        super(CustomCNN, self).__init__()
        self.conv_layers = nn.Sequential(
            # Первый сверточный блок
            nn.Conv2d(1, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),

```

```

# Второй сверточный блок
nn.Conv2d(32, 64, kernel_size=3, padding=1),
nn.ReLU(),
nn.MaxPool2d(2),

# Третий сверточный блок
nn.Conv2d(64, 128, kernel_size=3, padding=1),
nn.ReLU(),
nn.AdaptiveAvgPool2d((1, 1))
)

self.classifier = nn.Sequential(
    nn.Flatten(),
    nn.Linear(128, 64),
    nn.ReLU(),
    nn.Dropout(0.5),
    nn.Linear(64, 10)
)

```

```

def forward(self, x):
    x = self.conv_layers(x)
    x = self.classifier(x)
    return x

```

## # 5. ФУНКЦИЯ ОБУЧЕНИЯ (ПУНКТ 1)

```

def train_model(model, train_loader, test_loader, model_name):
    model = model.to(device)
    criterion = nn.CrossEntropyLoss() # Рекомендуемый критерий
    optimizer = optim.Adam(model.parameters(), lr=learning_rate) # Adam из ЛР №1

    train_losses = []
    test accuracies = []

    print(f"\n🌀 Обучение {model_name}")
    print(f"📊 Batch size: {batch_size}, LR: {learning_rate}")
    print("-" * 50)

    for epoch in range(num_epochs):
        # Обучение
        model.train()
        running_loss = 0.0
        start_time = time.time()

        for images, labels in tqdm(train_loader, desc=f'Epoch {epoch + 1}/{num_epochs}'):
            images, labels = images.to(device), labels.to(device)

            # Прямой проход

```

```
outputs = model(images)
loss = criterion(outputs, labels)
```

```
# Обратный проход
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

```
running_loss += loss.item()
```

```
# Валидация
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()
```

```
epoch_loss = running_loss / len(train_loader)
accuracy = 100 * correct / total
```

```
train_losses.append(epoch_loss)
test_accuracies.append(accuracy)
```

```
epoch_time = time.time() - start_time
```

```
print(f'Epoch {epoch + 1}/{num_epochs} | Loss: {epoch_loss:.4f} | '
      f'Accuracy: {accuracy:.2f}% | Time: {epoch_time:.2f}s')
```

```
return train_losses, test_accuracies
```

## # 6. ВИЗУАЛИЗАЦИЯ РЕЗУЛЬТАТОВ (ПУНКТ 1 И 2)

```
def plot_results(resnet_losses, resnet_accs, custom_losses, custom_accs):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))
```

```
# График потерь
ax1.plot(resnet_losses, 'b-', label='ResNet34', linewidth=2)
ax1.plot(custom_losses, 'r-', label='Custom CNN', linewidth=2)
ax1.set_title('Training Loss')
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Loss')
ax1.legend()
```

```
ax1.grid(True)
```

```
# График точности
```

```
ax2.plot(resnet_accs, 'b-', label='ResNet34', linewidth=2)
```

```
ax2.plot(custom_accs, 'r-', label='Custom CNN', linewidth=2)
```

```
ax2.set_title('Test Accuracy')
```

```
ax2.set_xlabel('Epoch')
```

```
ax2.set_ylabel('Accuracy (%)')
```

```
ax2.legend()
```

```
ax2.grid(True)
```

```
plt.tight_layout()
```

```
plt.savefig('training_comparison.png', dpi=300, bbox_inches='tight')
```

```
plt.show()
```

```
# 7. ВИЗУАЛИЗАЦИЯ ПРЕДСКАЗАНИЙ (ПУНКТ 4)
```

```
def visualize_predictions(model, test_loader, model_name, device):
```

```
    model.eval()
```

```
    data_iter = iter(test_loader)
```

```
    images, labels = next(data_iter)
```

```
    with torch.no_grad():
```

```
        images = images.to(device)
```

```
        outputs = model(images)
```

```
        _, predictions = torch.max(outputs, 1)
```

```
# Визуализация 8 примеров
```

```
fig, axes = plt.subplots(2, 4, figsize=(16, 8))
```

```
axes = axes.ravel()
```

```
for i in range(8):
```

```
    if model_name == "ResNet34":
```

```
        # Денормализация для ResNet
```

```
        img = images[i].cpu()
```

```
        img = img * torch.tensor([0.229, 0.224, 0.225]).view(3, 1, 1) + torch.tensor([0.485, 0.456, 0.406]).view(3, 1, 1)
```

```
        img = torch.clamp(img, 0, 1)
```

```
        axes[i].imshow(img.permute(1, 2, 0))
```

```
    else:
```

```
        # Денормализация для кастомной CNN
```

```
        img = images[i].cpu().squeeze()
```

```
        img = img * 0.5 + 0.5
```

```
        axes[i].imshow(img, cmap='gray')
```

```
    color = 'green' if predictions[i] == labels[i] else 'red'
```

```
    axes[i].set_title(f'True: {classes[labels[i]]}\nPred: {classes[predictions[i]]}',
```

```

        color=color, fontsize=10)
axes[i].axis('off')

plt.suptitle(f'{model_name} - Predictions on Test Images', fontsize=16)
plt.tight_layout()
plt.show()

