**Классификация печатного и рукописного текста**

**Задача**

На вход поступают отсканированные изображения, необходимо понять, что на них находится, рукописный текст или печатный. Это задача классификации.

**Модель**

Для классификации изображений будем использовать предобученную модель ResNeXt50.

model = models.resnext50\_32x4d(pretrained=**True**)

Для последнего dense-слоя установим размерность (2048, 2), так как мы предсказываем 2 класса.

model.fc = nn.Linear(2048, 2)

А также укажем модели использовать видеокарту.

device = torch.device(**"cuda:0"**)  
model.to(device)

**Аугментация**

Во время тренировки будем использовать аугментацию, так как модель будет использоваться для распознания отсканированных изображений, и данная аугментация поможет имитировать дефекты сканирования, а также неровно расположенные бумаги во время сканирования. В частности, будем использовать:

* отражение по горизонтали;
* изменение яркости и контраста;
* изменение гаммы;
* удаление частей изображения;
* шум;
* поворот на случайный угол;
* сжатие.

Также все изображения будут приведены к размеру 1024x1024.

**def** get\_train\_augmentations(image\_size):  
 **return** Compose([  
 Resize(image\_size, image\_size),  
 HorizontalFlip(p=0.5),  
 RandomBrightnessContrast(p=0.4, brightness\_limit=0.25, contrast\_limit=0.3),  
 RandomGamma(p=0.4),  
 CoarseDropout(p=0.1, max\_holes=8, max\_height=8, max\_width=8),  
 GaussNoise(p=0.1, var\_limit=(5.0, 50.0)),  
 ShiftScaleRotate(shift\_limit=0.1, scale\_limit=0.15, rotate\_limit=45, p=0.8),  
 ImageCompression(quality\_lower=80, quality\_upper=100, p=0.4),  
 Normalize(  
 mean=[0.485, 0.456, 0.406],  
 std=[0.229, 0.224, 0.225],  
 ),  
 ToTensorV2(),  
 ])

**Подготовка данных**

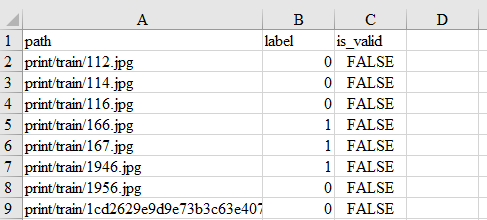
Разобьем данные на тренировочную и валидационную выборки. Так как в доступном наборе данных преобладает нецелевой класс, для обучения будем использовать урезанную выборку, чтобы соотношение классов было одинаковым.

В итоге используем данные следующей размерности:

* Класс 0 (печатный текст):
  + Тrain: 100;
  + Val: 30.
* Класс 1 (рукописный текст):
  + Тrain: 100;
  + Val: 30.

Составим csv файл, который будет содержать следующие столбцы:

* путь к изображению;
* класс;
* принадлежность изображения к валидационной выборке.



Сформируем тренировочный и валидационный DataFrame, а также классы изображений.

**def** preprocess\_data(path\_to\_csv):  
 df = pd.read\_csv(path\_to\_csv, delimiter=**';'**)  
 train\_df = df[df[**'is\_valid'**] == **False**]  
 val\_df = df[df[**'is\_valid'**] == **True**]  
 le = preprocessing.LabelEncoder()  
 le = le.fit(train\_df[**'label'**].values)  
 train\_labels = le.transform(train\_df[**'label'**].values)  
 val\_labels = le.transform(val\_df[**'label'**].values)  
 **return** train\_df, val\_df, train\_labels, val\_labels

train\_df, val\_df, train\_labels, val\_labels = preprocess\_data(**'/images.csv'**)

И, наконец, сформируем загрузчики данных для обучения и валидации.

train\_data = TextDataset(dataframe=train\_df,  
 labels=train\_labels,  
 path=**'/input'**,  
 transform=albumentations\_transform)

train\_loader = DataLoader(dataset=train\_data,  
 batch\_size=BATCH\_SIZE,  
 num\_workers=NUM\_WORKERS,  
 shuffle=**True**,  
 drop\_last=**False**)

Здесь TextDataset – класс, унаследованный от torch.utils.data.Dataset, в котором реализованы функции \_\_getitem\_\_ и \_\_len\_\_.

**Обучение**

Во время обучения будем использовать оптимизатор AdamW, критерий CrossEntropyLoss и планировщик CosineAnnealingWarmRestarts.

optimizer = torch.optim.AdamW(model.parameters(), lr=0.00001)  
criterion = nn.CrossEntropyLoss()  
scheduler = torch.optim.lr\_scheduler.CosineAnnealingWarmRestarts(optimizer=optimizer, T\_0=200)

В качестве метрики будем использовать точность (accuracy).

Код тренировки модели выглядит следующим образом:

**for** epoch **in** range(N\_EPOCHS):  
 model.train()  
 train\_loss = 0  
 train\_acc = 0  
  
 **for** i, (imgs, labels) **in** tqdm(enumerate(train\_loader), total=train\_len):  
 imgs = imgs.to(device)  
 labels = labels.to(device)  
 optimizer.zero\_grad()  
 output = model(imgs)  
 loss = criterion(output, labels)  
 loss.backward()  
 optimizer.step()  
 train\_loss += loss.item()  
 pred = torch.argmax(torch.softmax(output, 1), 1).cpu().detach().numpy()  
 true = labels.cpu().numpy()  
 train\_acc += accuracy\_score(true, pred)  
 scheduler.step(epoch + i / train\_len)

В нашем случае модель обучалась 50 эпох с размером батча 4. В результате тренировки модель достигла точности 98% на отложенной выборке.

**Классификация**

Для классификации изображений при помощи уже дообученной модели сначала изменим атрибут «pretrained» на «False».

model = models.resnext50\_32x4d(pretrained=**False**)  
model.fc = nn.Linear(2048, 2)  
*# model = nn.DataParallel(model, device\_ids=device\_ids, output\_device=device)*

Загрузим веса, полученные на дообучении и загрузим их.

checkpoint = torch.load(**'/model\_saved/weight\_best.pth'**)  
model.load\_state\_dict(checkpoint)  
model.to(device)  
criterion = nn.CrossEntropyLoss()

Для получения вероятности отнесения изображения к конкретному классу использем слой Softmax.

classifier = torch.nn.Softmax()

И, наконец, код классификации будет выглядеть следующим образом:

**with** torch.no\_grad():  
 imgs = imgs.to(device)  
 output\_test = model(imgs)preds = torch.argmax(classifier(output\_test), dim=1)preds\_array.append(preds)  
 preds\_paths.append(filepaths)  
 **for** index, predict **in** enumerate(preds):**if** predict:  
 hand.append(filepaths[index])  
 shutil.copy(filepaths[index], **r'/hand'**)  
 **else**:  
 shutil.copy(filepaths[index], **r'/print'**)

В результате работы скрипта, изображения из исходной папки будут отсортированы в папки hand и print соответственно.

**Библиотеки**

Для дообучения и инференса были использованы следующие библиотеки:

* Torch;
* Sklearn;
* Pandas;
* Albumentations;
* OpenCV;
* OS;
* Torchvision;
* Shutil.2