Подготовка окружения для работы

```
! gdown 10k8Hwn9kpK9SpK4IEj4-EaWQZqgYT5-Q
! pip install -r /content/requirements_2024_25_for_colab_small.txt
!pip install -U albumentations
!pip install lightning
!pip install tim
```

Установка необходимых библиотек

```
import timm
from pathlib import Path
import numpy as np
from typing import List
from tqdm.notebook import tqdm
from time import sleep
from PIL import Image
import IPython.display
from sklearn.metrics import balanced_accuracy_score
import gdown
import numpy as np
import copy
from json import dumps, load
import numpy as np
from os import environ
from os.path import join
from sys import argv
import random
import albumentations as A
#from torch.utils import data
import glob
from tqdm.auto import tqdm
import matplotlib.pyplot as plt
import numpy as np
import PIL.Image
import torch
import torch.nn.functional as F
from torch import nn
from json import dumps, load
from numpy import array
from os import environ
from os.path import join
from sys import argv
import random
from glob import glob
from torch.utils import data
from json import dump
from re import sub
from time import time
from traceback import format_exc
from os import makedirs
from os.path import basename, exists
from shutil import copytree
import torchvision.transforms.v2 as T
import albumentations.pytorch.transforms
import os
import lightning as L
import torchmetrics
import torchvision
```

Подключение предтренированной модели(EfficientNet)

```
def get_mobilenet_v2_torchvision(num_classes=9, transfer=True):

    model=timm.create_model(
            "tf_efficientnet_b3.ns_jft_in1k",
            pretrained=transfer,
            num_classes=num_classes,
        )
    first_unfrozen = -4
    parms=list(model.children())
```

```
    for child in parms:

        for param in child.parameters():
            param.requires_grad = True

    for child in parms[:first_unfrozen]:

        for param in child.parameters():
            param.requires_grad = False

    return model
a=get_mobilenet_v2_torchvision()
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), se
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
model.safetensors: 100%  49.3M/49.3M [00:00<00:00, 91.7MB/s]
```

```
EVALUATE_ONLY = True
TEST_ON_LARGE_DATASET = True
TISSUE_CLASSES = ('ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS', 'NORM', 'STR', 'TUM')
DATASETS_LINKS = {
    'train': '1P7zuFlv2iVgX95NWL_aeM46syaM6EwWE',
    'train_small': '1qd45xXfDwdZjktLFwQb-et-mAaFeCzOR',
    'train_tiny': '1I-2ZOuXLd4QwhZQQltp817Kn3J0Xgbui',
    'test': '14LvKrM8QXor9d0ZApGGRXSSxy4kW167b',
    'test_small': '1wbRsog0n7uGlHIPGLhyN-PMeT2kdQ21I',
    'test_tiny': '1viiB0s041CNsAK4itvX8PnYthJ-MDnQc'
}
```

Формирование собственного датасета(#LBL1,#LBL2)

```
BATCH_SIZE = 64
IMAGENET_MEAN = [0.485, 0.456, 0.406]
IMAGENET_STD = [0.229, 0.224, 0.225]
EPOCHS=60
NUM_WORKERS = os.cpu_count()
common = [
    A.ToFloat(max_value=255),
    A.Normalize(max_pixel_value=1.0, mean=IMAGENET_MEAN, std=IMAGENET_STD),
    A.pytorch.transforms.ToTensorV2(),

]
com = A.Compose(common)
def my_common_transform(image):
    p=com(image=image)

    return p["image"]
#LBL1
augmentations = [
    A.RandomBrightnessContrast(brightness_limit=0.3, contrast_limit=0.3, p=0.5),
    A.ToFloat(max_value=255),
    A.HorizontalFlip(p=0.3),
    A.VerticalFlip(p=0.4),
    A.Rotate(limit=(-30,30), p=0.5),
    A.Normalize(max_pixel_value=1.0, mean=IMAGENET_MEAN, std=IMAGENET_STD),
    A.pytorch.transforms.ToTensorV2(),

]
augs = A.Compose(augmentations)
def my_train_transform(image):
    p=augs(image=image)

    return p["image"]

class MyCustomDataset(torch.utils.data.Dataset):
    def __init__(
        self,
        mode,
        name,
        train_fraction=0.8,
        split_seed=42
```

```
            split_seed=42,
            transform=None,
        ):

            self.name = name
            self.is_loaded = False
            url = f"https://drive.google.com/uc?export=download&confirm=pbef&id={DATASETS_LINKS[name]}"
            output = f'{name}.npz'
            gdown.download(url, output, quiet=False)
            print(f'Loading dataset {self.name} from npz.')
            np_obj = np.load(f'{name}.npz')
            images = np_obj['data']
            labels = np_obj['labels']
            split = int(train_fraction * len(labels))
            self.n_files = images.shape[0]
            self.is_loaded = True
            print(f'Done. Dataset {name} consists of {self.n_files} images.')
            rng = random.Random(split_seed)
            indexes=np.arange(self.n_files)
            rng.shuffle(indexes)
            if mode == "train":
                indexes=indexes[:split]
                images = images[indexes]
                labels=labels[indexes]
            elif mode == "valid":   #LBL2
                indexes=indexes[split:]
                images = images[indexes]
                labels=labels[indexes]
            elif mode=="test":
                images = images
                labels=labels
            else:
                raise RuntimeError(f"Invalid mode: {mode!r}")
            self._images=images
            self._labels=labels
            self._len = len(labels)
            self.mode=mode
            assert self._labels.shape[0] == self._len

            if transform is None:
                transform = my_common_transform
            self._transform = transform

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

        def __getitem__(self, index):
            image = self._images[index]
            label = self._labels[index]
            # plt.imshow(image)
            # plt.show()
            # image=image.cpu().numpy().transpose(1,2,0)

            image= self._transform(image)
            return image, label
```

Реализация lr, изменяющегося по косинусу(#LBL3)

```
def cosine_annealing_lr(    #LBL3
    optimizer,
    total_steps,
):

    cos_annealing = torch.optim.lr_scheduler.CosineAnnealingLR(
        optimizer,
        T_max=total_steps,
    )
    return cos_annealing
```