```

#### # 8. КЛАССИФИКАЦИЯ ПРОИЗВОЛЬНЫХ ИЗОБРАЖЕНИЙ (ПУНКТ 4)

```

def classify_custom_image(model, image_path, model_name, transform):
    """Классификация произвольного изображения"""
    try:
        # Загрузка изображения
        if image_path.startswith('http'):
            response = requests.get(image_path)
            img = Image.open(io.BytesIO(response.content))
        else:
            img = Image.open(image_path)

        # Преобразование в grayscale если нужно
        if img.mode != 'L':
            img = img.convert('L')

        # Применение преобразований
        input_tensor = transform(img).unsqueeze(0).to(device)

        # Предсказание
        model.eval()
        with torch.no_grad():
            output = model(input_tensor)
            probabilities = torch.nn.functional.softmax(output[0], dim=0)
            predicted_class = torch.argmax(probabilities).item()
            confidence = probabilities[predicted_class].item()

        # Визуализация
        plt.figure(figsize=(10, 4))
        plt.subplot(1, 2, 1)
        plt.imshow(img, cmap='gray')
        plt.title('Input Image')
        plt.axis('off')
        plt.subplot(1, 2, 2)
        class_probs = probabilities.cpu().numpy()
        y_pos = np.arange(len(classes))
        plt.barh(y_pos, class_probs)
        plt.yticks(y_pos, classes)
        plt.xlabel('Probability')
        plt.title(f'{model_name}\nPrediction: {classes[predicted_class]}\nConfidence: {confidence:.2%}')
        plt.tight_layout()

```

```
plt.show()
```

```
return classes[predicted_class], confidence
```

```
except Exception as e:
```

```
    print(f"Ошибка: {e}")
```

```
    return None, 0
```

## # 9. ОСНОВНОЙ ПРОЦЕСС ОБУЧЕНИЯ И АНАЛИЗА

```
def main():
```

```
    print("=" * 70)
```

```
    print("ЛАБОРАТОРНАЯ РАБОТА: СРАВНЕНИЕ ПРЕДОБУЧЕННЫХ И КАСТОМНЫХ СЕТЕЙ")
```

```
    print("=" * 70)
```

```
    # Создание моделей
```

```
    print("\n1. СОЗДАНИЕ МОДЕЛЕЙ")
```

```
    resnet_model = create_resnet34()
```

```
    custom_model = CustomCNN()
```

```
    print(f"ResNet34 параметров: {sum(p.numel() for p in resnet_model.parameters()),}")
```

```
    print(f"Custom CNN параметров: {sum(p.numel() for p in custom_model.parameters()),}")
```

```
    # Обучение моделей (Пункт 1)
```

```
    print("\n2. ОБУЧЕНИЕ МОДЕЛЕЙ")
```

```
    print("=" * 50)
```

```
    resnet_losses, resnet_accuracies = train_model(
```

```
        resnet_model, train_loader_resnet, test_loader_resnet, "ResNet34"
```

```
)
```

```
    custom_losses, custom_accuracies = train_model(
```

```
        custom_model, train_loader_custom, test_loader_custom, "Custom CNN"
```

```
)
```

```
    # Визуализация результатов (Пункт 1)
```

```
    print("\n3. ВИЗУАЛИЗАЦИЯ РЕЗУЛЬТАТОВ ОБУЧЕНИЯ")
```

```
    plot_results(resnet_losses, resnet_accuracies, custom_losses, custom_accuracies)
```

```
    # Сравнение результатов (Пункт 2)
```

```
    print("\n4. СРАВНЕНИЕ РЕЗУЛЬТАТОВ")
```

```
    print("=" * 50)
```

```
    final_resnet_acc = resnet_accuracies[-1]
```

```
    final_custom_acc = custom_accuracies[-1]
```

```
    print(f"Final ResNet34 Accuracy: {final_resnet_acc:.2f}%")
```

```
    print(f"Final Custom CNN Accuracy: {final_custom_acc:.2f}%")
```

```
    print(f"Difference: {final_resnet_acc - final_custom_acc:+.2f}%")
```



```
# Анализ и State-of-the-art сравнение (Пункт 3)
print("\n5. АНАЛИЗ И STATE-OF-THE-ART СРАВНЕНИЕ")
print("=" * 50)
```

```
# State-of-the-art результаты для Fashion-MNIST
```

```
sotar_results = {
    "Лучшие известные результаты": 96.7,
    "Типичные ResNet результаты": 93.5,
    "Типичные CNN результаты": 92.0,
    "Базовые модели": 89.0
}
print("State-of-the-art результаты для Fashion-MNIST:")
for model_type, accuracy in sotar_results.items():
    print(f" {model_type}: {accuracy}%")
print(f"\nНаши результаты:")
print(
    f" ResNet34: {final_resnet_acc:.2f}% (отставание: {sotar_results['Типичные ResNet результаты'] - "
    f"final_resnet_acc:.2f}%)"
)
print(
    f" Custom CNN: {final_custom_acc:.2f}% (отставание: {sotar_results['Типичные CNN результаты'] - "
    f"final_custom_acc:.2f}%)"
)
```

```
# Визуализация работы моделей (Пункт 4)
print("\n6. ВИЗУАЛИЗАЦИЯ РАБОТЫ МОДЕЛЕЙ")
print("=" * 50)
```

```
print("Визуализация предсказаний на тестовых данных:")
visualize_predictions(resnet_model, test_loader_resnet, "ResNet34", device)
visualize_predictions(custom_model, test_loader_custom, "Custom CNN", device)
```

```
# Классификация произвольных изображений
print("\n7. КЛАССИФИКАЦИЯ ПРОИЗВОЛЬНЫХ ИЗОБРАЖЕНИЙ")
print("=" * 50)
```

```
# Примеры URL изображений для тестирования (можно заменить на свои)
```

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

```
print("Пример классификации произвольных изображений:")
for img_url in test_images:
    print(f"\nКлассификация изображения: {img_url}")
```

```
# ResNet34 классификация
```

```
resnet_pred, resnet_conf = classify_custom_image(
    resnet_model, img_url, "ResNet34", transform_resnet
)
```

```

if resnet_pred:
    print(f'ResNet34: {resnet_pred} (confidence: {resnet_conf:.2%})')

# Custom CNN классификация
custom_pred, custom_conf = classify_custom_image(
    custom_model, img_url, "Custom CNN", transform_custom
)
if custom_pred:
    print(f'Custom CNN: {custom_pred} (confidence: {custom_conf:.2%})')

