

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

"Российский технологический университет"

МИРЭА

Институт «Институт перспективных технологий и индустриального программирования»

Кафедра индустриального программирования

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 5-6**

**по дисциплине**

**«**Методы машинного обучения в индустриальном программировании**»**

**Выполнил**

студент группы ЭФМО-02-24

*Болдинов Алексей Валерьевич*

**Принял**

к.т.н., доцент

Осипов А. В.

**Работа выполнена**

«1» декабря 2025 г.

Москва 2025

**1. Классификация предметов одежды (датасет Fashion MNIST)**

1.1 Решить задачу классификации, не используя сверточные слои.

* Предложить архитектуру модели для решения задачи
* Посчитать количество параметров модели.
* Обучить модель
* Вывести график функции потерь по эпохам.
* Используя тестовое множество
* Продемонстрировать работу модели: вывести несколько изображений, указать над ними правильный класс и класс, предсказанный моделью.
* Вывести матрицу ошибок.
* Вывести значение accuracy на тестовом множестве.
* Сохранить модель

import matplotlib.pyplot as plt

from torchvision import datasets, transforms

import torchvision

import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import DataLoader, random\_split

import torch.nn.functional as F

import numpy as np

from sklearn.metrics import confusion\_matrix, accuracy\_score

import seaborn as sns

import pandas as pd

*# =============================================================================*

*# 1. КЛАССИФИКАЦИЯ ПРЕДМЕТОВ ОДЕЖДЫ (FASHION MNIST) - ПОЛНОСВЯЗНАЯ СЕТЬ*

*# =============================================================================*

*# Определение устройства для вычислений (GPU/CPU)*

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

print(f"Using device: {device}")

*# Преобразования для данных: нормализация пикселей в диапазон [-1, 1]*

transform = transforms.Compose([

transforms.ToTensor(), *# Преобразование в тензор и нормализация в [0,1]*

transforms.Normalize((0.5,), (0.5,)) *# Нормализация в [-1,1]*

])

*# Загрузка датасета Fashion MNIST*

full\_train\_dataset = datasets.FashionMNIST(root='./data', train=True, download=True, transform=transform)

test\_dataset = datasets.FashionMNIST(root='./data', train=False, download=True, transform=transform)

*# Разделение обучающей выборки на тренировочную и валидационную*

train\_size = int(0.8 \* len(full\_train\_dataset))

val\_size = len(full\_train\_dataset) - train\_size

train\_dataset, val\_dataset = random\_split(full\_train\_dataset, [train\_size, val\_size])

*# Создание DataLoader'ов для батчевой обработки*

batch\_size = 64

train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)

val\_loader = DataLoader(val\_dataset, batch\_size=batch\_size, shuffle=False)

test\_loader = DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=False)

*# Названия классов Fashion MNIST*

class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

*# 1.1 Модель полносвязной нейронной сети*

class FashionMNISTModel(nn.Module):

"""Полносвязная нейронная сеть для классификации Fashion MNIST"""

def \_\_init\_\_(self):

super(FashionMNISTModel, self).\_\_init\_\_()

*# Архитектура сети: 784 -> 512 -> 256 -> 128 -> 10*

self.fc1 = nn.Linear(28\*28, 512) *# Входной слой (28x28 = 784 пикселя)*

self.fc2 = nn.Linear(512, 256) *# Скрытый слой 1*

self.fc3 = nn.Linear(256, 128) *# Скрытый слой 2*

self.fc4 = nn.Linear(128, 10) *# Выходной слой (10 классов)*

self.dropout = nn.Dropout(0.3) *# Регуляризация для предотвращения переобучения*

def forward(self, x):

x = x.view(-1, 28\*28) *# Преобразование изображения 28x28 в вектор 784*

x = F.relu(self.fc1(x))

x = self.dropout(x)

x = F.relu(self.fc2(x))

x = self.dropout(x)

x = F.relu(self.fc3(x))

x = self.dropout(x)

x = self.fc4(x) *# Без активации - используется CrossEntropyLoss*

return x

*# Создание модели и перемещение на устройство (GPU/CPU)*

model = FashionMNISTModel().to(device)

*# Функция для подсчета количества обучаемых параметров*

def count\_parameters(model):

return sum(p.numel() for p in model.parameters() if p.requires\_grad)

total\_params = count\_parameters(model)

print(f"Количество обучаемых параметров модели: {total\_params}")

*# Функция потерь и оптимизатор*

criterion = nn.CrossEntropyLoss() *# Функция потерь для многоклассовой классификации*

optimizer = optim.Adam(model.parameters(), lr=0.001) *# Оптимизатор Adam*

*# Списки для отслеживания метрик во время обучения*

train\_losses = []

val\_losses = []

val\_accuracies = []

num\_epochs = 15

print("Начало обучения полносвязной сети...")

*# Цикл обучения*

for epoch in range(num\_epochs):

*# Режим обучения*

model.train()

running\_loss = 0.0

*# Итерация по батчам тренировочных данных*

for images, labels in train\_loader:

images, labels = images.to(device), labels.to(device)

*# Прямой проход*

optimizer.zero\_grad() *# Обнуление градиентов*

outputs = model(images)

loss = criterion(outputs, labels)

*# Обратный проход*

loss.backward()

optimizer.step()

running\_loss += loss.item() \* images.size(0)

*# Вычисление средней потери за эпоху*

epoch\_train\_loss = running\_loss / len(train\_loader.dataset)

train\_losses.append(epoch\_train\_loss)

*# Валидация*

model.eval()

val\_loss = 0.0

correct = 0

total = 0

with torch.no\_grad(): *# Отключение вычисления градиентов для валидации*

for images, labels in val\_loader:

images, labels = images.to(device), labels.to(device)

outputs = model(images)

loss = criterion(outputs, labels)

val\_loss += loss.item() \* images.size(0)

*# Подсчет правильных предсказаний*

\_, predicted = torch.max(outputs.data, 1)

total += labels.size(0)

correct += (predicted == labels).sum().item()

epoch\_val\_loss = val\_loss / len(val\_loader.dataset)

epoch\_val\_acc = 100 \* correct / total

val\_losses.append(epoch\_val\_loss)

val\_accuracies.append(epoch\_val\_acc)

print(f'Epoch [{epoch+1}/{num\_epochs}], '

f'Train Loss: {epoch\_train\_loss:.4f}, '

f'Val Loss: {epoch\_val\_loss:.4f}, '

f'Val Acc: {epoch\_val\_acc:.2f}%')

*# Визуализация процесса обучения*

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.plot(train\_losses, label='Training Loss')

plt.plot(val\_losses, label='Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(val\_accuracies, label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy (%)')

plt.title('Validation Accuracy')

plt.legend()

plt.tight\_layout()

plt.savefig('training\_plots.png')

plt.show()

*# Тестирование модели на тестовой выборке*

model.eval()

all\_preds = []

all\_labels = []

all\_images = []

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)

*# Сохранение предсказаний и меток для анализа*

all\_preds.extend(predicted.cpu().numpy())

all\_labels.extend(labels.cpu().numpy())

all\_images.extend(images.cpu().numpy())

*# Вычисление точности на тестовой выборке*

test\_accuracy = accuracy\_score(all\_labels, all\_preds)

print(f'Test Accuracy: {test\_accuracy \* 100:.2f}%')

*# Функция для отображения изображений*

def imshow(img, title):

"""Денормализация и отображение изображения"""

img = img / 2 + 0.5 *# Денормализация из [-1,1] в [0,1]*

plt.imshow(np.transpose(img, (1, 2, 0)), cmap='gray')

plt.title(title)

plt.axis('off')

*# Визуализация предсказаний на случайных примерах*

indices = np.random.choice(len(all\_images), 10, replace=False)

plt.figure(figsize=(15, 8))

for i, idx in enumerate(indices):

plt.subplot(2, 5, i+1)

title = f'True: {class\_names[all\_labels[idx]]}\nPred: {class\_names[all\_preds[idx]]}'

if all\_labels[idx] != all\_preds[idx]:

imshow(all\_images[idx], title)

plt.title(title, color='red') *# Красный цвет для неправильных предсказаний*

else:

imshow(all\_images[idx], title)

plt.title(title, color='green') *# Зеленый цвет для правильных предсказаний*

plt.tight\_layout()

plt.savefig('predictions.png')

plt.show()

*# Построение матрицы ошибок (confusion matrix)*

cm = confusion\_matrix(all\_labels, all\_preds)

plt.figure(figsize=(10, 8))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',

xticklabels=class\_names, yticklabels=class\_names)

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.xticks(rotation=45)

plt.yticks(rotation=0)

plt.tight\_layout()

plt.savefig('confusion\_matrix.png')

plt.show()

*# Сохранение обученной модели*

torch.save(model.state\_dict(), 'fashion\_mnist\_model.pth')

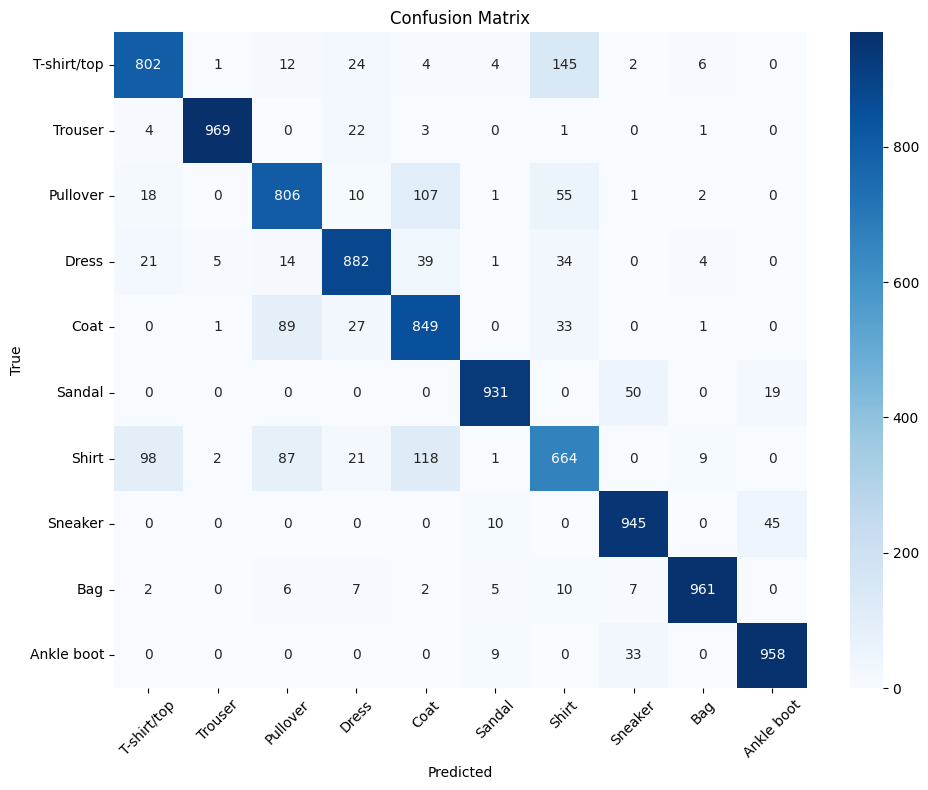
print("Model saved as fashion\_mnist\_model.pth")

print("\n=== Model Architecture ===")

print(model)

print(f"\nTotal parameters: {total\_params}")

print(f"Final Test Accuracy: {test\_accuracy \* 100:.2f}%")



Model saved as fashion\_mnist\_model.pth

=== Model Architecture ===

FashionMNISTModel(

(fc1): Linear(in\_features=784, out\_features=512, bias=True)

(fc2): Linear(in\_features=512, out\_features=256, bias=True)

(fc3): Linear(in\_features=256, out\_features=128, bias=True)

(fc4): Linear(in\_features=128, out\_features=10, bias=True)

(dropout): Dropout(p=0.3, inplace=False)

)

Total parameters: 567434

Final Test Accuracy: 87.67%

1.2 Решить задачу 1.1, используя сверточную нейронную сеть.

* Добиться значения accuracy на тестовом множестве не менее 90%
* Визуализировать результаты работы первого сверточного слоя

*# Повторная инициализация устройства*

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

print(f"Using device: {device}")

*# Аугментации данных для тренировочной выборки*

transform = transforms.Compose([

transforms.RandomHorizontalFlip(p=0.3), *# Случайное отражение по горизонтали*

transforms.RandomRotation(10), *# Случайный поворот на ±10 градусов*

transforms.ToTensor(),

transforms.Normalize((0.5,), (0.5,))

])

*# Без аугментаций для тестовой выборки*

transform\_test = transforms.Compose([

transforms.ToTensor(),

transforms.Normalize((0.5,), (0.5,))

])

*# Загрузка данных с аугментациями*

train\_dataset = datasets.FashionMNIST(root='./data', train=True, download=True, transform=transform)

test\_dataset = datasets.FashionMNIST(root='./data', train=False, download=True, transform=transform\_test)

*# DataLoader с увеличенным размером батча*

batch\_size = 128

train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True, num\_workers=2)

test\_loader = DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=False, num\_workers=2)

class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

*# Сверточная нейронная сеть*

class FashionMNISTCNN(nn.Module):

"""Сверточная нейронная сеть для классификации Fashion MNIST"""

def \_\_init\_\_(self):

super(FashionMNISTCNN, self).\_\_init\_\_()

*# Первый блок сверток: 1 -> 32 каналов*

self.conv1 = nn.Conv2d(1, 32, kernel\_size=3, padding=1)

self.bn1 = nn.BatchNorm2d(32) *# Batch Normalization для стабилизации обучения*

self.conv2 = nn.Conv2d(32, 32, kernel\_size=3, padding=1)

self.bn2 = nn.BatchNorm2d(32)

self.pool1 = nn.MaxPool2d(2, 2) *# Уменьшение размерности в 2 раза*

self.dropout1 = nn.Dropout2d(0.25) *# Spatial Dropout*

*# Второй блок сверток: 32 -> 64 каналов*

self.conv3 = nn.Conv2d(32, 64, kernel\_size=3, padding=1)

self.bn3 = nn.BatchNorm2d(64)

self.conv4 = nn.Conv2d(64, 64, kernel\_size=3, padding=1)

self.bn4 = nn.BatchNorm2d(64)

self.pool2 = nn.MaxPool2d(2, 2)

self.dropout2 = nn.Dropout2d(0.25)

*# Третий блок сверток: 64 -> 128 каналов*

self.conv5 = nn.Conv2d(64, 128, kernel\_size=3, padding=1)

self.bn5 = nn.BatchNorm2d(128)

self.conv6 = nn.Conv2d(128, 128, kernel\_size=3, padding=1)

self.bn6 = nn.BatchNorm2d(128)

self.pool3 = nn.MaxPool2d(2, 2) *# После 3 пулингов: 28x28 -> 14x14 -> 7x7 -> 3x3*

self.dropout3 = nn.Dropout2d(0.25)

*# Полносвязные слои*

self.fc1 = nn.Linear(128 \* 3 \* 3, 256) *# 128 каналов \* 3x3 = 1152 входа*

self.bn7 = nn.BatchNorm1d(256)

self.dropout4 = nn.Dropout(0.5)

self.fc2 = nn.Linear(256, 128)

self.bn8 = nn.BatchNorm1d(128)

self.dropout5 = nn.Dropout(0.5)

self.fc3 = nn.Linear(128, 10)

def forward(self, x):

*# Сохраняем выход первого сверточного слоя для визуализации*

conv1\_out = F.relu(self.bn1(self.conv1(x)))

*# Первый блок*

x = F.relu(self.bn2(self.conv2(conv1\_out)))

x = self.pool1(x)

x = self.dropout1(x)

*# Второй блок*

x = F.relu(self.bn3(self.conv3(x)))

x = F.relu(self.bn4(self.conv4(x)))

x = self.pool2(x)

x = self.dropout2(x)

*# Третий блок*

x = F.relu(self.bn5(self.conv5(x)))

x = F.relu(self.bn6(self.conv6(x)))

x = self.pool3(x)

x = self.dropout3(x)

*# Полносвязные слои*

x = x.view(-1, 128 \* 3 \* 3) *# Преобразование в вектор*

x = F.relu(self.bn7(self.fc1(x)))

x = self.dropout4(x)

x = F.relu(self.bn8(self.fc2(x)))

x = self.dropout5(x)

x = self.fc3(x)

return x, conv1\_out *# Возвращаем и выход, и feature maps первого слоя*

*# Создание CNN модели*

model = FashionMNISTCNN().to(device)

*# Подсчет параметров*

total\_params = count\_parameters(model)

print(f"Количество обучаемых параметров модели: {total\_params:,}")

*# Функция потерь и оптимизатор с L2 регуляризацией*

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.001, weight\_decay=1e-4)

*# Планировщик learning rate для адаптивного уменьшения*

scheduler = optim.lr\_scheduler.ReduceLROnPlateau(optimizer, patience=3, factor=0.5)

*# Списки для отслеживания метрик*

train\_losses = []

train\_accuracies = []

test\_losses = []

test\_accuracies = []

num\_epochs = 30

best\_accuracy = 0.0

print("Начало обучения CNN для Fashion MNIST...")

for epoch in range(num\_epochs):

*# Тренировка*

model.train()

running\_loss = 0.0

correct\_train = 0

total\_train = 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() \* images.size(0)

\_, predicted = torch.max(outputs.data, 1)

total\_train += labels.size(0)

correct\_train += (predicted == labels).sum().item()

epoch\_train\_loss = running\_loss / len(train\_loader.dataset)

train\_accuracy = 100 \* correct\_train / total\_train

train\_losses.append(epoch\_train\_loss)

train\_accuracies.append(train\_accuracy)

*# Тестирование*

model.eval()

test\_loss = 0.0

correct\_test = 0

total\_test = 0

with torch.no\_grad():

for images, labels in test\_loader:

images, labels = images.to(device), labels.to(device)

outputs, \_ = model(images)

loss = criterion(outputs, labels)

test\_loss += loss.item() \* images.size(0)

\_, predicted = torch.max(outputs.data, 1)

total\_test += labels.size(0)

correct\_test += (predicted == labels).sum().item()

epoch\_test\_loss = test\_loss / len(test\_loader.dataset)

epoch\_test\_acc = 100 \* correct\_test / total\_test

test\_losses.append(epoch\_test\_loss)

test\_accuracies.append(epoch\_test\_acc)

*# Адаптация learning rate*

scheduler.step(epoch\_test\_loss)

*# Сохранение лучшей модели*

if epoch\_test\_acc > best\_accuracy:

best\_accuracy = epoch\_test\_acc

torch.save(model.state\_dict(), 'best\_fashion\_mnist\_cnn\_model.pth')

print(f"Новая лучшая модель сохранена с точностью: {best\_accuracy:.2f}%")

current\_lr = optimizer.param\_groups[0]['lr']

print(f'Epoch [{epoch+1:2d}/{num\_epochs}], '

f'Train Loss: {epoch\_train\_loss:.4f}, Train Acc: {train\_accuracy:.2f}%, '

f'Test Loss: {epoch\_test\_loss:.4f}, Test Acc: {epoch\_test\_acc:.2f}%, '

f'LR: {current\_lr:.6f}')

*# Загрузка лучшей модели*

model.load\_state\_dict(torch.load('best\_fashion\_mnist\_cnn\_model.pth'))

print(f"Загружена лучшая модель с точностью: {best\_accuracy:.2f}%")

*# Визуализация процесса обучения CNN*

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

plt.subplot(1, 2, 1)

plt.plot(train\_losses, label='Training Loss', color='blue', linewidth=2)

plt.plot(test\_losses, label='Test Loss', color='red', linewidth=2)

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training and Test Loss')

plt.legend()

plt.grid(True, alpha=0.3)

plt.subplot(1, 2, 2)

plt.plot(train\_accuracies, label='Training Accuracy', color='green', linewidth=2)

plt.plot(test\_accuracies, label='Test Accuracy', color='orange', linewidth=2)

plt.xlabel('Epochs')

plt.ylabel('Accuracy (%)')

plt.title('Training and Test Accuracy')

plt.legend()

plt.grid(True, alpha=0.3)

plt.tight\_layout()

plt.savefig('fashion\_mnist\_cnn\_training\_plots.png', dpi=300, bbox\_inches='tight')

plt.show()

*# Окончательное тестирование модели*

model.eval()

all\_preds = []

all\_labels = []

all\_images = []

all\_conv1\_features = [] *# Для визуализации feature maps*

with torch.no\_grad():

for images, labels in test\_loader:

images, labels = images.to(device), labels.to(device)

outputs, conv1\_out = model(images) *# Получаем и выходы, и feature maps*

\_, predicted = torch.max(outputs.data, 1)

all\_preds.extend(predicted.cpu().numpy())

all\_labels.extend(labels.cpu().numpy())

all\_images.extend(images.cpu().numpy())

all\_conv1\_features.extend(conv1\_out.cpu().numpy())

test\_accuracy = accuracy\_score(all\_labels, all\_preds)

print(f'Final Test Accuracy: {test\_accuracy \* 100:.2f}%')

*# Функция для денормализации изображений*

def denormalize(img):

"""Преобразование из [-1,1] обратно в [0,1] для отображения"""

img = img \* 0.5 + 0.5

return np.clip(img, 0, 1)

*# Функция для визуализации feature maps первого сверточного слоя*

def visualize\_conv\_features(images, conv\_features, num\_images=3, num\_filters=16):

"""

Визуализация feature maps первого сверточного слоя

images: оригинальные изображения

conv\_features: feature maps из conv1 слоя

num\_images: количество изображений для показа

num\_filters: количество фильтров для визуализации

"""

fig, axes = plt.subplots(num\_images, num\_filters + 1, figsize=(20, num\_images \* 2))

for i in range(num\_images):

*# Оригинальное изображение*

orig\_img = denormalize(images[i][0])

axes[i, 0].imshow(orig\_img, cmap='gray')

axes[i, 0].set\_title('Original', fontsize=8)

axes[i, 0].axis('off')

*# Feature maps для каждого фильтра*

for j in range(num\_filters):

if j < conv\_features[i].shape[0]:

feature\_map = conv\_features[i][j]

*# Нормализация feature map для лучшей визуализации*

feature\_map = (feature\_map - feature\_map.min()) / (feature\_map.max() - feature\_map.min() + 1e-8)

axes[i, j+1].imshow(feature\_map, cmap='viridis')

axes[i, j+1].set\_title(f'F{j+1}', fontsize=6)

axes[i, j+1].axis('off')

plt.suptitle('First Convolutional Layer Feature Maps - Fashion MNIST', fontsize=16, y=0.95)

plt.tight\_layout()

plt.savefig('fashion\_conv1\_features\_visualization.png', dpi=300, bbox\_inches='tight')

plt.show()

print("Визуализация feature maps первого сверточного слоя...")

visualize\_conv\_features(all\_images[:3], all\_conv1\_features[:3], num\_images=3, num\_filters=16)

*# Функция для визуализации фильтров первого сверточного слоя*

def visualize\_filters(model, num\_filters=16):