Обучение модели (#LBL4,#LBL5,#LBL6)

```
def train_model():

  def get_cls_model(input_shape=(1, 40, 100)):
    """
```

```
    :param input_shape: tuple (n_rows, n_cols, n_channels)
            input shape of image for classification
    :return: nn model for classification
    """
ds_train = MyCustomDataset(name="train",mode="train",transform=my_train_transform)
ds_valid = MyCustomDataset(name="train",mode="valid")
STEPS=ds_train.__len__()//BATCH_SIZE
def get_mobilenet_v2_torchvision(num_classes=9, transfer=True):

    model=timm.create_model(
            "tf_efficientnet_b3.ns_jft_in1k",
            pretrained=transfer,
            num_classes=num_classes,
        )


    return model

class Classifier(L.LightningModule):

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.steps=STEPS
        self.lr = 0.0001
        self.alpha=1.0
        self.wdc= 0 #1.2e-4
        self.model = self.get_model()
        self.loss_fn = nn.CrossEntropyLoss()
        self.accuracy = torchmetrics.classification.Accuracy(
            task="multiclass",
            num_classes=9,
        )

    def get_model(self):
        return get_mobilenet_v2_torchvision()



    def training_step(self, batch):
        return self._step(batch, "train")

    def validation_step(self, batch):
        return self._step(batch, "valid")

    def _step(self, batch, kind):
        x, y = batch

        p = self.model(x)
        loss = self.loss_fn(p, y)
        accs = self.accuracy(p.argmax(axis=-1), y)

        return self._log_metrics(loss, accs, kind)

    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters(), lr=self.lr,weight_decay=self.wdc)

        steps_per_epoch = self.steps
        total_steps = EPOCHS * steps_per_epoch

        scheduler = cosine_annealing_lr(
            optimizer,
            total_steps=total_steps
        )

        lr_scheduler = {
            "scheduler": scheduler,
            "interval": "step",
            "frequency": 1,
        }
        return [optimizer], [lr_scheduler]

    def _log_metrics(self, loss, accs, kind):
        metrics = {}
        if loss is not None:
            metrics[f"{kind}_loss"] = loss
        if accs is not None:
            metrics[f"{kind}_accs"] = accs
        self.log_dict(
```

```
                metrics,
                prog_bar=True,
                logger=True,
                on_step=kind == "train",
                on_epoch=True,
            )
            return loss



    dl_train = torch.utils.data.DataLoader(
    ds_train,
    batch_size=BATCH_SIZE,
    shuffle=True,
    num_workers=NUM_WORKERS,
    )
    dl_valid = torch.utils.data.DataLoader(
        ds_valid,
        batch_size=BATCH_SIZE,
        shuffle=False,
        num_workers=NUM_WORKERS,
    )
    callbacks = [
          L.pytorch.callbacks.TQDMProgressBar(leave=True),
          L.pytorch.callbacks.LearningRateMonitor(),
          L.pytorch.callbacks.ModelCheckpoint(
              filename="{epoch}-{valid_accs:.3f}",
              monitor="valid_accs",
              mode="max",
              save_top_k=1,
              save_last=True,
          )
      ]
    trainer = L.Trainer(
          callbacks=callbacks,
          max_epochs=EPOCHS,
          default_root_dir="runs",

      )
    Model=Classifier()
    #last_ckpt = "runs/lightning_logs/version_6/checkpoints/epoch=55-valid_accs=0.977.ckpt"
    trainer.fit(Model, dl_train, dl_valid)#, ckpt_path=last_ckpt)
    # train model
    model=Model.model
    return model


train_model()
```

```
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1P7zuFlv2iVgX95NWL_aeM46syaM6EwWE
To: /content/train.npz
100%|██████████| 2.10G/2.10G [00:22<00:00, 94.8MB/s]
Loading dataset train from npz.
Done. Dataset train consists of 18000 images.
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1P7zuFlv2iVgX95NWL_aeM46syaM6EwWE
To: /content/train.npz
100%|██████████| 2.10G/2.10G [00:12<00:00, 163MB/s]
Loading dataset train from npz.
Done. Dataset train consists of 18000 images.
INFO: GPU available: True (cuda), used: True
INFO:lightning.pytorch.utilities.rank_zero:GPU available: True (cuda), used: True
INFO: TPU available: False, using: 0 TPU cores
INFO:lightning.pytorch.utilities.rank_zero:TPU available: False, using: 0 TPU cores
INFO: HPU available: False, using: 0 HPUs
INFO:lightning.pytorch.utilities.rank_zero:HPU available: False, using: 0 HPUs
INFO: LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:lightning.pytorch.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:
  | Name     | Type               | Params | Mode
-----------------------------------------------------
0 | model    | EfficientNet       | 10.7 M | train
1 | loss_fn  | CrossEntropyLoss   | 0      | train
2 | accuracy | MulticlassAccuracy | 0      | train
-----------------------------------------------------
10.7 M    Trainable params
0         Non-trainable params
10.7 M    Total params
42.840    Total estimated model params size (MB)
535       Modules in train mode
0         Modules in eval mode
INFO:lightning.pytorch.callbacks.model_summary:
  | Name     | Type               | Params | Mode
-----------------------------------------------------
0 | model    | EfficientNet       | 10.7 M | train
1 | loss_fn  | CrossEntropyLoss   | 0      | train
2 | accuracy | MulticlassAccuracy | 0      | train
-----------------------------------------------------
10.7 M    Trainable params
0         Non-trainable params
10.7 M    Total params
42.840    Total estimated model params size (MB)
535       Modules in train mode
0         Modules in eval mode
```

Training: 0/? [00:09<?, ?it/s]

Epoch 0: 100%

225/225 [02:37<00:00, 1.43it/s, v_num=0, train_loss_step=0.189, train_accs_step=0.969, valid_loss=0.125, valid_accs=0.963, train_loss_epoch=0.433,

Epoch 1: 100%

225/225 [02:41<00:00, 1.40it/s, v_num=0, train_loss_step=0.116, train_accs_step=0.969, valid_loss=0.0943, valid_accs=0.973, train_loss_epoch=0.131,

Epoch 2: 100%

225/225 [02:43<00:00, 1.38it/s, v_num=0, train_loss_step=0.137, train_accs_step=0.969, valid_loss=0.0767, valid_accs=0.979, train_loss_epoch=0.0868

Epoch 3: 100%

225/225 [02:40<00:00, 1.40it/s, v_num=0, train_loss_step=0.114, train_accs_step=0.984, valid_loss=0.0697, valid_accs=0.981, train_loss_epoch=0.0575

Epoch 4: 100%

225/225 [02:43<00:00, 1.38it/s, v_num=0, train_loss_step=0.080, train_accs_step=0.984, valid_loss=0.060, valid_accs=0.983, train_loss_epoch=0.047,

Epoch 5: 100%

225/225 [02:41<00:00, 1.40it/s, v_num=0, train_loss_step=0.020, train_accs_step=0.984, valid_loss=0.0648, valid_accs=0.982, train_loss_epoch=0.0422

Epoch 6: 100%

225/225 [02:41<00:00, 1.40it/s, v_num=0, train_loss_step=0.0228, train_accs_step=0.984, valid_loss=0.0579, valid_accs=0.985, train_loss_epoch=0.032

Epoch 7: 100%

225/225 [02:43<00:00, 1.38it/s, v_num=0, train_loss_step=0.141, train_accs_step=0.969, valid_loss=0.0645, valid_accs=0.984, train_loss_epoch=0.0289

Epoch 8: 100%

225/225 [02:41<00:00, 1.40it/s, v_num=0, train_loss_step=0.0508, train_accs_step=0.984, valid_loss=0.0528, valid_accs=0.984, train_loss_epoch=0.024

Epoch 9: 100%

225/225 [02:41<00:00, 1.39it/s, v_num=0, train_loss_step=0.0694, train_accs_step=0.969, valid_loss=0.0515, valid_accs=0.986, train_loss_epoch=0.023

Epoch 10: 100%

225/225 [02:42<00:00, 1.38it/s, v_num=0, train_loss_step=0.0712, train_accs_step=0.969, valid_loss=0.0589, valid_accs=0.985, train_loss_epoch=0.018

Epoch 11: 100%

225/225 [02:40<00:00, 1.40it/s, v_num=0, train_loss_step=0.0114, train_accs_step=1.000, valid_loss=0.0516, valid_accs=0.986, train_loss_epoch=0.020

Epoch 12: 100%

225/225 [02:42<00:00, 1.38it/s, v_num=0, train_loss_step=0.0356, train_accs_step=0.984, valid_loss=0.0553, valid_accs=0.986, train_loss_epoch=0.018

Epoch 13: 100%

225/225 [02:41<00:00, 1.40it/s, v_num=0, train_loss_step=0.0024, train_accs_step=1.000, valid_loss=0.0624, valid_accs=0.985, train_loss_epoch=0.016

