AI Essentials: Theory and Applications
Final Project Report

Animals Pictures Classification Problem: ML Models Comparison

Eldar Iseev
AAI-2501M

https://github.com/xoyon/Ai-Essentials-Final-Eldar-.git

1. Введение

Классификация изображений из датасета Oxford-IIIT Pet требует устойчивых признаков для различия пород. Датасет плохо стандартизирован: ракурсы, освещение, пропорции и шум сильно варьируются. Сравнение простых и глубоких моделей показывает, как архитектурная сложность и предобученные признаки влияют на точность.

2. Данные и подготовка

Использовался датасет Oxford-IIIT Pet, содержащий 3680 изображений, распределённых по 37 классам. Данные были разделены на обучающую (80%), валидационную (10%) и тестовую (10%) выборки. Далее изображения масштабировались, нормализовались и преобразовывались для использования в тренировке.

```
train_transforms = transforms.Compose([
    transforms.RandomResizedCrop(IMG_SIZE),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(0.1,0.1,0.1),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485,0.456,0.406], std=[0.229,0.224,0.225]),
])

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

3. Описание и реализация моделей

RandomForest

Для классического подхода был выбран алгоритм RandomForest, который обучался на признаках, извлечённых из предобученного ResNet18. В коде использовался слой *avgpool* для получения векторных представлений изображений фиксированной размерности. Эти признаки затем подавались на вход ансамблю деревьев решений. Такой метод позволил использовать богатые визуальные представления, полученные на ImageNet, без необходимости дообучения самой нейросети. RandomForest был выбран как базовый ориентир, поскольку он хорошо работает на табличных данных и позволяет оценить качество признаков, извлечённых из глубокой модели.

```python
# Модель 1: RandomForest — извлечение признаков через предобученный ResNet18 (avgpool)
fe_extractor = timm.create_model("resnet18", pretrained=True, num_classes=0, global_pool="avg")
fe_extractor.eval()
fe_extractor.to(device)

def extract_features(loader, model, device):
    feats = []
    labels = []
    with torch.no_grad():
        for imgs, lbl in loader:
            imgs = imgs.to(device)
            f = model(imgs)  # output shape (N, feat_dim)
            feats.append(f.cpu().numpy())
            labels.append(lbl.numpy())
    feats = np.concatenate(feats, axis=0)
    labels = np.concatenate(labels, axis=0)
    return feats, labels

# Экстракция признаков
train_feats, train_labels = extract_features(train_loader, fe_extractor, device)
val_feats, val_labels = extract_features(val_loader, fe_extractor, device)
test_feats, test_labels = extract_features(test_loader, fe_extractor, device)

# Тренировка и оценка RandomForest
rf = RandomForestClassifier(n_estimators=200, n_jobs=-1, random_state=SEED)
rf.fit(train_feats, train_labels)
val_preds = rf.predict(val_feats)
print("RandomForest val accuracy:", accuracy_score(val_labels, val_preds))
joblib.dump(rf, os.path.join(MODEL_DIR, "random_forest_oxford_pet.joblib"))
```

CNN

Была реализована компактная архитектура CNN, включающая последовательность сверточных блоков с увеличением числа каналов, нормализацией и функцией активации ReLU. Для повышения устойчивости обучения добавлялись dropout-слои. Важным элементом стали residual-блоки, которые позволяли сохранять информацию из предыдущих слоёв и бороться с проблемой затухающих градиентов. В конце сети использовался адаптивный pooling, переводящий карты признаков в фиксированный размер, и полносвязный классификатор. Реализация задумывалась как проверка возможностей «ручной» архитектуры, созданной специально для задачи, без опоры на предобученные веса.

```
# Модель 2: CNN
class MyCNN(nn.Module):
    def __init__(self, num_classes):
        super().__init__()

        def conv_block(in_channels, out_channels, kernel_size=3, stride=1, padding=1, dropout=0.3):
            return nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding),
                nn.BatchNorm2d(out_channels),
                nn.ReLU(inplace=True),
                nn.Dropout(dropout)
            )

        # Residual блок
        class ResidualBlock(nn.Module):
            def __init__(self, channels):
                super().__init__()
                self.conv1 = nn.Conv2d(channels, channels, 3, padding=1)
                self.bn1 = nn.BatchNorm2d(channels)
                self.relu = nn.ReLU(inplace=True)
                self.conv2 = nn.Conv2d(channels, channels, 3, padding=1)
                self.bn2 = nn.BatchNorm2d(channels)
            def forward(self, x):
                identity = x
                out = self.relu(self.bn1(self.conv1(x)))
                out = self.bn2(self.conv2(out))
                out += identity
                return self.relu(out)

        self.feature = nn.Sequential(
            conv_block(3, 32),
            nn.MaxPool2d(2),

            conv_block(32, 64),
            ResidualBlock(64),
            nn.MaxPool2d(2),

            conv_block(64, 128),
            ResidualBlock(128),
            nn.MaxPool2d(2),

            conv_block(128, 256),
            ResidualBlock(256),
            nn.AdaptiveAvgPool2d(1)
```

ResNet50

Использовалась предобученная модель ResNet50, в которой финальный полносвязный слой был заменён на классификатор для 37 классов. В коде применялись оптимизатор AdamW, регуляризация через weight decay, а также планировщик ReduceLROnPlateau, снижающий скорость обучения при отсутствии улучшений. Дополнительно была реализована ранняя остановка, что позволило избежать переобучения и стабилизировать процесс обучения. ResNet50 был выбран как проверенный стандарт в задачах классификации изображений, способный извлекать богатые и устойчивые признаки.