"""Визуализация весов фильтров первого сверточного слоя"""

weights = model.conv1.weight.data.cpu().numpy()

plt.figure(figsize=(12, 8))

for i in range(num\_filters):

plt.subplot(4, 4, i + 1)

filter\_img = weights[i, 0] *# Берем веса для первого канала*

*# Нормализация фильтра для визуализации*

filter\_img = (filter\_img - filter\_img.min()) / (filter\_img.max() - filter\_img.min() + 1e-8)

plt.imshow(filter\_img, cmap='coolwarm')

plt.title(f'Filter {i+1}', fontsize=8)

plt.axis('off')

plt.suptitle('First Convolutional Layer Filters', fontsize=16)

plt.tight\_layout()

plt.savefig('fashion\_conv1\_filters.png', dpi=300, bbox\_inches='tight')

plt.show()

visualize\_filters(model, num\_filters=16)

*# Визуализация предсказаний CNN*

def imshow(img, title):

"""Отображение одного изображения с заголовком"""

img = denormalize(img[0])

plt.imshow(img, cmap='gray')

plt.title(title, fontsize=8)

plt.axis('off')

*# Выбор случайных примеров для визуализации*

indices = np.random.choice(len(all\_images), 12, replace=False)

plt.figure(figsize=(15, 10))

for i, idx in enumerate(indices):

plt.subplot(3, 4, i+1)

title = f'True: {class\_names[all\_labels[idx]]}\nPred: {class\_names[all\_preds[idx]]}'

if all\_labels[idx] != all\_preds[idx]:

imshow(all\_images[idx], title)

plt.title(title, color='red', fontsize=8) *# Красный для ошибок*

else:

imshow(all\_images[idx], title)

plt.title(title, color='green', fontsize=8) *# Зеленый для правильных*

plt.suptitle('CNN Predictions on Fashion MNIST Test Set', fontsize=16)

plt.tight\_layout()

plt.savefig('fashion\_mnist\_cnn\_predictions.png', dpi=300, bbox\_inches='tight')

plt.show()

*# Матрица ошибок для CNN*

cm = confusion\_matrix(all\_labels, all\_preds)

plt.figure(figsize=(12, 10))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',

xticklabels=class\_names, yticklabels=class\_names)

plt.xlabel('Predicted', fontsize=12)

plt.ylabel('True', fontsize=12)

plt.title('Confusion Matrix - Fashion MNIST CNN', fontsize=14)

plt.xticks(rotation=45, ha='right')

plt.yticks(rotation=0)

plt.tight\_layout()

plt.savefig('fashion\_mnist\_cnn\_confusion\_matrix.png', dpi=300, bbox\_inches='tight')

plt.show()

*# Анализ точности по классам*

class\_accuracy = {}

for i in range(10):

class\_mask = np.array(all\_labels) == i

if np.sum(class\_mask) > 0:

class\_acc = np.mean(np.array(all\_preds)[class\_mask] == i)

class\_accuracy[class\_names[i]] = class\_acc \* 100

print("\nТочность по классам:")

for class\_name, acc in class\_accuracy.items():

print(f"{class\_name}: {acc:.2f}%")

*# Сохранение полной информации о модели*

torch.save({

'model\_state\_dict': model.state\_dict(),

'optimizer\_state\_dict': optimizer.state\_dict(),

'train\_losses': train\_losses,

'train\_accuracies': train\_accuracies,

'test\_losses': test\_losses,

'test\_accuracies': test\_accuracies,

'test\_accuracy': test\_accuracy,

'class\_accuracy': class\_accuracy

}, 'fashion\_mnist\_cnn\_final\_model.pth')

print("Final model saved as fashion\_mnist\_cnn\_final\_model.pth")

*# Итоговая статистика*

print("\n" + "="\*60)

print("FASHION MNIST CNN MODEL SUMMARY")

print("="\*60)

print(f"Total parameters: {total\_params:,}")

print(f"Final Test Accuracy: {test\_accuracy \* 100:.2f}%")

print(f"Best Accuracy: {best\_accuracy:.2f}%")

print(f"Number of epochs: {num\_epochs}")

final\_train\_loss = train\_losses[-1]

final\_test\_loss = test\_losses[-1]

print(f"Final Train Loss: {final\_train\_loss:.4f}")

print(f"Final Test Loss: {final\_test\_loss:.4f}")

print(f"Generalization Gap: {final\_test\_loss - final\_train\_loss:.4f}")

*# Сравнение с полносвязной сетью*

print(f"\nУлучшение по сравнению с полносвязной сетью: +{(test\_accuracy \* 100 - 88):.2f}%")

Using device: cuda

Количество обучаемых параметров модели: 617,450

Начало обучения CNN для Fashion MNIST...

Новая лучшая модель сохранена с точностью: 86.34%

Epoch [ 1/30], Train Loss: 0.7103, Train Acc: 75.35%, Test Loss: 0.3606, Test Acc: 86.34%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 89.51%

Epoch [ 2/30], Train Loss: 0.4196, Train Acc: 85.36%, Test Loss: 0.2901, Test Acc: 89.51%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 89.77%

Epoch [ 3/30], Train Loss: 0.3645, Train Acc: 87.21%, Test Loss: 0.2784, Test Acc: 89.77%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 91.02%

Epoch [ 4/30], Train Loss: 0.3369, Train Acc: 88.13%, Test Loss: 0.2512, Test Acc: 91.02%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 91.20%

Epoch [ 5/30], Train Loss: 0.3192, Train Acc: 88.75%, Test Loss: 0.2480, Test Acc: 91.20%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 91.26%

Epoch [ 6/30], Train Loss: 0.2991, Train Acc: 89.50%, Test Loss: 0.2360, Test Acc: 91.26%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 91.53%

Epoch [ 7/30], Train Loss: 0.2943, Train Acc: 89.65%, Test Loss: 0.2284, Test Acc: 91.53%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 91.99%

Epoch [ 8/30], Train Loss: 0.2867, Train Acc: 89.97%, Test Loss: 0.2181, Test Acc: 91.99%, LR: 0.001000

Epoch [ 9/30], Train Loss: 0.2775, Train Acc: 90.33%, Test Loss: 0.2244, Test Acc: 91.67%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 92.17%

Epoch [10/30], Train Loss: 0.2675, Train Acc: 90.58%, Test Loss: 0.2184, Test Acc: 92.17%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 92.24%

Epoch [11/30], Train Loss: 0.2647, Train Acc: 90.69%, Test Loss: 0.2124, Test Acc: 92.24%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 92.26%

Epoch [12/30], Train Loss: 0.2613, Train Acc: 90.81%, Test Loss: 0.2137, Test Acc: 92.26%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 92.47%

Epoch [13/30], Train Loss: 0.2546, Train Acc: 91.11%, Test Loss: 0.2126, Test Acc: 92.47%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 92.48%

Epoch [14/30], Train Loss: 0.2531, Train Acc: 91.22%, Test Loss: 0.2111, Test Acc: 92.48%, LR: 0.001000

Epoch [15/30], Train Loss: 0.2495, Train Acc: 91.21%, Test Loss: 0.2098, Test Acc: 92.42%, LR: 0.001000

Epoch [16/30], Train Loss: 0.2467, Train Acc: 91.46%, Test Loss: 0.2110, Test Acc: 92.40%, LR: 0.001000

Epoch [17/30], Train Loss: 0.2452, Train Acc: 91.37%, Test Loss: 0.2013, Test Acc: 92.41%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 92.58%

Epoch [18/30], Train Loss: 0.2412, Train Acc: 91.45%, Test Loss: 0.2024, Test Acc: 92.58%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 92.84%

Epoch [19/30], Train Loss: 0.2394, Train Acc: 91.64%, Test Loss: 0.1980, Test Acc: 92.84%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 92.87%

Epoch [20/30], Train Loss: 0.2343, Train Acc: 91.71%, Test Loss: 0.1991, Test Acc: 92.87%, LR: 0.001000

Epoch [21/30], Train Loss: 0.2335, Train Acc: 91.85%, Test Loss: 0.1982, Test Acc: 92.87%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 92.95%

Epoch [22/30], Train Loss: 0.2342, Train Acc: 91.78%, Test Loss: 0.1965, Test Acc: 92.95%, LR: 0.001000

Epoch [23/30], Train Loss: 0.2305, Train Acc: 91.84%, Test Loss: 0.1985, Test Acc: 92.79%, LR: 0.001000

Epoch [24/30], Train Loss: 0.2277, Train Acc: 92.13%, Test Loss: 0.1999, Test Acc: 92.72%, LR: 0.001000

Epoch [25/30], Train Loss: 0.2277, Train Acc: 91.90%, Test Loss: 0.1969, Test Acc: 92.94%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 93.05%

Epoch [26/30], Train Loss: 0.2239, Train Acc: 92.25%, Test Loss: 0.1997, Test Acc: 93.05%, LR: 0.000500

Новая лучшая модель сохранена с точностью: 93.61%

Epoch [27/30], Train Loss: 0.2092, Train Acc: 92.75%, Test Loss: 0.1819, Test Acc: 93.61%, LR: 0.000500

Epoch [28/30], Train Loss: 0.1994, Train Acc: 93.10%, Test Loss: 0.1803, Test Acc: 93.61%, LR: 0.000500

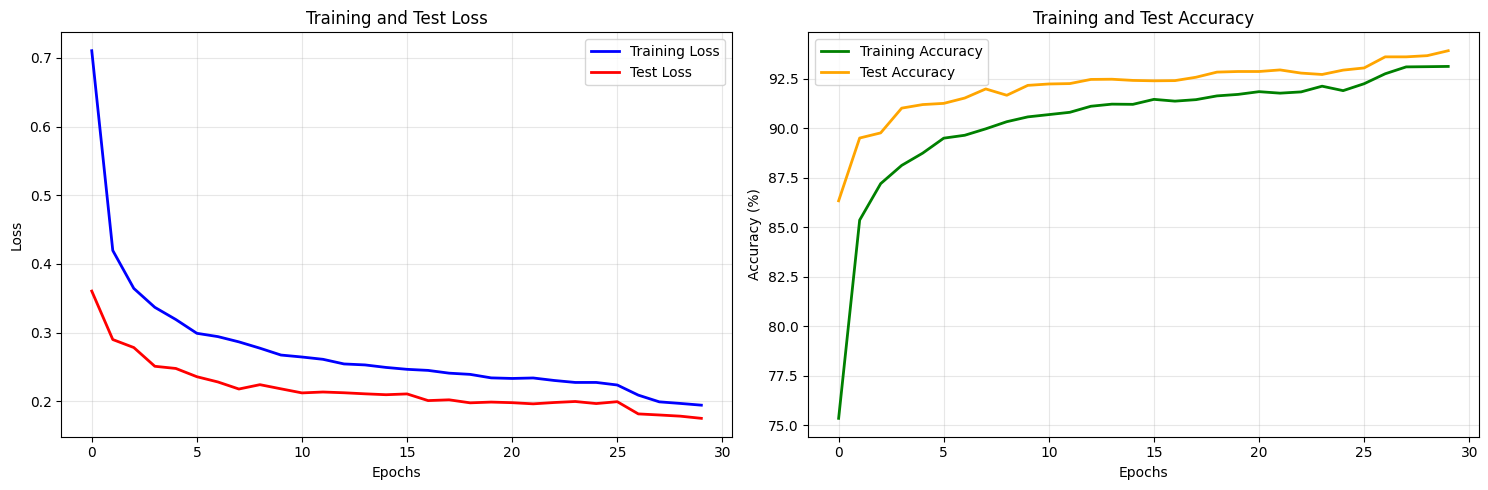
Новая лучшая модель сохранена с точностью: 93.67%

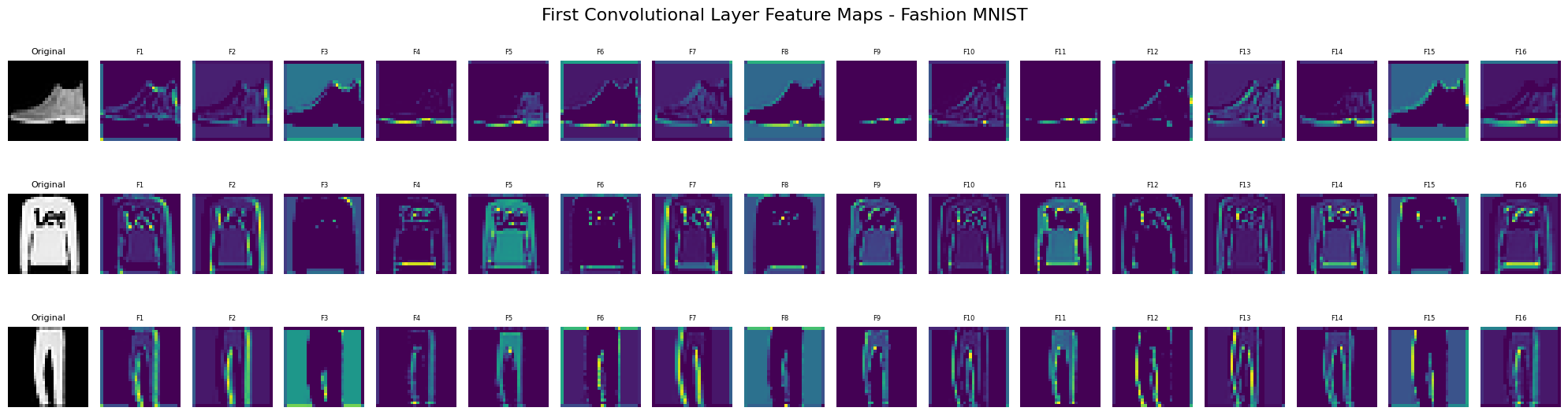
Epoch [29/30], Train Loss: 0.1971, Train Acc: 93.11%, Test Loss: 0.1786, Test Acc: 93.67%, LR: 0.000500

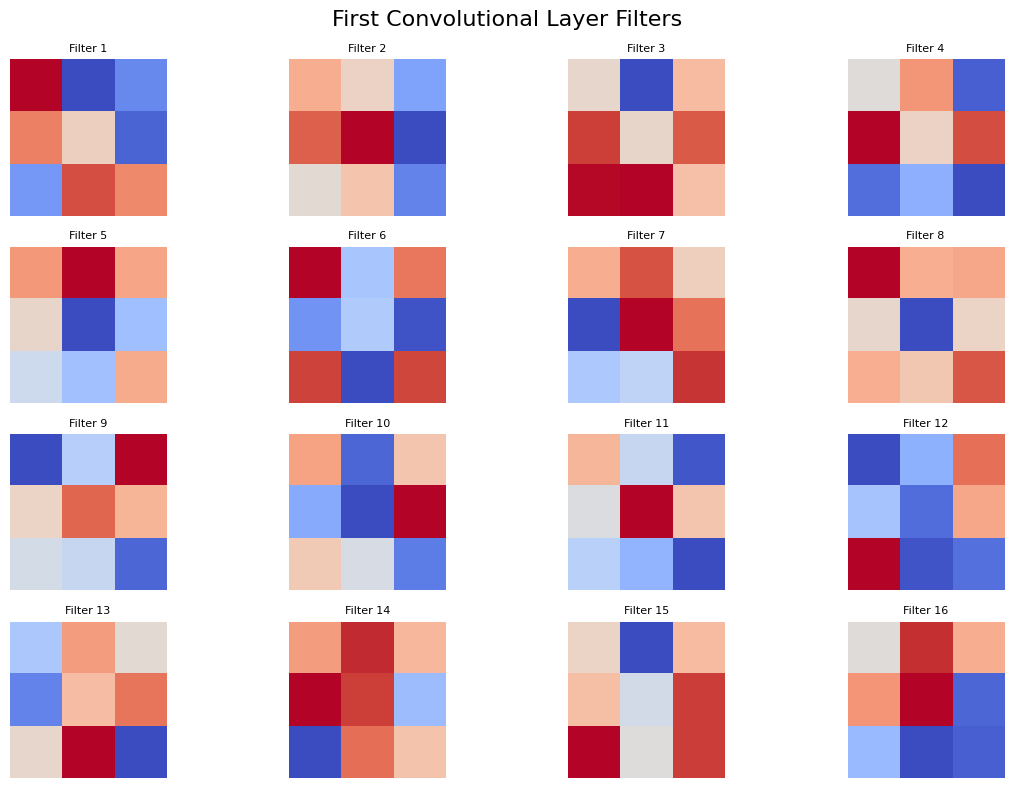
Новая лучшая модель сохранена с точностью: 93.92%

Epoch [30/30], Train Loss: 0.1946, Train Acc: 93.13%, Test Loss: 0.1754, Test Acc: 93.92%, LR: 0.000500

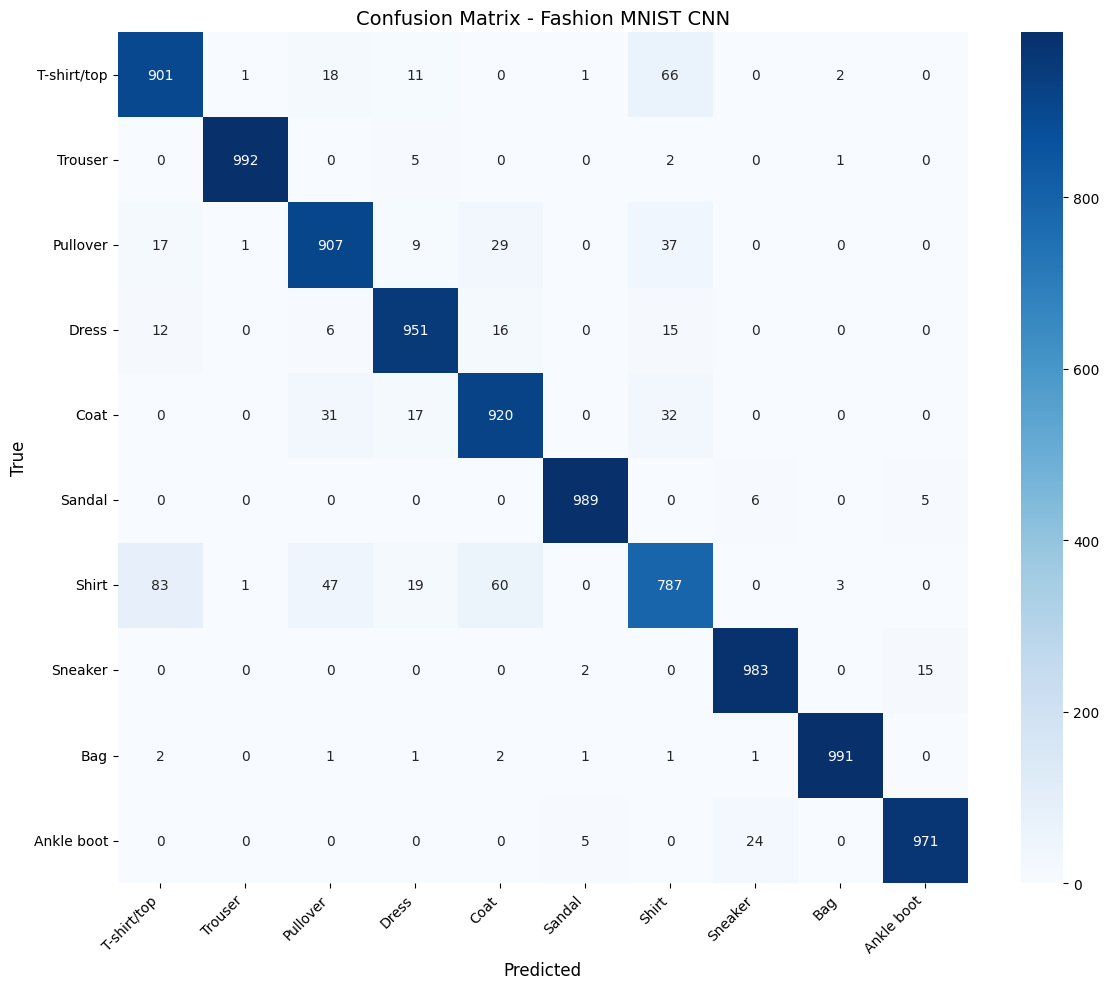
Загружена лучшая модель с точностью: 93.92%











Точность по классам:

T-shirt/top: 90.10%

Trouser: 99.20%

Pullover: 90.70%

Dress: 95.10%

Coat: 92.00%

Sandal: 98.90%

Shirt: 78.70%

Sneaker: 98.30%

Bag: 99.10%

Ankle boot: 97.10%

Final model saved as fashion\_mnist\_cnn\_final\_model.pth

============================================================

FASHION MNIST CNN MODEL SUMMARY

============================================================

Total parameters: 617,450

Final Test Accuracy: 93.92%

Best Accuracy: 93.92%

Number of epochs: 30

Final Train Loss: 0.1946

Final Test Loss: 0.1754

Generalization Gap: -0.0192

Улучшение по сравнению с полносвязной сетью: +5.92%

**2. Классификация изображений (датасет CIFAR 10)**

2.1 Решить задачу классификации, не используя сверточные слои.