Epoch 14: 100%

225/225 [02:42<00:00, 1.38it/s, v_num=0, train_loss_step=0.0842, train_accs_step=0.984, valid_loss=0.0551, valid_accs=0.988, train_loss_epoch=0.013

Epoch 15: 100%

225/225 [02:41<00:00, 1.39it/s, v_num=0, train_loss_step=0.00752, train_accs_step=1.000, valid_loss=0.0498, valid_accs=0.987, train_loss_epoch=0.01

Epoch 16: 100%

225/225 [02:45<00:00, 1.36it/s, v_num=0, train_loss_step=0.0144, train_accs_step=1.000, valid_loss=0.060, valid_accs=0.987, train_loss_epoch=0.0141

Epoch 17: 100%

225/225 [02:42<00:00, 1.38it/s, v_num=0, train_loss_step=0.00165, train_accs_step=1.000, valid_loss=0.0609, valid_accs=0.985, train_loss_epoch=0.01

Epoch 18: 100%

225/225 [02:41<00:00, 1.39it/s, v_num=0, train_loss_step=0.00163, train_accs_step=1.000, valid_loss=0.0643, valid_accs=0.987, train_loss_epoch=0.01

Epoch 19: 100%

225/225 [02:42<00:00, 1.38it/s, v_num=0, train_loss_step=0.000297, train_accs_step=1.000, valid_loss=0.0578, valid_accs=0.986, train_loss_epoch=0.0

Epoch 20: 100%

225/225 [02:41<00:00, 1.39it/s, v_num=0, train_loss_step=0.000205, train_accs_step=1.000, valid_loss=0.0531, valid_accs=0.989, train_loss_epoch=0.0

Epoch 21: 100%

225/225 [02:42<00:00, 1.39it/s, v_num=0, train_loss_step=0.000592, train_accs_step=1.000, valid_loss=0.0566, valid_accs=0.989, train_loss_epoch=0.0

Epoch 22: 100%

225/225 [02:42<00:00, 1.38it/s, v_num=0, train_loss_step=0.000103, train_accs_step=1.000, valid_loss=0.057, valid_accs=0.989, train_loss_epoch=0.00

Epoch 23: 100%

225/225 [02:41<00:00, 1.39it/s, v_num=0, train_loss_step=0.0618, train_accs_step=0.969, valid_loss=0.0593, valid_accs=0.988, train_loss_epoch=0.009

Epoch 24: 100%

225/225 [02:42<00:00, 1.38it/s, v_num=0, train_loss_step=0.00371, train_accs_step=1.000, valid_loss=0.0545, valid_accs=0.989, train_loss_epoch=0.00

Epoch 25: 100%

225/225 [02:43<00:00, 1.38it/s, v_num=0, train_loss_step=0.0192, train_accs_step=1.000, valid_loss=0.0449, valid_accs=0.990, train_loss_epoch=0.009

Epoch 26: 100%

225/225 [02:40<00:00, 1.40it/s, v_num=0, train_loss_step=0.0247, train_accs_step=0.984, valid_loss=0.0448, valid_accs=0.991, train_loss_epoch=0.009

Epoch 27: 100%

225/225 [02:43<00:00, 1.38it/s, v_num=0, train_loss_step=0.000433, train_accs_step=1.000, valid_loss=0.048, valid_accs=0.988, train_loss_epoch=0.00

Epoch 28: 100%

225/225 [02:40<00:00, 1.40it/s, v_num=0, train_loss_step=0.0209, train_accs_step=0.984, valid_loss=0.0443, valid_accs=0.990, train_loss_epoch=0.006

Epoch 29: 100%

225/225 [02:43<00:00, 1.38it/s, v_num=0, train_loss_step=0.00273, train_accs_step=1.000, valid_loss=0.053, valid_accs=0.990, train_loss_epoch=0.004

Epoch 30: 100%

225/225 [02:42<00:00, 1.39it/s, v_num=0, train_loss_step=4.14e-5, train_accs_step=1.000, valid_loss=0.0582, valid_accs=0.988, train_loss_epoch=0.00597, train_accs_epoch=0.998]

Epoch 31: 100%

225/225 [02:42<00:00, 1.38it/s, v_num=0, train_loss_step=0.00355, train_accs_step=1.000, valid_loss=0.0498, valid_accs=0.989, train_loss_epoch=0.0(

Epoch 32: 100%

225/225 [02:42<00:00, 1.38it/s, v_num=0, train_loss_step=0.00132, train_accs_step=1.000, valid_loss=0.0517, valid_accs=0.990, train_loss_epoch=0.0(

Epoch 33: 100%

225/225 [02:42<00:00, 1.39it/s, v_num=0, train_loss_step=2.56e-5, train_accs_step=1.000, valid_loss=0.0494, valid_accs=0.990, train_loss_epoch=0.00608, train_accs_epoch=0.998]

Epoch 34: 100%

225/225 [02:44<00:00, 1.37it/s, v_num=0, train_loss_step=0.00102, train_accs_step=1.000, valid_loss=0.0394, valid_accs=0.990, train_loss_epoch=0.0(

Epoch 35: 100%

225/225 [02:41<00:00, 1.39it/s, v_num=0, train_loss_step=0.0014, train_accs_step=1.000, valid_loss=0.0355, valid_accs=0.991, train_loss_epoch=0.003

Epoch 36: 100%

225/225 [02:44<00:00, 1.37it/s, v_num=0, train_loss_step=0.0124, train_accs_step=1.000, valid_loss=0.0402, valid_accs=0.991, train_loss_epoch=0.002

Epoch 37: 100%

225/225 [02:43<00:00, 1.37it/s, v_num=0, train_loss_step=0.000102, train_accs_step=1.000, valid_loss=0.0464, valid_accs=0.990, train_loss_epoch=0.(

Epoch 38: 100%

225/225 [02:43<00:00, 1.38it/s, v_num=0, train_loss_step=2.29e-5, train_accs_step=1.000, valid_loss=0.0418, valid_accs=0.991, train_loss_epoch=0.00257, train_accs_epoch=0.999]

Epoch 39: 100%

225/225 [02:42<00:00, 1.39it/s, v_num=0, train_loss_step=2.98e-5, train_accs_step=1.000, valid_loss=0.0457, valid_accs=0.989, train_loss_epoch=0.00239, train_accs_epoch=0.999]

Epoch 40: 100%

225/225 [02:44<00:00, 1.37it/s, v_num=0, train_loss_step=0.000869, train_accs_step=1.000, valid_loss=0.0466, valid_accs=0.990, train_loss_epoch=0.(

Epoch 41: 100%

225/225 [02:42<00:00, 1.38it/s, v_num=0, train_loss_step=0.00116, train_accs_step=1.000, valid_loss=0.0568, valid_accs=0.989, train_loss_epoch=0.0(

Epoch 42: 100%

225/225 [02:41<00:00, 1.40it/s, v_num=0, train_loss_step=0.00144, train_accs_step=1.000, valid_loss=0.0512, valid_accs=0.990, train_loss_epoch=0.0(

Epoch 43: 100%

225/225 [02:43<00:00, 1.38it/s, v_num=0, train_loss_step=2.89e-5, train_accs_step=1.000, valid_loss=0.0476, valid_accs=0.990, train_loss_epoch=0.0014, train_accs_epoch=1.000]

Epoch 44: 100%

225/225 [02:41<00:00, 1.40it/s, v_num=0, train_loss_step=1.73e-5, train_accs_step=1.000, valid_loss=0.0433, valid_accs=0.991, train_loss_epoch=0.00204, train_accs_epoch=1.000]