```python
# Модель 3: ResNet50 с дообучением
from torchvision.models import resnet50, ResNet50_Weights

# предобученные веса
resnet = resnet50(weights=ResNet50_Weights.IMAGENET1K_V2)
resnet.fc = nn.Linear(resnet.fc.in_features, num_classes)

resnet = resnet.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.AdamW(resnet.parameters(), lr=LR, weight_decay=1e-4)

best_acc = 0.0
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode="max", factor=0.5, patience=2)

early_patience = 2
no_improve_epochs = 0
epochs_resnet = 10

for epoch in range(epochs_resnet):
    tr_loss, tr_acc = train_one_epoch(resnet, train_loader, criterion, optimizer, device)
    val_loss, val_acc = eval_model(resnet, val_loader, criterion, device)
    scheduler.step(val_acc)
    print(f"ResNet Epoch {epoch+1}/{epochs_resnet} | train acc {tr_acc:.4f} | val acc {val_acc:.4f}")

    if val_acc > best_acc:
        best_acc = val_acc
        torch.save(resnet.state_dict(), os.path.join(MODEL_DIR, "resnet50_oxford_pet.pth"))
        no_improve_epochs = 0
    else:
        no_improve_epochs += 1
        if no_improve_epochs >= early_patience:
            print(f"Early stopping на эпохе {epoch+1} (нет улучшений {early_patience} эпох подряд)")
            break
```

EfficientNet-B0

Эта архитектура была выбрана благодаря своей эффективности и сбалансированному увеличению глубины, ширины и разрешения. EfficientNet использует принцип compound scaling, что позволяет достигать высокой точности при меньшем числе параметров. В коде применялась предобученная версия из библиотеки timm, дообученная на датасете. Такой выбор был обусловлен стремлением протестировать современную CNN, оптимизированную по производительности и точности.

```python
# Модель 4: EfficientNet (timm) — efficientnet_b0
effnet = timm.create_model("efficientnet_b0", pretrained=True, num_classes=num_classes)
effnet = effnet.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.AdamW(effnet.parameters(), lr=LR, weight_decay=1e-4)

