Competition

Шаронова Александра ФКИ501

Датасет

Аугментации из библиотеки Albumentations:

- цвет
- контраст


```python
class pizza_dataset(Dataset):
    def __init__(self,transform=None,mode='train'):
        self.mode = mode
        self.data_dir = None
        if self.mode == 'train':
            self.data_dir = 'train/train_256/'
        if self.mode =='test':
            self.data_dir = 'test/test_256/'
        self.transform = transform
        self.images = glob.glob(self.data_dir+'*')
        if self.mode == 'train':
            self.class_labels  = pd.read_csv('train.csv')

    def __len__(self):
        return len(self.images)

    def __getitem__(self,ind):
        if self.mode =='train':
            image = np.array(Image.open(self.images[ind]))
            if self.transform is not None:
                image = self.transform(image=image)
                image = image['image']
            cls = self.class_labels['label'][ind]
            return image,torch.tensor(cls,dtype = torch.long)
        if self.mode == 'test':
            image = np.array(Image.open(self.images[ind]))
            if self.transform is not None:
                image = self.transform(image=image)
                image = image['image']
            return image

train_transform = A.Compose([
    A.RandomBrightnessContrast(p=0.2,brightness_limit=0.2,contrast_limit=0.2),
    A.HueSaturationValue(hue_shift_limit=10,sat_shift_limit=15,val_shift_limit=10,p=0.2),
    A.Normalize(mean=[0.66223061, 0.53081928, 0.40509563], std=[0.21449153, 0.22278006, 0.208873  ]),
    ToTensorV2(),
])

test_transform = A.Compose([
    A.Normalize(mean=[0.66223061, 0.53081928, 0.40509563], std=[0.21449153, 0.22278006, 0.208873  ]),
    ToTensorV2(),

])
```

Пайплайн обучения:

Lightning с лучшей на валидации моделью.

```python
def train_model(model,log_save_dir,exp_name,max_epochs = 10,monitor="MulticlassAccuracy/val",
                return_lightning_model = True,opt_cls = torch.optim.AdamW,lr=1e-3,opt_type = 'AdamW'):
    checkpoint_callback = ModelCheckpoint(
        monitor=monitor, mode="max", filename="model"
    )
    if opt_type !='ranger21':
        pl_model = LModel(model,optimizer_cls = opt_cls,lr = lr,max_epochs=max_epochs)
    else:
        pl_model = LModel(model,
        optimizer_cls= partial(my_optimizer_ranger,
                               num_epochs=max_epochs,
                               num_batches_per_epoch=len(train_loader),
                               lr=lr),
        lr=lr)

    trainer = L.Trainer(
        max_epochs=max_epochs,
        callbacks=[checkpoint_callback],
        num_sanity_val_steps=0,
        log_every_n_steps=10,
        logger=L.pytorch.loggers.TensorBoardLogger(save_dir=log_save_dir,name=exp_name),
        enable_progress_bar=True,
        precision="16-mixed"
    )
    trainer.fit(model = pl_model,train_dataloaders=train_loader,val_dataloaders=val_loader)
    if return_lightning_model:
        return pl_model
    else:
        return pl_model.model
```

```python
class LModel(L.LightningModule):
    def __init__(self, model, lr=0.001,optimizer_cls=torch.optim.AdamW,max_epochs = 30):
        super().__init__()
        self.save_hyperparameters(logger=False, ignore=["model"])
        self.lr = lr
        self.model = model
        self.criterion = nn.CrossEntropyLoss()
        metrics = MetricCollection([
            MulticlassAccuracy(num_classes=46,),
            MulticlassF1Score(num_classes=46,),

        ])
        self.train_metrics = metrics.clone(postfix="/train")
        self.val_metrics = metrics.clone(postfix="/val")
        self.test_metrics = metrics.clone(postfix="/test")
        self.optimizer_cls = optimizer_cls
        self.max_epochs = max_epochs

    def forward(self, x):
        return self.model(x)

    def configure_optimizers(self):
        optimizer = self.optimizer_cls(
            self.model.parameters(),
            lr=self.lr,
            )
        scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
            optimizer,
        )
        return {
            "optimizer": optimizer,
            "lr_scheduler": {
                "scheduler": scheduler,
                "interval": "epoch",
                "monitor": "loss"
            },
        }

    def training_step(self, batch, batch_idx):
        x,y = batch
        out = self.model(x)
        loss = self.criterion(out, y)
        self.train_metrics.update(out, y)
        self.log("loss", loss, prog_bar=True)
        return loss

    def validation_step(self, batch, batch_idx):
        x, y = batch
        out = self.model(x)
        self.val_metrics.update(out, y)

    def on_train_epoch_end(self):
        self.log_dict(self.train_metrics.compute())
        self.train_metrics.reset()

        val_metrics = self.val_metrics.compute()
        self.log_dict(val_metrics)
        self.val_metrics.reset()

    def test_step(self, batch, batch_idx):
        x, y = batch
        out = self.model(x)
        self.test_metrics.update(out, y)

    def on_test_epoch_end(self):
        self.log_dict(self.test_metrics.compute())
        self.test_metrics.reset()
```