Epoch 45: 100%

225/225 [02:42<00:00, 1.38it/s, v_num=0, train_loss_step=0.00217, train_accs_step=1.000, valid_loss=0.0432, valid_accs=0.991, train_loss_epoch=0.0(

Epoch 46: 100%

225/225 [02:43<00:00, 1.38it/s, v_num=0, train_loss_step=0.00589, train_accs_step=1.000, valid_loss=0.0418, valid_accs=0.990, train_loss_epoch=0.0(

Epoch 47: 100%

225/225 [02:41<00:00, 1.39it/s, v_num=0, train_loss_step=1.2e-5, train_accs_step=1.000, valid_loss=0.0434, valid_accs=0.991, train_loss_epoch=0.00184, train_accs_epoch=1.000]

Epoch 48: 100%

225/225 [02:42<00:00, 1.39it/s, v_num=0, train_loss_step=0.00053, train_accs_step=1.000, valid_loss=0.0453, valid_accs=0.991, train_loss_epoch=0.0(

Epoch 49: 100%

225/225 [02:43<00:00, 1.37it/s, v_num=0, train_loss_step=0.00171, train_accs_step=1.000, valid_loss=0.0412, valid_accs=0.991, train_loss_epoch=0.0(

Epoch 50: 100%

225/225 [02:43<00:00, 1.38it/s, v_num=0, train_loss_step=0.000309, train_accs_step=1.000, valid_loss=0.0451, valid_accs=0.991, train_loss_epoch=0.(

Epoch 51: 100%

225/225 [02:42<00:00, 1.39it/s, v_num=0, train_loss_step=0.0067, train_accs_step=1.000, valid_loss=0.0428, valid_accs=0.992, train_loss_epoch=0.001

Epoch 52: 100%

225/225 [02:41<00:00, 1.39it/s, v_num=0, train_loss_step=2.18e-5, train_accs_step=1.000, valid_loss=0.0424, valid_accs=0.992, train_loss_epoch=0.00129, train_accs_epoch=1.000]

