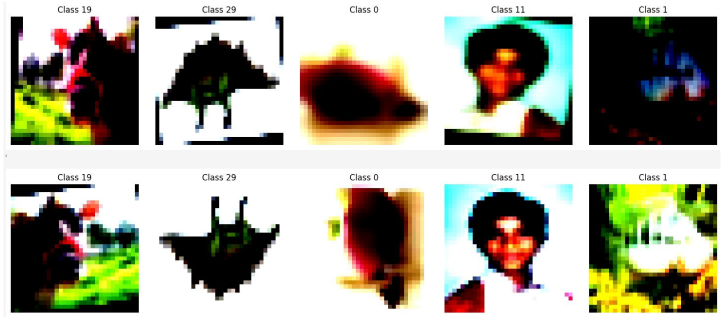
Research task

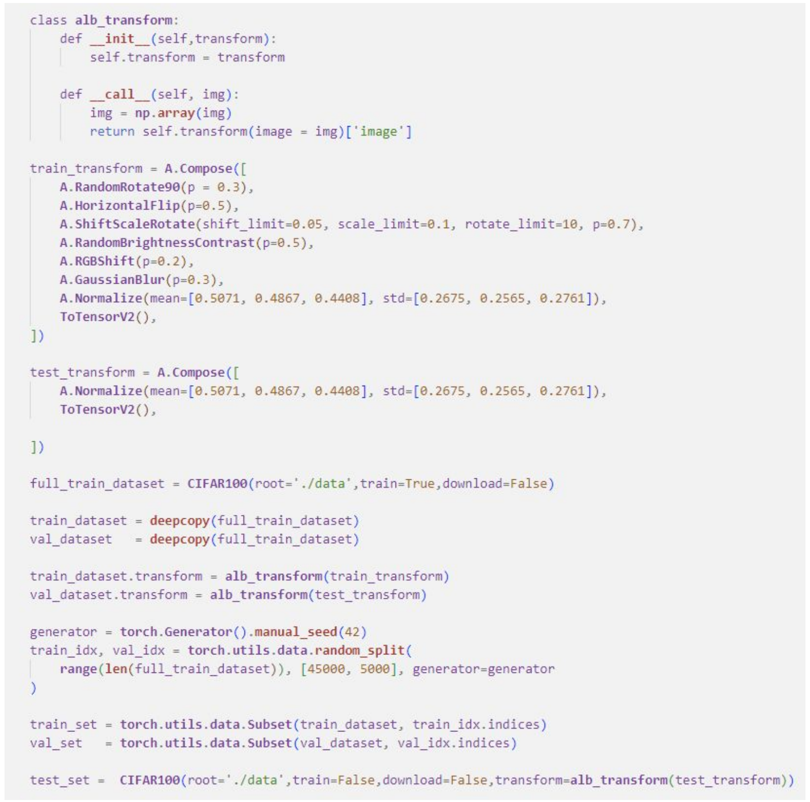
Шаронова Александра ФКИ 501

Работа с датасетом

Аугментации из библиотеки Albumentations:

- геометрические
- цвет/контраст
- размытие
- нормализация



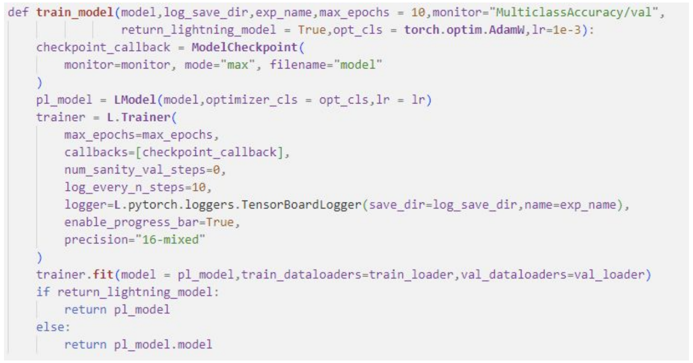


```
class alb_transform:
    def __init__(self,transform):
        self.transform = transform

    def __call__(self, img):
        img = np.array(img)
        return self.transform(image = img)['image']

train_transform = A.Compose([
    A.RandomRotate90(p = 0.3),
    A.HorizontalFlip(p=0.5),
    A.ShiftScaleRotate(shift_limit=0.05, scale_limit=0.1, rotate_limit=10, p=0.7),
    A.RandomBrightnessContrast(p=0.5),
    A.RGBShift(p=0.2),
    A.GaussianBlur(p=0.3),
    A.Normalize(mean=[0.5071, 0.4867, 0.4408], std=[0.2675, 0.2565, 0.2761]),
    ToTensorV2(),
])

test_transform = A.Compose([
    A.Normalize(mean=[0.5071, 0.4867, 0.4408], std=[0.2675, 0.2565, 0.2761]),
    ToTensorV2(),

])

full_train_dataset = CIFAR100(root='./data',train=True,download=False)

train_dataset = deepcopy(full_train_dataset)
val_dataset   = deepcopy(full_train_dataset)

train_dataset.transform = alb_transform(train_transform)
val_dataset.transform = alb_transform(test_transform)

generator = torch.Generator().manual_seed(42)
train_idx, val_idx = torch.utils.data.random_split(
    range(len(full_train_dataset)), [45000, 5000], generator=generator
)

train_set = torch.utils.data.Subset(train_dataset, train_idx.indices)
val_set   = torch.utils.data.Subset(val_dataset, val_idx.indices)

test_set =  CIFAR100(root='./data',train=False,download=False,transform=alb_transform(test_transform))
```