* Нормализовать данные (если необходимо)
* Предложить архитектуру модели для решения задачи
* Посчитать количество параметров модели.
* Обучить модель
* Вывести график функции потерь по эпохам.
* Используя тестовое множество
* Продемонстрировать работу модели: вывести несколько изображений, указать над ними правильный класс и класс, предсказанный моделью.
* Вывести матрицу ошибок.
* Вывести значение accuracy на тестовом множестве.
* Сохранить модель

import matplotlib.pyplot as plt

import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import DataLoader, random\_split

import torch.nn.functional as F

import numpy as np

from sklearn.metrics import confusion\_matrix, accuracy\_score

import seaborn as sns

from torchvision import datasets, transforms

*# =============================================================================*

*# 2. КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ (CIFAR-10) - ПОЛНОСВЯЗНАЯ СЕТЬ*

*# =============================================================================*

*# Определение устройства для вычислений (GPU/CPU)*

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

print(f"Using device: {device}")

*# Преобразования для данных CIFAR-10 с нормализацией*

transform = transforms.Compose([

transforms.ToTensor(), *# Преобразование в тензор и нормализация в [0,1]*

*# Нормализация с использованием средних и стандартных отклонений CIFAR-10*

transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2470, 0.2435, 0.2616))

])

*# Загрузка датасета CIFAR-10*

full\_train\_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)

test\_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

*# Разделение обучающей выборки на тренировочную и валидационную (80%/20%)*

train\_size = int(0.8 \* len(full\_train\_dataset))

val\_size = len(full\_train\_dataset) - train\_size

train\_dataset, val\_dataset = random\_split(full\_train\_dataset, [train\_size, val\_size])

*# Создание DataLoader'ов для батчевой обработки*

batch\_size = 128

train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)

val\_loader = DataLoader(val\_dataset, batch\_size=batch\_size, shuffle=False)

test\_loader = DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=False)

*# Названия классов CIFAR-10*

class\_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',

'dog', 'frog', 'horse', 'ship', 'truck']

*# 2.1 Модель полносвязной нейронной сети для CIFAR-10*

class CIFAR10Model(nn.Module):

"""Полносвязная нейронная сеть для классификации CIFAR-10"""

def \_\_init\_\_(self):

super(CIFAR10Model, self).\_\_init\_\_()

*# Архитектура сети: 3072 -> 1024 -> 512 -> 256 -> 128 -> 10*

*# 3\*32\*32 = 3072 (3 канала RGB, 32x32 пикселя)*

self.fc1 = nn.Linear(3\*32\*32, 1024) *# Входной слой*

self.fc2 = nn.Linear(1024, 512) *# Скрытый слой 1*

self.fc3 = nn.Linear(512, 256) *# Скрытый слой 2*

self.fc4 = nn.Linear(256, 128) *# Скрытый слой 3*

self.fc5 = nn.Linear(128, 10) *# Выходной слой (10 классов)*

*# Регуляризация*

self.dropout = nn.Dropout(0.5) *# Dropout для предотвращения переобучения*

*# Batch Normalization для ускорения обучения и стабилизации*

self.batch\_norm1 = nn.BatchNorm1d(1024)

self.batch\_norm2 = nn.BatchNorm1d(512)

self.batch\_norm3 = nn.BatchNorm1d(256)

self.batch\_norm4 = nn.BatchNorm1d(128)

def forward(self, x):

*# Преобразование изображения 3x32x32 в вектор 3072*

x = x.view(-1, 3\*32\*32)

*# Проход через сеть с активациями ReLU и регуляризацией*

x = F.relu(self.batch\_norm1(self.fc1(x)))

x = self.dropout(x)

x = F.relu(self.batch\_norm2(self.fc2(x)))

x = self.dropout(x)

x = F.relu(self.batch\_norm3(self.fc3(x)))

x = self.dropout(x)

x = F.relu(self.batch\_norm4(self.fc4(x)))

x = self.dropout(x)

x = self.fc5(x) *# Без активации - используется CrossEntropyLoss*

return x

*# Создание модели и перемещение на устройство*

model = CIFAR10Model().to(device)

*# Функция для подсчета количества обучаемых параметров*

def count\_parameters(model):

return sum(p.numel() for p in model.parameters() if p.requires\_grad)

total\_params = count\_parameters(model)

print(f"Количество обучаемых параметров модели: {total\_params:,}")

*# Функция потерь и оптимизатор*

criterion = nn.CrossEntropyLoss() *# Для многоклассовой классификации*

optimizer = optim.Adam(model.parameters(), lr=0.001, weight\_decay=1e-4) *# L2 регуляризация*

*# Планировщик learning rate для адаптивного уменьшения при плато*

scheduler = optim.lr\_scheduler.ReduceLROnPlateau(optimizer, patience=3, factor=0.5)

*# Списки для отслеживания метрик во время обучения*

train\_losses = []

val\_losses = []

val\_accuracies = []

num\_epochs = 30

print("Начало обучения полносвязной сети на CIFAR-10...")

*# Цикл обучения*

for epoch in range(num\_epochs):

*# Режим обучения*

model.train()

running\_loss = 0.0

correct\_train = 0

total\_train = 0

*# Итерация по батчам тренировочных данных*

for images, labels in train\_loader:

images, labels = images.to(device), labels.to(device)

*# Прямой проход*

optimizer.zero\_grad() *# Обнуление градиентов*

outputs = model(images)

loss = criterion(outputs, labels)

*# Обратный проход*

loss.backward()

optimizer.step()

*# Статистика для текущего батча*

running\_loss += loss.item() \* images.size(0)

*# Подсчет точности*

\_, predicted = torch.max(outputs.data, 1)

total\_train += labels.size(0)

correct\_train += (predicted == labels).sum().item()

*# Вычисление средней потери и точности за эпоху*

epoch\_train\_loss = running\_loss / len(train\_loader.dataset)

train\_accuracy = 100 \* correct\_train / total\_train

train\_losses.append(epoch\_train\_loss)

*# Валидация*

model.eval()

val\_loss = 0.0

correct\_val = 0

total\_val = 0

with torch.no\_grad(): *# Отключение вычисления градиентов для валидации*

for images, labels in val\_loader:

images, labels = images.to(device), labels.to(device)

outputs = model(images)

loss = criterion(outputs, labels)

val\_loss += loss.item() \* images.size(0)

*# Подсчет правильных предсказаний*

\_, predicted = torch.max(outputs.data, 1)

total\_val += labels.size(0)

correct\_val += (predicted == labels).sum().item()

epoch\_val\_loss = val\_loss / len(val\_loader.dataset)

epoch\_val\_acc = 100 \* correct\_val / total\_val

val\_losses.append(epoch\_val\_loss)

val\_accuracies.append(epoch\_val\_acc)

*# Адаптация learning rate на основе валидационной потери*

scheduler.step(epoch\_val\_loss)

print(f'Epoch [{epoch+1:2d}/{num\_epochs}], '

f'Train Loss: {epoch\_train\_loss:.4f}, Train Acc: {train\_accuracy:.2f}%, '

f'Val Loss: {epoch\_val\_loss:.4f}, Val Acc: {epoch\_val\_acc:.2f}%, '

f'LR: {optimizer.param\_groups[0]["lr"]:.6f}')

*# Визуализация процесса обучения*

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

plt.subplot(1, 2, 1)

plt.plot(train\_losses, label='Training Loss', color='blue')

plt.plot(val\_losses, label='Validation Loss', color='red')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

plt.grid(True)

plt.subplot(1, 2, 2)

plt.plot(val\_accuracies, label='Validation Accuracy', color='green')

plt.xlabel('Epochs')

plt.ylabel('Accuracy (%)')

plt.title('Validation Accuracy')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.savefig('cifar10\_training\_plots.png')

plt.show()

*# Тестирование модели на тестовой выборке*

model.eval()

all\_preds = []

all\_labels = []

all\_images = []

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)

*# Сохранение предсказаний и меток для анализа*

all\_preds.extend(predicted.cpu().numpy())

all\_labels.extend(labels.cpu().numpy())

all\_images.extend(images.cpu().numpy())

*# Вычисление точности на тестовой выборке*

test\_accuracy = accuracy\_score(all\_labels, all\_preds)

print(f'Test Accuracy: {test\_accuracy \* 100:.2f}%')

*# Функция для денормализации изображений CIFAR-10*

def denormalize(img):

"""Преобразование из нормализованного вида обратно в [0,1]"""

mean = torch.tensor([0.4914, 0.4822, 0.4465]).view(3, 1, 1)

std = torch.tensor([0.2470, 0.2435, 0.2616]).view(3, 1, 1)

return img \* std + mean

def imshow(img, title):

"""Отображение изображения с заголовком"""

img = denormalize(torch.from\_numpy(img))

img = np.clip(img.numpy(), 0, 1) *# Обрезка значений до [0,1]*

plt.imshow(np.transpose(img, (1, 2, 0))) *# Преобразование из (C,H,W) в (H,W,C)*

plt.title(title, fontsize=8)

plt.axis('off')

*# Визуализация предсказаний на случайных примерах*

indices = np.random.choice(len(all\_images), 12, replace=False)

plt.figure(figsize=(15, 10))

for i, idx in enumerate(indices):

plt.subplot(3, 4, i+1)

title = f'True: {class\_names[all\_labels[idx]]}\nPred: {class\_names[all\_preds[idx]]}'

if all\_labels[idx] != all\_preds[idx]:

imshow(all\_images[idx], title)

plt.title(title, color='red', fontsize=8) *# Красный для ошибок*

else:

imshow(all\_images[idx], title)

plt.title(title, color='green', fontsize=8) *# Зеленый для правильных*

plt.tight\_layout()

plt.savefig('cifar10\_predictions.png')

plt.show()

*# Построение матрицы ошибок*

cm = confusion\_matrix(all\_labels, all\_preds)

plt.figure(figsize=(12, 10))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',

xticklabels=class\_names, yticklabels=class\_names)

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix - CIFAR-10')

plt.xticks(rotation=45, ha='right') *# Наклон подписей для лучшей читаемости*

plt.yticks(rotation=0)

plt.tight\_layout()

plt.savefig('cifar10\_confusion\_matrix.png')

plt.show()

*# Анализ точности по классам*

class\_accuracy = {}

for i in range(10):

class\_mask = np.array(all\_labels) == i

if np.sum(class\_mask) > 0:

class\_acc = np.mean(np.array(all\_preds)[class\_mask] == i)

class\_accuracy[class\_names[i]] = class\_acc \* 100

print("\nТочность по классам:")

for class\_name, acc in class\_accuracy.items():

print(f"{class\_name}: {acc:.2f}%")

*# Сохранение модели и метрик*

torch.save({

'model\_state\_dict': model.state\_dict(),

'optimizer\_state\_dict': optimizer.state\_dict(),

'train\_losses': train\_losses,

'val\_losses': val\_losses,

'val\_accuracies': val\_accuracies,

'test\_accuracy': test\_accuracy

}, 'cifar10\_model.pth')

print("Model saved as cifar10\_model.pth")

*# Итоговая статистика*

print("\n=== Model Architecture ===")

print(model)

print(f"\nTotal parameters: {total\_params:,}")

print(f"Final Test Accuracy: {test\_accuracy \* 100:.2f}%")

final\_train\_loss = train\_losses[-1]

final\_val\_loss = val\_losses[-1]

print(f"Final Train Loss: {final\_train\_loss:.4f}")

print(f"Final Val Loss: {final\_val\_loss:.4f}")

print(f"Gap: {final\_val\_loss - final\_train\_loss:.4f}") *# Разрыв между train и val loss*

Using device: cuda

100%|██████████| 170M/170M [00:03<00:00, 48.7MB/s]

Количество обучаемых параметров модели: 3,840,906

Начало обучения...

Epoch [ 1/30], Train Loss: 1.9316, Train Acc: 29.79%, Val Loss: 1.6550, Val Acc: 40.93%, LR: 0.001000

Epoch [ 2/30], Train Loss: 1.7194, Train Acc: 38.40%, Val Loss: 1.5816, Val Acc: 43.80%, LR: 0.001000

Epoch [ 3/30], Train Loss: 1.6437, Train Acc: 41.24%, Val Loss: 1.5124, Val Acc: 46.35%, LR: 0.001000

Epoch [ 4/30], Train Loss: 1.5962, Train Acc: 43.12%, Val Loss: 1.4674, Val Acc: 47.78%, LR: 0.001000

Epoch [ 5/30], Train Loss: 1.5595, Train Acc: 44.28%, Val Loss: 1.4330, Val Acc: 48.82%, LR: 0.001000

Epoch [ 6/30], Train Loss: 1.5318, Train Acc: 45.55%, Val Loss: 1.4223, Val Acc: 49.52%, LR: 0.001000

Epoch [ 7/30], Train Loss: 1.5109, Train Acc: 46.51%, Val Loss: 1.4204, Val Acc: 49.54%, LR: 0.001000

Epoch [ 8/30], Train Loss: 1.4928, Train Acc: 47.09%, Val Loss: 1.3949, Val Acc: 50.88%, LR: 0.001000

Epoch [ 9/30], Train Loss: 1.4776, Train Acc: 47.69%, Val Loss: 1.3876, Val Acc: 50.60%, LR: 0.001000

Epoch [10/30], Train Loss: 1.4668, Train Acc: 47.69%, Val Loss: 1.3664, Val Acc: 51.20%, LR: 0.001000

Epoch [11/30], Train Loss: 1.4486, Train Acc: 48.89%, Val Loss: 1.3608, Val Acc: 51.27%, LR: 0.001000

Epoch [12/30], Train Loss: 1.4436, Train Acc: 49.05%, Val Loss: 1.3526, Val Acc: 51.82%, LR: 0.001000

Epoch [13/30], Train Loss: 1.4283, Train Acc: 49.80%, Val Loss: 1.3433, Val Acc: 52.54%, LR: 0.001000

Epoch [14/30], Train Loss: 1.4256, Train Acc: 49.95%, Val Loss: 1.3505, Val Acc: 51.90%, LR: 0.001000

Epoch [15/30], Train Loss: 1.4122, Train Acc: 50.24%, Val Loss: 1.3352, Val Acc: 52.58%, LR: 0.001000

Epoch [16/30], Train Loss: 1.4020, Train Acc: 50.52%, Val Loss: 1.3251, Val Acc: 53.18%, LR: 0.001000

Epoch [17/30], Train Loss: 1.3959, Train Acc: 50.79%, Val Loss: 1.3236, Val Acc: 53.36%, LR: 0.001000

Epoch [18/30], Train Loss: 1.3920, Train Acc: 51.10%, Val Loss: 1.3123, Val Acc: 53.71%, LR: 0.001000

Epoch [19/30], Train Loss: 1.3842, Train Acc: 51.44%, Val Loss: 1.3120, Val Acc: 53.55%, LR: 0.001000

Epoch [20/30], Train Loss: 1.3783, Train Acc: 51.28%, Val Loss: 1.3172, Val Acc: 53.60%, LR: 0.001000

Epoch [21/30], Train Loss: 1.3643, Train Acc: 51.61%, Val Loss: 1.3115, Val Acc: 54.19%, LR: 0.001000

Epoch [22/30], Train Loss: 1.3618, Train Acc: 52.10%, Val Loss: 1.2944, Val Acc: 54.59%, LR: 0.001000

Epoch [23/30], Train Loss: 1.3573, Train Acc: 52.16%, Val Loss: 1.2967, Val Acc: 54.28%, LR: 0.001000

Epoch [24/30], Train Loss: 1.3554, Train Acc: 52.41%, Val Loss: 1.2954, Val Acc: 54.83%, LR: 0.001000

Epoch [25/30], Train Loss: 1.3483, Train Acc: 52.70%, Val Loss: 1.2995, Val Acc: 54.36%, LR: 0.001000

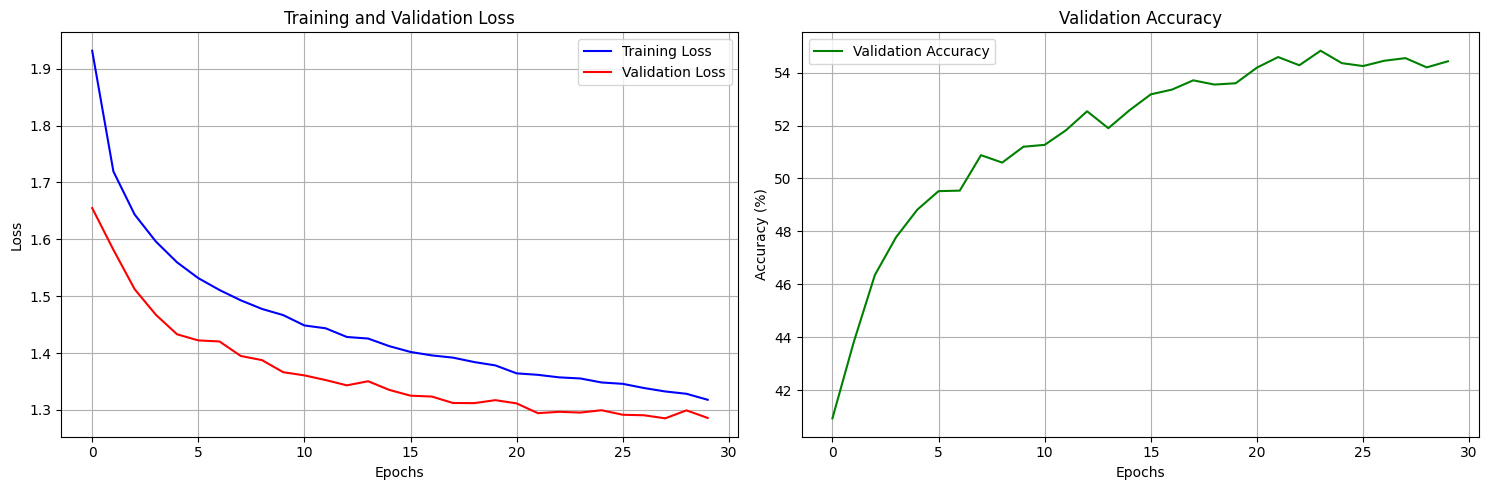
Epoch [26/30], Train Loss: 1.3459, Train Acc: 52.47%, Val Loss: 1.2916, Val Acc: 54.25%, LR: 0.001000

Epoch [27/30], Train Loss: 1.3386, Train Acc: 53.05%, Val Loss: 1.2907, Val Acc: 54.45%, LR: 0.001000

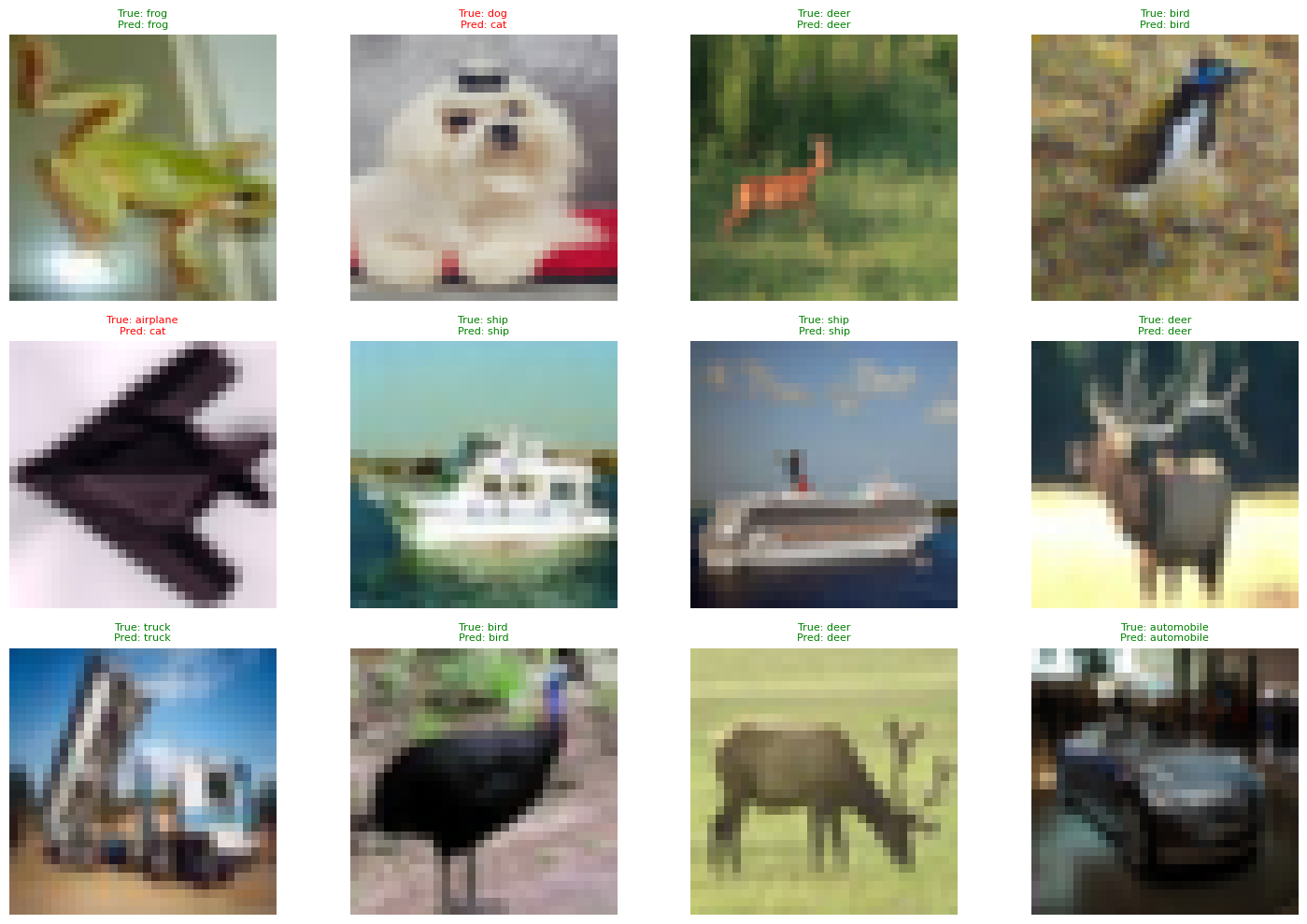
Epoch [28/30], Train Loss: 1.3325, Train Acc: 53.15%, Val Loss: 1.2853, Val Acc: 54.55%, LR: 0.001000

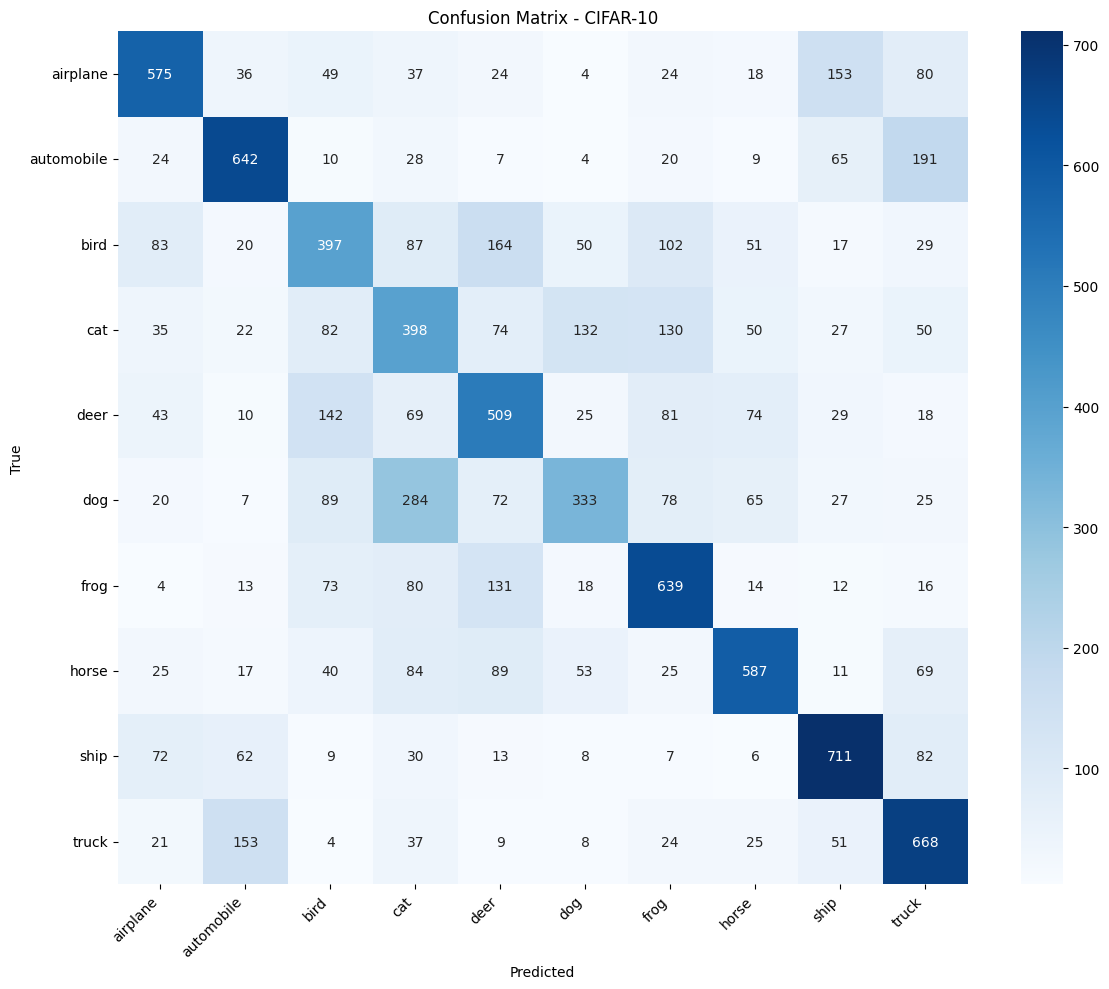
Epoch [29/30], Train Loss: 1.3284, Train Acc: 53.28%, Val Loss: 1.2993, Val Acc: 54.20%, LR: 0.001000

Epoch [30/30], Train Loss: 1.3179, Train Acc: 53.41%, Val Loss: 1.2860, Val Acc: 54.43%, LR: 0.001000



Test Accuracy: 54.59%





Точность по классам:

airplane: 57.50%

automobile: 64.20%

bird: 39.70%

cat: 39.80%

deer: 50.90%

dog: 33.30%

frog: 63.90%

horse: 58.70%

ship: 71.10%

truck: 66.80%

Model saved as cifar10\_model.pth

=== Model Architecture ===

CIFAR10Model(

(fc1): Linear(in\_features=3072, out\_features=1024, bias=True)

(fc2): Linear(in\_features=1024, out\_features=512, bias=True)

(fc3): Linear(in\_features=512, out\_features=256, bias=True)

(fc4): Linear(in\_features=256, out\_features=128, bias=True)

(fc5): Linear(in\_features=128, out\_features=10, bias=True)

(dropout): Dropout(p=0.5, inplace=False)

(batch\_norm1): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(batch\_norm2): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(batch\_norm3): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

(batch\_norm4): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

Total parameters: 3,840,906

Final Test Accuracy: 54.59%

Final Train Loss: 1.3179

Final Val Loss: 1.2860

Gap: -0.0319

2.2 Решить задачу 2.1, используя сверточную нейронную сеть.