Epoch 53: 100%

225/225 [02:42<00:00, 1.38it/s, v_num=0, train_loss_step=0.000636, train_accs_step=1.000, valid_loss=0.0426, valid_accs=0.991, train_loss_epoch=0.(

```
Epoch 54: 100%
 225/225 [02:43<00:00,  1.38it/s, v_num=0, train_loss_step=4.27e-
5, train_accs_step=1.000, valid_loss=0.0412, valid_accs=0.992, train_loss_epoch=0.00168, train_accs_epoch=0.999]

Epoch 55: 100%
 225/225 [02:43<00:00,  1.38it/s, v_num=0, train_loss_step=8.17e-
5, train_accs_step=1.000, valid_loss=0.0412, valid_accs=0.991, train_loss_epoch=0.00125, train_accs_epoch=1.000]

Epoch 56: 100%
 225/225 [02:41<00:00,  1.39it/s, v_num=0, train_loss_step=6.26e-
5, train_accs_step=1.000, valid_loss=0.0435, valid_accs=0.992, train_loss_epoch=0.00121, train_accs_epoch=1.000]

Epoch 57:  62%
 225/225 [02:42<00:00,  1.38it/s, v_num=0, train_loss_step=0.00134, train_accs_step=1.000, valid_loss=0.0459, valid_accs=0.991, train_loss_epoch=0.0(
Validation: |          | 0/? [00:00<?, ?it/s]
Epoch 58: 100%
 225/225 [02:43<00:00,  1.38it/s, v_num=0, train_loss_step=0.000288, train_accs_step=1.000, valid_loss=0.0447, valid_accs=0.991, train_loss_epoch=0.(

Epoch 59: 100%
 225/225 [02:43<00:00,  1.38it/s, v_num=0, train_loss_step=1.53e-
5, train_accs_step=1.000, valid_loss=0.0411, valid_accs=0.992, train_loss_epoch=0.00142, train_accs_epoch=1.000]
INFO: `Trainer.fit` stopped: `max_epochs=60` reached.
INFO:lightning.pytorch.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs=60` reached.
EfficientNet(
  (conv_stem): Conv2dSame(3, 40, kernel_size=(3, 3), stride=(2, 2), bias=False)
  (bn1): BatchNormAct2d(
    40, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
    (drop): Identity()
    (act): SiLU(inplace=True)
  )
  (blocks): Sequential(
    (0): Sequential(
      (0): DepthwiseSeparableConv(
        (conv_dw): Conv2d(40, 40, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=40, bias=False)
        (bn1): BatchNormAct2d(
          40, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
          (drop): Identity()
          (act): SiLU(inplace=True)
        )
        (aa): Identity()
        (se): SqueezeExcite(
          (conv_reduce): Conv2d(40, 10, kernel_size=(1, 1), stride=(1, 1))
          (act1): SiLU(inplace=True)
          (conv_expand): Conv2d(10, 40, kernel_size=(1, 1), stride=(1, 1))
          (gate): Sigmoid()
        )
        (conv_pw): Conv2d(40, 24, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn2): BatchNormAct2d(
          24, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
          (drop): Identity()
          (act): Identity()
        )
        (drop_path): Identity()
      )
      (1): DepthwiseSeparableConv(
        (conv_dw): Conv2d(24, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=24, bias=False)
        (bn1): BatchNormAct2d(
          24, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
          (drop): Identity()
          (act): SiLU(inplace=True)
        )
        (aa): Identity()
        (se): SqueezeExcite(
          (conv_reduce): Conv2d(24, 6, kernel_size=(1, 1), stride=(1, 1))
          (act1): SiLU(inplace=True)
          (conv_expand): Conv2d(6, 24, kernel_size=(1, 1), stride=(1, 1))
          (gate): Sigmoid()
        )
```

```
      (conv_pw): Conv2d(24, 24, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn2): BatchNormAct2d(
        24, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): Identity()
      )
      (drop_path): Identity()
    )
  )
  (1): Sequential(
    (0): InvertedResidual(
      (conv_pw): Conv2d(24, 144, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNormAct2d(
        144, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (conv_dw): Conv2dSame(144, 144, kernel_size=(3, 3), stride=(2, 2), groups=144, bias=False)
      (bn2): BatchNormAct2d(
        144, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (aa): Identity()
      (se): SqueezeExcite(
        (conv_reduce): Conv2d(144, 6, kernel_size=(1, 1), stride=(1, 1))
        (act1): SiLU(inplace=True)
        (conv_expand): Conv2d(6, 144, kernel_size=(1, 1), stride=(1, 1))
        (gate): Sigmoid()
      )
      (conv_pwl): Conv2d(144, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNormAct2d(
        32, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): Identity()
      )
      (drop_path): Identity()
    )
    (1): InvertedResidual(
      (conv_pw): Conv2d(32, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNormAct2d(
        192, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (conv_dw): Conv2d(192, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=192, bias=False)
      (bn2): BatchNormAct2d(
        192, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (aa): Identity()
      (se): SqueezeExcite(
        (conv_reduce): Conv2d(192, 8, kernel_size=(1, 1), stride=(1, 1))
        (act1): SiLU(inplace=True)
        (conv_expand): Conv2d(8, 192, kernel_size=(1, 1), stride=(1, 1))
        (gate): Sigmoid()
      )
      (conv_pwl): Conv2d(192, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNormAct2d(
        32, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): Identity()
      )
      (drop_path): Identity()
    )
    (2): InvertedResidual(
      (conv_pw): Conv2d(32, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNormAct2d(
        192, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (conv_dw): Conv2d(192, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=192, bias=False)
      (bn2): BatchNormAct2d(
        192, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (aa): Identity()
      (se): SqueezeExcite(
        (conv_reduce): Conv2d(192, 8, kernel_size=(1, 1), stride=(1, 1))
        (act1): SiLU(inplace=True)
        (conv_expand): Conv2d(8, 192, kernel_size=(1, 1), stride=(1, 1))
        (gate): Sigmoid()
```

```
      )
      (conv_pwl): Conv2d(192, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNormAct2d(
        32, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): Identity()
      )
      (drop_path): Identity()
    )
  )
  (2): Sequential(
    (0): InvertedResidual(
      (conv_pw): Conv2d(32, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNormAct2d(
        192, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (conv_dw): Conv2dSame(192, 192, kernel_size=(5, 5), stride=(2, 2), groups=192, bias=False)
      (bn2): BatchNormAct2d(
        192, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (aa): Identity()
      (se): SqueezeExcite(
        (conv_reduce): Conv2d(192, 8, kernel_size=(1, 1), stride=(1, 1))
        (act1): SiLU(inplace=True)
        (conv_expand): Conv2d(8, 192, kernel_size=(1, 1), stride=(1, 1))
        (gate): Sigmoid()
      )
      (conv_pwl): Conv2d(192, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNormAct2d(
        48, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): Identity()
      )
      (drop_path): Identity()
    )
    (1): InvertedResidual(
      (conv_pw): Conv2d(48, 288, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNormAct2d(
        288, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (conv_dw): Conv2d(288, 288, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), groups=288, bias=False)
      (bn2): BatchNormAct2d(
        288, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (aa): Identity()
      (se): SqueezeExcite(
        (conv_reduce): Conv2d(288, 12, kernel_size=(1, 1), stride=(1, 1))
        (act1): SiLU(inplace=True)
        (conv_expand): Conv2d(12, 288, kernel_size=(1, 1), stride=(1, 1))
        (gate): Sigmoid()
      )
      (conv_pwl): Conv2d(288, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNormAct2d(
        48, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): Identity()
      )
      (drop_path): Identity()
    )
    (2): InvertedResidual(
      (conv_pw): Conv2d(48, 288, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNormAct2d(
        288, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (conv_dw): Conv2d(288, 288, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), groups=288, bias=False)
      (bn2): BatchNormAct2d(
        288, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (aa): Identity()
      (se): SqueezeExcite(
        (conv_reduce): Conv2d(288, 12, kernel_size=(1, 1), stride=(1, 1))
        (act1): SiLU(inplace=True)
        (conv_expand): Conv2d(12, 288, kernel_size=(1, 1), stride=(1, 1))
        (gate): Sigmoid()
```

```
      (gate): Sigmoid()
    )
    (conv_pwl): Conv2d(288, 48, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNormAct2d(
      48, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): Identity()
    )
    (drop_path): Identity()
  )
)
(3): Sequential(
  (0): InvertedResidual(
    (conv_pw): Conv2d(48, 288, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNormAct2d(
      288, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): SiLU(inplace=True)
    )
    (conv_dw): Conv2dSame(288, 288, kernel_size=(3, 3), stride=(2, 2), groups=288, bias=False)
    (bn2): BatchNormAct2d(
      288, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): SiLU(inplace=True)
    )
    (aa): Identity()
    (se): SqueezeExcite(
      (conv_reduce): Conv2d(288, 12, kernel_size=(1, 1), stride=(1, 1))
      (act1): SiLU(inplace=True)
      (conv_expand): Conv2d(12, 288, kernel_size=(1, 1), stride=(1, 1))
      (gate): Sigmoid()
    )
    (conv_pwl): Conv2d(288, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNormAct2d(
      96, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): Identity()
    )
    (drop_path): Identity()
  )
  (1): InvertedResidual(
    (conv_pw): Conv2d(96, 576, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNormAct2d(
      576, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): SiLU(inplace=True)
    )
    (conv_dw): Conv2d(576, 576, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=576, bias=False)
    (bn2): BatchNormAct2d(
      576, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): SiLU(inplace=True)
    )
    (aa): Identity()
    (se): SqueezeExcite(
      (conv_reduce): Conv2d(576, 24, kernel_size=(1, 1), stride=(1, 1))
      (act1): SiLU(inplace=True)
      (conv_expand): Conv2d(24, 576, kernel_size=(1, 1), stride=(1, 1))
      (gate): Sigmoid()
    )
    (conv_pwl): Conv2d(576, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNormAct2d(
      96, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): Identity()
    )
    (drop_path): Identity()
  )
  (2): InvertedResidual(
    (conv_pw): Conv2d(96, 576, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNormAct2d(
      576, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): SiLU(inplace=True)
    )
    (conv_dw): Conv2d(576, 576, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=576, bias=False)
    (bn2): BatchNormAct2d(
      576, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): SiLU(inplace=True)
    )
    (aa): Identity()
    (se): SqueezeExcite(
      (conv_reduce): Conv2d(576, 24, kernel_size=(1, 1), stride=(1, 1))
      (act1): SiLU(inplace=True)
      (conv_expand): Conv2d(24, 576, kernel_size=(1, 1), stride=(1, 1))
```

```
          (gate): Sigmoid()
        )
        (conv_pwl): Conv2d(576, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNormAct2d(
          96, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
          (drop): Identity()
          (act): Identity()
        )
        (drop_path): Identity()
      )
      (3): InvertedResidual(
        (conv_pw): Conv2d(96, 576, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNormAct2d(
          576, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
          (drop): Identity()
          (act): SiLU(inplace=True)
        )
        (conv_dw): Conv2d(576, 576, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=576, bias=False)
        (bn2): BatchNormAct2d(
          576, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
          (drop): Identity()
          (act): SiLU(inplace=True)
        )
        (aa): Identity()
        (se): SqueezeExcite(
          (conv_reduce): Conv2d(576, 24, kernel_size=(1, 1), stride=(1, 1))
          (act1): SiLU(inplace=True)
          (conv_expand): Conv2d(24, 576, kernel_size=(1, 1), stride=(1, 1))
          (gate): Sigmoid()
        )
        (conv_pwl): Conv2d(576, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNormAct2d(
          96, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
          (drop): Identity()
          (act): Identity()
        )
        (drop_path): Identity()
      )
      (4): InvertedResidual(
        (conv_pw): Conv2d(96, 576, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNormAct2d(
          576, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
          (drop): Identity()
          (act): SiLU(inplace=True)
        )
        (conv_dw): Conv2d(576, 576, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=576, bias=False)
        (bn2): BatchNormAct2d(
          576, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
          (drop): Identity()
          (act): SiLU(inplace=True)
        )
        (aa): Identity()
        (se): SqueezeExcite(
          (conv_reduce): Conv2d(576, 24, kernel_size=(1, 1), stride=(1, 1))
          (act1): SiLU(inplace=True)
          (conv_expand): Conv2d(24, 576, kernel_size=(1, 1), stride=(1, 1))
          (gate): Sigmoid()
        )
        (conv_pwl): Conv2d(576, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNormAct2d(
          96, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
          (drop): Identity()
          (act): Identity()
        )
        (drop_path): Identity()
      )
    )
    (4): Sequential(
      (0): InvertedResidual(
        (conv_pw): Conv2d(96, 576, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNormAct2d(
          576, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
          (drop): Identity()
          (act): SiLU(inplace=True)
        )
        (conv_dw): Conv2d(576, 576, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), groups=576, bias=False)
        (bn2): BatchNormAct2d(
          576, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
          (drop): Identity()
          (act): SiLU(inplace=True)
        )
        (aa): Identity()
        (se): SqueezeExcite(
          (conv_reduce): Conv2d(576, 24, kernel_size=(1, 1), stride=(1, 1))
          (act1): SiLU(inplace=True)
          (conv_expand): Conv2d(24, 576, kernel_size=(1, 1), stride=(1, 1))
```

```
      (conv_expand): Conv2d(24, 576, kernel_size=(1, 1), stride=(1, 1))
      (gate): Sigmoid()
    )
    (conv_pwl): Conv2d(576, 136, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNormAct2d(
      136, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): Identity()
    )
    (drop_path): Identity()
  )
  (1): InvertedResidual(
    (conv_pw): Conv2d(136, 816, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNormAct2d(
      816, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): SiLU(inplace=True)
    )
    (conv_dw): Conv2d(816, 816, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), groups=816, bias=False)
    (bn2): BatchNormAct2d(
      816, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): SiLU(inplace=True)
    )
    (aa): Identity()
    (se): SqueezeExcite(
      (conv_reduce): Conv2d(816, 34, kernel_size=(1, 1), stride=(1, 1))
      (act1): SiLU(inplace=True)
      (conv_expand): Conv2d(34, 816, kernel_size=(1, 1), stride=(1, 1))
      (gate): Sigmoid()
    )
    (conv_pwl): Conv2d(816, 136, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNormAct2d(
      136, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): Identity()
    )
    (drop_path): Identity()
  )
  (2): InvertedResidual(
    (conv_pw): Conv2d(136, 816, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNormAct2d(
      816, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): SiLU(inplace=True)
    )
    (conv_dw): Conv2d(816, 816, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), groups=816, bias=False)
    (bn2): BatchNormAct2d(
      816, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): SiLU(inplace=True)
    )
    (aa): Identity()
    (se): SqueezeExcite(
      (conv_reduce): Conv2d(816, 34, kernel_size=(1, 1), stride=(1, 1))
      (act1): SiLU(inplace=True)
      (conv_expand): Conv2d(34, 816, kernel_size=(1, 1), stride=(1, 1))
      (gate): Sigmoid()
    )
    (conv_pwl): Conv2d(816, 136, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNormAct2d(
      136, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): Identity()
    )
    (drop_path): Identity()
  )
  (3): InvertedResidual(
    (conv_pw): Conv2d(136, 816, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNormAct2d(
      816, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): SiLU(inplace=True)
    )
    (conv_dw): Conv2d(816, 816, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), groups=816, bias=False)
    (bn2): BatchNormAct2d(
      816, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): SiLU(inplace=True)
    )
    (aa): Identity()
    (se): SqueezeExcite(
      (conv_reduce): Conv2d(816, 34, kernel_size=(1, 1), stride=(1, 1))
      (act1): SiLU(inplace=True)
      (conv_expand): Conv2d(34, 816, kernel_size=(1, 1), stride=(1, 1))
      (gate): Sigmoid()
```

```
      )
      (conv_pwl): Conv2d(816, 136, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNormAct2d(
        136, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): Identity()
      )
      (drop_path): Identity()
    )
    (4): InvertedResidual(
      (conv_pw): Conv2d(136, 816, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNormAct2d(
        816, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (conv_dw): Conv2d(816, 816, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), groups=816, bias=False)
      (bn2): BatchNormAct2d(
        816, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (aa): Identity()
      (se): SqueezeExcite(
        (conv_reduce): Conv2d(816, 34, kernel_size=(1, 1), stride=(1, 1))
        (act1): SiLU(inplace=True)
        (conv_expand): Conv2d(34, 816, kernel_size=(1, 1), stride=(1, 1))
        (gate): Sigmoid()
      )
      (conv_pwl): Conv2d(816, 136, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNormAct2d(
        136, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): Identity()
      )
      (drop_path): Identity()
    )
  )
  (5): Sequential(
    (0): InvertedResidual(
      (conv_pw): Conv2d(136, 816, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNormAct2d(
        816, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (conv_dw): Conv2dSame(816, 816, kernel_size=(5, 5), stride=(2, 2), groups=816, bias=False)
      (bn2): BatchNormAct2d(
        816, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (aa): Identity()
      (se): SqueezeExcite(
        (conv_reduce): Conv2d(816, 34, kernel_size=(1, 1), stride=(1, 1))
        (act1): SiLU(inplace=True)
        (conv_expand): Conv2d(34, 816, kernel_size=(1, 1), stride=(1, 1))
        (gate): Sigmoid()
      )
      (conv_pwl): Conv2d(816, 232, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNormAct2d(
        232, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): Identity()
      )
      (drop_path): Identity()
    )
    (1): InvertedResidual(
      (conv_pw): Conv2d(232, 1392, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNormAct2d(
        1392, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (conv_dw): Conv2d(1392, 1392, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), groups=1392, bias=False)
      (bn2): BatchNormAct2d(
        1392, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (aa): Identity()
      (se): SqueezeExcite(
        (conv_reduce): Conv2d(1392, 58, kernel_size=(1, 1), stride=(1, 1))
        (act1): SiLU(inplace=True)
        (conv_expand): Conv2d(58, 1392, kernel_size=(1, 1), stride=(1, 1))
```