epochs_effnet = 10

best_acc = 0.0
for epoch in range(epochs_effnet):
    tr_loss, tr_acc = train_one_epoch(effnet, train_loader, criterion, optimizer, device)
    val_loss, val_acc = eval_model(effnet, val_loader, criterion, device)
    scheduler.step(val_acc)
    print(f"EffNet Epoch {epoch+1}/{epochs_effnet} | train acc {tr_acc:.4f} | val acc {val_acc:.4f}")
    if val_acc > best_acc:
        best_acc = val_acc
        torch.save(effnet.state_dict(), os.path.join(MODEL_DIR, "efficientnet_b0_oxford_pet.pth"))
print("Best EfficientNet val acc:", best_acc)
```

Vision Transformer (ViT)

ViT рассматривает изображение как последовательность патчей фиксированного размера и применяет механизм внимания для выявления глобальных зависимостей. В коде использовалась предобученная модель *vit_base_patch16_224*, адаптированная под 37 классов. Несмотря на то, что трансформеры требуют больших объемов данных для раскрытия своего потенциала, использование предобученных весов позволило протестировать данный подход на сравнительно небольшом датасете. ViT был включён в эксперименты как представитель нового поколения архитектур, чтобы оценить их применимость в условиях ограниченных данных.

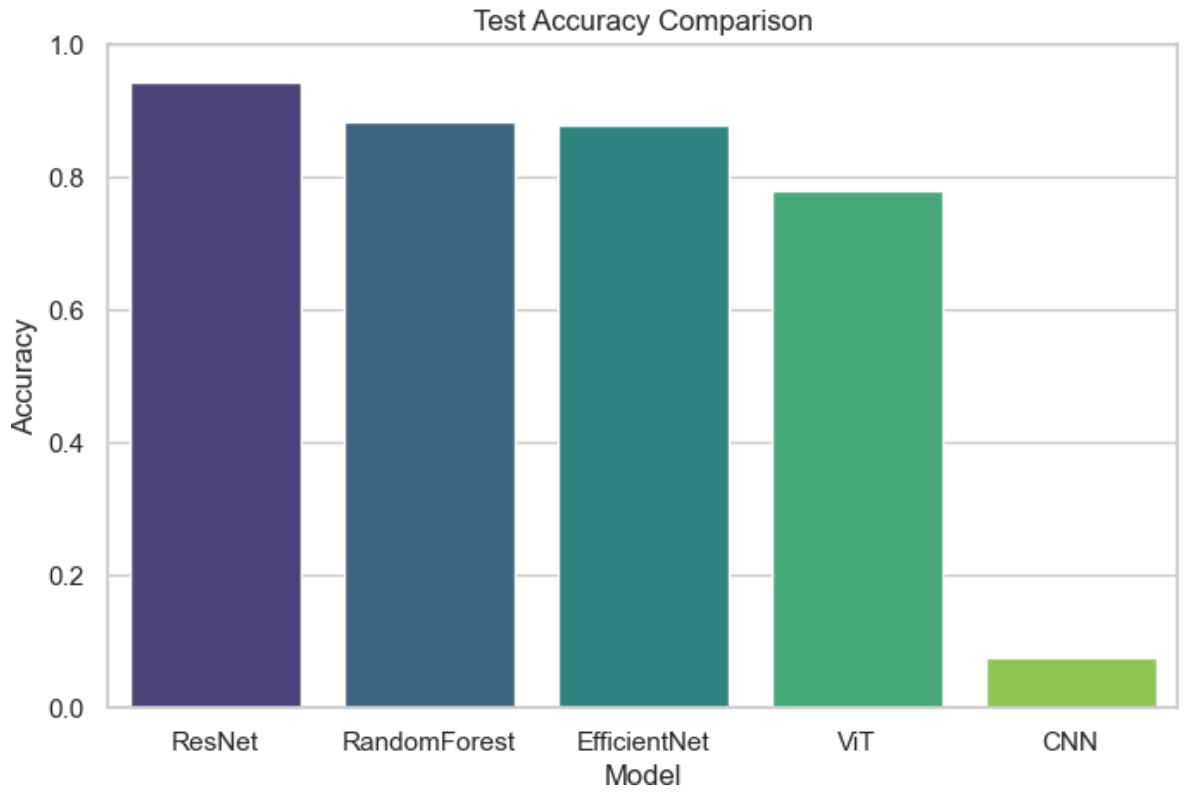
```
# Модель 5: ViT (timm) — vit_base_patch16_224
vit = timm.create_model("vit_base_patch16_224", pretrained=True, num_classes=num_classes)
vit = vit.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.AdamW(vit.parameters(), lr=LR, weight_decay=1e-4)

epochs_vit = 8

best_acc = 0.0
for epoch in range(epochs_vit):
    tr_loss, tr_acc = train_one_epoch(vit, train_loader, criterion, optimizer, device)
    val_loss, val_acc = eval_model(vit, val_loader, criterion, device)
    scheduler.step(val_acc)
    print(f"ViT Epoch {epoch+1}/{epochs_vit} | train acc {tr_acc:.4f} | val acc {val_acc:.4f}")
    if val_acc > best_acc:
        best_acc = val_acc
        torch.save(vit.state_dict(), os.path.join(MODEL_DIR, "vit_base_oxford_pet.pth"))
print("Best ViT val acc:", best_acc)
```

4. Результаты

	Model	Accuracy	Macro F1	Precision	Recall
0	ResNet	0.942935	0.936149	0.944829	0.936341
1	RandomForest	0.883152	0.870210	0.884662	0.869302
2	EfficientNet	0.877717	0.868136	0.876684	0.870694
3	ViT	0.779891	0.766280	0.825881	0.768417
4	CNN	0.076087	0.054233	0.094686	0.080269



5. Обсуждение

Результаты показали явное преимущество предобученных глубоких моделей. ResNet50 оказался лучшим, достигнув точности около 94%, что подтверждает его надежность и способность извлекать богатые признаки. RandomForest и EfficientNet показали схожие результаты, около 88%, что делает их конкурентоспособными, но менее точными. ViT продемонстрировал более низкую точность (78%), что связано с ограниченным размером датасета: трансформеры раскрывают потенциал при больших объёмах данных. Собственная CNN показала крайне низкие результаты (7%), что объясняется недостаточной глубиной и сложностью архитектуры для такой задачи. Таким образом, предобученные CNN остаются наиболее надежным выбором для классификации изображений при ограниченных данных.

6. Заключение

Эксперименты подтвердили, что использование предобученных моделей критически важно для достижения высокой точности на небольших датасетах. ResNet50 оказался наиболее успешным решением, EfficientNet и RandomForest показали достойные результаты, а ViT продемонстрировал

потенциал, но ограниченный данными. Собственная CNN не справилась с задачей, что подчеркивает важность архитектурной сложности и предобученных весов.

В целом, для классификации изображений животных на Oxford-IIIT Pet оптимальным выбором является ResNet50, сочетающий высокую точность и устойчивость обучения.