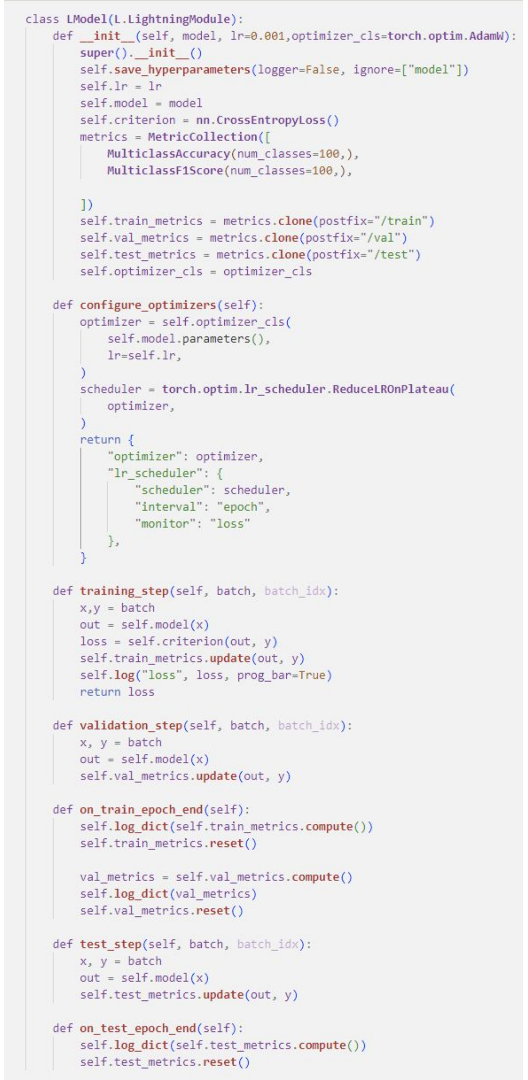
Пайплайн обучения:

Lightning:

- сохранялась лучшая на валидации модель



```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):
    checkpoint_callback = ModelCheckpoint(
        monitor=monitor, mode="max", filename="model"
    )
    pl_model = LModel(model,optimizer_cls = opt_cls,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
```



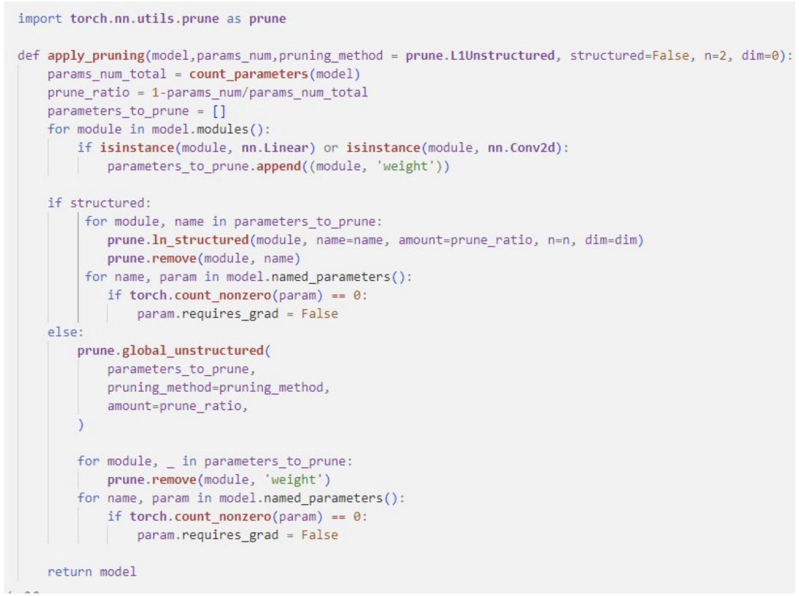
```python
class LModel(L.LightningModule):
    def __init__(self, model, lr=0.001,optimizer_cls=torch.optim.AdamW):
        super().__init__()
        self.save_hyperparameters(logger=False, ignore=["model"])
        self.lr = lr
        self.model = model
        self.criterion = nn.CrossEntropyLoss()
        metrics = MetricCollection([
            MulticlassAccuracy(num_classes=100,),
            MulticlassF1Score(num_classes=100,),

        ])
        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

    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()
```

Пайплайн обучения:

Pruning (для заданного количества обучаемых параметров):