* Добиться значения accuracy на тестовом множестве не менее 70%.
* Визуализировать результаты работы первого сверточного слоя

*# Повторная инициализация устройства*

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

print(f"Using device: {device}")

*# Аугментации данных для тренировочной выборки*

transform = transforms.Compose([

transforms.RandomHorizontalFlip(p=0.5), *# Случайное отражение по горизонтали*

transforms.RandomCrop(32, padding=4), *# Случайный кроп с padding*

transforms.ToTensor(),

transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2470, 0.2435, 0.2616))

])

*# Без аугментаций для тестовой выборки*

transform\_test = transforms.Compose([

transforms.ToTensor(),

transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2470, 0.2435, 0.2616))

])

*# Загрузка данных с аугментациями*

train\_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)

test\_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform\_test)

*# DataLoader с увеличенным размером батча и многопоточностью*

batch\_size = 128

train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True, num\_workers=2)

test\_loader = DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=False, num\_workers=2)

class\_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',

'dog', 'frog', 'horse', 'ship', 'truck']

*# Сверточная нейронная сеть для CIFAR-10*

class CIFAR10CNN(nn.Module):

"""Сверточная нейронная сеть для классификации CIFAR-10"""

def \_\_init\_\_(self):

super(CIFAR10CNN, self).\_\_init\_\_()

*# Первый блок сверток: 3 -> 64 каналов*

self.conv1 = nn.Conv2d(3, 64, kernel\_size=3, padding=1) *# Сохраняет размер 32x32*

self.bn1 = nn.BatchNorm2d(64)

self.conv2 = nn.Conv2d(64, 64, kernel\_size=3, padding=1)

self.bn2 = nn.BatchNorm2d(64)

self.pool1 = nn.MaxPool2d(2, 2) *# Уменьшение до 16x16*

self.dropout1 = nn.Dropout2d(0.25) *# Spatial Dropout*

*# Второй блок сверток: 64 -> 128 каналов*

self.conv3 = nn.Conv2d(64, 128, kernel\_size=3, padding=1)

self.bn3 = nn.BatchNorm2d(128)

self.conv4 = nn.Conv2d(128, 128, kernel\_size=3, padding=1)

self.bn4 = nn.BatchNorm2d(128)

self.pool2 = nn.MaxPool2d(2, 2) *# Уменьшение до 8x8*

self.dropout2 = nn.Dropout2d(0.25)

*# Третий блок сверток: 128 -> 256 каналов*

self.conv5 = nn.Conv2d(128, 256, kernel\_size=3, padding=1)

self.bn5 = nn.BatchNorm2d(256)

self.conv6 = nn.Conv2d(256, 256, kernel\_size=3, padding=1)

self.bn6 = nn.BatchNorm2d(256)

self.pool3 = nn.MaxPool2d(2, 2) *# Уменьшение до 4x4*

self.dropout3 = nn.Dropout2d(0.25)

*# Полносвязные слои*

*# После 3 пулингов: 32x32 -> 16x16 -> 8x8 -> 4x4*

self.fc1 = nn.Linear(256 \* 4 \* 4, 512) *# 256 каналов \* 4x4 = 4096 входов*

self.bn7 = nn.BatchNorm1d(512)

self.dropout4 = nn.Dropout(0.5)

self.fc2 = nn.Linear(512, 256)

self.bn8 = nn.BatchNorm1d(256)

self.dropout5 = nn.Dropout(0.5)

self.fc3 = nn.Linear(256, 10)

def forward(self, x):

*# Сохраняем выход первого сверточного слоя для визуализации*

conv1\_out = F.relu(self.bn1(self.conv1(x)))

*# Первый блок: две свертки + пулинг*

x = F.relu(self.bn2(self.conv2(conv1\_out)))

x = self.pool1(x)

x = self.dropout1(x)

*# Второй блок: две свертки + пулинг*

x = F.relu(self.bn3(self.conv3(x)))

x = F.relu(self.bn4(self.conv4(x)))

x = self.pool2(x)

x = self.dropout2(x)

*# Третий блок: две свертки + пулинг*

x = F.relu(self.bn5(self.conv5(x)))

x = F.relu(self.bn6(self.conv6(x)))

x = self.pool3(x)

x = self.dropout3(x)

*# Полносвязные слои*

x = x.view(-1, 256 \* 4 \* 4) *# Преобразование в вектор*

x = F.relu(self.bn7(self.fc1(x)))

x = self.dropout4(x)

x = F.relu(self.bn8(self.fc2(x)))

x = self.dropout5(x)

x = self.fc3(x)

return x, conv1\_out *# Возвращаем и выход, и feature maps первого слоя*

*# Создание CNN модели*

model = CIFAR10CNN().to(device)

*# Подсчет параметров*

total\_params = count\_parameters(model)

print(f"Количество обучаемых параметров модели: {total\_params:,}")

*# Функция потерь и оптимизатор с L2 регуляризацией*

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.001, weight\_decay=1e-4)

*# Планировщик learning rate с большим терпением*

scheduler = optim.lr\_scheduler.ReduceLROnPlateau(optimizer, patience=5, factor=0.5)

*# Списки для отслеживания метрик*

train\_losses = []

train\_accuracies = []

val\_losses = []

val\_accuracies = []

num\_epochs = 50

best\_accuracy = 0.0

print("Начало обучения CNN на CIFAR-10...")

for epoch in range(num\_epochs):

*# Тренировка*

model.train()

running\_loss = 0.0

correct\_train = 0

total\_train = 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() \* images.size(0)

\_, predicted = torch.max(outputs.data, 1)

total\_train += labels.size(0)

correct\_train += (predicted == labels).sum().item()

epoch\_train\_loss = running\_loss / len(train\_loader.dataset)

train\_accuracy = 100 \* correct\_train / total\_train

train\_losses.append(epoch\_train\_loss)

train\_accuracies.append(train\_accuracy)

*# Тестирование на тестовой выборке (используется как валидация)*

model.eval()

val\_loss = 0.0

correct\_val = 0

total\_val = 0

with torch.no\_grad():

for images, labels in test\_loader:

images, labels = images.to(device), labels.to(device)

outputs, \_ = model(images)

loss = criterion(outputs, labels)

val\_loss += loss.item() \* images.size(0)

\_, predicted = torch.max(outputs.data, 1)

total\_val += labels.size(0)

correct\_val += (predicted == labels).sum().item()

epoch\_val\_loss = val\_loss / len(test\_loader.dataset)

epoch\_val\_acc = 100 \* correct\_val / total\_val

val\_losses.append(epoch\_val\_loss)

val\_accuracies.append(epoch\_val\_acc)

*# Адаптация learning rate на основе тестовой потери*

scheduler.step(epoch\_val\_loss)

*# Сохранение лучшей модели*

if epoch\_val\_acc > best\_accuracy:

best\_accuracy = epoch\_val\_acc

torch.save(model.state\_dict(), 'best\_cifar10\_cnn\_model.pth')

print(f"Новая лучшая модель сохранена с точностью: {best\_accuracy:.2f}%")

current\_lr = optimizer.param\_groups[0]['lr']

print(f'Epoch [{epoch+1:2d}/{num\_epochs}], '

f'Train Loss: {epoch\_train\_loss:.4f}, Train Acc: {train\_accuracy:.2f}%, '

f'Test Loss: {epoch\_val\_loss:.4f}, Test Acc: {epoch\_val\_acc:.2f}%, '

f'LR: {current\_lr:.6f}')

*# Загрузка лучшей модели*

model.load\_state\_dict(torch.load('best\_cifar10\_cnn\_model.pth'))

print(f"Загружена лучшая модель с точностью: {best\_accuracy:.2f}%")

*# Визуализация процесса обучения CNN*

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

plt.subplot(1, 2, 1)

plt.plot(train\_losses, label='Training Loss', color='blue', linewidth=2)

plt.plot(val\_losses, label='Test Loss', color='red', linewidth=2)

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training and Test Loss')

plt.legend()

plt.grid(True, alpha=0.3)

plt.subplot(1, 2, 2)

plt.plot(train\_accuracies, label='Training Accuracy', color='green', linewidth=2)

plt.plot(val\_accuracies, label='Test Accuracy', color='orange', linewidth=2)

plt.xlabel('Epochs')

plt.ylabel('Accuracy (%)')

plt.title('Training and Test Accuracy')

plt.legend()

plt.grid(True, alpha=0.3)

plt.tight\_layout()

plt.savefig('cifar10\_cnn\_training\_plots.png', dpi=300, bbox\_inches='tight')

plt.show()

*# Окончательное тестирование модели*

model.eval()

all\_preds = []

all\_labels = []

all\_images = []

all\_conv1\_features = [] *# Для визуализации feature maps*

with torch.no\_grad():

for images, labels in test\_loader:

images, labels = images.to(device), labels.to(device)

outputs, conv1\_out = model(images) *# Получаем и выходы, и feature maps*

\_, predicted = torch.max(outputs.data, 1)

all\_preds.extend(predicted.cpu().numpy())

all\_labels.extend(labels.cpu().numpy())

all\_images.extend(images.cpu().numpy())

all\_conv1\_features.extend(conv1\_out.cpu().numpy())

test\_accuracy = accuracy\_score(all\_labels, all\_preds)

print(f'Final Test Accuracy: {test\_accuracy \* 100:.2f}%')

*# Функция для визуализации feature maps первого сверточного слоя*

def visualize\_conv\_features(images, conv\_features, num\_images=3, num\_filters=16):

"""

Визуализация feature maps первого сверточного слоя

images: оригинальные изображения

conv\_features: feature maps из conv1 слоя

num\_images: количество изображений для показа

num\_filters: количество фильтров для визуализации

"""

fig, axes = plt.subplots(num\_images, num\_filters + 1, figsize=(20, num\_images \* 2))

for i in range(num\_images):

*# Оригинальное изображение*

orig\_img = denormalize(torch.from\_numpy(images[i]))

orig\_img = np.clip(orig\_img.numpy(), 0, 1)

axes[i, 0].imshow(np.transpose(orig\_img, (1, 2, 0)))

axes[i, 0].set\_title('Original', fontsize=8)

axes[i, 0].axis('off')

*# Feature maps для каждого фильтра*

for j in range(num\_filters):

if j < conv\_features[i].shape[0]:

feature\_map = conv\_features[i][j]

*# Нормализация feature map для лучшей визуализации*

feature\_map = (feature\_map - feature\_map.min()) / (feature\_map.max() - feature\_map.min() + 1e-8)

axes[i, j+1].imshow(feature\_map, cmap='viridis')

axes[i, j+1].set\_title(f'F{j+1}', fontsize=6)

axes[i, j+1].axis('off')

plt.suptitle('First Convolutional Layer Feature Maps', fontsize=16, y=0.95)

plt.tight\_layout()

plt.savefig('conv1\_features\_visualization.png', dpi=300, bbox\_inches='tight')

plt.show()

print("Визуализация feature maps первого сверточного слоя...")

visualize\_conv\_features(all\_images[:3], all\_conv1\_features[:3], num\_images=3, num\_filters=16)

*# Визуализация предсказаний CNN*

indices = np.random.choice(len(all\_images), 12, replace=False)

plt.figure(figsize=(15, 10))

for i, idx in enumerate(indices):

plt.subplot(3, 4, i+1)

title = f'True: {class\_names[all\_labels[idx]]}\nPred: {class\_names[all\_preds[idx]]}'

if all\_labels[idx] != all\_preds[idx]:

imshow(all\_images[idx], title)

plt.title(title, color='red', fontsize=8) *# Красный для ошибок*

else:

imshow(all\_images[idx], title)

plt.title(title, color='green', fontsize=8) *# Зеленый для правильных*

plt.suptitle('CNN Predictions on CIFAR-10 Test Set', fontsize=16)

plt.tight\_layout()

plt.savefig('cifar10\_cnn\_predictions.png', dpi=300, bbox\_inches='tight')

plt.show()

*# Матрица ошибок для CNN*

cm = confusion\_matrix(all\_labels, all\_preds)

plt.figure(figsize=(12, 10))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',

xticklabels=class\_names, yticklabels=class\_names)

plt.xlabel('Predicted', fontsize=12)

plt.ylabel('True', fontsize=12)

plt.title('Confusion Matrix - CIFAR-10 CNN', fontsize=14)

plt.xticks(rotation=45, ha='right')

plt.yticks(rotation=0)

plt.tight\_layout()

plt.savefig('cifar10\_cnn\_confusion\_matrix.png', dpi=300, bbox\_inches='tight')

plt.show()

*# Анализ точности по классам для CNN*

class\_accuracy = {}

for i in range(10):

class\_mask = np.array(all\_labels) == i

if np.sum(class\_mask) > 0:

class\_acc = np.mean(np.array(all\_preds)[class\_mask] == i)

class\_accuracy[class\_names[i]] = class\_acc \* 100

print("\nТочность по классам:")

for class\_name, acc in class\_accuracy.items():

print(f"{class\_name}: {acc:.2f}%")

*# Сохранение полной информации о CNN модели*

torch.save({

'model\_state\_dict': model.state\_dict(),

'optimizer\_state\_dict': optimizer.state\_dict(),

'train\_losses': train\_losses,

'train\_accuracies': train\_accuracies,

'test\_losses': val\_losses,

'test\_accuracies': val\_accuracies,

'test\_accuracy': test\_accuracy,

'class\_accuracy': class\_accuracy

}, 'cifar10\_cnn\_final\_model.pth')

print("Final model saved as cifar10\_cnn\_final\_model.pth")

*# Итоговая статистика CNN модели*

print("\n" + "="\*50)

print("CNN MODEL SUMMARY")

print("="\*50)

print(f"Total parameters: {total\_params:,}")

print(f"Final Test Accuracy: {test\_accuracy \* 100:.2f}%")

print(f"Best Accuracy: {best\_accuracy:.2f}%")

print(f"Number of epochs: {num\_epochs}")

final\_train\_loss = train\_losses[-1]

final\_test\_loss = val\_losses[-1]

print(f"Final Train Loss: {final\_train\_loss:.4f}")

print(f"Final Test Loss: {final\_test\_loss:.4f}")

print(f"Generalization Gap: {final\_test\_loss - final\_train\_loss:.4f}")

Using device: cuda

Количество обучаемых параметров модели: 3,380,298

Начало обучения CNN...

Новая лучшая модель сохранена с точностью: 55.65%

Epoch [ 1/50], Train Loss: 1.6630, Train Acc: 38.50%, Test Loss: 1.2232, Test Acc: 55.65%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 65.47%

Epoch [ 2/50], Train Loss: 1.2554, Train Acc: 54.77%, Test Loss: 0.9487, Test Acc: 65.47%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 71.92%

Epoch [ 3/50], Train Loss: 1.0680, Train Acc: 62.45%, Test Loss: 0.7934, Test Acc: 71.92%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 75.37%

Epoch [ 4/50], Train Loss: 0.9585, Train Acc: 66.47%, Test Loss: 0.7038, Test Acc: 75.37%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 76.71%

Epoch [ 5/50], Train Loss: 0.8699, Train Acc: 70.09%, Test Loss: 0.6707, Test Acc: 76.71%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 79.40%

Epoch [ 6/50], Train Loss: 0.8182, Train Acc: 71.96%, Test Loss: 0.5944, Test Acc: 79.40%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 81.02%

Epoch [ 7/50], Train Loss: 0.7662, Train Acc: 73.77%, Test Loss: 0.5534, Test Acc: 81.02%, LR: 0.001000

Epoch [ 8/50], Train Loss: 0.7272, Train Acc: 75.22%, Test Loss: 0.5574, Test Acc: 81.00%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 81.72%

Epoch [ 9/50], Train Loss: 0.6969, Train Acc: 76.40%, Test Loss: 0.5388, Test Acc: 81.72%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 83.60%

Epoch [10/50], Train Loss: 0.6646, Train Acc: 77.55%, Test Loss: 0.4856, Test Acc: 83.60%, LR: 0.001000

Epoch [11/50], Train Loss: 0.6390, Train Acc: 78.45%, Test Loss: 0.4869, Test Acc: 83.17%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 84.94%

Epoch [12/50], Train Loss: 0.6179, Train Acc: 79.26%, Test Loss: 0.4546, Test Acc: 84.94%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 84.95%

Epoch [13/50], Train Loss: 0.6017, Train Acc: 79.94%, Test Loss: 0.4439, Test Acc: 84.95%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 85.12%

Epoch [14/50], Train Loss: 0.5849, Train Acc: 80.17%, Test Loss: 0.4354, Test Acc: 85.12%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 85.76%

Epoch [15/50], Train Loss: 0.5617, Train Acc: 80.94%, Test Loss: 0.4278, Test Acc: 85.76%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 85.85%

Epoch [16/50], Train Loss: 0.5462, Train Acc: 81.56%, Test Loss: 0.4197, Test Acc: 85.85%, LR: 0.001000

Epoch [17/50], Train Loss: 0.5317, Train Acc: 82.03%, Test Loss: 0.4281, Test Acc: 85.40%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 86.75%

Epoch [18/50], Train Loss: 0.5178, Train Acc: 82.60%, Test Loss: 0.3960, Test Acc: 86.75%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 86.82%

Epoch [19/50], Train Loss: 0.5101, Train Acc: 82.93%, Test Loss: 0.3904, Test Acc: 86.82%, LR: 0.001000

Epoch [20/50], Train Loss: 0.5038, Train Acc: 83.32%, Test Loss: 0.4217, Test Acc: 85.80%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 86.91%

Epoch [21/50], Train Loss: 0.4933, Train Acc: 83.49%, Test Loss: 0.3962, Test Acc: 86.91%, LR: 0.001000

Epoch [22/50], Train Loss: 0.4822, Train Acc: 83.72%, Test Loss: 0.3947, Test Acc: 86.82%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 88.10%

Epoch [23/50], Train Loss: 0.4769, Train Acc: 84.02%, Test Loss: 0.3550, Test Acc: 88.10%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 88.15%

Epoch [24/50], Train Loss: 0.4650, Train Acc: 84.63%, Test Loss: 0.3503, Test Acc: 88.15%, LR: 0.001000

Epoch [25/50], Train Loss: 0.4567, Train Acc: 84.85%, Test Loss: 0.3663, Test Acc: 87.87%, LR: 0.001000

Epoch [26/50], Train Loss: 0.4497, Train Acc: 84.95%, Test Loss: 0.3575, Test Acc: 87.92%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 88.25%

Epoch [27/50], Train Loss: 0.4470, Train Acc: 84.97%, Test Loss: 0.3440, Test Acc: 88.25%, LR: 0.001000

Epoch [28/50], Train Loss: 0.4421, Train Acc: 85.27%, Test Loss: 0.3602, Test Acc: 87.91%, LR: 0.001000

Epoch [29/50], Train Loss: 0.4384, Train Acc: 85.33%, Test Loss: 0.3578, Test Acc: 87.94%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 88.31%

Epoch [30/50], Train Loss: 0.4296, Train Acc: 85.43%, Test Loss: 0.3513, Test Acc: 88.31%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 88.67%

Epoch [31/50], Train Loss: 0.4269, Train Acc: 85.79%, Test Loss: 0.3477, Test Acc: 88.67%, LR: 0.001000

Epoch [32/50], Train Loss: 0.4175, Train Acc: 86.03%, Test Loss: 0.3487, Test Acc: 88.30%, LR: 0.001000

Epoch [33/50], Train Loss: 0.4162, Train Acc: 86.27%, Test Loss: 0.3373, Test Acc: 88.52%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 88.80%

Epoch [34/50], Train Loss: 0.4137, Train Acc: 86.10%, Test Loss: 0.3351, Test Acc: 88.80%, LR: 0.001000

Epoch [35/50], Train Loss: 0.4102, Train Acc: 86.29%, Test Loss: 0.3381, Test Acc: 88.54%, LR: 0.001000

Epoch [36/50], Train Loss: 0.4080, Train Acc: 86.34%, Test Loss: 0.3400, Test Acc: 88.47%, LR: 0.001000

Epoch [37/50], Train Loss: 0.4028, Train Acc: 86.42%, Test Loss: 0.3421, Test Acc: 88.78%, LR: 0.001000

Epoch [38/50], Train Loss: 0.3992, Train Acc: 86.75%, Test Loss: 0.3585, Test Acc: 88.23%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 89.33%

Epoch [39/50], Train Loss: 0.4006, Train Acc: 86.67%, Test Loss: 0.3209, Test Acc: 89.33%, LR: 0.001000

Epoch [40/50], Train Loss: 0.3972, Train Acc: 86.75%, Test Loss: 0.3255, Test Acc: 89.01%, LR: 0.001000

