Face Recognition Project

О проекте

В данном проекте проводилось исследование различных механизмов распознавания лиц, в частности функционала, отличающего одни лица от других. Были протестированы стратегии обучения, функции потерь, архитектуры моделей и механизмы предобработки данных.

Данные

Для обучения и тестирования использовался датасет CelebA, содержащий изображения 500 разных людей.

Примеры изображений из датасета CelebA-500

Так как в проекте используется только система распознавания, изображения из исходного датасета были предварительно преобразованы так, что на них

оставались только лица, выровненные по опорным точкам.

При обучении модели использовалась сокращенная версия датасета из 12к изображений. Для борьбы с переобучением использовались аугментации по цвету, что позволило увеличить размер датасета в 6 раз.

Был опробован вариант комбинирования разных аугментаций, однако таким образом в датасете измененных изображений становилось сильно больше, чем истинных, что делало обучение нестабильным. Поэтому во всех экспериментах по умолчанию использовался вариант из 5 основных аугментаций из библиотеки PyTorch:

```
"augmentations":[T.Grayscale(3), T.ColorJitter(brightness=.5, hu
T.GaussianBlur(kernel_size=(5, 9), sigma=(0.1, 5.)),T.RandomPost
T.RandomEqualize()]
```

Базовая архитектура

В качестве базовой предобученной модели были опробованы следующие варианты из модуля torchvision.models: resnet38, resnet34, resnet50, efficientnet_b0, efficientnet_v2_m. Лучше всего по качеству и стабильности работы оказался resnet34.

Последний классификационный слой заменялся на другой линейный слой для контроля размерности эмбеддингов. При этом эксперименты показали, что один линейный слой в качестве последнего работает лучше, чем последовательность из нескольких - качество классификации при использовании одного слоя было в 1.5 раза больше. После линейного слоя добавлялся классификационный слой.

Стратегии обучения

Оптимизатор

В качестве отправной точки экспериментов использовался Adam с $lr = 3 * 10^{-4}$. В ходе экспериментов сравнивались Adam и AdamW с различными значениями

weight_decay в диапазоне $10^{-6} - 1$. Лучше всего по качеству и стабильности работы оказался AdamW с weight_decay=0.01.

Стратегия изменения шага

Переменный шаг градиентного спуска использовался для борьбы с переобучением. Тестировалось циклическое изменение шага с различными значениями гиперпараметров:

```
"scheduler":torch.optim.lr_scheduler.CosineAnnealingWarmRestarts
"scheduler_args":{"T_0":3, "eta_min":3e-5},
```

Применение такого расписания сказывалось на обучении следующим образом: при резком повышении шага после очередного цикла качество на обучении и валидации резко снижалось, а затем снова начинало расти. При этом в масштабах всего обучения качество стабильно росло. Ожидалось, что периодическое резкое увеличение шага позволит выйти из локальных минимумов и повысить обобщающую способность модели. Однако через 10-15 эпох резкое увеличение шага не вызывало изменения качества на тестовой выборке, и отрыв от качества на валидационной выборке не сокращался, то есть переобучение продолжало происходить. Таким образом, изменение шага не помогло с переобучением, при этом само обучение дестабилизировалось, поэтому от этого приема пришлось отказаться.

Заморозка слоев модели

Заморозка слоев также использовалась для подавления переобучения. Была опробована заморозка первых сверточных слоев на всем цикле обучения и заморозка всех слоев кроме последнего с последовательной разморозкой. Такой прием замедлял обучение, но переобучение не исчезало, поэтому от него тоже пришлось отказаться. При этом обнаружился тот факт, что постоянная заморозка первого слоя только усиливает переобучение.

Функции потерь

Cross Entropy Loss

ArcFace Loss

Данный метод обучения позволил достичь точности 0.7 на валидационной выборке к 43 эпохе. Были использованы следующие гиперпараметры:

```
batch_size = 16
path = "/content/celebA_train_500"
train_data_params = {
    "path":path,
    "augmentations":[T.Grayscale(3), T.ColorJitter(brightness=.5
                     T.RandomPosterize(bits=2), T.RandomEqualize
    "compose_augmentations":False
}
train_dataset = FaceDataset(train_data_params, train=True)
train_dataloader = DataLoader(train_dataset, batch_size=batch_si

trainer_args = {
    "train_dataloader":train_dataloader,
    "val_dataloader":val_dataloader,
    "score":AccuracyScore(),
    "optimizer":torch.optim.AdamW,
    "optimizer_args":{"lr":3e-4, "weight_decay":0.01},
    "scheduler":None,
    "scheduler_args":None,
    "freezer":None,
    "freezer_args":None,
    "n_epochs":50,
    "device":"cuda" if torch.cuda.is_available() else "cpu",
    "backup_path":"/content/drive/MyDrive/Face Recognition/recog
}

trainer = Trainer(trainer_args)

model_args = {
```

```
    "encoder":resnet34(),
    "encoder_emb_size":512,
    "encoder_last_layer_name":"fc",
    "n_classes":500,
    "s":5,
    "m":0.1
}
model = Recognizer_ArcFaceLoss(model_args)
```

```
Epoch 43. Train loss: 2.0487506076954087, val loss: 3.3176913356149442, train score: 0.9997571468220611, val score: 0.7018487858719642
Training: 100% 3603/3603 [04:38<00:00, 16.12it/s]
Validation: 100% 151/151 [00:06<00:00, 20.00it/s]
Epoch 44. Train loss: 2.048690522540915, val loss: 3.3230715397967403, train score: 0.9998265334443205, val score: 0.7043322295805733
```