```

```

# Выводы (Пункт 3)
print("\n8. ВЫВОДЫ")
print("=" * 50)

```

```

print("📊 АНАЛИЗ РЕЗУЛЬТАТОВ:")
print(f'1. Эффективность обучения:')
print(f'   • ResNet34 достиг точности {final_resnet_acc:.2f}%')
print(f'   • Custom CNN достиг точности {final_custom_acc:.2f}%')

```

```

print(f'2. Сравнение архитектур:')
print(f'   • ResNet34 показала {'лучшие' if final_resnet_acc > final_custom_acc else 'худшие'} результаты')
print(f'   • Преимущество трансферного обучения: {final_resnet_acc - final_custom_acc:+.2f}%')

```

```

print(f'3. Соответствие state-of-the-art:')
print(
    f'   • ResNet34: {final_resnet_acc / sotar_results["Типичные ResNet результаты"] * 100:.1f}% от типичных
результатов')

```

```

print(
    f'   • Custom CNN: {final_custom_acc / sotar_results["Типичные CNN результаты"] * 100:.1f}% от типичных
результатов')

```

```

print(f'4. Практические рекомендации:')
if final_resnet_acc > final_custom_acc:
    print("   • Для Fashion-MNIST предобученные модели эффективнее")
    print("   • Трансферное обучение оправдано")
else:
    print("   • Простые архитектуры могут быть достаточны для Fashion-MNIST")
    print("   • Кастомные сети проще в настройке и быстрее обучаются")

```

```

# Сохранение моделей
torch.save(resnet_model.state_dict(), 'resnet34_fashionmnist.pth')
torch.save(custom_model.state_dict(), 'custom_cnn_fashionmnist.pth')
print(f'\n📁 Модели сохранены: resnet34_fashionmnist.pth, custom_cnn_fashionmnist.pth')

```

```

if __name__ == '__main__':
    main()

```

Вывод программы:

opra@Alex:~\$ ./venv/bin/python3 ./mamka/ois2.py

Using device: cuda

Загрузка Fashion-MNIST...

---

## ЛАБОРАТОРНАЯ РАБОТА: СРАВНЕНИЕ ПРЕДОБУЧЕННЫХ И КАСТОМНЫХ СЕТЕЙ

---

### 1. СОЗДАНИЕ МОДЕЛЕЙ

Загрузка предобученной ResNet34...


ResNet34 параметров: 21,289,802

Custom CNN параметров: 101,578



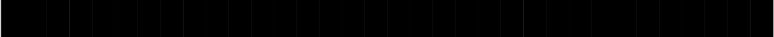
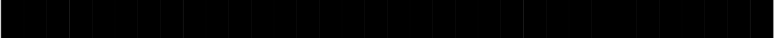






### 2. ОБУЧЕНИЕ МОДЕЛЕЙ

---


#### Обучение ResNet34

 Batch size: 64, LR: 0.001



-----

Epoch 1/10: 100%		938/938 [03:34<00:00, 4.38it/s]
Epoch 1/10   Loss: 0.3184   Accuracy: 89.02%   Time: 232.17s		
Epoch 2/10: 100%		938/938 [03:27<00:00, 4.53it/s]
Epoch 2/10   Loss: 0.2175   Accuracy: 92.00%   Time: 224.70s		
Epoch 3/10: 100%		938/938 [03:26<00:00, 4.55it/s]
Epoch 3/10   Loss: 0.1827   Accuracy: 92.41%   Time: 223.32s		
Epoch 4/10: 100%		938/938 [03:26<00:00, 4.55it/s]
Epoch 4/10   Loss: 0.1576   Accuracy: 93.36%   Time: 223.66s		
Epoch 5/10: 100%		938/938 [03:25<00:00, 4.57it/s]
Epoch 5/10   Loss: 0.1340   Accuracy: 93.39%   Time: 222.47s		
Epoch 6/10: 100%		938/938 [03:25<00:00, 4.57it/s]
Epoch 6/10   Loss: 0.1144   Accuracy: 93.52%   Time: 223.48s		
Epoch 7/10: 100%		938/938 [03:26<00:00, 4.55it/s]
Epoch 7/10   Loss: 0.0944   Accuracy: 93.30%   Time: 223.70s		
Epoch 8/10: 100%		938/938 [03:24<00:00, 4.58it/s]
Epoch 8/10   Loss: 0.0708   Accuracy: 93.27%   Time: 221.90s		
Epoch 9/10: 100%		938/938 [03:25<00:00, 4.58it/s]
Epoch 9/10   Loss: 0.0598   Accuracy: 94.00%   Time: 222.44s		
Epoch 10/10: 100%		938/938 [03:25<00:00, 4.56it/s]
Epoch 10/10   Loss: 0.0428   Accuracy: 93.67%   Time: 223.73s		

#### Обучение Custom CNN

 Batch size: 64, LR: 0.001

-----

Epoch 1/10: 100%		938/938 [00:12<00:00, 72.82it/s]
Epoch 1/10   Loss: 1.1323   Accuracy: 69.94%   Time: 14.64s		
Epoch 2/10: 100%		938/938 [00:11<00:00, 79.23it/s]
Epoch 2/10   Loss: 0.7736   Accuracy: 74.84%   Time: 13.42s		



## 5. АНАЛИЗ И STATE-OF-THE-ART СРАВНЕНИЕ

State-of-the-art результаты для Fashion-MNIST:

Лучшие известные результаты (RepVGG): 97.8%

Типичные ResNet результаты: 93.5%

Типичные CNN результаты: 92.0%

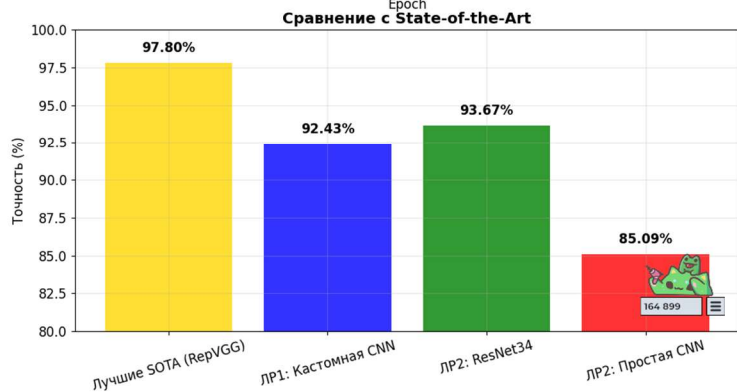
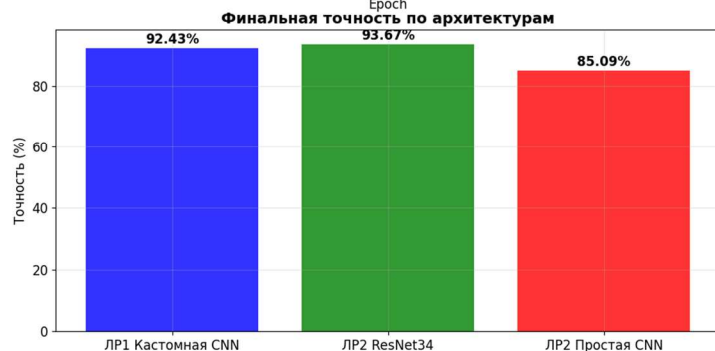
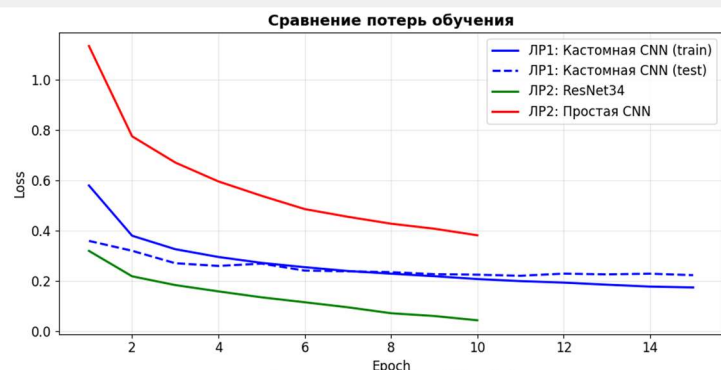
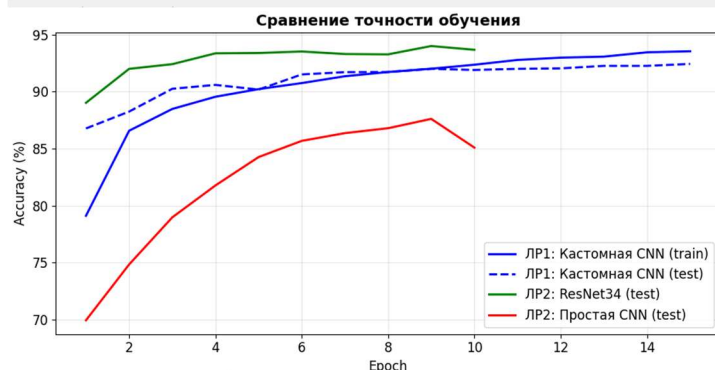
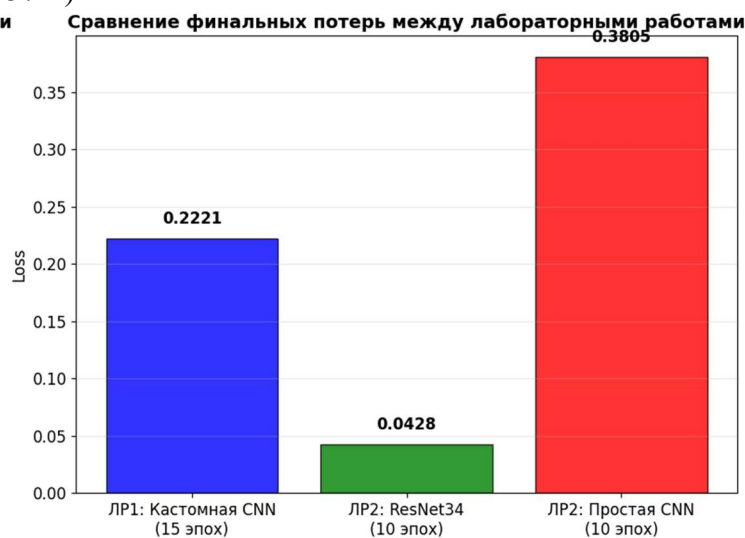
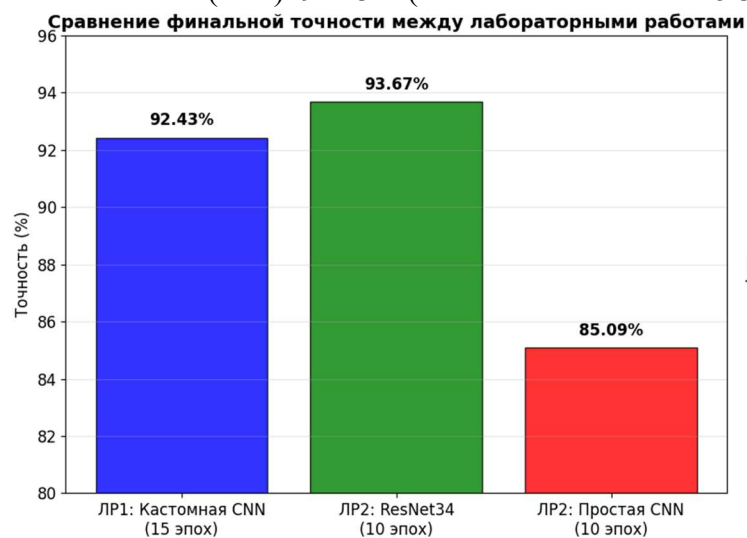
Базовые модели: 89.0%