EfficientNet-B1

```
efficientnet_model = torchvision.models.efficientnet_b1(pretrained = True)

efficientnet_model.classifier[1] = nn.Linear(efficientnet_model.classifier[1].in_features, 46)
```

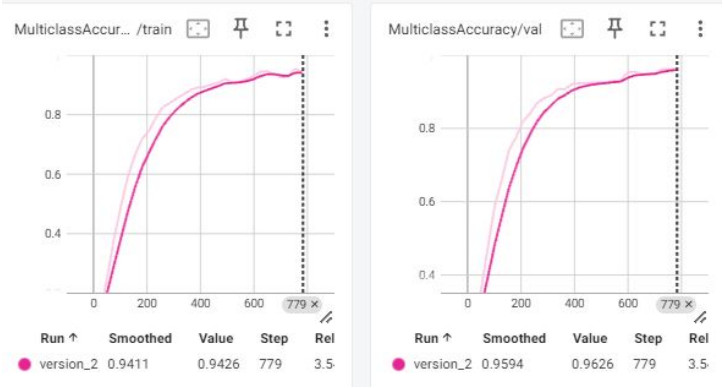
Параметры обучения:

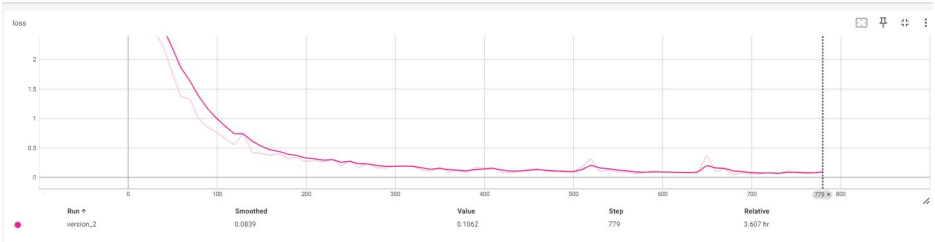
- num_epochs - 30
- learning rate = 1e-4
- оптимизатор - AdamW

Количество обучаемых параметров: 6.6M

Скоры:

- public - 0.97003
- private - 0.97378





MobileNet-V3(modified)

Параметры обучения:

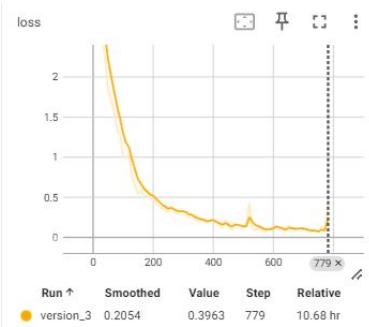
- num_epochs - 30
- learning rate = 1e-4
- оптимизатор - AdamW

Количество обучаемых параметров: 3M

Скоры:

- public - 0.96161
- private - 0.96816

```python
class ModifiedMobileNet(nn.Module):
    def __init__(self, num_classes=46, dropout=0.3, model_type='small'):
        super().__init__()
        if model_type == 'small':
            mobilenet = torchvision.models.mobilenet_v3_small(pretrained=True)
        elif model_type == 'large':
            mobilenet = torchvision.models.mobilenet_v3_large(pretrained=True)
        else:
            raise ValueError(f"Unsupported model_type: {model_type}")

        self.features = mobilenet.features
        self.pool = nn.AdaptiveAvgPool2d(1)

        last_channel = mobilenet.classifier[0].in_features

        self.bn = nn.BatchNorm2d(last_channel)
        self.relu = nn.ReLU(inplace=True)
        self.dropout = nn.Dropout(dropout)
        self.fc = nn.Linear(last_channel, num_classes)

    def forward(self, x):
        x = self.features(x)
        x = self.pool(x)
        x = self.bn(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x


mobilenet_model = ModifiedMobileNet(model_type='large')
```



DenseNet-121

Параметры обучения:

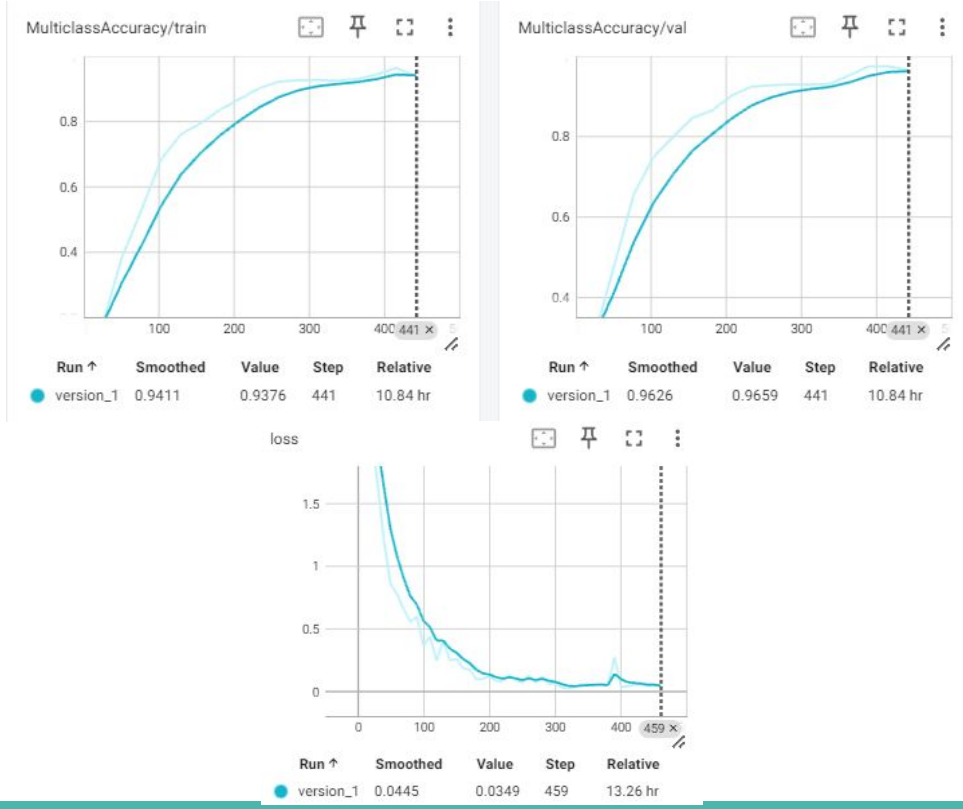
- num_epochs - 17
- learning rate = 1e-4
- оптимизатор - AdamW

Количество обучаемых параметров: 7M

Скоры:

- public - 0.93913
- private - 0.94382

```
densenet_model = torchvision.models.densenet121(pretrained = True)

densenet_model.classifier = nn.Linear(densenet_model.classifier.in_features, 46)
```



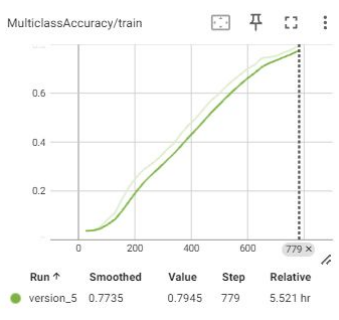
EffecientNet-B1 with Ranger21

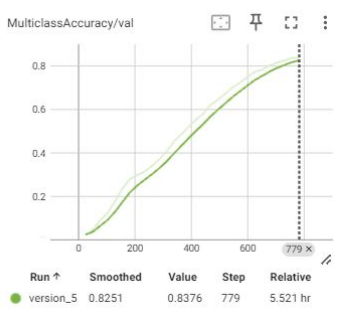
Параметры обучения:

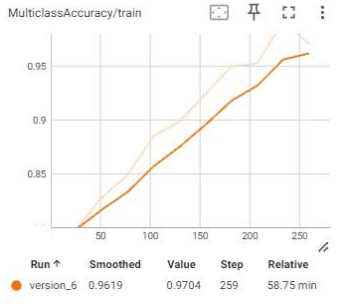
- Первый этап:
  - num_epochs - 30
  - learning rate = 1e-4
  - оптимизатор - Ranger21
- Второй этап:
  - num_epochs - 10
  - learning_rate = 1e-3
  - оптимизатор - Ranger21

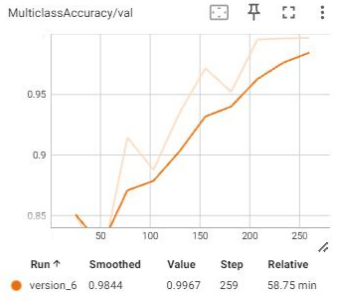
Скоры:

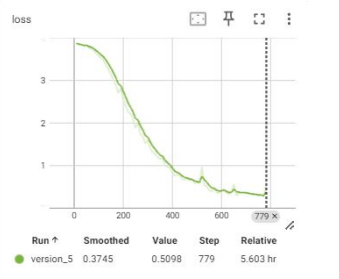
- public - 0.96722
- private - 0.97284

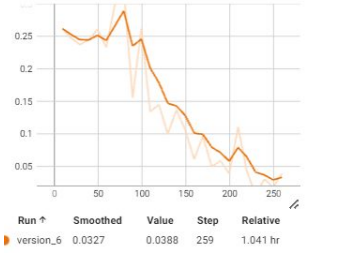




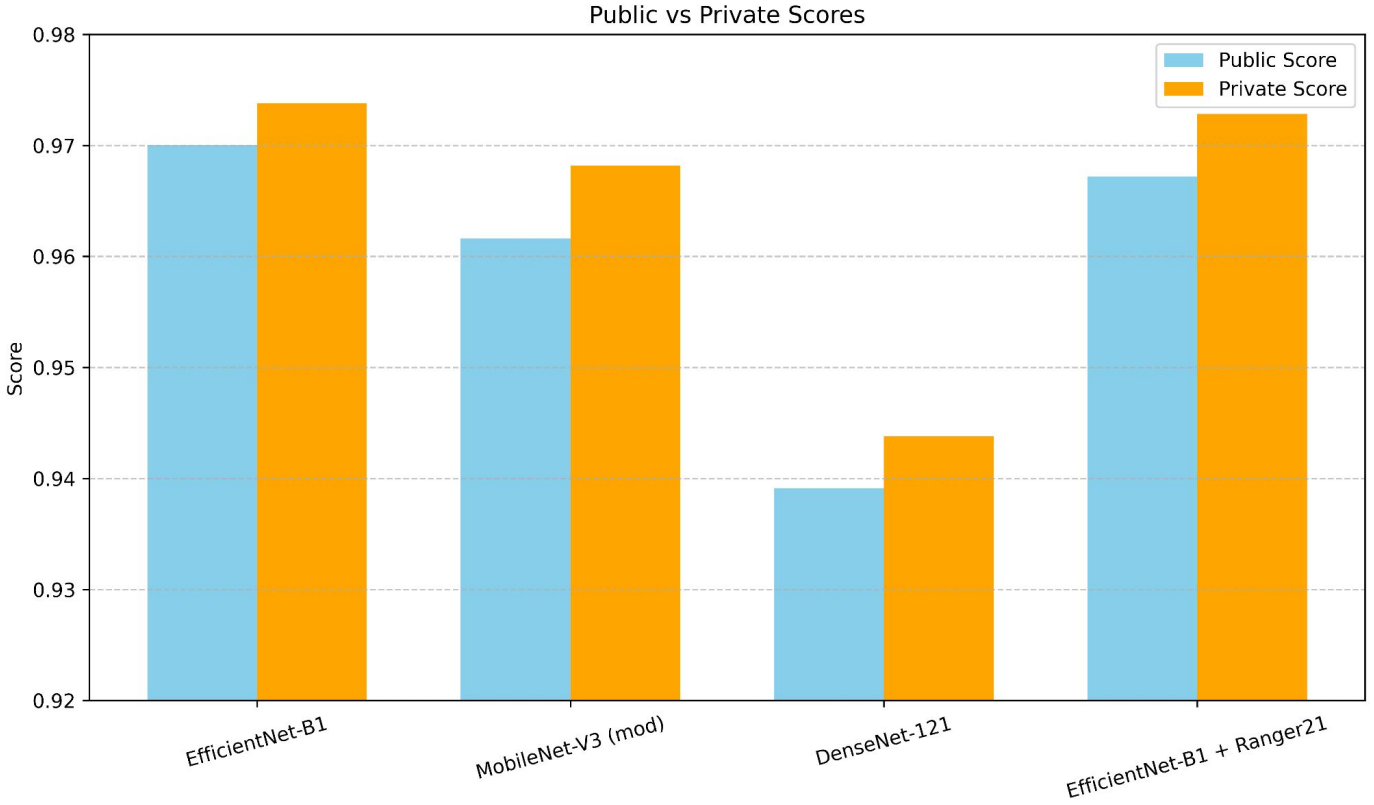








Итог


Public vs Private Scores
Public Score
Private Score
Score
0.98
0.97
0.96
0.95
0.94
0.93
0.92
EfficientNet-B1
MobileNet-V3 (mod)
DenseNet-121
EfficientNet-B1 + Ranger21