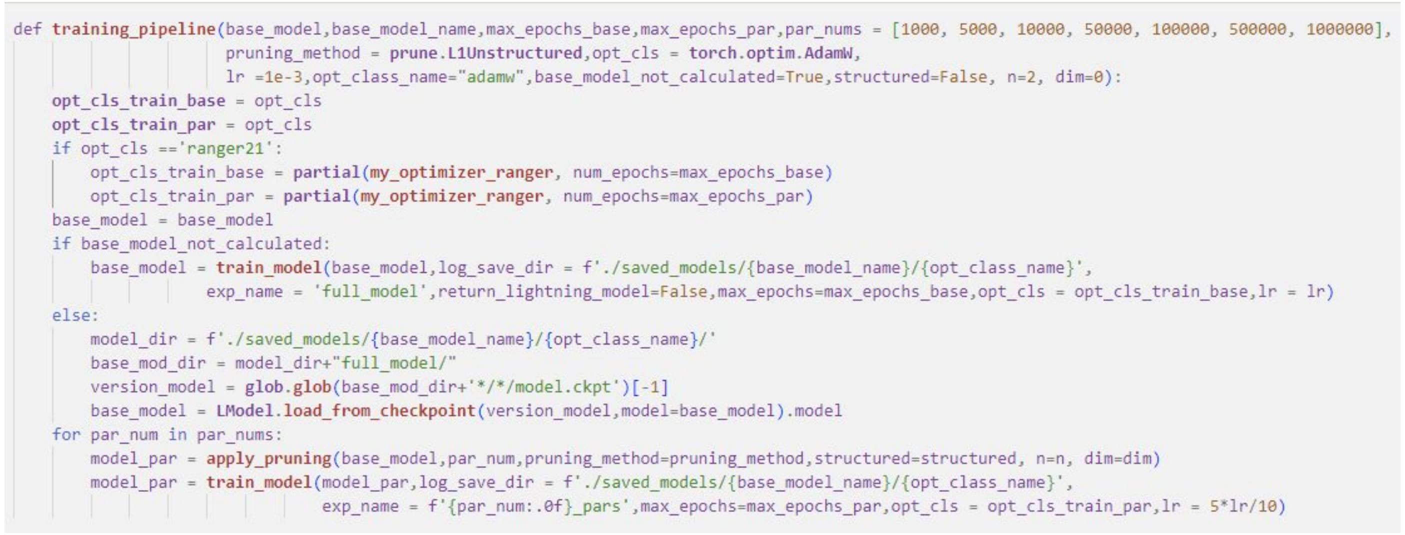
- prune.L1Unstructured

- prune.ln_structured



```python
import torch.nn.utils.prune as prune

def apply_pruning(model,params_num,pruning_method = prune.L1Unstructured, structured=False, n=2, dim=0):
    params_num_total = count_parameters(model)
    prune_ratio = 1-params_num/params_num_total
    parameters_to_prune = []
    for module in model.modules():
        if isinstance(module, nn.Linear) or isinstance(module, nn.Conv2d):
            parameters_to_prune.append((module, 'weight'))

    if structured:
         for module, name in parameters_to_prune:
            prune.ln_structured(module, name=name, amount=prune_ratio, n=n, dim=dim)
            prune.remove(module, name)
         for name, param in model.named_parameters():
            if torch.count_nonzero(param) == 0:
                param.requires_grad = False
    else:
        prune.global_unstructured(
            parameters_to_prune,
            pruning_method=pruning_method,
            amount=prune_ratio,
        )

        for module, _ in parameters_to_prune:
            prune.remove(module, 'weight')
        for name, param in model.named_parameters():
            if torch.count_nonzero(param) == 0:
                param.requires_grad = False

    return model
```

Пайплайн обучения



```
def training_pipeline(base_model,base_model_name,max_epochs_base,max_epochs_par,par_nums = [1000, 5000, 10000, 50000, 100000, 500000, 1000000],
                      pruning_method = prune.L1Unstructured,opt_cls = torch.optim.AdamW,
                      lr =1e-3,opt_class_name="adamw",base_model_not_calculated=True,structured=False, n=2, dim=0):
    opt_cls_train_base = opt_cls
    opt_cls_train_par = opt_cls
    if opt_cls =='ranger21':
        opt_cls_train_base = partial(my_optimizer_ranger, num_epochs=max_epochs_base)
        opt_cls_train_par = partial(my_optimizer_ranger, num_epochs=max_epochs_par)
    base_model = base_model
    if base_model_not_calculated:
        base_model = train_model(base_model,log_save_dir = f'./saved_models/{base_model_name}/{opt_class_name}',
                    exp_name = 'full_model',return_lightning_model=False,max_epochs=max_epochs_base,opt_cls = opt_cls_train_base,lr = lr)
    else:
        model_dir = f'./saved_models/{base_model_name}/{opt_class_name}/'
        base_mod_dir = model_dir+"full_model/"
        version_model = glob.glob(base_mod_dir+'*/*/model.ckpt')[-1]
        base_model = LModel.load_from_checkpoint(version_model,model=base_model).model
    for par_num in par_nums:
        model_par = apply_pruning(base_model,par_num,pruning_method=pruning_method,structured=structured, n=n, dim=dim)
        model_par = train_model(model_par,log_save_dir = f'./saved_models/{base_model_name}/{opt_class_name}',
                            exp_name = f'{par_num:.0f}_pars',max_epochs=max_epochs_par,opt_cls = opt_cls_train_par,lr = 5*lr/10)
```

Число эпох обучения модели до прунинга - 20 эпох.
Дообучение модели после прунинга - 10 эпох.

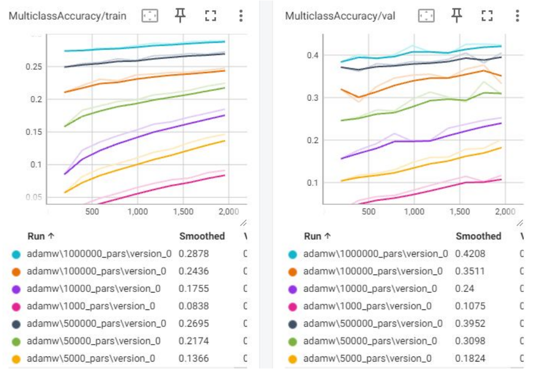
Используемые архитектуры: ResNet-18

Изменения:

- нет свертки 7x7 в начале.
- нет Max Pooling в начале.