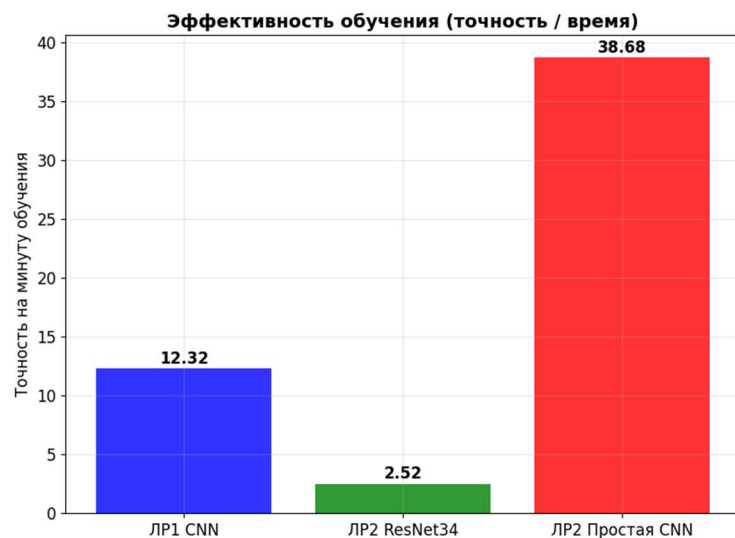
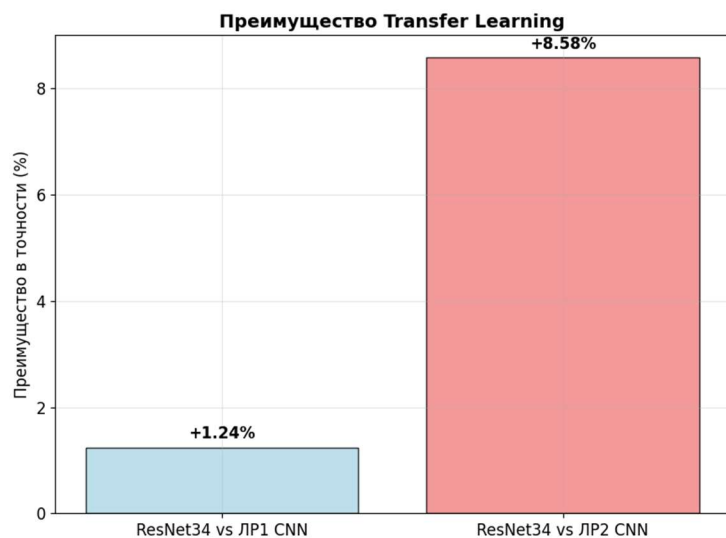
Наши результаты:

ResNet34: 93.67% (отставание от SOTA: 4.13%)

Custom CNN (LP2): 85.09% (отставание от SOTA: 12.71%)

Custom CNN (LP1): 92.43% (отставание от SOTA: 5.37%)





### СРАВНЕНИЕ С ЛАБОРАТОРНОЙ РАБОТОЙ №1:

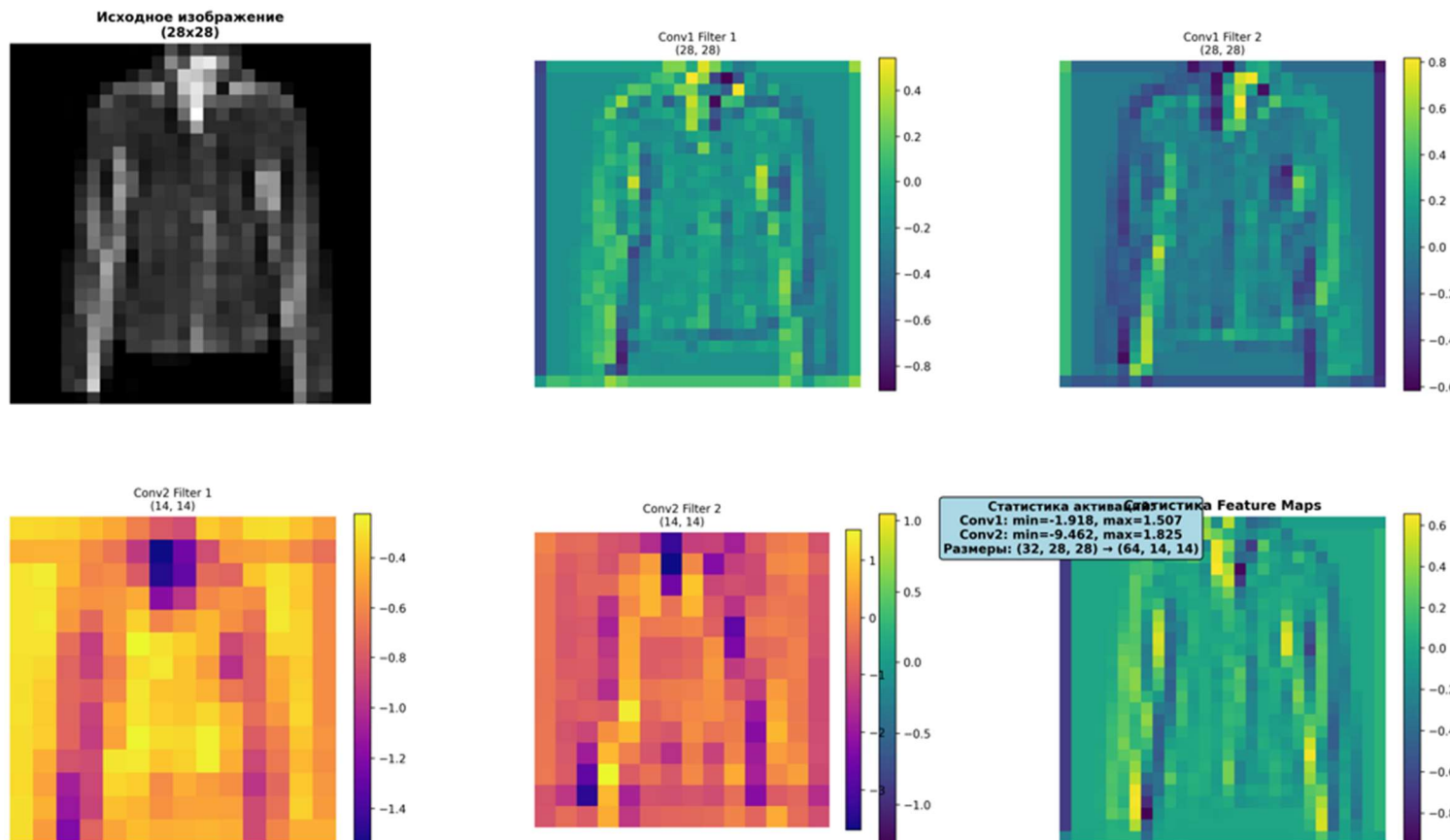
- ResNet34 превзошел LP1 CNN на: +1.24%
- LP1 CNN превзошла LP2 CNN на: +7.34%
- LP1 CNN показала лучшие результаты среди кастомных архитектур