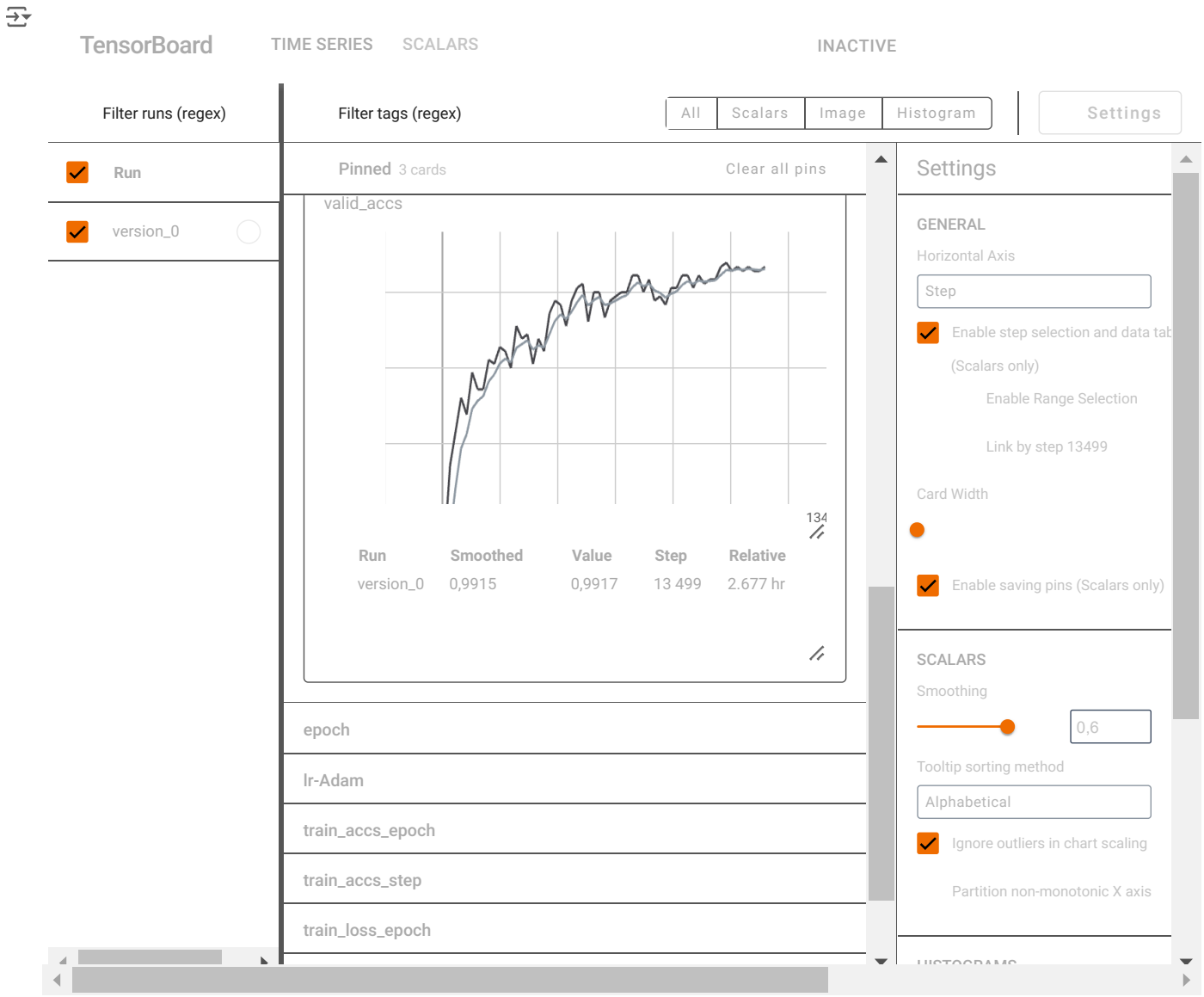
```
      (gate): Sigmoid()
    )
    (conv_pwl): Conv2d(1392, 232, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNormAct2d(
      232, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): Identity()
    )
    (drop_path): Identity()
  )
  (2): InvertedResidual(
    (conv_pw): Conv2d(232, 1392, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNormAct2d(
      1392, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): SiLU(inplace=True)
    )
    (conv_dw): Conv2d(1392, 1392, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), groups=1392, bias=False)
    (bn2): BatchNormAct2d(
      1392, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): SiLU(inplace=True)
    )
    (aa): Identity()
    (se): SqueezeExcite(
      (conv_reduce): Conv2d(1392, 58, kernel_size=(1, 1), stride=(1, 1))
      (act1): SiLU(inplace=True)
      (conv_expand): Conv2d(58, 1392, kernel_size=(1, 1), stride=(1, 1))
      (gate): Sigmoid()
    )
    (conv_pwl): Conv2d(1392, 232, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNormAct2d(
      232, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): Identity()
    )
    (drop_path): Identity()
  )
  (3): InvertedResidual(
    (conv_pw): Conv2d(232, 1392, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNormAct2d(
      1392, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): SiLU(inplace=True)
    )
    (conv_dw): Conv2d(1392, 1392, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), groups=1392, bias=False)
    (bn2): BatchNormAct2d(
      1392, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): SiLU(inplace=True)
    )
    (aa): Identity()
    (se): SqueezeExcite(
      (conv_reduce): Conv2d(1392, 58, kernel_size=(1, 1), stride=(1, 1))
      (act1): SiLU(inplace=True)
      (conv_expand): Conv2d(58, 1392, kernel_size=(1, 1), stride=(1, 1))
      (gate): Sigmoid()
    )
    (conv_pwl): Conv2d(1392, 232, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNormAct2d(
      232, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): Identity()
    )
    (drop_path): Identity()
  )
  (4): InvertedResidual(
    (conv_pw): Conv2d(232, 1392, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNormAct2d(
      1392, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): SiLU(inplace=True)
    )
    (conv_dw): Conv2d(1392, 1392, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), groups=1392, bias=False)
    (bn2): BatchNormAct2d(
      1392, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
      (drop): Identity()
      (act): SiLU(inplace=True)
    )
    (aa): Identity()
    (se): SqueezeExcite(
      (conv_reduce): Conv2d(1392, 58, kernel_size=(1, 1), stride=(1, 1))
      (act1): SiLU(inplace=True)
      (conv_expand): Conv2d(58, 1392, kernel_size=(1, 1), stride=(1, 1))
      (gate): Sigmoid()
    )
```

```
      (conv_pwl): Conv2d(1392, 232, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNormAct2d(
        232, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): Identity()
      )
      (drop_path): Identity()
    )
    (5): InvertedResidual(
      (conv_pw): Conv2d(232, 1392, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNormAct2d(
        1392, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (conv_dw): Conv2d(1392, 1392, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), groups=1392, bias=False)
      (bn2): BatchNormAct2d(
        1392, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (aa): Identity()
      (se): SqueezeExcite(
        (conv_reduce): Conv2d(1392, 58, kernel_size=(1, 1), stride=(1, 1))
        (act1): SiLU(inplace=True)
        (conv_expand): Conv2d(58, 1392, kernel_size=(1, 1), stride=(1, 1))
        (gate): Sigmoid()
      )
      (conv_pwl): Conv2d(1392, 232, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNormAct2d(
        232, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): Identity()
      )
      (drop_path): Identity()
    )
  )
  (6): Sequential(
    (0): InvertedResidual(
      (conv_pw): Conv2d(232, 1392, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNormAct2d(
        1392, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (conv_dw): Conv2d(1392, 1392, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=1392, bias=False)
      (bn2): BatchNormAct2d(
        1392, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (aa): Identity()
      (se): SqueezeExcite(
        (conv_reduce): Conv2d(1392, 58, kernel_size=(1, 1), stride=(1, 1))
        (act1): SiLU(inplace=True)
        (conv_expand): Conv2d(58, 1392, kernel_size=(1, 1), stride=(1, 1))
        (gate): Sigmoid()
      )
      (conv_pwl): Conv2d(1392, 384, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNormAct2d(
        384, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): Identity()
      )
      (drop_path): Identity()
    )
    (1): InvertedResidual(
      (conv_pw): Conv2d(384, 2304, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNormAct2d(
        2304, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (conv_dw): Conv2d(2304, 2304, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=2304, bias=False)
      (bn2): BatchNormAct2d(
        2304, eps=0.001, momentum=0.1, affine=True, track_running_stats=True
        (drop): Identity()
        (act): SiLU(inplace=True)
      )
      (aa): Identity()
      (se): SqueezeExcite(
        (conv_reduce): Conv2d(2304, 96, kernel_size=(1, 1), stride=(1, 1))
```

```
%load_ext tensorboard
```