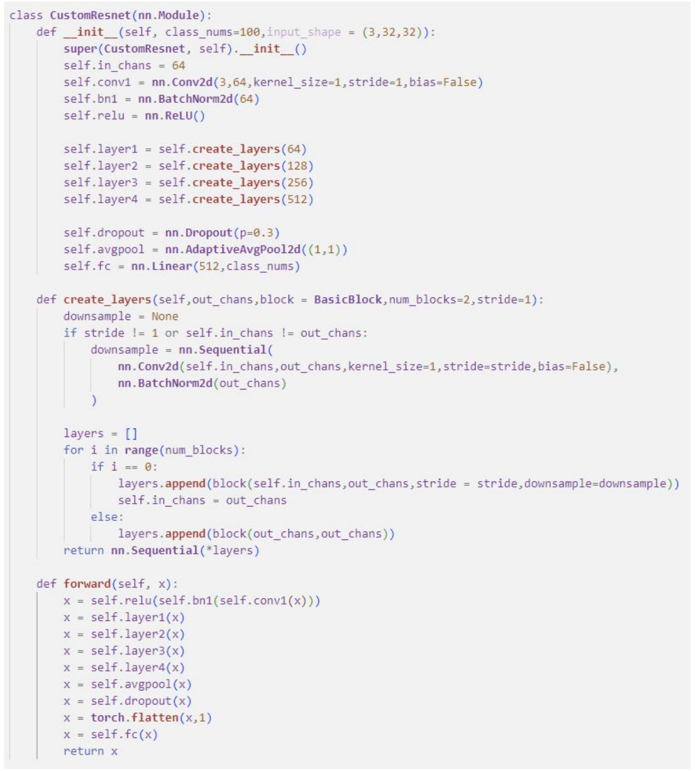
Структура блоков не менялась.

Прунинг - prune.L1Unstructured



Run ↑	Smoothed
● adamw\1000000_pars\version_0	0.2878
● adamw\100000_pars\version_0	0.2436
● adamw\10000_pars\version_0	0.1755
● adamw\1000_pars\version_0	0.0838
● adamw\500000_pars\version_0	0.2695
● adamw\50000_pars\version_0	0.2174
● adamw\5000_pars\version_0	0.1366

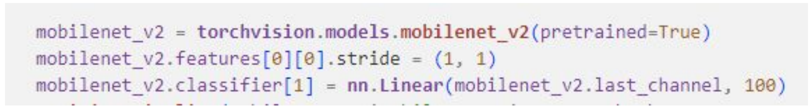
Run ↑	Smoothed
● adamw\1000000_pars\version_0	0.4208
● adamw\100000_pars\version_0	0.3511
● adamw\10000_pars\version_0	0.24
● adamw\1000_pars\version_0	0.1075
● adamw\500000_pars\version_0	0.3952
● adamw\50000_pars\version_0	0.3098
● adamw\5000_pars\version_0	0.1824



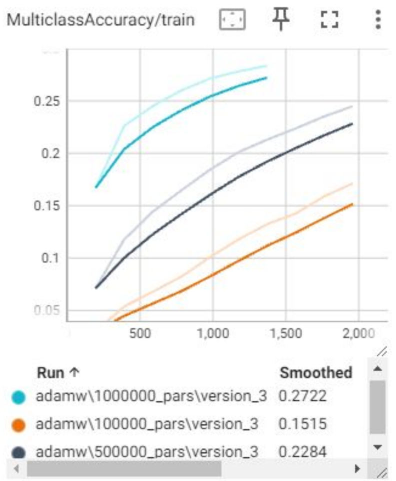
```python
class CustomResnet(nn.Module):
    def __init__(self, class_nums=100,input_shape = (3,32,32)):
        super(CustomResnet, self).__init__()
        self.in_chans = 64
        self.conv1 = nn.Conv2d(3,64,kernel_size=1,stride=1,bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU()

        self.layer1 = self.create_layers(64)
        self.layer2 = self.create_layers(128)
        self.layer3 = self.create_layers(256)
        self.layer4 = self.create_layers(512)

        self.dropout = nn.Dropout(p=0.3)
        self.avgpool = nn.AdaptiveAvgPool2d((1,1))
        self.fc = nn.Linear(512,class_nums)

    def create_layers(self,out_chans,block = BasicBlock,num_blocks=2,stride=1):
        downsample = None
        if stride != 1 or self.in_chans != out_chans:
            downsample = nn.Sequential(
                nn.Conv2d(self.in_chans,out_chans,kernel_size=1,stride=stride,bias=False),
                nn.BatchNorm2d(out_chans)
            )

        layers = []
        for i in range(num_blocks):
            if i == 0:
                layers.append(block(self.in_chans,out_chans,stride = stride,downsample=downsample))
                self.in_chans = out_chans
            else:
                layers.append(block(out_chans,out_chans))
        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.relu(self.bn1(self.conv1(x)))
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.avgpool(x)
        x = self.dropout(x)
        x = torch.flatten(x,1)
        x = self.fc(x)
        return x
```

Используемые архитектуры: MobileNetV2

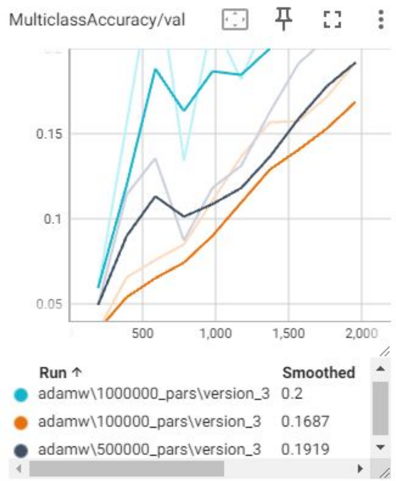
- Без stride=2 в начальной свертки
- изменен последний слой
- Число обучаемых параметров: 100k,500k,1M
- Прунинг prune.ln_structed



```
mobilenet_v2 = torchvision.models.mobilenet_v2(pretrained=True)
mobilenet_v2.features[0][0].stride = (1, 1)
mobilenet_v2.classifier[1] = nn.Linear(mobilenet_v2.last_channel, 100)
```