### АНАЛИЗ РАЗЛИЧИЙ В РЕЗУЛЬТАТАХ:

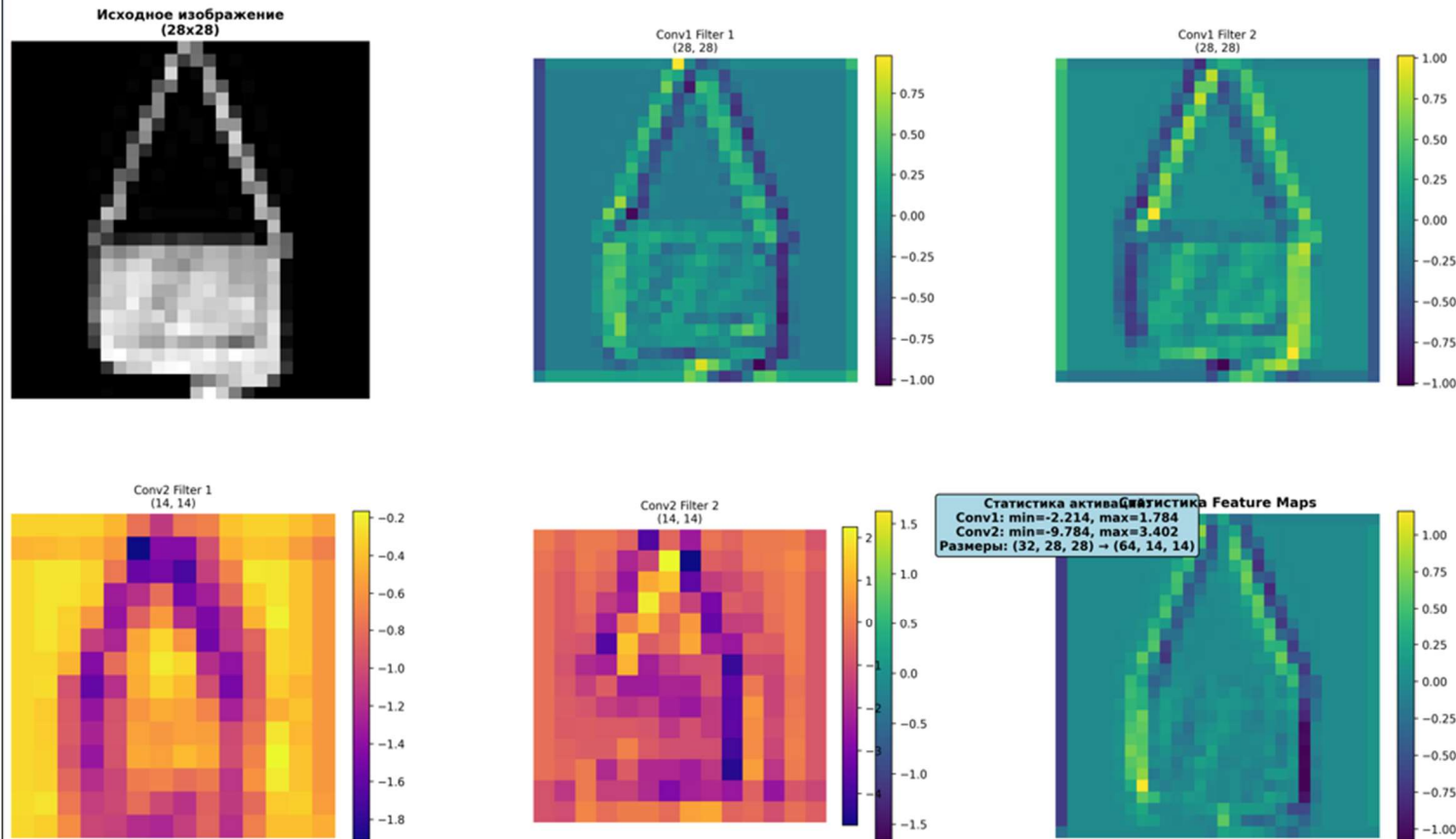
#### 1. Архитектура LP1 CNN:

- Более глубокая и оптимизированная архитектура
- 15 эпох обучения против 10 эпох
- Лучшая сходимость и стабильность