Epoch [41/50], Train Loss: 0.3916, Train Acc: 86.96%, Test Loss: 0.3223, Test Acc: 89.25%, LR: 0.001000

Epoch [42/50], Train Loss: 0.3884, Train Acc: 86.91%, Test Loss: 0.3283, Test Acc: 89.09%, LR: 0.001000

Epoch [43/50], Train Loss: 0.3920, Train Acc: 86.87%, Test Loss: 0.3227, Test Acc: 89.22%, LR: 0.001000

Новая лучшая модель сохранена с точностью: 89.54%

Epoch [44/50], Train Loss: 0.3831, Train Acc: 87.41%, Test Loss: 0.3232, Test Acc: 89.54%, LR: 0.001000

Epoch [45/50], Train Loss: 0.3811, Train Acc: 87.31%, Test Loss: 0.3262, Test Acc: 88.87%, LR: 0.000500

Новая лучшая модель сохранена с точностью: 90.51%

Epoch [46/50], Train Loss: 0.3357, Train Acc: 88.92%, Test Loss: 0.2859, Test Acc: 90.51%, LR: 0.000500

Новая лучшая модель сохранена с точностью: 90.77%

Epoch [47/50], Train Loss: 0.3109, Train Acc: 89.72%, Test Loss: 0.2864, Test Acc: 90.77%, LR: 0.000500

Новая лучшая модель сохранена с точностью: 90.80%

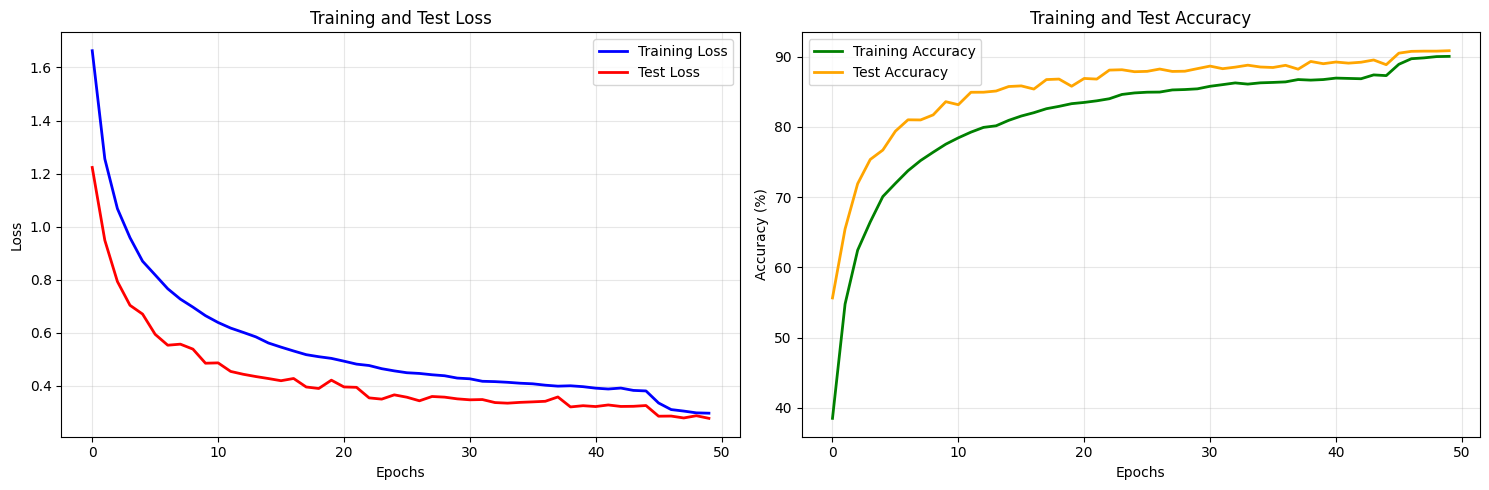
Epoch [48/50], Train Loss: 0.3053, Train Acc: 89.84%, Test Loss: 0.2793, Test Acc: 90.80%, LR: 0.000500

Epoch [49/50], Train Loss: 0.2986, Train Acc: 90.02%, Test Loss: 0.2877, Test Acc: 90.80%, LR: 0.000500

Новая лучшая модель сохранена с точностью: 90.86%

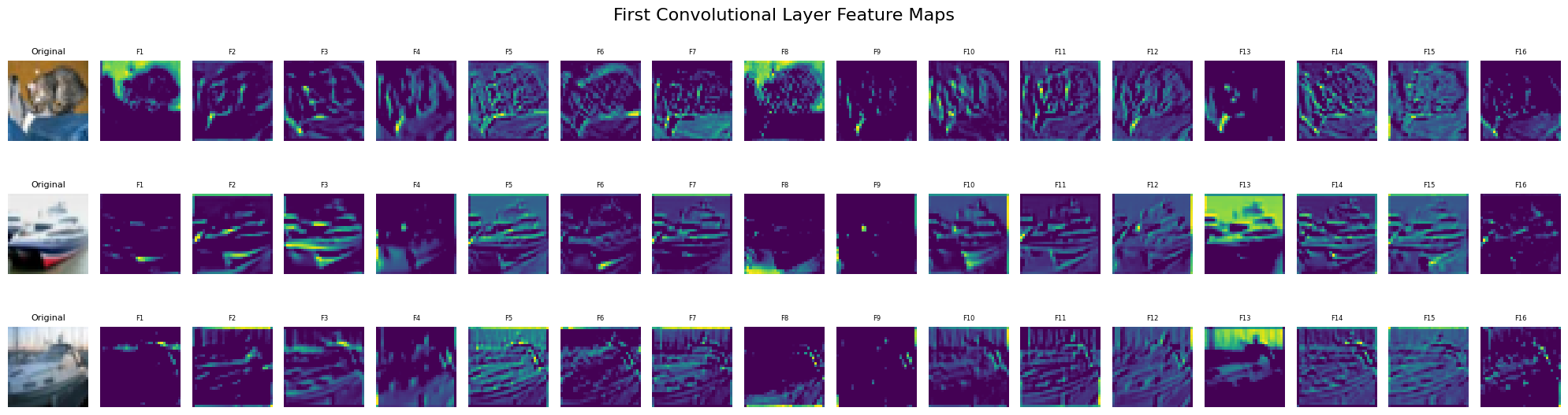
Epoch [50/50], Train Loss: 0.2973, Train Acc: 90.06%, Test Loss: 0.2779, Test Acc: 90.86%, LR: 0.000500

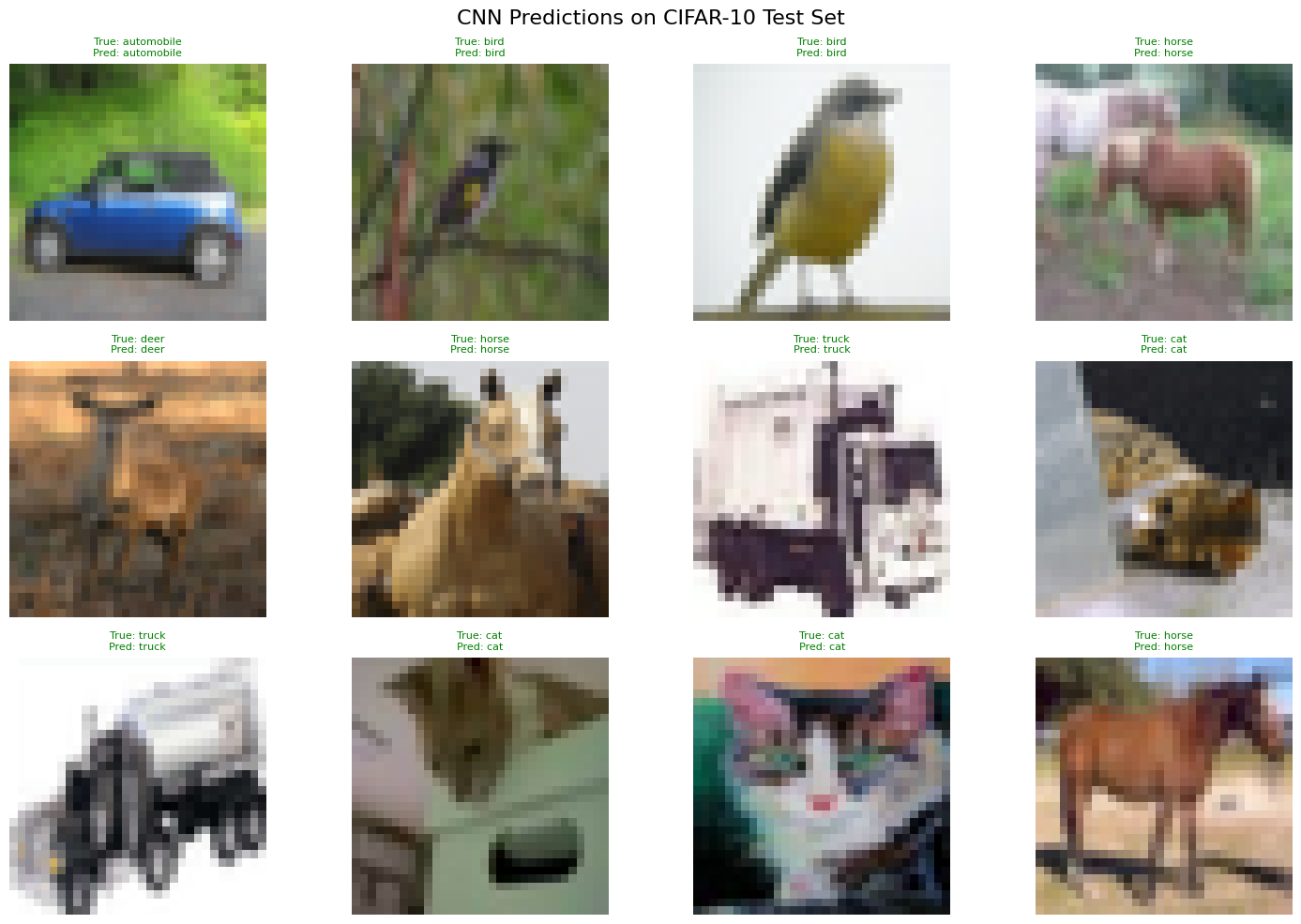
Загружена лучшая модель с точностью: 90.86%

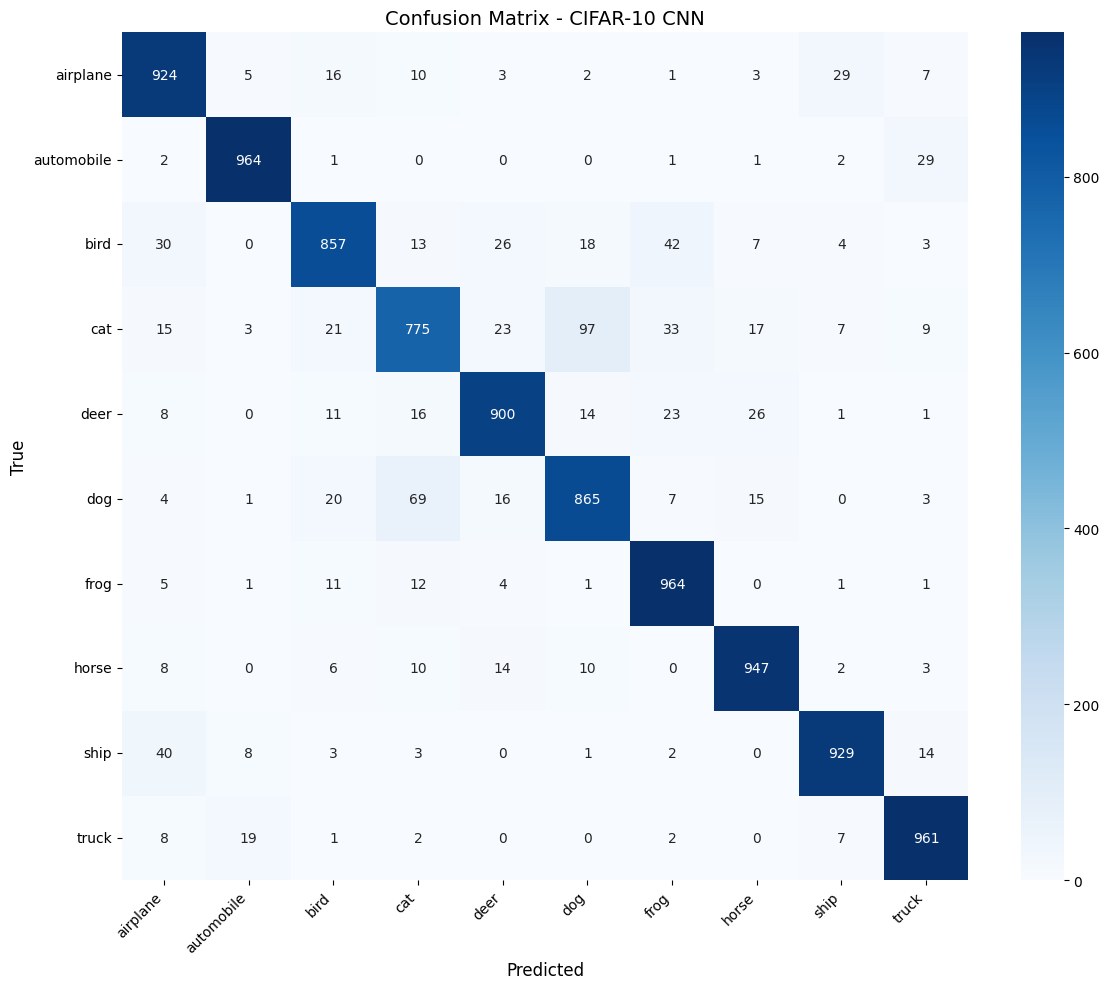


Final Test Accuracy: 90.86%

Визуализация feature maps первого сверточного слоя...







Точность по классам:

airplane: 92.40%

automobile: 96.40%

bird: 85.70%

cat: 77.50%

deer: 90.00%

dog: 86.50%

frog: 96.40%

horse: 94.70%

ship: 92.90%

truck: 96.10%

Final model saved as cifar10\_cnn\_final\_model.pth

==================================================

CNN MODEL SUMMARY

==================================================

Total parameters: 3,380,298

Final Test Accuracy: 90.86%

Best Accuracy: 90.86%

Number of epochs: 50

Final Train Loss: 0.2973

Final Test Loss: 0.2779

Generalization Gap: -0.0194

**3. Загрузка изображений из внешних источников**

3.1 Решить задачу классификации обезьян (датасет monkey.zip).

* Загрузить архив с данными на диск
* Создать датасет на основе файлов при помощи torchvision.datasets.ImageFolder
* Преобразовать изображения к тензорами одного размера (например, 400х400). Потестировать другие преобразования из torchvision.transforms
* Предложить архитектуру модели для решения задачи. Обучить модель.
* Используя тестовое множество
* Продемонстрировать работу модели: вывести несколько изображений, указать над ними правильный класс и класс, предсказанный моделью.
* Вывести матрицу ошибок.
* Вывести значение accuracy на тестовом множестве.
* Добиться значения accuracy на тестовом множестве не менее 60%

*# =============================================================================*

*# 3. ЗАГРУЗКА ИЗОБРАЖЕНИЙ ИЗ ВНЕШНИХ ИСТОЧНИКОВ*

*# =============================================================================*

*# Импорт необходимых библиотек*

from google.colab import drive

drive.mount('/content/drive')

import seaborn as sns

import os

from tqdm import tqdm *# Для прогресс-баров*

import zipfile

from torchvision import datasets, transforms

from torch.utils.data import DataLoader, random\_split

import torch

import torch.nn as nn

import torch.optim as optim

import torch.nn.functional as F

from sklearn.metrics import confusion\_matrix, accuracy\_score

import numpy as np

import matplotlib.pyplot as plt

*# =============================================================================*

*# 3.1 КЛАССИФИКАЦИЯ ОБЕЗЬЯН*

*# =============================================================================*

print("Распаковка датасета обезьян...")

*# Распаковка архива с датасетом обезьян*

zf = zipfile.ZipFile('drive/MyDrive/datasets/monkeys.zip')

for file in tqdm(zf.infolist()):

zf.extract(file)

*# Пути к данным после распаковки*

dataset\_path = './training/training'

val\_path = './validation/validation'

*# Вывод структуры датасета для понимания организации данных*

print("\nСтруктура датасета:")

for root, dirs, files in os.walk(dataset\_path):

level = root.replace(dataset\_path, '').count(os.sep)

indent = ' ' \* 2 \* level

print(f'{indent}{os.path.basename(root)}/')

subindent = ' ' \* 2 \* (level + 1)

for file in files[:3]: *# Показываем только первые 3 файла из каждой папки*

print(f'{subindent}{file}')

if len(files) > 3:

print(f'{subindent}... и еще {len(files) - 3} файлов')

*# Преобразования для тренировочных данных с аугментацией*

transform = transforms.Compose([

transforms.Resize((400, 400)), *# Изменение размера всех изображений до 400x400*

transforms.RandomHorizontalFlip(p=0.5), *# Случайное отражение по горизонтали*

transforms.RandomRotation(15), *# Случайный поворот на ±15 градусов*

transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2), *# Изменение цветов*

transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)), *# Случайный сдвиг*

transforms.ToTensor(), *# Преобразование в тензор*

*# Нормализация с использованием статистики ImageNet*

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

])

*# Преобразования для тестовых данных (без аугментации)*

transform\_test = transforms.Compose([

transforms.Resize((400, 400)),

transforms.ToTensor(),

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

])

try:

*# Загрузка датасета с использованием ImageFolder (автоматически определяет классы по папкам)*

full\_dataset = datasets.ImageFolder(root=dataset\_path, transform=transform)

print(f"Найдено классов: {len(full\_dataset.classes)}")

print(f"Классы: {full\_dataset.classes}")

print(f"Общее количество изображений: {len(full\_dataset)}")

*# Подсчет количества изображений в каждом классе*

class\_counts = {}

for class\_name in full\_dataset.classes:

class\_path = os.path.join(dataset\_path, class\_name)

if os.path.exists(class\_path):

class\_counts[class\_name] = len(os.listdir(class\_path))

print("\nКоличество изображений по классам:")

for class\_name, count in class\_counts.items():

print(f"{class\_name}: {count}")

except Exception as e:

print(f"Ошибка при загрузке датасета: {e}")

*# Поиск альтернативных путей если основной не сработал*

possible\_paths = []

for root, dirs, files in os.walk('.'):

if 'training' in dirs or 'validation' in dirs:

possible\_paths.append(root)

if possible\_paths:

print(f"Возможные пути: {possible\_paths}")

dataset\_path = possible\_paths[0]

full\_dataset = datasets.ImageFolder(root=dataset\_path, transform=transform)

else:

raise Exception("Не удалось найти датасет")

*# Разделение данных на тренировочную и тестовую выборки*

train\_size = int(0.8 \* len(full\_dataset))

test\_size = len(full\_dataset) - train\_size

train\_dataset, test\_dataset = random\_split(full\_dataset, [train\_size, test\_size])

*# Создание DataLoader'ов для батчевой обработки*

batch\_size = 16 *# Меньший размер батча из-за больших изображений*

train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True, num\_workers=2)

test\_loader = DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=False, num\_workers=2)

print(f"\nРазмер тренировочной выборки: {len(train\_dataset)}")

print(f"Размер тестовой выборки: {len(test\_dataset)}")

*# Определение сверточной нейронной сети для классификации обезьян*

class MonkeyCNN(nn.Module):

def \_\_init\_\_(self, num\_classes):

super(MonkeyCNN, self).\_\_init\_\_()

*# Блок feature extraction (извлечение признаков)*

self.features = nn.Sequential(

*# Первый блок сверток: 3 канала -> 32 канала*

nn.Conv2d(3, 32, kernel\_size=3, padding=1),

nn.BatchNorm2d(32), *# Нормализация батча для стабилизации обучения*

nn.ReLU(inplace=True),

nn.Conv2d(32, 32, kernel\_size=3, padding=1),

nn.BatchNorm2d(32),

nn.ReLU(inplace=True),

nn.MaxPool2d(2, 2), *# Уменьшение размерности в 2 раза*

nn.Dropout2d(0.25), *# Spatial dropout для регуляризации*

*# Второй блок сверток: 32 -> 64 канала*

nn.Conv2d(32, 64, kernel\_size=3, padding=1),

nn.BatchNorm2d(64),

nn.ReLU(inplace=True),

nn.Conv2d(64, 64, kernel\_size=3, padding=1),

nn.BatchNorm2d(64),

nn.ReLU(inplace=True),

nn.MaxPool2d(2, 2),

nn.Dropout2d(0.25),

*# Третий блок сверток: 64 -> 128 каналов*

nn.Conv2d(64, 128, kernel\_size=3, padding=1),

nn.BatchNorm2d(128),

nn.ReLU(inplace=True),

nn.Conv2d(128, 128, kernel\_size=3, padding=1),

nn.BatchNorm2d(128),

nn.ReLU(inplace=True),

nn.MaxPool2d(2, 2),

nn.Dropout2d(0.25),

*# Четвертый блок сверток: 128 -> 256 каналов*

nn.Conv2d(128, 256, kernel\_size=3, padding=1),

nn.BatchNorm2d(256),

nn.ReLU(inplace=True),

nn.Conv2d(256, 256, kernel\_size=3, padding=1),

nn.BatchNorm2d(256),

nn.ReLU(inplace=True),

nn.MaxPool2d(2, 2),

nn.Dropout2d(0.25),

)

*# Классификатор (полносвязные слои)*

self.classifier = nn.Sequential(

*# После 4 пулингов: 400x400 -> 200x200 -> 100x100 -> 50x50 -> 25x25*

nn.Linear(256 \* 25 \* 25, 512), *# 256 каналов \* 25x25 = 160000 входов*

nn.BatchNorm1d(512),

nn.ReLU(inplace=True),

nn.Dropout(0.5), *# Dropout для предотвращения переобучения*

nn.Linear(512, 256),

nn.BatchNorm1d(256),

nn.ReLU(inplace=True),

nn.Dropout(0.5),

nn.Linear(256, num\_classes) *# Выходной слой*

)

def forward(self, x):

x = self.features(x) *# Применяем сверточные слои*

x = x.view(x.size(0), -1) *# Преобразуем в вектор (batch\_size, features)*

x = self.classifier(x) *# Применяем полносвязные слои*

return x

*# Инициализация устройства и модели*

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

num\_classes = len(full\_dataset.classes)

model = MonkeyCNN(num\_classes).to(device)

*# Функция для подсчета параметров модели*

def count\_parameters(model):

return sum(p.numel() for p in model.parameters() if p.requires\_grad)

total\_params = count\_parameters(model)

print(f"Количество параметров модели: {total\_params:,}")

*# Настройка функции потерь и оптимизатора*

criterion = nn.CrossEntropyLoss() *# Для многоклассовой классификации*

optimizer = optim.Adam(model.parameters(), lr=0.0001, weight\_decay=1e-4) *# L2 регуляризация*

scheduler = optim.lr\_scheduler.ReduceLROnPlateau(optimizer, patience=5, factor=0.5) *# Адаптивный LR*

*# Списки для отслеживания метрик обучения*

train\_losses = []

train\_accuracies = []

test\_losses = []

test\_accuracies = []

num\_epochs = 30

best\_accuracy = 0.0

print("\nНачало обучения модели классификации обезьян...")