Построение графиков(lr, accuracy на train, accuracy на valid)#LBL7

```
%tensorboard --logdir runs/lightning_logs #LBL7
```



Класс модели для её загрузки

```
STEPS=10
class Classifier(L.LightningModule):

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.steps=STEPS
        self.lr = 0.001
        self.alpha=1.0
        self.wdc= 0 #1.2e-4
        self.model = self.get_model()
        self.loss_fn = nn.CrossEntropyLoss()
        self.accuracy = torchmetrics.classification.Accuracy(
            task="multiclass",
            num_classes=9,
        )

    def get_model(self):
        return get_mobilenet_v2_torchvision()

    # def configure_optimizers(self):
    #     return torch.optim.Adam(self.parameters(), lr=self.lr,weight_decay=self.wdc)

    def training_step(self, batch):
```

```
        return self._step(batch, "train")

    def validation_step(self, batch):
        return self._step(batch, "valid")

    def _step(self, batch, kind):
        x, y = batch

        p = self.model(x)
        loss = self.loss_fn(p, y)
        accs = self.accuracy(p.argmax(axis=-1), y)

        return self._log_metrics(loss, accs, kind)

    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters(), lr=self.lr,weight_decay=self.wdc)

        steps_per_epoch = self.steps
        total_steps = EPOCHS * steps_per_epoch

        scheduler = cosine_annealing_lr(
            optimizer,
            total_steps=total_steps
        )

        lr_scheduler = {
            "scheduler": scheduler,
            "interval": "step",
            "frequency": 1,
        }
        return [optimizer], [lr_scheduler]

    def _log_metrics(self, loss, accs, kind):
        metrics = {}
        if loss is not None:
            metrics[f"{kind}_loss"] = loss
        if accs is not None:
            metrics[f"{kind}_accs"] = accs
        self.log_dict(
            metrics,
            prog_bar=True,
            logger=True,
            on_step=kind == "train",
            on_epoch=True,
        )
        return loss
```