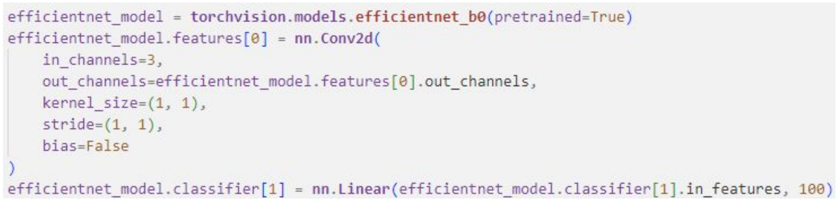
	Run ↑	Smoothed
●	adamw\1000000_pars\version_3	0.2722
●	adamw\100000_pars\version_3	0.1515
●	adamw\500000_pars\version_3	0.2284



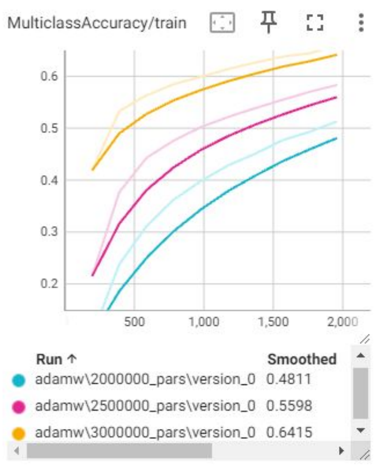
	Run ↑	Smoothed
●	adamw\1000000_pars\version_3	0.2
●	adamw\100000_pars\version_3	0.1687
●	adamw\500000_pars\version_3	0.1919

Используемые архитектуры: EfficientNet-B0

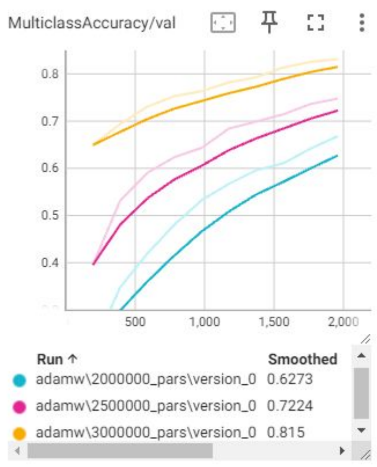
- Изменена начальная свертка.
- последний слой.
- Число обучаемых параметров: 2M,2.5M,3M.
- Прунинг prune.ln_structed.



```
efficientnet_model = torchvision.models.efficientnet_b0(pretrained=True)
efficientnet_model.features[0] = nn.Conv2d(
    in_channels=3,
    out_channels=efficientnet_model.features[0].out_channels,
    kernel_size=(1, 1),
    stride=(1, 1),
    bias=False
)
efficientnet_model.classifier[1] = nn.Linear(efficientnet_model.classifier[1].in_features, 100)
```



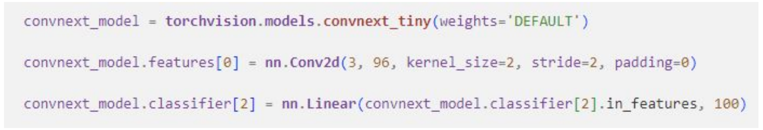
Run ↑	Smoothed
● adamw\2000000_pars\version_0	0.4811
● adamw\2500000_pars\version_0	0.5598
● adamw\3000000_pars\version_0	0.6415



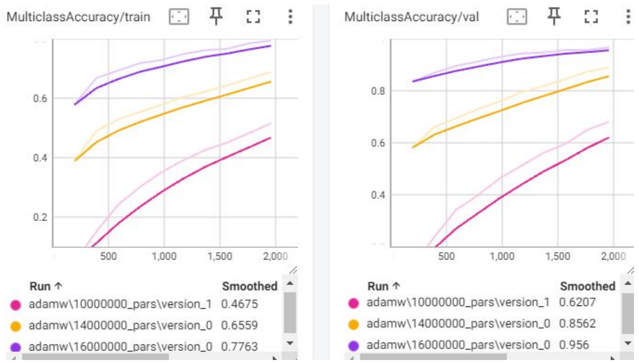
Run ↑	Smoothed
● adamw\2000000_pars\version_0	0.6273
● adamw\2500000_pars\version_0	0.7224
● adamw\3000000_pars\version_0	0.815

Используемые архитектуры:ConvNeXT(tiny)

- начальная свертка
- последний слой
- Число обучаемых параметров: 10M,14M,16M
- Прунинг: prune.ln_structed



```
convnext_model = torchvision.models.convnext_tiny(weights='DEFAULT')

convnext_model.features[0] = nn.Conv2d(3, 96, kernel_size=2, stride=2, padding=0)

convnext_model.classifier[2] = nn.Linear(convnext_model.classifier[2].in_features, 100)
```