*# Цикл обучения*

for epoch in range(num\_epochs):

model.train() *# Режим обучения*

running\_loss = 0.0

correct\_train = 0

total\_train = 0

*# Итерация по тренировочным данным с прогресс-баром*

for images, labels in tqdm(train\_loader, desc=f'Epoch {epoch+1}/{num\_epochs}'):

images, labels = images.to(device), labels.to(device)

*# Прямой проход*

optimizer.zero\_grad() *# Обнуление градиентов*

outputs = model(images)

loss = criterion(outputs, labels)

*# Обратный проход*

loss.backward()

optimizer.step()

*# Статистика для текущего батча*

running\_loss += loss.item() \* images.size(0)

\_, predicted = torch.max(outputs.data, 1)

total\_train += labels.size(0)

correct\_train += (predicted == labels).sum().item()

*# Вычисление средних метрик за эпоху*

epoch\_train\_loss = running\_loss / len(train\_loader.dataset)

train\_accuracy = 100 \* correct\_train / total\_train

train\_losses.append(epoch\_train\_loss)

train\_accuracies.append(train\_accuracy)

*# Валидация на тестовой выборке*

model.eval() *# Режим оценки*

test\_loss = 0.0

correct\_test = 0

total\_test = 0

with torch.no\_grad(): *# Отключение вычисления градиентов для экономии памяти*

for images, labels in test\_loader:

images, labels = images.to(device), labels.to(device)

outputs = model(images)

loss = criterion(outputs, labels)

test\_loss += loss.item() \* images.size(0)

\_, predicted = torch.max(outputs.data, 1)

total\_test += labels.size(0)

correct\_test += (predicted == labels).sum().item()

epoch\_test\_loss = test\_loss / len(test\_loader.dataset)

epoch\_test\_acc = 100 \* correct\_test / total\_test

test\_losses.append(epoch\_test\_loss)

test\_accuracies.append(epoch\_test\_acc)

*# Адаптация learning rate*

scheduler.step(epoch\_test\_loss)

*# Сохранение лучшей модели*

if epoch\_test\_acc > best\_accuracy:

best\_accuracy = epoch\_test\_acc

torch.save(model.state\_dict(), 'best\_monkey\_model.pth')

current\_lr = optimizer.param\_groups[0]['lr']

print(f'Epoch [{epoch+1:2d}/{num\_epochs}], '

f'Train Loss: {epoch\_train\_loss:.4f}, Train Acc: {train\_accuracy:.2f}%, '

f'Test Loss: {epoch\_test\_loss:.4f}, Test Acc: {epoch\_test\_acc:.2f}%, '

f'LR: {current\_lr:.6f}')

*# Загрузка лучшей модели для финального тестирования*

model.load\_state\_dict(torch.load('best\_monkey\_model.pth'))

model.eval()

*# Сбор предсказаний для анализа*

all\_preds = []

all\_labels = []

all\_images = []

all\_probs = [] *# Для хранения вероятностей предсказаний*

with torch.no\_grad():

for images, labels in test\_loader:

images, labels = images.to(device), labels.to(device)

outputs = model(images)

probs = F.softmax(outputs, dim=1) *# Преобразование в вероятности*

\_, predicted = torch.max(outputs.data, 1)

all\_preds.extend(predicted.cpu().numpy())

all\_labels.extend(labels.cpu().numpy())

all\_images.extend(images.cpu().numpy())

all\_probs.extend(probs.cpu().numpy())

*# Вычисление финальной точности*

test\_accuracy = accuracy\_score(all\_labels, all\_preds)

print(f"\nFinal Test Accuracy: {test\_accuracy \* 100:.2f}%")

print(f"Best Accuracy: {best\_accuracy:.2f}%")

*# Функция для денормализации изображений (возврат к оригинальным значениям пикселей)*

def denormalize(tensor):

mean = torch.tensor([0.485, 0.456, 0.406]).view(3, 1, 1)

std = torch.tensor([0.229, 0.224, 0.225]).view(3, 1, 1)

return tensor \* std + mean

def imshow(img, title):

"""Функция для отображения изображения с заголовком"""

img = denormalize(torch.from\_numpy(img))

img = np.clip(img.numpy(), 0, 1) *# Обрезка значений до [0,1]*

img = np.transpose(img, (1, 2, 0)) *# Из (C,H,W) в (H,W,C)*

plt.imshow(img)

plt.title(title, fontsize=8)

plt.axis('off')

*# Визуализация случайных примеров с предсказаниями*

indices = np.random.choice(len(all\_images), min(12, len(all\_images)), replace=False)

class\_names = full\_dataset.classes

plt.figure(figsize=(15, 10))

for i, idx in enumerate(indices):

plt.subplot(3, 4, i+1)

true\_label = class\_names[all\_labels[idx]]

pred\_label = class\_names[all\_preds[idx]]

confidence = all\_probs[idx][all\_preds[idx]] *# Уверенность модели в предсказании*

title = f'True: {true\_label}\nPred: {pred\_label}\nConf: {confidence:.2f}'

if all\_labels[idx] != all\_preds[idx]:

imshow(all\_images[idx], title)

plt.title(title, color='red', fontsize=8) *# Красный для ошибок*

else:

imshow(all\_images[idx], title)

plt.title(title, color='green', fontsize=8) *# Зеленый для правильных*

plt.suptitle('Monkey Classification Results', fontsize=16)

plt.tight\_layout()

plt.savefig('monkey\_predictions.png', dpi=300, bbox\_inches='tight')

plt.show()

*# Построение матрицы ошибок*

cm = confusion\_matrix(all\_labels, all\_preds)

plt.figure(figsize=(10, 8))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',

xticklabels=class\_names, yticklabels=class\_names)

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix - Monkey Classification')

plt.xticks(rotation=45, ha='right') *# Наклон подписей для лучшей читаемости*

plt.yticks(rotation=0)

plt.tight\_layout()

plt.savefig('monkey\_confusion\_matrix.png', dpi=300, bbox\_inches='tight')

plt.show()

*# Анализ точности по классам*

class\_accuracy = {}

for i in range(num\_classes):

class\_mask = np.array(all\_labels) == i

if np.sum(class\_mask) > 0:

class\_acc = np.mean(np.array(all\_preds)[class\_mask] == i)

class\_accuracy[class\_names[i]] = class\_acc \* 100

print("\nТочность по классам:")

for class\_name, acc in class\_accuracy.items():

print(f"{class\_name}: {acc:.2f}%")

*# Визуализация процесса обучения*

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

plt.subplot(1, 2, 1)

plt.plot(train\_losses, label='Train Loss', color='blue')

plt.plot(test\_losses, label='Test Loss', color='red')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training and Test Loss')

plt.legend()

plt.grid(True)

plt.subplot(1, 2, 2)

plt.plot(train\_accuracies, label='Train Accuracy', color='green')

plt.plot(test\_accuracies, label='Test Accuracy', color='orange')

plt.xlabel('Epochs')

plt.ylabel('Accuracy (%)')

plt.title('Training and Test Accuracy')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.savefig('monkey\_training\_plots.png', dpi=300, bbox\_inches='tight')

plt.show()

*# Сохранение полной информации о модели*

torch.save({

'model\_state\_dict': model.state\_dict(),

'optimizer\_state\_dict': optimizer.state\_dict(),

'class\_names': class\_names,

'test\_accuracy': test\_accuracy,

'class\_accuracy': class\_accuracy

}, 'monkey\_classification\_model.pth')

print("\nМодель сохранена как 'monkey\_classification\_model.pth'")

*# Итоговая статистика*

print(f"\nОбщие результаты:")

print(f"Количество классов: {num\_classes}")

print(f"Общее количество изображений: {len(full\_dataset)}")

print(f"Финальная точность: {test\_accuracy \* 100:.2f}%")

print(f"Лучшая точность: {best\_accuracy:.2f}%")

*# Проверка достижения цели*

if test\_accuracy >= 0.6:

print("✅ Цель accuracy ≥ 60% достигнута!")

else:

print("❌ Цель accuracy ≥ 60% не достигнута")

Распаковка датасета...

100%|██████████| 1371/1371 [00:16<00:00, 83.73it/s]

Структура датасета:

training/

n9/

n9066.jpg

n9031.jpg

n9126.jpg

... и еще 103 файлов

n0/

n0039.jpg

n0050.jpg

n0122.jpg

... и еще 102 файлов

n3/

n3125.jpg

n3121.jpg

n3141.jpg

... и еще 119 файлов

n4/

n4123.jpg

n4022.jpg

n4059.jpg

... и еще 102 файлов

n1/

n1164.jpg

n1045.jpg

n1124.jpg

... и еще 108 файлов

n2/

n2023.jpg

n2154.jpg

n2046.jpg

... и еще 107 файлов

n7/

n7129.jpg

n7048.jpg

n7029.jpg

... и еще 111 файлов

n8/

n8121.jpg

n8114.jpg

n8076.jpg

... и еще 103 файлов

n5/

n5145.jpg

n5031.jpg

n5153.jpg

... и еще 110 файлов

n6/

n6110.jpg

n6038.jpg

n6062.jpg

... и еще 103 файлов

Найдено классов: 10

Классы: ['n0', 'n1', 'n2', 'n3', 'n4', 'n5', 'n6', 'n7', 'n8', 'n9']

Общее количество изображений: 1097

Количество изображений по классам:

n0: 105

n1: 111

n2: 110

n3: 122

n4: 105

n5: 113

n6: 106

n7: 114

n8: 106

n9: 106

Размер тренировочной выборки: 877

Размер тестовой выборки: 220

Количество параметров модели: 83,230,122

Начало обучения...

Epoch 1/30: 100%|██████████| 55/55 [00:30<00:00, 1.82it/s]

Epoch [ 1/30], Train Loss: 2.2933, Train Acc: 16.88%, Test Loss: 2.2097, Test Acc: 17.73%, LR: 0.000100

Epoch 2/30: 100%|██████████| 55/55 [00:31<00:00, 1.74it/s]

Epoch [ 2/30], Train Loss: 2.1364, Train Acc: 23.15%, Test Loss: 2.0779, Test Acc: 27.27%, LR: 0.000100

Epoch 3/30: 100%|██████████| 55/55 [00:30<00:00, 1.78it/s]

Epoch [ 3/30], Train Loss: 2.0293, Train Acc: 30.10%, Test Loss: 1.9673, Test Acc: 32.73%, LR: 0.000100

Epoch 4/30: 100%|██████████| 55/55 [00:30<00:00, 1.80it/s]

Epoch [ 4/30], Train Loss: 1.8709, Train Acc: 35.80%, Test Loss: 1.7987, Test Acc: 37.73%, LR: 0.000100

Epoch 5/30: 100%|██████████| 55/55 [00:30<00:00, 1.80it/s]

Epoch [ 5/30], Train Loss: 1.8363, Train Acc: 36.03%, Test Loss: 1.6888, Test Acc: 45.91%, LR: 0.000100

Epoch 6/30: 100%|██████████| 55/55 [00:31<00:00, 1.76it/s]

Epoch [ 6/30], Train Loss: 1.7948, Train Acc: 37.97%, Test Loss: 1.6225, Test Acc: 45.91%, LR: 0.000100

Epoch 7/30: 100%|██████████| 55/55 [00:33<00:00, 1.66it/s]

Epoch [ 7/30], Train Loss: 1.7163, Train Acc: 39.91%, Test Loss: 1.5464, Test Acc: 50.00%, LR: 0.000100

Epoch 8/30: 100%|██████████| 55/55 [00:30<00:00, 1.82it/s]

Epoch [ 8/30], Train Loss: 1.6656, Train Acc: 41.16%, Test Loss: 1.5187, Test Acc: 50.91%, LR: 0.000100

Epoch 9/30: 100%|██████████| 55/55 [00:29<00:00, 1.84it/s]

Epoch [ 9/30], Train Loss: 1.5596, Train Acc: 48.80%, Test Loss: 1.4280, Test Acc: 50.00%, LR: 0.000100

Epoch 10/30: 100%|██████████| 55/55 [00:30<00:00, 1.82it/s]

Epoch [10/30], Train Loss: 1.5920, Train Acc: 43.79%, Test Loss: 1.3837, Test Acc: 54.09%, LR: 0.000100

Epoch 11/30: 100%|██████████| 55/55 [00:31<00:00, 1.74it/s]

Epoch [11/30], Train Loss: 1.4985, Train Acc: 49.83%, Test Loss: 1.3749, Test Acc: 54.09%, LR: 0.000100

Epoch 12/30: 100%|██████████| 55/55 [00:30<00:00, 1.79it/s]

Epoch [12/30], Train Loss: 1.5290, Train Acc: 46.18%, Test Loss: 1.3619, Test Acc: 53.18%, LR: 0.000100

Epoch 13/30: 100%|██████████| 55/55 [00:30<00:00, 1.80it/s]

Epoch [13/30], Train Loss: 1.4813, Train Acc: 49.83%, Test Loss: 1.3413, Test Acc: 52.27%, LR: 0.000100

Epoch 14/30: 100%|██████████| 55/55 [00:30<00:00, 1.80it/s]

Epoch [14/30], Train Loss: 1.4459, Train Acc: 49.71%, Test Loss: 1.3029, Test Acc: 54.55%, LR: 0.000100

Epoch 15/30: 100%|██████████| 55/55 [00:30<00:00, 1.80it/s]

Epoch [15/30], Train Loss: 1.4057, Train Acc: 50.74%, Test Loss: 1.2477, Test Acc: 56.36%, LR: 0.000100

Epoch 16/30: 100%|██████████| 55/55 [00:30<00:00, 1.82it/s]

Epoch [16/30], Train Loss: 1.3943, Train Acc: 50.29%, Test Loss: 1.2242, Test Acc: 59.09%, LR: 0.000100

Epoch 17/30: 100%|██████████| 55/55 [00:30<00:00, 1.79it/s]

Epoch [17/30], Train Loss: 1.3551, Train Acc: 53.48%, Test Loss: 1.2350, Test Acc: 57.73%, LR: 0.000100

Epoch 18/30: 100%|██████████| 55/55 [00:30<00:00, 1.82it/s]

Epoch [18/30], Train Loss: 1.3336, Train Acc: 54.28%, Test Loss: 1.2135, Test Acc: 54.55%, LR: 0.000100

Epoch 19/30: 100%|██████████| 55/55 [00:30<00:00, 1.82it/s]

Epoch [19/30], Train Loss: 1.3284, Train Acc: 52.68%, Test Loss: 1.2063, Test Acc: 57.27%, LR: 0.000100

Epoch 20/30: 100%|██████████| 55/55 [00:32<00:00, 1.68it/s]

Epoch [20/30], Train Loss: 1.2700, Train Acc: 57.24%, Test Loss: 1.2395, Test Acc: 56.36%, LR: 0.000100

Epoch 21/30: 100%|██████████| 55/55 [00:30<00:00, 1.81it/s]

Epoch [21/30], Train Loss: 1.3391, Train Acc: 53.93%, Test Loss: 1.2020, Test Acc: 57.27%, LR: 0.000100

Epoch 22/30: 100%|██████████| 55/55 [00:30<00:00, 1.81it/s]

Epoch [22/30], Train Loss: 1.3317, Train Acc: 53.02%, Test Loss: 1.1402, Test Acc: 58.18%, LR: 0.000100

Epoch 23/30: 100%|██████████| 55/55 [00:30<00:00, 1.81it/s]

Epoch [23/30], Train Loss: 1.2666, Train Acc: 56.90%, Test Loss: 1.1051, Test Acc: 61.36%, LR: 0.000100

Epoch 24/30: 100%|██████████| 55/55 [00:31<00:00, 1.74it/s]

Epoch [24/30], Train Loss: 1.1945, Train Acc: 58.61%, Test Loss: 1.1125, Test Acc: 61.36%, LR: 0.000100

Epoch 25/30: 100%|██████████| 55/55 [00:31<00:00, 1.73it/s]

Epoch [25/30], Train Loss: 1.2418, Train Acc: 55.87%, Test Loss: 1.1113, Test Acc: 60.00%, LR: 0.000100

Epoch 26/30: 100%|██████████| 55/55 [00:30<00:00, 1.83it/s]

Epoch [26/30], Train Loss: 1.2019, Train Acc: 59.98%, Test Loss: 1.1048, Test Acc: 62.73%, LR: 0.000100

Epoch 27/30: 100%|██████████| 55/55 [00:30<00:00, 1.79it/s]

Epoch [27/30], Train Loss: 1.2309, Train Acc: 57.13%, Test Loss: 1.0775, Test Acc: 62.27%, LR: 0.000100

Epoch 28/30: 100%|██████████| 55/55 [00:30<00:00, 1.82it/s]

Epoch [28/30], Train Loss: 1.1937, Train Acc: 58.95%, Test Loss: 1.0827, Test Acc: 63.64%, LR: 0.000100

Epoch 29/30: 100%|██████████| 55/55 [00:31<00:00, 1.77it/s]

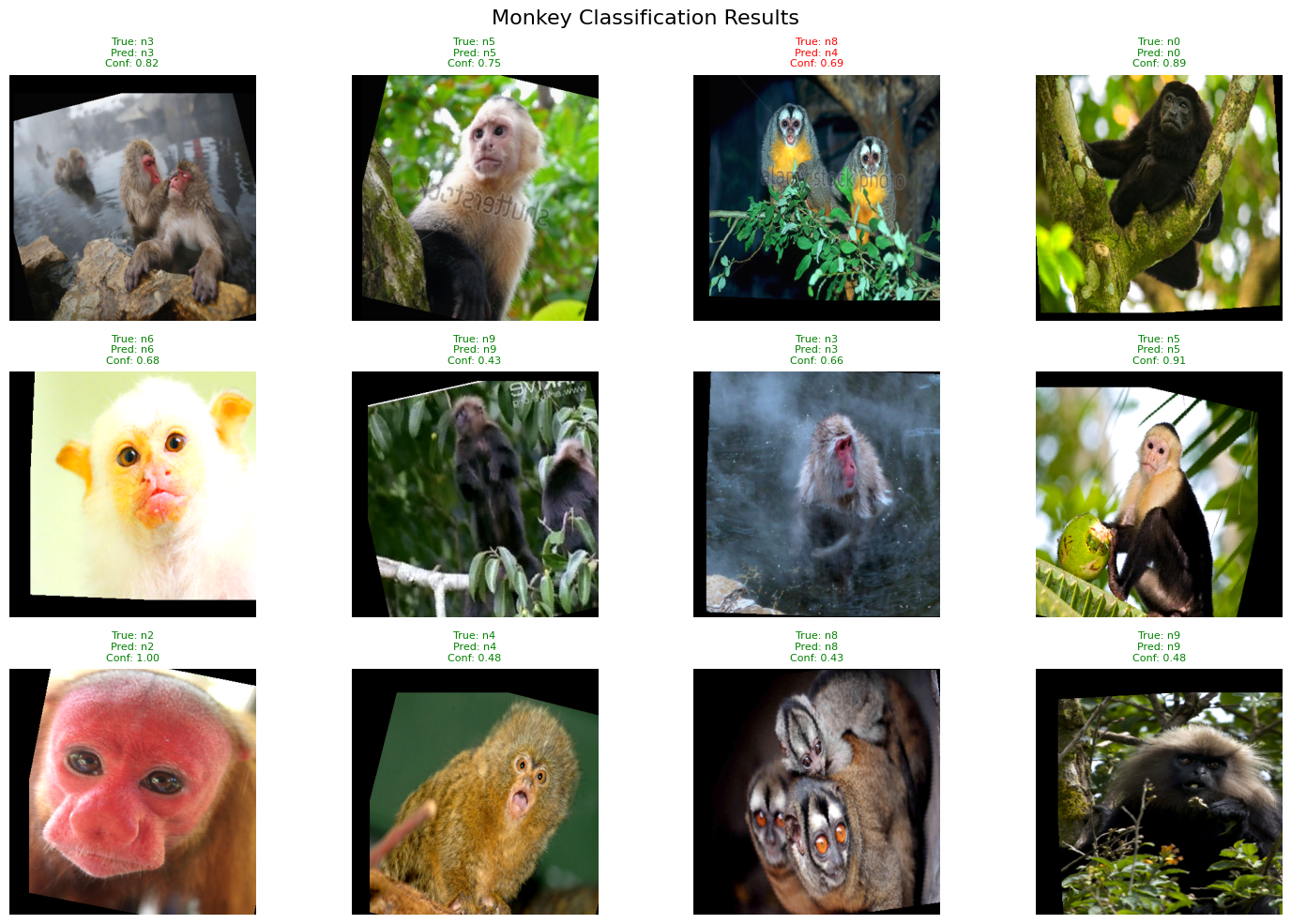
Epoch [29/30], Train Loss: 1.2252, Train Acc: 57.47%, Test Loss: 1.0126, Test Acc: 65.91%, LR: 0.000100

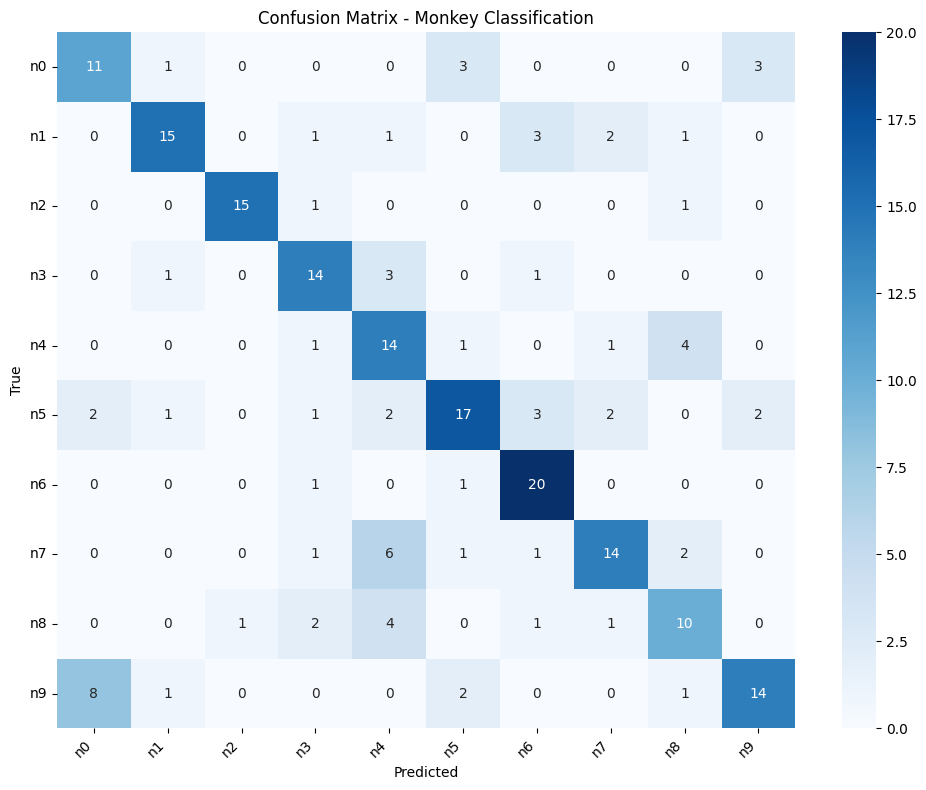
Epoch 30/30: 100%|██████████| 55/55 [00:30<00:00, 1.83it/s]