Сохранение модели с чекпоинта, на котором значение accuracy на валидации максимально(#LBL8)

```
best_ckpt = "runs/lightning_logs/version_0/checkpoints/epoch=52-valid_accs=0.992.ckpt"  #LBL8

#load best checkpoint
my_training_module = Classifier.load_from_checkpoint(best_ckpt)
model=my_training_module.model
sd = model.state_dict()
torch.save(sd, "Classifier.pt")
```

Тестирование полученной модели

```
class Dataset:

    def __init__(self, name):
        self.name = name
        self.is_loaded = False
        url = f"https://drive.google.com/uc?export=download&confirm=pbef&id={DATASETS_LINKS[name]}"
        output = f'{name}.npz'
        gdown.download(url, output, quiet=False)
        print(f'Loading dataset {self.name} from npz.')
        np_obj = np.load(f'{name}.npz')
        self.images = np_obj['data']
        self.labels = np_obj['labels']
        self.n_files = self.images.shape[0]
        self.is_loaded = True
```

```
            print(f'Done. Dataset {name} consists of {self.n_files} images.')

        def image(self, i):
            # read i-th image in dataset and return it as numpy array
            if self.is_loaded:
                return self.images[i, :, :, :]

        def images_seq(self, n=None):
            # sequential access to images inside dataset (is needed for testing)
            for i in range(self.n_files if not n else n):
                yield self.image(i)

        def random_image_with_label(self):
            # get random image with label from dataset
            i = np.random.randint(self.n_files)
            return self.image(i), self.labels[i]

        def random_batch_with_labels(self, n):
            # create random batch of images with labels (is needed for training)
            indices = np.random.choice(self.n_files, n)
            imgs = []
            for i in indices:
                img = self.image(i)
                imgs.append(self.image(i))
            logits = np.array([self.labels[i] for i in indices])
            return np.stack(imgs), logits

        def image_with_label(self, i: int):
            # return i-th image with label from dataset
            return self.image(i), self.labels[i]


    class Metrics:

        @staticmethod
        def accuracy(gt: List[int], pred: List[int]):
            assert len(gt) == len(pred), 'gt and prediction should be of equal length'
            return sum(int(i[0] == i[1]) for i in zip(gt, pred)) / len(gt)

        @staticmethod
        def accuracy_balanced(gt: List[int], pred: List[int]):
            return balanced_accuracy_score(gt, pred)

        @staticmethod
        def print_all(gt: List[int], pred: List[int], info: str):
            print(f'metrics for {info}:')
            print('\t accuracy {:.4f}:'.format(Metrics.accuracy(gt, pred)))
            print('\t balanced accuracy {:.4f}:'.format(Metrics.accuracy_balanced(gt, pred)))


    class Model:

        def __init__(self):
            # todo
            self.model = get_mobilenet_v2_torchvision()

        def save(self, name: str):
            # todo
            pass
            # example demonstrating saving the model to PROJECT_DIR folder on gdrive with name 'name'
            sd = self.model.state_dict()
            torch.save(sd, f'/content/drive/MyDrive/{name}.pt')
            #np.savez(f'/content/drive/MyDrive/{name}.npz', data=arr)

        def load(self, name: str):
            # todo

            # example demonstrating loading the model with name 'name' from gdrive using link
            DEVICE = torch.device("cpu")
            #model=get_mobilenet_v2_torchvision(num_classes=9, transfer=False)

            name_to_id_dict = {
                'best': '11AU2Ty9OKI2DrssSo-KK7iU0t1lzAvrA'
            }
            output = f'{name}.pt'
            gdown.download(f'https://drive.google.com/uc?id={name_to_id_dict[name]}', output, quiet=False)
            #np_obj = np.load(f'{name}.npz')
            sd = torch.load(
```

```
            f"{name}.pt",
            map_location=DEVICE,
            weights_only=True,)
        self.model.load_state_dict(sd);
        self.model = self.model.to(DEVICE).eval()
```

```
    def test_on_dataset(self, dataset: Dataset, limit=None):
        # you can upgrade this code if you want to speed up testing using batches

        predictions = []
        n = dataset.n_files if not limit else int(dataset.n_files * limit)
        for img in tqdm(dataset.images_seq(n), total=n):
            predictions.append(self.test_on_image(img))
        return predictions

    def test_on_image(self, img: np.ndarray):
        # todo: replace this code
        IMAGENET_MEAN = [0.485, 0.456, 0.406]
        IMAGENET_STD = [0.229, 0.224, 0.225]
        common = [
            A.ToFloat(max_value=255),
            A.Normalize(max_pixel_value=1.0, mean=IMAGENET_MEAN, std=IMAGENET_STD),
            A.pytorch.transforms.ToTensorV2(),

        ]
        com = A.Compose(common)
        def my_common_transform(image):
            p=com(image=image)
            return p["image"]

        img=my_common_transform(img)
        with torch.no_grad():
          p=self.model(img[None,:,:,:])

        p=p.cpu().numpy()[0]
        # points=points.clip(0,NETWORK_SIZE[0]-0.005)
        p=p.argmax(axis=-1)
        #matches = p == label
        return p
```

Загрузка модели

```
model = Model()
if not EVALUATE_ONLY:
    # model.train(d_train)
    model.save('best')
else:
    #todo: your link goes here
    model.load('best')
```

```
Downloading...
From (original): https://drive.google.com/uc?id=11AU2Ty9OKI2DrssSo-KK7iU0t1lzAvrA
From (redirected): https://drive.google.com/uc?id=11AU2Ty9OKI2DrssSo-KK7iU0t1lzAvrA&confirm=t&uuid=f3066919-cd8c-43cb-a9bb-6
To: /content/best.pt
100%|██████████| 43.4M/43.4M [00:01<00:00, 43.2MB/s]
```

Загрузка тестового набора данных

```
d_test = Dataset('test')
```

```
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=14LvKrM8QXor9d0ZApGGRXSSxy4kW167b
To: /content/test.npz
100%|██████████| 525M/525M [00:07<00:00, 72.7MB/s]
Loading dataset test from npz.
Done. Dataset test consists of 4500 images.
```

Получение результатов на тесте

```
pred = model.test_on_dataset(d_test)
Metrics.print_all(d_test.labels, pred, 'test')
```

```
100%                                                4500/4500 [11:13<00:00,  5.91it/s]
metrics for test:
        accuracy 0.9918:
```

Построение матрицы ошибок(#LBL9)

```
import pandas as pd
a=np.hstack((np.array(d_test.labels)[:,None],np.array(pred)[:,None]))  #LBL9
b=np.zeros((9,9))
classes=list(TISSUE_CLASSES)
for i in range(a.shape[0]):
  b[a[i,0],a[i,1]]+=1
confusion_matrix=pd.DataFrame(data=b,
          index=TISSUE_CLASSES,
          columns=TISSUE_CLASSES)
confusion_matrix.index.names=['true\pred']
print(confusion_matrix)
```