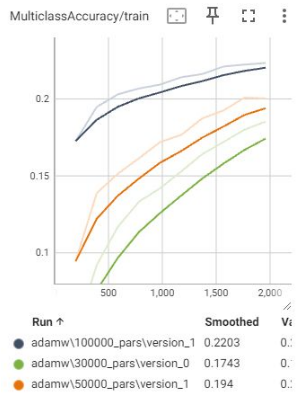


Run ↑	Smoothed
● adamw\10000000_pars\version_1	0.4675
● adamw\14000000_pars\version_0	0.6559
● adamw\16000000_pars\version_0	0.7763

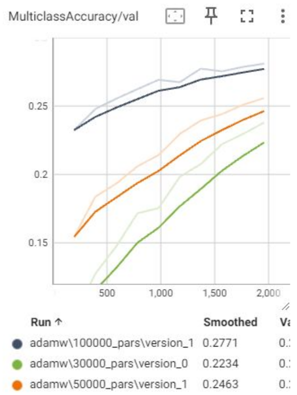
Run ↑	Smoothed
● adamw\10000000_pars\version_1	0.6207
● adamw\14000000_pars\version_0	0.8562
● adamw\16000000_pars\version_0	0.956

Используемые архитектуры: кастомные

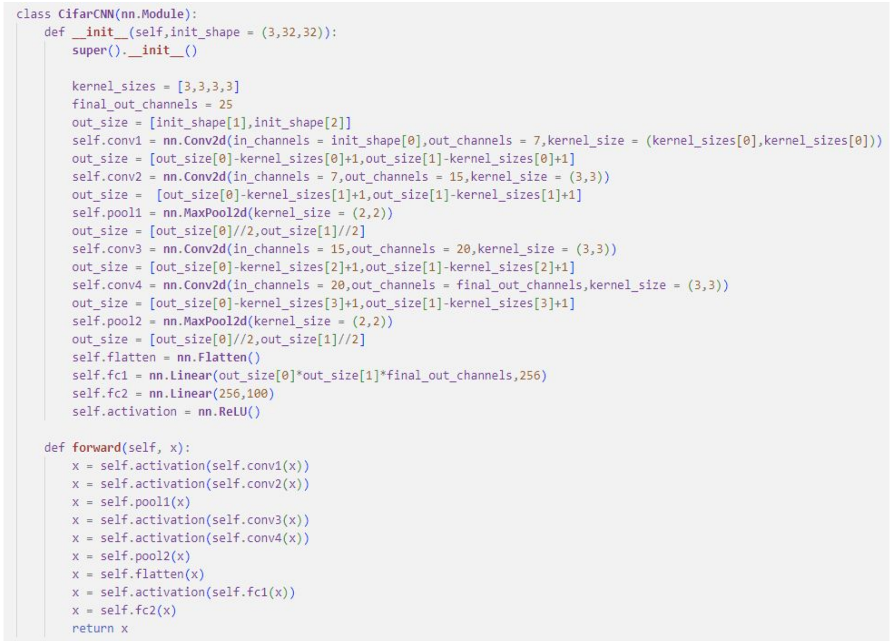
- Число обучаемых параметров: 30k,50k,100k
- Прунинг: prune.ln_structed



Run ↑	Smoothed	Va
● adamw\100000_pars\version_1	0.2203	0.:
● adamw\30000_pars\version_0	0.1743	0.:
● adamw\50000_pars\version_1	0.194	0.:



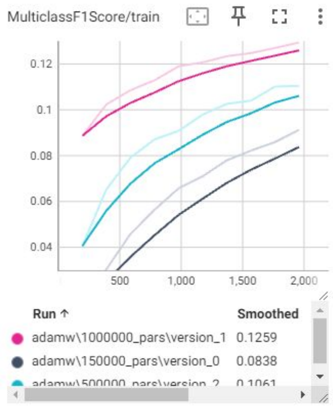
Run ↑	Smoothed	Va
● adamw\100000_pars\version_1	0.2771	0.:
● adamw\30000_pars\version_0	0.2234	0.:
● adamw\50000_pars\version_1	0.2463	0.:



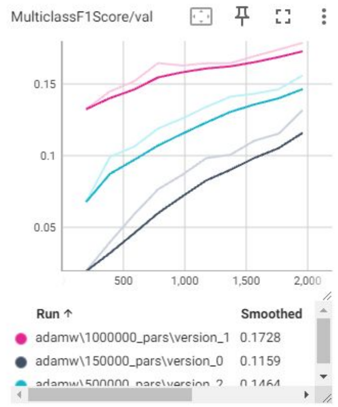
```
class CifarCNN(nn.Module):
    def __init__(self,init_shape = (3,32,32)):
        super().__init__()

        kernel_sizes = [3,3,3,3]
        final_out_channels = 25
        out_size = [init_shape[1],init_shape[2]]
        self.conv1 = nn.Conv2d(in_channels = init_shape[0],out_channels = 7,kernel_size = (kernel_sizes[0],kernel_sizes[0]))
        out_size = [out_size[0]-kernel_sizes[0]+1,out_size[1]-kernel_sizes[0]+1]
        self.conv2 = nn.Conv2d(in_channels = 7,out_channels = 15,kernel_size = (3,3))
        out_size =  [out_size[0]-kernel_sizes[1]+1,out_size[1]-kernel_sizes[1]+1]
        self.pool1 = nn.MaxPool2d(kernel_size = (2,2))
        out_size = [out_size[0]//2,out_size[1]//2]
        self.conv3 = nn.Conv2d(in_channels = 15,out_channels = 20,kernel_size = (3,3))
        out_size = [out_size[0]-kernel_sizes[2]+1,out_size[1]-kernel_sizes[2]+1]
        self.conv4 = nn.Conv2d(in_channels = 20,out_channels = final_out_channels,kernel_size = (3,3))
        out_size = [out_size[0]-kernel_sizes[3]+1,out_size[1]-kernel_sizes[3]+1]
        self.pool2 = nn.MaxPool2d(kernel_size = (2,2))
        out_size = [out_size[0]//2,out_size[1]//2]
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(out_size[0]*out_size[1]*final_out_channels,256)
        self.fc2 = nn.Linear(256,100)
        self.activation = nn.ReLU()

    def forward(self, x):
        x = self.activation(self.conv1(x))
        x = self.activation(self.conv2(x))
        x = self.pool1(x)
        x = self.activation(self.conv3(x))
        x = self.activation(self.conv4(x))
        x = self.pool2(x)
        x = self.flatten(x)
        x = self.activation(self.fc1(x))
        x = self.fc2(x)
        return x
```