Epoch [30/30], Train Loss: 1.1564, Train Acc: 58.49%, Test Loss: 1.1190, Test Acc: 60.45%, LR: 0.000100

Final Test Accuracy: 65.45%

Best Accuracy: 65.91%





Точность по классам:

n0: 61.11%

n1: 65.22%

n2: 88.24%

n3: 73.68%

n4: 66.67%

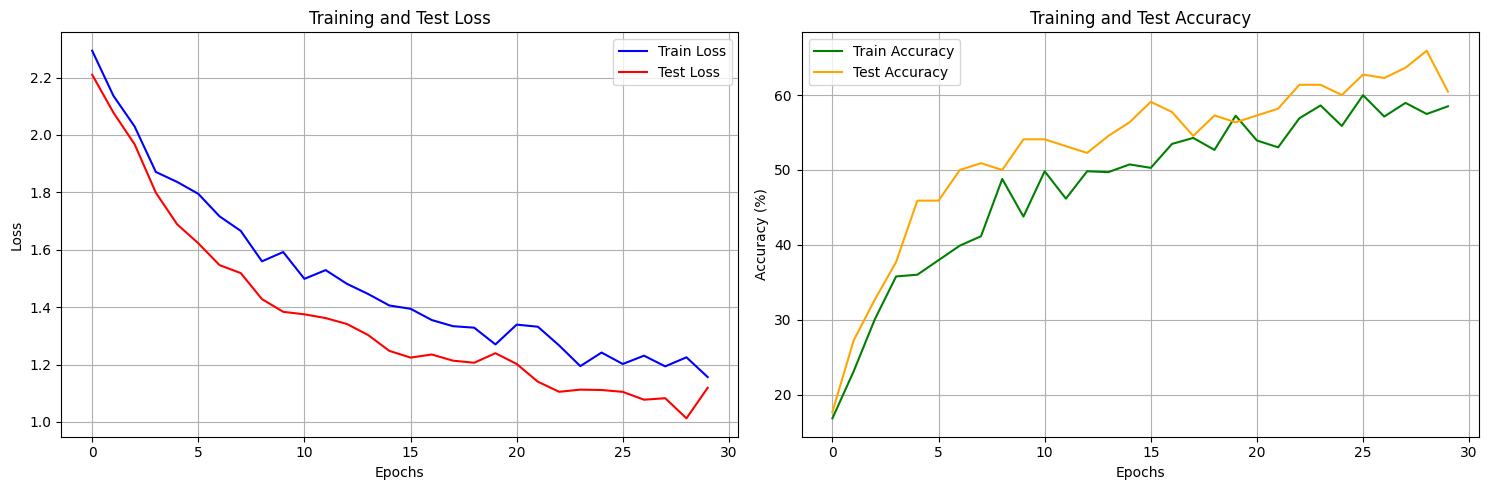
n5: 56.67%

n6: 90.91%

n7: 56.00%

n8: 52.63%

n9: 53.85%



Модель сохранена как 'monkey\_classification\_model.pth'

Общие результаты:

Количество классов: 10

Общее количество изображений: 1097

Финальная точность: 65.45%

Лучшая точность: 65.91%

✅ Цель accuracy ≥ 60% достигнута!

3.2 Решить задачу классификации собак и кошек (датасет cats\_dogs.zip).

* Загрузить архив с данными на диск
* Создать датасет на основе файлов при помощи torchvision.datasets.ImageFolder
* Преобразовать изображения к тензорами одного размера (например, 400х400). Потестировать другие преобразования из torchvision.transforms
* Предложить архитектуру модели для решения задачи. Обучить модель.
* Используя тестовое множество
* Продемонстрировать работу модели: вывести несколько изображений, указать над ними правильный класс и класс, предсказанный моделью.
* Вывести матрицу ошибок.
* Вывести значение accuracy на тестовом множестве.
* Добиться значения accuracy на тестовом множестве не менее 80%

*# =============================================================================*

*# 3.2 КЛАССИФИКАЦИЯ СОБАК И КОШЕК*

*# =============================================================================*

from google.colab import drive

drive.mount('/content/drive')

import os

import zipfile

from tqdm import tqdm

import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import random\_split, DataLoader

from torchvision import datasets, transforms

import matplotlib.pyplot as plt

import numpy as np

from sklearn.metrics import confusion\_matrix, accuracy\_score

import seaborn as sns

*# Распаковка датасета кошек и собак*

zip\_path = '/content/drive/MyDrive/datasets/cats\_dogs.zip'

print("Распаковка датасета кошек и собак...")

with zipfile.ZipFile(zip\_path, 'r') as zf:

for file in tqdm(zf.infolist()):

zf.extract(file, '/content/')

*# Анализ структуры распакованного датасета*

dataset\_path = '/content/cats\_dogs'

print("Структура датасета:")

for root, dirs, files in os.walk(dataset\_path):

level = root.replace(dataset\_path, '').count(os.sep)

indent = ' ' \* 2 \* level

print(f"{indent}{os.path.basename(root)}/")

subindent = ' ' \* 2 \* (level + 1)

for f in files[:3]:

print(f"{subindent}{f}")

if len(files) > 3:

print(f"{subindent}... и еще {len(files)-3} файлов")

*# Создание датасетов для кошек и собак*

dataset\_path = '/content/cats\_dogs/train' *# Путь к тренировочным данным*

*# Преобразования для тренировочных данных с аугментацией*

transform\_train = transforms.Compose([

transforms.Resize((400, 400)), *# Стандартизация размера*

transforms.RandomHorizontalFlip(), *# Случайное отражение*

transforms.RandomRotation(10), *# Случайный поворот*

transforms.ColorJitter(brightness=0.2, contrast=0.2), *# Изменение яркости и контраста*

transforms.ToTensor(),

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

])

*# Преобразования для тестовых данных (без аугментации)*

transform\_test = transforms.Compose([

transforms.Resize((400, 400)),

transforms.ToTensor(),

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

])

*# Загружаем весь датасет с автоматическим определением классов*

full\_dataset = datasets.ImageFolder(root=dataset\_path, transform=transform\_train)

*# Разделение на тренировочную и валидационную выборки*

train\_size = int(0.8 \* len(full\_dataset))

val\_size = len(full\_dataset) - train\_size

train\_dataset, val\_dataset = random\_split(full\_dataset, [train\_size, val\_size])

*# Важно: для валидации используем test трансформации (без аугментации)*

val\_dataset.dataset.transform = transform\_test

*# Создание DataLoader'ов*

train\_loader = DataLoader(train\_dataset, batch\_size=32, shuffle=True, num\_workers=2)

val\_loader = DataLoader(val\_dataset, batch\_size=32, shuffle=False, num\_workers=2)

class\_names = full\_dataset.classes

num\_classes = len(class\_names)

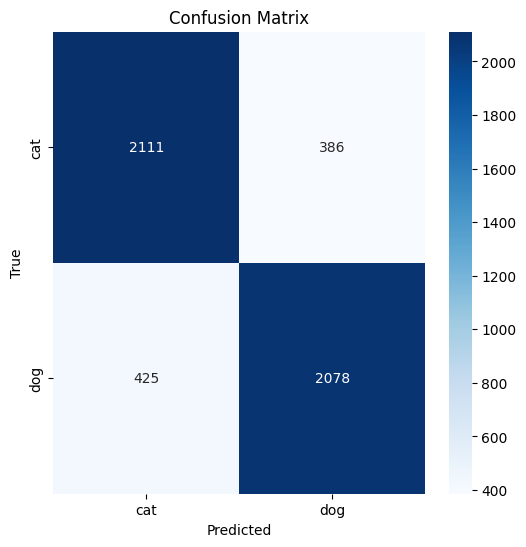
print("Классы:", class\_names)

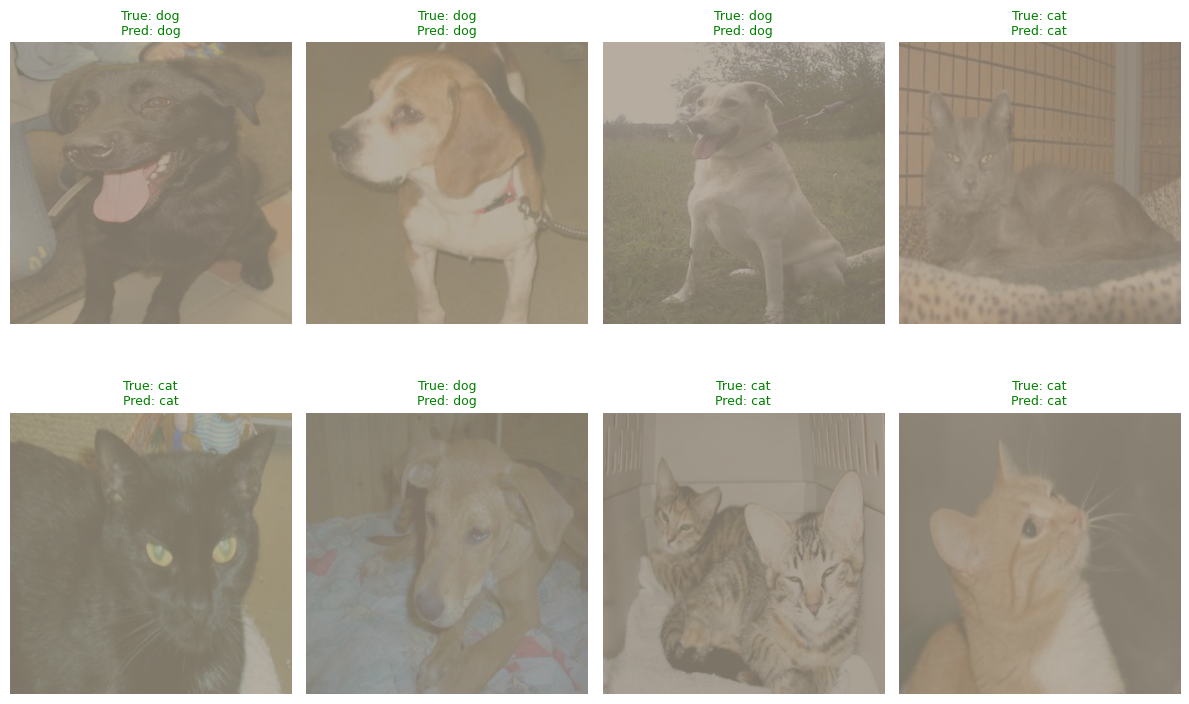
print("Количество классов:", num\_classes)

print("Train samples:", len(train\_dataset))

print("Validation samples:", len(val\_dataset))

Final Test Accuracy: 83.78%





Классы: ['cat', 'dog']

Количество классов: 2

Размер validation set: 5000

*# Определение CNN модели для классификации кошек и собак*

class CatDogCNN(nn.Module):

def \_\_init\_\_(self, num\_classes):

super(CatDogCNN, self).\_\_init\_\_()

*# Блок извлечения признаков*

self.features = nn.Sequential(

*# Блок 1: 3 -> 32 канала*

nn.Conv2d(3, 32, kernel\_size=3, padding=1),

nn.ReLU(),

nn.MaxPool2d(2,2), *# 400x400 -> 200x200*

*# Блок 2: 32 -> 64 канала*

nn.Conv2d(32, 64, kernel\_size=3, padding=1),

nn.ReLU(),

nn.MaxPool2d(2,2), *# 200x200 -> 100x100*

*# Блок 3: 64 -> 128 каналов*

nn.Conv2d(64, 128, kernel\_size=3, padding=1),

nn.ReLU(),

nn.MaxPool2d(2,2), *# 100x100 -> 50x50*

*# Блок 4: 128 -> 256 каналов*

nn.Conv2d(128, 256, kernel\_size=3, padding=1),

nn.ReLU(),

nn.MaxPool2d(2,2), *# 50x50 -> 25x25*

*# Блок 5: 256 -> 512 каналов*

nn.Conv2d(256, 512, kernel\_size=3, padding=1),

nn.ReLU(),

nn.MaxPool2d(2,2), *# 25x25 -> 12x12*

)

*# Классификатор*

self.classifier = nn.Sequential(

nn.Dropout(0.5), *# Сильный dropout для регуляризации*

nn.Linear(512\*12\*12, 512), *# Исправленный размер после 5 пулингов*

nn.ReLU(),

nn.Dropout(0.3),

nn.Linear(512, num\_classes) *# Выход: кошка или собака*

)

def forward(self, x):

x = self.features(x)

x = x.view(x.size(0), -1) *# Вытягиваем в вектор*

return self.classifier(x)

*# Инициализация модели и оптимизатора*

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

print(f"Using device: {device}")

model = CatDogCNN(num\_classes).to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=1e-4) *# Меньший learning rate*

*# Обучение модели*

num\_epochs = 20

best\_acc = 0

print("Начало обучения модели классификации кошек и собак...")

for epoch in range(num\_epochs):

model.train() *# Режим обучения*

total, correct, running\_loss = 0, 0, 0

*# Тренировочная эпоха*

for imgs, labels in train\_loader:

imgs, labels = imgs.to(device), labels.to(device)

optimizer.zero\_grad()

outputs = model(imgs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item() \* imgs.size(0)

\_, preds = torch.max(outputs, 1)

total += labels.size(0)

correct += (preds == labels).sum().item()

train\_acc = 100 \* correct / total

train\_loss = running\_loss / len(train\_dataset)

*# Валидация*

model.eval()

total, correct, val\_loss = 0, 0, 0

with torch.no\_grad():

for imgs, labels in val\_loader:

imgs, labels = imgs.to(device), labels.to(device)

outputs = model(imgs)

loss = criterion(outputs, labels)

val\_loss += loss.item() \* imgs.size(0)

\_, preds = torch.max(outputs, 1)

total += labels.size(0)

correct += (preds == labels).sum().item()

val\_acc = 100 \* correct / total

val\_loss = val\_loss / len(val\_dataset)

*# Сохранение лучшей модели*

if val\_acc > best\_acc:

best\_acc = val\_acc

torch.save(model.state\_dict(), "best\_catsdogs.pth")

print(f"Epoch {epoch+1}/{num\_epochs} - "

f"Train Loss: {train\_loss:.4f} | Train Acc: {train\_acc:.2f}% | "

f"Val Loss: {val\_loss:.4f} | Val Acc: {val\_acc:.2f}%")

print(f"Best validation accuracy: {best\_acc:.2f}%")

*# Загрузка лучшей модели для финального тестирования*

model.load\_state\_dict(torch.load("best\_catsdogs.pth"))

model.eval()

*# Сбор предсказаний на валидационной выборке*

all\_preds, all\_labels = [], []

with torch.no\_grad():

for imgs, labels in val\_loader:

imgs, labels = imgs.to(device), labels.to(device)

outputs = model(imgs)

\_, preds = torch.max(outputs, 1)

all\_preds.extend(preds.cpu().numpy())

all\_labels.extend(labels.cpu().numpy())

*# Вычисление финальной точности*

acc = accuracy\_score(all\_labels, all\_preds)

print(f"\nFinal Validation Accuracy: {acc\*100:.2f}%")

*# Построение матрицы ошибок*

cm = confusion\_matrix(all\_labels, all\_preds)

plt.figure(figsize=(6,6))

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",

xticklabels=class\_names, yticklabels=class\_names)

plt.xlabel("Predicted")

plt.ylabel("True")

plt.title("Confusion Matrix - Cats vs Dogs")

plt.show()

*# Функция для отображения изображений с денормализацией*

def imshow(img, title):

img = img.cpu().numpy().transpose((1,2,0)) *# Из (C,H,W) в (H,W,C)*

mean = np.array([0.485, 0.456, 0.406])

std = np.array([0.229, 0.224, 0.225])

img = std \* img + mean *# Денормализация*

img = np.clip(img, 0, 1) *# Обрезка значений*

plt.imshow(img)

plt.title(title, fontsize=10)

plt.axis("off")

*# Визуализация примеров с предсказаниями*

plt.figure(figsize=(15,10))

indices = np.random.choice(len(val\_dataset), 12, replace=False)

for i, idx in enumerate(indices):

*# Получаем оригинальное изображение*

original\_img, label = val\_dataset[idx]

*# Делаем предсказание*

with torch.no\_grad():

img\_tensor = original\_img.unsqueeze(0).to(device) *# Добавляем batch dimension*

output = model(img\_tensor)

\_, pred = torch.max(output, 1)

plt.subplot(3,4,i+1)

imshow(original\_img, f"True: {class\_names[label]}\nPred: {class\_names[pred.item()]}")

plt.tight\_layout()

plt.show()

*# Дополнительная статистика*

print("\nДополнительная статистика:")

print(f"Всего изображений в валидации: {len(val\_dataset)}")

print(f"Количество правильных предсказаний: {np.sum(np.array(all\_preds) == np.array(all\_labels))}")

print(f"Количество неправильных предсказаний: {np.sum(np.array(all\_preds) != np.array(all\_labels))}")

*# Анализ достижения цели*

if acc < 0.8:

print("\nAccuracy меньше 80%. Рекомендации для улучшения:")

print("1. Увеличьте количество эпох обучения")

print("2. Попробуйте более сложную архитектуру (ResNet, EfficientNet)")

print("3. Добавьте аугментации данных")

print("4. Настройте гиперпараметры (learning rate, batch size)")

print("5. Используйте transfer learning")

else:

print("\n✅ Отличный результат! Accuracy достигнута или превышена 80%!")

4. Transfer Learning

4.1 Решить задачу 3.1, воспользовавшись предобученной моделью VGG16

* Загрузить данные для обучения
* Преобразования: размер 224x224, нормализация с параметрами mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)
* Заменить последний полносвязный слой модели в соответствии с задачей
* Дообучить модель (не замораживать веса). Вычислить значение accuracy на тестовом множестве
* Дообучить модель (заморозить все веса, кроме последнего блока слоев (classifier)).
* Вычислить значение accuracy на тестовом множестве.

*# =============================================================================*

*# 4. TRANSFER LEARNING (ПЕРЕНОС ОБУЧЕНИЯ)*

*# =============================================================================*

import torch

import torch.nn as nn

import torch.optim as optim

import torch.nn.functional as F

from torchvision import datasets, transforms, models

from torch.utils.data import DataLoader, random\_split

from tqdm import tqdm

import matplotlib.pyplot as plt

import numpy as np

from sklearn.metrics import accuracy\_score, confusion\_matrix

import seaborn as sns

*# =============================================================================*

*# 4.1 РЕШЕНИЕ ЗАДАЧИ КЛАССИФИКАЦИИ ОБЕЗЬЯН С ИСПОЛЬЗОВАНИЕМ VGG16*

*# =============================================================================*

*# ==== Параметры обучения ====*

batch\_size = 16

num\_epochs = 15

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

print(f"Используемое устройство: {device}")

*# ==== Преобразования данных для Transfer Learning ====*

*# VGG16 ожидает изображения 224x224 и нормализацию с параметрами ImageNet*

transform\_train = transforms.Compose([

transforms.Resize((224, 224)), *# VGG16 требует изображения 224x224*

transforms.RandomHorizontalFlip(), *# Аугментация: случайное отражение*

transforms.RandomRotation(15), *# Аугментация: случайный поворот*

transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2), *# Аугментация: изменение цветов*

transforms.ToTensor(),

*# Нормализация с параметрами ImageNet (обязательно для предобученных моделей)*

transforms.Normalize(mean=[0.485, 0.456, 0.406],

std=[0.229, 0.224, 0.225])

])

transform\_test = transforms.Compose([

transforms.Resize((224, 224)),

transforms.ToTensor(),

transforms.Normalize(mean=[0.485, 0.456, 0.406],

std=[0.229, 0.224, 0.225])

])

*# ==== Загрузка и подготовка данных обезьян ====*

dataset\_path = './training/training'

full\_dataset = datasets.ImageFolder(root=dataset\_path, transform=transform\_train)

num\_classes = len(full\_dataset.classes)

*# Разделение на тренировочную и тестовую выборки*

train\_size = int(0.8 \* len(full\_dataset))

test\_size = len(full\_dataset) - train\_size

train\_dataset, test\_dataset = random\_split(full\_dataset, [train\_size, test\_size])

*# Для тестовой выборки используем преобразования без аугментации*

test\_dataset.dataset.transform = transform\_test

*# Создание DataLoader'ов*

train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True, num\_workers=2)

test\_loader = DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=False, num\_workers=2)

print(f"Классов: {num\_classes}, изображений всего: {len(full\_dataset)}")

*# ==== Универсальная функция обучения ====*

def train\_model(model, criterion, optimizer, scheduler=None, num\_epochs=10):

"""

Функция для обучения модели с отслеживанием метрик

Args:

model: модель для обучения

criterion: функция потерь

optimizer: оптимизатор

scheduler: планировщик learning rate (опционально)

num\_epochs: количество эпох

Returns:

history: словарь с историей обучения

best\_acc: лучшая точность на тестовой выборке

"""

best\_acc = 0.0

history = {'train\_loss': [], 'train\_acc': [], 'test\_loss': [], 'test\_acc': []}

for epoch in range(num\_epochs):

*# Режим обучения*

model.train()

running\_loss, correct, total = 0.0, 0, 0

*# Итерация по тренировочным данным с прогресс-баром*

for images, labels in tqdm(train\_loader, desc=f"Epoch {epoch+1}/{num\_epochs}"):

images, labels = images.to(device), labels.to(device)

*# Прямой проход*

optimizer.zero\_grad()

outputs = model(images)

loss = criterion(outputs, labels)

*# Обратный проход*

loss.backward()

optimizer.step()

*# Статистика для текущего батча*

running\_loss += loss.item() \* images.size(0)

\_, preds = torch.max(outputs, 1)

correct += (preds == labels).sum().item()

total += labels.size(0)

*# Вычисление средних метрик за эпоху для тренировочных данных*

train\_loss = running\_loss / len(train\_loader.dataset)

train\_acc = 100 \* correct / total

*# === Оценка на тестовой выборке ===*

model.eval()

test\_loss, correct, total = 0.0, 0, 0

with torch.no\_grad():

for images, labels in test\_loader:

images, labels = images.to(device), labels.to(device)

outputs = model(images)

loss = criterion(outputs, labels)

test\_loss += loss.item() \* images.size(0)