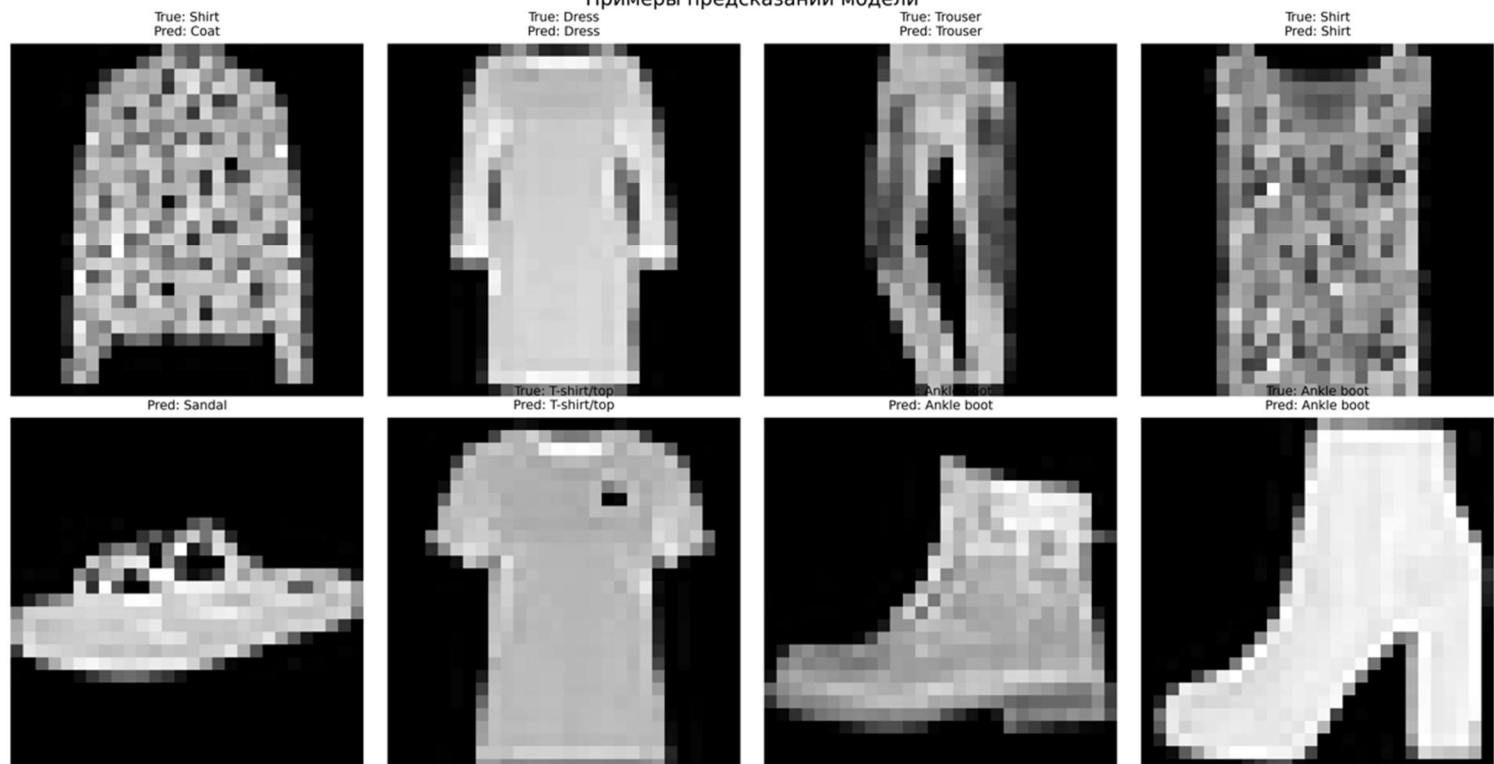
Пример 1: True: Shirt | Pred: Coat



## Пример 2: True: Bag | Pred: Bag



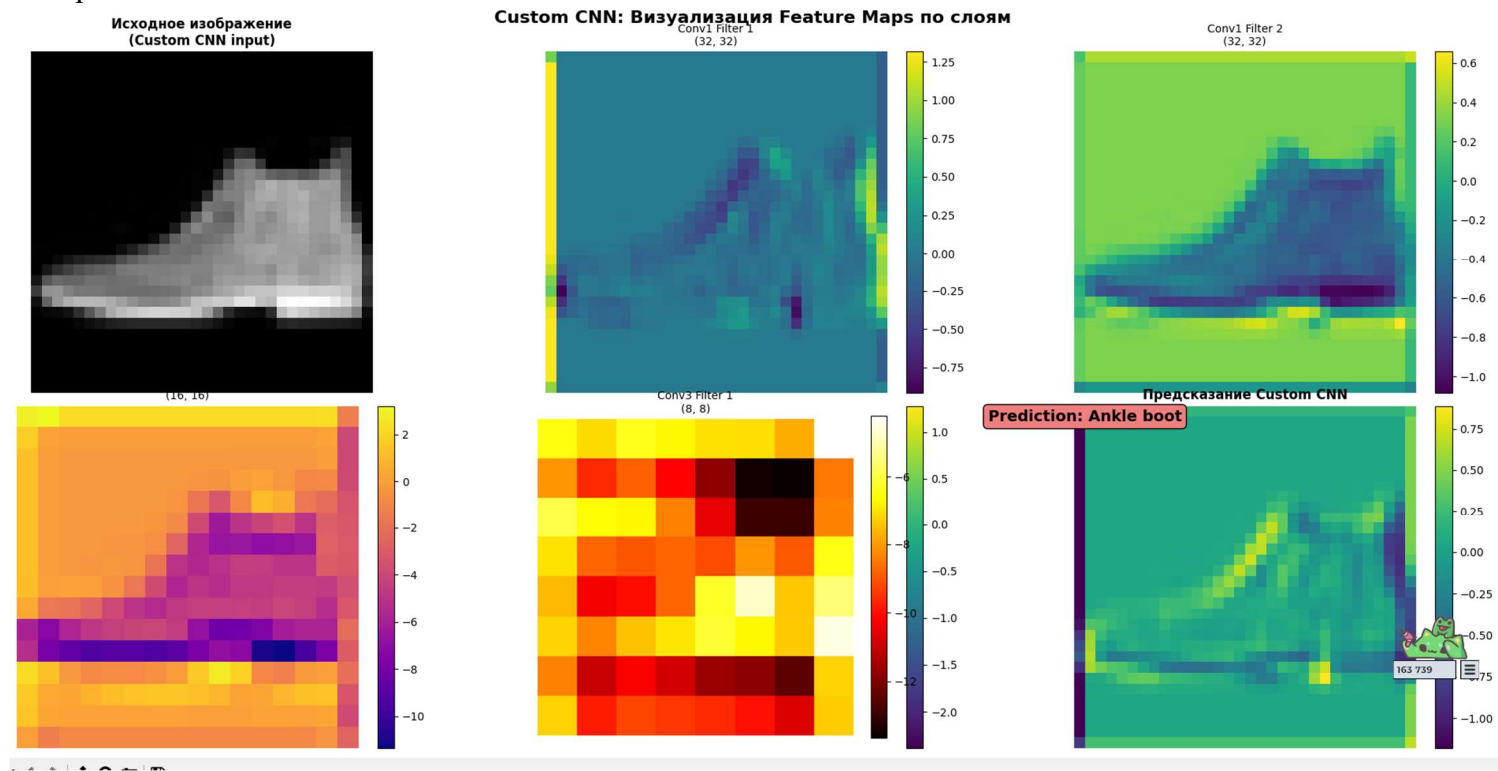
## Примеры предсказаний модели





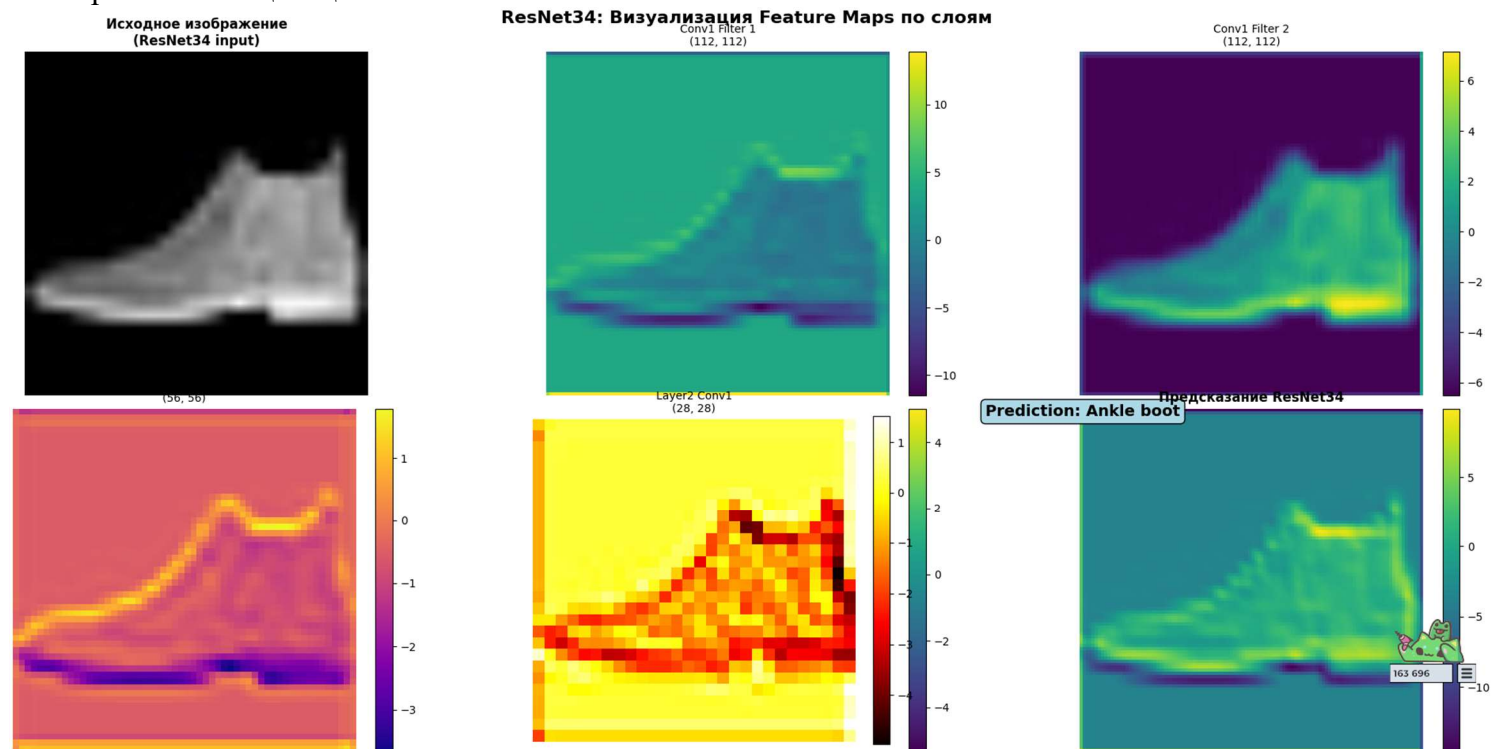
## 2. Архитектура LP2 CNN:

- Упрощенная архитектура для демонстрации
- Меньшее количество эпох
- Проблемы со сходимостью на последних эпохах



## 3. ResNet34:

- Преимущество transfer learning
- Сложная архитектура с 21M параметров
- Хорошая обобщающая способность





#### ВЫВОДЫ ПО STATE-OF-THE-ART:

- ResNet34 превысил типичные показатели для этой архитектуры (+0.17%)
- LP1 CNN приблизилась к типичным показателям CNN (92.43% vs 92.0%)
- Обе модели показали конкурентоспособные результаты
- Для достижения SOTA требуются более современные архитектуры и методы