Используемые архитектуры:кастомные

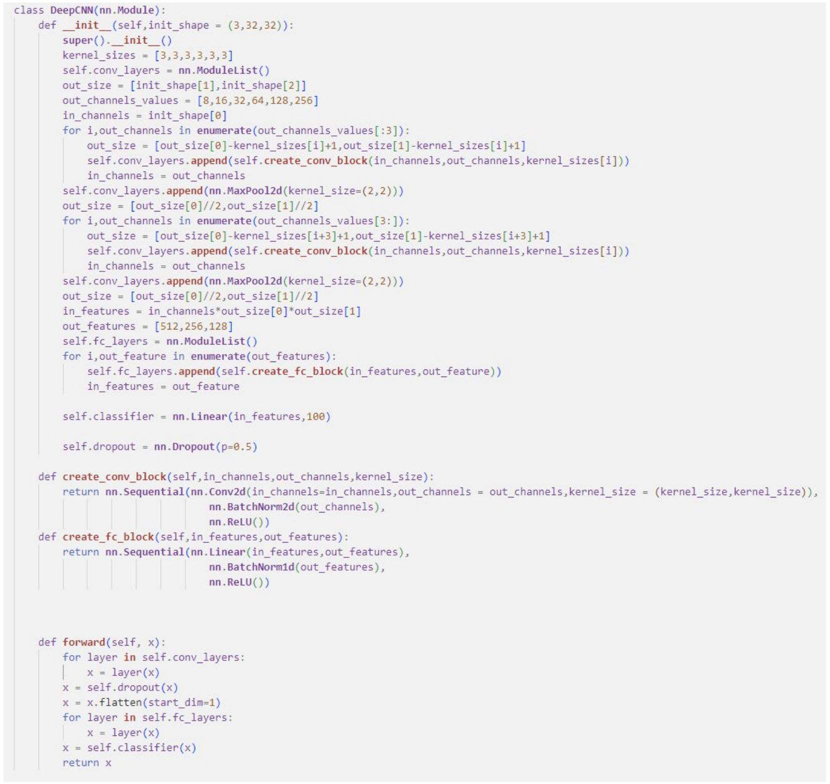
- Число обучаемых параметров: 30k,50k,100k
- Прунинг: prune.ln_structed



Run ↑	Smoothed
● adamw\1000000_pars\version_1	0.1259
● adamw\150000_pars\version_0	0.0838
● adamw\500000_pars\version_2	0.1061



Run ↑	Smoothed
● adamw\1000000_pars\version_1	0.1728
● adamw\150000_pars\version_0	0.1159
● adamw\500000_pars\version_2	0.1464