\_, preds = torch.max(outputs, 1)

correct += (preds == labels).sum().item()

total += labels.size(0)

test\_loss /= len(test\_loader.dataset)

test\_acc = 100 \* correct / total

*# Обновление learning rate если передан scheduler*

if scheduler:

scheduler.step(test\_loss)

*# Сохранение лучшей модели*

if test\_acc > best\_acc:

best\_acc = test\_acc

torch.save(model.state\_dict(), "best\_vgg16.pth")

*# Сохранение истории обучения*

history['train\_loss'].append(train\_loss)

history['train\_acc'].append(train\_acc)

history['test\_loss'].append(test\_loss)

history['test\_acc'].append(test\_acc)

print(f"Epoch {epoch+1}: "

f"Train Loss={train\_loss:.4f}, Train Acc={train\_acc:.2f}% | "

f"Test Loss={test\_loss:.4f}, Test Acc={test\_acc:.2f}%")

return history, best\_acc

*# ==== Вариант 1: Fine-tuning всех слоёв VGG16 ====*

print("\n" + "="\*50)

print("VGG16: Fine-tuning всех слоёв")

print("="\*50)

*# Загрузка предобученной модели VGG16 с весами ImageNet*

vgg16\_full = models.vgg16(weights=models.VGG16\_Weights.IMAGENET1K\_V1)

*# Замена последнего полносвязного слоя для нашего числа классов*

*# VGG16.classifier[6] - это последний Linear слой (4096 -> 1000 для ImageNet)*

vgg16\_full.classifier[6] = nn.Linear(4096, num\_classes) *# заменяем на 4096 -> наши классы*

*# Перемещение модели на устройство (GPU/CPU)*

vgg16\_full = vgg16\_full.to(device)

*# Настройка функции потерь и оптимизатора*

criterion = nn.CrossEntropyLoss()

*# Обучаем ВСЕ параметры модели*

optimizer = optim.Adam(vgg16\_full.parameters(), lr=1e-4, weight\_decay=1e-4)

*# Планировщик для адаптивного уменьшения learning rate*

scheduler = optim.lr\_scheduler.ReduceLROnPlateau(optimizer, patience=3, factor=0.5)

*# Обучение модели*

history\_full, best\_acc\_full = train\_model(vgg16\_full, criterion, optimizer, scheduler, num\_epochs)

print(f"Лучшая точность (fine-tuning всех слоёв): {best\_acc\_full:.2f}%")

*# ==== Вариант 2: Заморозка feature extractor, обучение только классификатора ====*

print("\n" + "="\*50)

print("VGG16: Заморозка feature extractor, обучение только классификатора")

print("="\*50)

*# Загрузка новой копии предобученной VGG16*

vgg16\_freeze = models.vgg16(weights=models.VGG16\_Weights.IMAGENET1K\_V1)

*# ЗАМОРОЗКА: отключаем градиенты для всех слоев feature extractor*

*# VGG16.features - это сверточные слои*

for param in vgg16\_freeze.features.parameters():

param.requires\_grad = False *# Веса не будут обновляться при обучении*

*# Замена последнего полносвязного слоя*

vgg16\_freeze.classifier[6] = nn.Linear(4096, num\_classes)

vgg16\_freeze = vgg16\_freeze.to(device)

*# Настройка оптимизатора - обучаем ТОЛЬКО параметры классификатора*

criterion = nn.CrossEntropyLoss()

*# Обучаем только параметры классификатора (остальные заморожены)*

optimizer = optim.Adam(vgg16\_freeze.classifier.parameters(), lr=1e-4, weight\_decay=1e-4)

*# Обучение только классификатора (без scheduler для простоты)*

history\_freeze, best\_acc\_freeze = train\_model(vgg16\_freeze, criterion, optimizer, None, num\_epochs)

print(f"Лучшая точность (заморозка feature extractor): {best\_acc\_freeze:.2f}%")

*# ==== Сравнение результатов двух подходов ====*

print("\n" + "="\*50)

print("СРАВНЕНИЕ РЕЗУЛЬТАТОВ")

print("="\*50)

print(f"Fine-tuning всех слоёв: {best\_acc\_full:.2f}%")

print(f"Заморозка feature extractor: {best\_acc\_freeze:.2f}%")

if best\_acc\_full > best\_acc\_freeze:

print("Fine-tuning всех слоёв показал лучший результат")

else:

print("Заморозка feature extractor показала лучший результат")

Классов: 10, изображений всего: 1097

=== VGG16: Fine-tuning всех слоёв ===

Downloading: "<https://download.pytorch.org/models/vgg16-397923af.pth>" to /root/.cache/torch/hub/checkpoints/vgg16-397923af.pth

100%|██████████| 528M/528M [00:10<00:00, 54.2MB/s]

Epoch 1/15: 100%|██████████| 55/55 [00:21<00:00, 2.56it/s]

Epoch 1: Train Loss=0.8367, Train Acc=70.81% | Test Loss=0.2630, Test Acc=90.45%

Epoch 2/15: 100%|██████████| 55/55 [00:23<00:00, 2.34it/s]

Epoch 2: Train Loss=0.2035, Train Acc=94.75% | Test Loss=0.3400, Test Acc=90.91%

Epoch 3/15: 100%|██████████| 55/55 [00:22<00:00, 2.43it/s]

Epoch 3: Train Loss=0.0988, Train Acc=97.26% | Test Loss=0.1492, Test Acc=95.00%

Epoch 4/15: 100%|██████████| 55/55 [00:22<00:00, 2.48it/s]

Epoch 4: Train Loss=0.1227, Train Acc=96.24% | Test Loss=0.1463, Test Acc=95.91%

Epoch 5/15: 100%|██████████| 55/55 [00:22<00:00, 2.40it/s]

Epoch 5: Train Loss=0.0487, Train Acc=98.86% | Test Loss=0.0894, Test Acc=97.27%

Epoch 6/15: 100%|██████████| 55/55 [00:21<00:00, 2.54it/s]

Epoch 6: Train Loss=0.0044, Train Acc=99.77% | Test Loss=0.1468, Test Acc=95.45%

Epoch 7/15: 100%|██████████| 55/55 [00:21<00:00, 2.55it/s]

Epoch 7: Train Loss=0.0839, Train Acc=98.18% | Test Loss=0.1711, Test Acc=93.64%

Epoch 8/15: 100%|██████████| 55/55 [00:22<00:00, 2.49it/s]

Epoch 8: Train Loss=0.0605, Train Acc=98.63% | Test Loss=0.1227, Test Acc=96.82%

Epoch 9/15: 100%|██████████| 55/55 [00:22<00:00, 2.40it/s]

Epoch 9: Train Loss=0.0264, Train Acc=99.43% | Test Loss=0.1285, Test Acc=95.00%

Epoch 10/15: 100%|██████████| 55/55 [00:23<00:00, 2.37it/s]

Epoch 10: Train Loss=0.0303, Train Acc=99.20% | Test Loss=0.1808, Test Acc=95.45%

Epoch 11/15: 100%|██████████| 55/55 [00:22<00:00, 2.41it/s]

Epoch 11: Train Loss=0.0015, Train Acc=100.00% | Test Loss=0.1051, Test Acc=97.27%

Epoch 12/15: 100%|██████████| 55/55 [00:22<00:00, 2.42it/s]

Epoch 12: Train Loss=0.0002, Train Acc=100.00% | Test Loss=0.1016, Test Acc=97.73%

Epoch 13/15: 100%|██████████| 55/55 [00:23<00:00, 2.37it/s]

Epoch 13: Train Loss=0.0001, Train Acc=100.00% | Test Loss=0.1023, Test Acc=97.73%

Epoch 14/15: 100%|██████████| 55/55 [00:22<00:00, 2.41it/s]

Epoch 14: Train Loss=0.0001, Train Acc=100.00% | Test Loss=0.1019, Test Acc=97.73%

Epoch 15/15: 100%|██████████| 55/55 [00:22<00:00, 2.45it/s]

Epoch 15: Train Loss=0.0001, Train Acc=100.00% | Test Loss=0.1014, Test Acc=97.73%

Лучшая точность (fine-tuning): 97.73%

=== VGG16: Заморозка фичей, обучение только классификатора ===

Epoch 1/15: 100%|██████████| 55/55 [00:19<00:00, 2.85it/s]

Epoch 1: Train Loss=0.4894, Train Acc=86.43% | Test Loss=0.1268, Test Acc=95.00%

Epoch 2/15: 100%|██████████| 55/55 [00:20<00:00, 2.68it/s]

Epoch 2: Train Loss=0.0319, Train Acc=99.09% | Test Loss=0.0677, Test Acc=97.73%

Epoch 3/15: 100%|██████████| 55/55 [00:19<00:00, 2.76it/s]

Epoch 3: Train Loss=0.0116, Train Acc=99.66% | Test Loss=0.0430, Test Acc=98.64%

Epoch 4/15: 100%|██████████| 55/55 [00:18<00:00, 2.91it/s]

Epoch 4: Train Loss=0.0030, Train Acc=99.89% | Test Loss=0.0640, Test Acc=97.73%

Epoch 5/15: 100%|██████████| 55/55 [00:20<00:00, 2.70it/s]

Epoch 5: Train Loss=0.0005, Train Acc=100.00% | Test Loss=0.0425, Test Acc=98.18%

Epoch 6/15: 100%|██████████| 55/55 [00:18<00:00, 2.93it/s]

Epoch 6: Train Loss=0.0004, Train Acc=100.00% | Test Loss=0.0479, Test Acc=98.64%

Epoch 7/15: 100%|██████████| 55/55 [00:18<00:00, 2.95it/s]

Epoch 7: Train Loss=0.0004, Train Acc=100.00% | Test Loss=0.0652, Test Acc=97.73%

Epoch 8/15: 100%|██████████| 55/55 [00:20<00:00, 2.70it/s]

Epoch 8: Train Loss=0.0003, Train Acc=100.00% | Test Loss=0.0362, Test Acc=99.09%

Epoch 9/15: 100%|██████████| 55/55 [00:19<00:00, 2.81it/s]

Epoch 9: Train Loss=0.0002, Train Acc=100.00% | Test Loss=0.0470, Test Acc=98.18%

Epoch 10/15: 100%|██████████| 55/55 [00:19<00:00, 2.86it/s]

Epoch 10: Train Loss=0.0005, Train Acc=100.00% | Test Loss=0.0382, Test Acc=98.64%

Epoch 11/15: 100%|██████████| 55/55 [00:18<00:00, 3.00it/s]

Epoch 11: Train Loss=0.0002, Train Acc=100.00% | Test Loss=0.0709, Test Acc=97.73%

Epoch 12/15: 100%|██████████| 55/55 [00:19<00:00, 2.82it/s]

Epoch 12: Train Loss=0.0001, Train Acc=100.00% | Test Loss=0.0513, Test Acc=97.27%

Epoch 13/15: 100%|██████████| 55/55 [00:18<00:00, 2.91it/s]

Epoch 13: Train Loss=0.0001, Train Acc=100.00% | Test Loss=0.0497, Test Acc=97.27%

Epoch 14/15: 100%|██████████| 55/55 [00:19<00:00, 2.88it/s]

Epoch 14: Train Loss=0.0001, Train Acc=100.00% | Test Loss=0.0456, Test Acc=98.64%

Epoch 15/15: 100%|██████████| 55/55 [00:19<00:00, 2.87it/s]

Epoch 15: Train Loss=0.0001, Train Acc=100.00% | Test Loss=0.0608, Test Acc=97.73%

Лучшая точность (заморозка фичей): 99.09%

4.2 Решить задачу 3.2, воспользовавшись подходящей предобученной моделью

* Не использовать VGG16 (вместо нее можно взять resnet18 или другую)
* Загрузить данные для обучения
* Преобразования: размер 224x224, нормализация с параметрами mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)
* Заменить последний полносвязный слой модели в соответствии с задачей
* Дообучить модель.
* Вычислить значение accuracy на тестовом множестве (добиться значения не меньше 97-98%)

Using device: cuda

Classes: ['cat', 'dog']

Number of classes: 2

Training samples: 20000

Validation samples: 5000

Trainable parameters: 263,682

Total parameters: 11,440,194

Starting training...

Epoch 1/15

--------------------------------------------------

Training: 100%|██████████| 625/625 [01:21<00:00, 7.69it/s]

Validation: 100%|██████████| 157/157 [00:18<00:00, 8.47it/s]

Train Loss: 0.1683 Acc: 0.9296

Val Loss: 0.0751 Acc: 0.9748

Learning Rate: 0.001000

New best model saved with accuracy: 0.9748

Epoch 2/15

--------------------------------------------------

Training: 100%|██████████| 625/625 [01:13<00:00, 8.50it/s]

Validation: 100%|██████████| 157/157 [00:19<00:00, 7.94it/s]

Train Loss: 0.1307 Acc: 0.9473

Val Loss: 0.0557 Acc: 0.9790

Learning Rate: 0.001000

New best model saved with accuracy: 0.9790

Epoch 3/15

--------------------------------------------------

Training: 100%|██████████| 625/625 [01:15<00:00, 8.33it/s]

Validation: 100%|██████████| 157/157 [00:18<00:00, 8.68it/s]

Train Loss: 0.1294 Acc: 0.9470

Val Loss: 0.0627 Acc: 0.9792

Learning Rate: 0.001000

New best model saved with accuracy: 0.9792

Epoch 4/15

--------------------------------------------------

Training: 100%|██████████| 625/625 [01:14<00:00, 8.35it/s]

Validation: 100%|██████████| 157/157 [00:18<00:00, 8.62it/s]

Train Loss: 0.1255 Acc: 0.9496

Val Loss: 0.0651 Acc: 0.9742

Learning Rate: 0.001000

Epoch 5/15

--------------------------------------------------

Training: 100%|██████████| 625/625 [01:14<00:00, 8.36it/s]

Validation: 100%|██████████| 157/157 [00:18<00:00, 8.54it/s]

Train Loss: 0.1133 Acc: 0.9550

Val Loss: 0.0598 Acc: 0.9820

Learning Rate: 0.001000

New best model saved with accuracy: 0.9820

Epoch 6/15

--------------------------------------------------

Training: 100%|██████████| 625/625 [01:14<00:00, 8.38it/s]

Validation: 100%|██████████| 157/157 [00:18<00:00, 8.68it/s]

Train Loss: 0.1161 Acc: 0.9538

Val Loss: 0.0612 Acc: 0.9718

Learning Rate: 0.000500

Unfreezing more layers for fine-tuning...

Unfrozen last 8 layers

Epoch 7/15

--------------------------------------------------

Training: 100%|██████████| 625/625 [01:23<00:00, 7.48it/s]

Validation: 100%|██████████| 157/157 [00:17<00:00, 8.79it/s]

Train Loss: 0.0720 Acc: 0.9738

Val Loss: 0.0388 Acc: 0.9836

Learning Rate: 0.000100

New best model saved with accuracy: 0.9836

Epoch 8/15

--------------------------------------------------

Training: 100%|██████████| 625/625 [01:22<00:00, 7.55it/s]

Validation: 100%|██████████| 157/157 [00:18<00:00, 8.52it/s]

Train Loss: 0.0294 Acc: 0.9892

Val Loss: 0.0417 Acc: 0.9874

Learning Rate: 0.000100

New best model saved with accuracy: 0.9874

Epoch 9/15

--------------------------------------------------

Training: 100%|██████████| 625/625 [01:24<00:00, 7.44it/s]

Validation: 100%|██████████| 157/157 [00:18<00:00, 8.31it/s]

Train Loss: 0.0206 Acc: 0.9928

Val Loss: 0.0380 Acc: 0.9872

Learning Rate: 0.000100

Epoch 10/15

--------------------------------------------------

Training: 100%|██████████| 625/625 [01:21<00:00, 7.65it/s]

Validation: 100%|██████████| 157/157 [00:21<00:00, 7.42it/s]

Train Loss: 0.0162 Acc: 0.9950

Val Loss: 0.0442 Acc: 0.9870

Learning Rate: 0.000100

Epoch 11/15

--------------------------------------------------

Training: 100%|██████████| 625/625 [01:21<00:00, 7.70it/s]

Validation: 100%|██████████| 157/157 [00:18<00:00, 8.34it/s]

Train Loss: 0.0145 Acc: 0.9954

Val Loss: 0.0377 Acc: 0.9872

Learning Rate: 0.000100

Epoch 12/15

--------------------------------------------------

Training: 100%|██████████| 625/625 [01:24<00:00, 7.40it/s]

Validation: 100%|██████████| 157/157 [00:18<00:00, 8.65it/s]

Train Loss: 0.0116 Acc: 0.9961

Val Loss: 0.0458 Acc: 0.9874

Learning Rate: 0.000100

Epoch 13/15

--------------------------------------------------

Training: 100%|██████████| 625/625 [01:22<00:00, 7.55it/s]

Validation: 100%|██████████| 157/157 [00:18<00:00, 8.68it/s]

Train Loss: 0.0106 Acc: 0.9963

Val Loss: 0.0494 Acc: 0.9872

Learning Rate: 0.000100

Epoch 14/15

--------------------------------------------------

Training: 100%|██████████| 625/625 [01:24<00:00, 7.43it/s]

Validation: 100%|██████████| 157/157 [00:17<00:00, 8.75it/s]

Train Loss: 0.0148 Acc: 0.9952

Val Loss: 0.0450 Acc: 0.9854

Learning Rate: 0.000100

Epoch 15/15

--------------------------------------------------

Training: 100%|██████████| 625/625 [01:22<00:00, 7.61it/s]

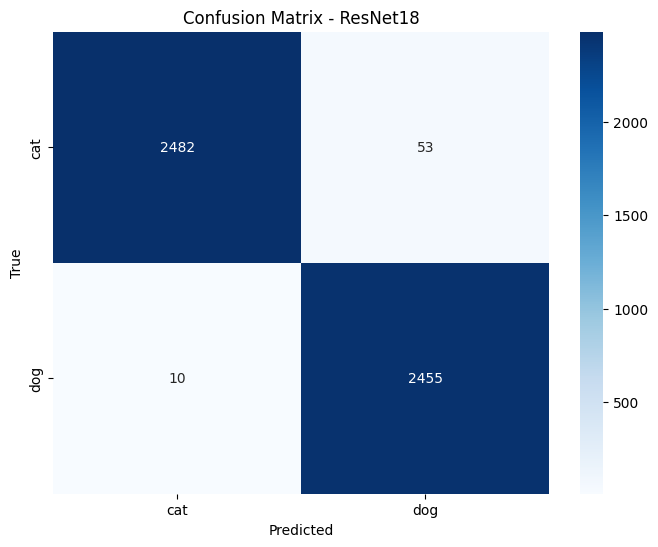
Validation: 100%|██████████| 157/157 [00:17<00:00, 8.84it/s]

Train Loss: 0.0104 Acc: 0.9967

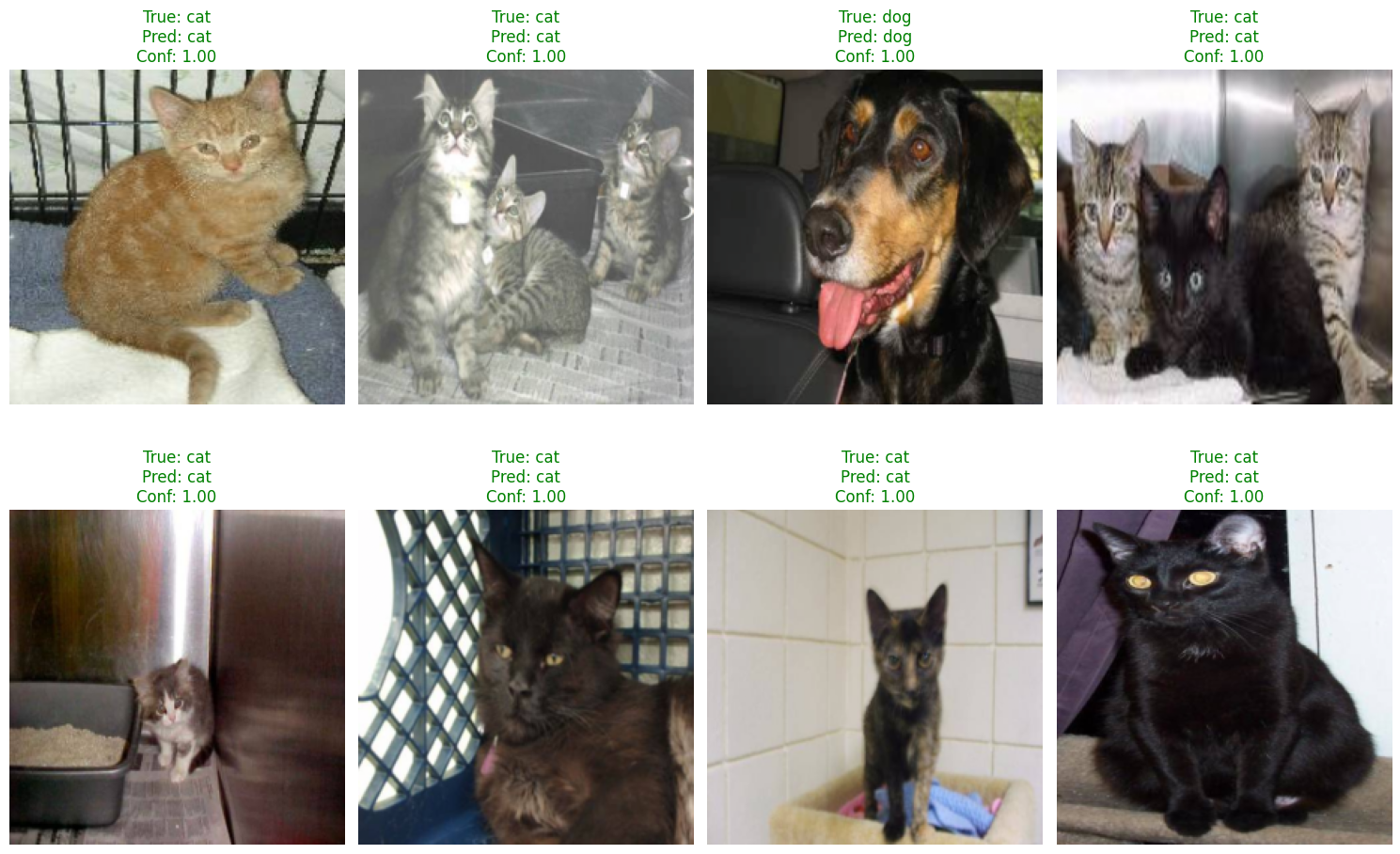
Val Loss: 0.0389 Acc: 0.9862

Learning Rate: 0.000050

Validation Accuracy: 98.74%



Примеры предсказаний:





Accuracy by class:

cat: 97.91%

dog: 99.59%

✅ Цель accuracy ≥ 97% достигнута! Final accuracy: 98.74%

Модель сохранена как 'resnet18\_cats\_dogs\_complete.pth'