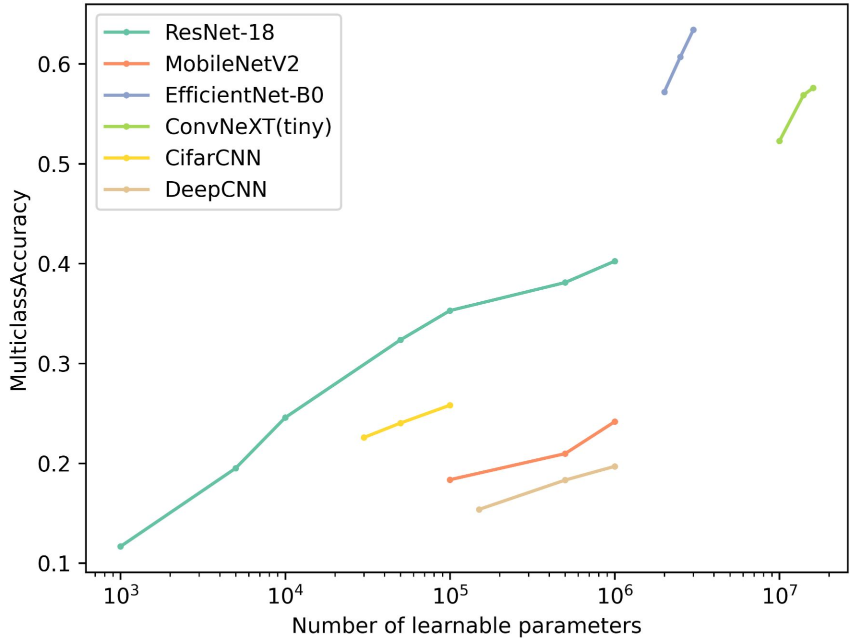


```
class DeepCNN(nn.Module):
    def __init__(self,init_shape = (3,32,32)):
        super().__init__()
        kernel_sizes = [3,3,3,3,3,3]
        self.conv_layers = nn.ModuleList()
        out_size = [init_shape[1],init_shape[2]]
        out_channels_values = [8,16,32,64,128,256]
        in_channels = init_shape[0]
        for i,out_channels in enumerate(out_channels_values[:3]):
            out_size = [out_size[0]-kernel_sizes[i]+1,out_size[1]-kernel_sizes[i]+1]
            self.conv_layers.append(self.create_conv_block(in_channels,out_channels,kernel_sizes[i]))
            in_channels = out_channels
        self.conv_layers.append(nn.MaxPool2d(kernel_size=(2,2)))
        out_size = [out_size[0]//2,out_size[1]//2]
        for i,out_channels in enumerate(out_channels_values[3:]):
            out_size = [out_size[0]-kernel_sizes[i+3]+1,out_size[1]-kernel_sizes[i+3]+1]
            self.conv_layers.append(self.create_conv_block(in_channels,out_channels,kernel_sizes[i]))
            in_channels = out_channels
        self.conv_layers.append(nn.MaxPool2d(kernel_size=(2,2)))
        out_size = [out_size[0]//2,out_size[1]//2]
        in_features = in_channels*out_size[0]*out_size[1]
        out_features = [512,256,128]
        self.fc_layers = nn.ModuleList()
        for i,out_feature in enumerate(out_features):
            self.fc_layers.append(self.create_fc_block(in_features,out_feature))
            in_features = out_feature

        self.classifier = nn.Linear(in_features,100)

        self.dropout = nn.Dropout(p=0.5)

    def create_conv_block(self,in_channels,out_channels,kernel_size):
        return nn.Sequential(nn.Conv2d(in_channels=in_channels,out_channels = out_channels,kernel_size = (kernel_size,kernel_size)),
                            nn.BatchNorm2d(out_channels),
                            nn.ReLU())
    def create_fc_block(self,in_features,out_features):
        return nn.Sequential(nn.Linear(in_features,out_features),
                            nn.BatchNorm1d(out_features),
                            nn.ReLU())



    def forward(self, x):
        for layer in self.conv_layers:
            x = layer(x)
        x = self.dropout(x)
        x = x.flatten(start_dim=1)
        for layer in self.fc_layers:
            x = layer(x)
        x = self.classifier(x)
        return x
```

Результаты обучения с ограничением на число параметров

Для теста бралась сохраненная модель с максимальным качеством на валидации.

Максимальный скор (3M) EfficientNet-B0 - 0.63



Результаты обучения: полные модели

CifarCNN: 194357

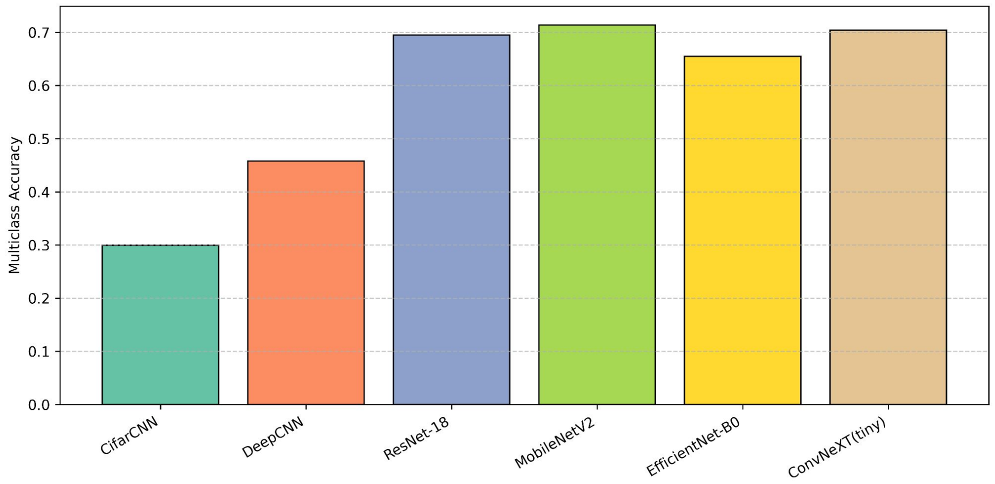
DeepCNN: 1750836

ResNet-18: 11208996

MobileNetV2: 2317860

EfficientNet-B0: 4092864

ConvNeXT(tiny): 27870628



Максимальный скор - 0.71 у MobileNetV2

Доп. значения метрик для графика

```
ResNet-18: [0.11659999191761017, 0.19499999284744263, 0.24570000171661377, 0.32349997758865356, 0.3528999984264374, 0.38100001215934753, 0.40240001678466797]

MobileNetV2: [0.1834999918937683, 0.20960000157356262, 0.24160000681877136]

EfficientNet-B0: [0.5719000101089478, 0.6068999767303467, 0.6341000199317932]

ConvNeXT(tiny): [0.5227000117301941, 0.5687999725341797, 0.5759000182151794]

CifarCNN: [0.2257000058889389, 0.240200012922287, 0.2581000030040741]

DeepCNN: [0.15370000898838043, 0.18310000002384186, 0.19689999520778656]

-------------------------------------------------------------------------------------------------------------------------------------------------------------------
----

CifarCNN: 0.29930001497268677

DeepCNN: 0.4580000042915344

ResNet-18: 0.6949000358581543

MobileNetV2: 0.7136000394821167

EfficientNet-B0: 0.6552000045776367

ConvNeXT(tiny): 0.7042999863624573
```