ification-of-histological-images-3

November 26, 2023

№1

:

[ ]:
```
!pip install -q tqdm
!pip install --upgrade --no-cache-dir gdown
```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages
(4.7.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from gdown) (3.13.1)
Requirement already satisfied: requests[socks] in
/usr/local/lib/python3.10/dist-packages (from gdown) (2.31.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from gdown) (1.16.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from gdown) (4.66.1)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-
packages (from gdown) (4.11.2)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-
packages (from beautifulsoup4->gdown) (2.5)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests[socks]->gdown) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown)
(2023.7.22)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in
/usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.7.1)
```

Google Drive :

[ ]:
```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

, , (gdrive) :

```
[ ]: EVALUATE_ONLY = True
TEST_ON_LARGE_DATASET = True
TISSUE_CLASSES = ('ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS', 'NORM', 'STR',␣
 ↪'TUM')
#                  ,
DATASETS_LINKS = {
    'train': '1XtQzVQ5XbrfxpLHJuL0XBGJ5U7CS-cLi',
    'train_small': '1qd45xXfDwdZjktLFwQb-et-mAaFeCzOR',
    'train_tiny': '1I-2ZOuXLd4QwhZQQltp817Kn3J0Xgbui',
    'test': '1RfPou3pFKpuHDJZ-D9XDFzgvwpUBFlDr',
    'test_small': '1wbRsogOn7uGlHIPGLhyN-PMeT2kdQ2lI',
    'test_tiny': '1viiBOs041CNsAK4itvX8PnYthJ-MDnQc'
}

MY_DATASETS_LINKS = {
    'train': '1_IGG3T6R9plow38LTFBCf-2HHDZNSXXo',
    'train_small': '1W1LEIVpLTHk8fO_bXeHh-rNUuFTb_aiq',
    'train_tiny': '11AeOrMHidGyYAKCydoU5HMX4y8TTtldI',
    'test': '11YvaW-tT6GtSN5uX6IukmBl1nZR88eev',
    'test_small': '1yizFhQtif2_1A7gtnvCSKhGiatLY7FqQ',
    'test_tiny': '1ek3WB-xOa_-q5dhSGMYI8HNbgcZoRZCy'
}
```

:

```
[ ]: 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
```

0.0.1 Dataset

, , ( ).

```
[ ]: import numpy as np
import matplotlib.pyplot as plt

from PIL import Image
```

```
import torch
import torch.nn as nn
import torch.optim as optim

import torchvision
from torchvision import datasets, models, transforms
from torch.utils.data import Dataset, DataLoader

class MyDataset(Dataset):
    def __init__(self, name):
        self.name = name
        self.mode = 'train' if 'train' in self.name.lower() else 'test'
        self.is_loaded = False
        # url = f"https://drive.google.com/uc?
 ↪export=download&confirm=pbef&id={DATASETS_LINKS[name]}"
        url = f"https://drive.google.com/uc?
 ↪export=download&confirm=pbef&id={MY_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.data = np_obj


        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 get_data(self):
      return self.data


    def __len__(self):
        return self.n_files

    def image(self, i):
        # read i-th image in dataset and return it as numpy array
        if self.is_loaded:
            return Image.fromarray(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 __getitem__(self, i):
      resnet_transforms = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224), #                                   224*224
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
      ])
      #       11.                    ,
      #                                ,
      #    ,                   ,                        ,
      #    -

      #                              ,
      # augmentation = transforms.RandomChoice([
      #           transforms.Compose([
      #               transforms.RandomCrop(224),
      #               transforms.CenterCrop(224),
      #           ]),
      #           # transforms.RandomRotation(degrees=(-10, 10)),
      #           transforms.RandomHorizontalFlip(p=0.5),
      # ])
      # transform_with_augmentation = transforms.Compose([
      #   augmentation,
      #   transforms.Resize(size=(224, 224), antialias=True),
      #   transforms.CenterCrop(224),
      #   transforms.ToTensor(),
      #   transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
      # ])
      # if (self.mode == 'train'):
      #   return (transform_with_augmentation(self.image(i))), self.labels[i]
      # else:
      return (resnet_transforms(self.image(i))), self.labels[i]
```

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

0.0.2 Dataset

, . , . ,
.

[ ]:
```
# d_train_tiny = MyDataset('train_tiny')

# img, lbl = d_train_tiny.random_image_with_label()

# print(img)
# print(f'Got numpy array of shape {img.shape}, and label with code {lbl}.')
# print(f'Label code corresponds to {TISSUE_CLASSES[lbl]} class.')

# pil_img = Image.fromarray(img)
# IPython.display.display(pil_img)
```

0.0.3 Metrics

, : 1. , 2. .

[ ]:
```
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)))
```

0.0.4 Model

, .

save, load .

.

, , . , ,

, , ,

, .

, , : 1. ; 2. ; 3.

; 4. - ( );

5. (,); 6. ,

(,); 7. /

( ); 8. ;

9. ; 10.

( ) 11. . .

.

.

[ ]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,␣
↪recall_score, f1_score
def evaluate(model, dataloader, loss_fn):

    losses = []

    num_correct = 0
    num_elements = 0

    for i, batch in enumerate(dataloader):

        X_batch, y_batch = batch
        num_elements += len(y_batch)


        with torch.no_grad():
            logits = model(X_batch.to(device))

            loss = loss_fn(logits, y_batch.to(device))
            losses.append(loss.item())

            y_pred = torch.argmax(logits, dim=1)


            num_correct += torch.sum(y_pred.cpu() == y_batch)
```

```
    accuracy = num_correct / num_elements

    return accuracy.numpy(), np.mean(losses)

def plot_confusion_matrix(y_true, y_pred, classes):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(len(classes), len(classes)))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=classes,␣
 ↪yticklabels=classes)
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Confusion Matrix')
    plt.show()
```

[ ]:
```
import torch
import torch.nn as nn
import torch.optim as optim

import torchvision
from torchvision import datasets, models, transforms
from IPython import display
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

[ ]:
```
class Model:

    def __init__(self):
        num_freeze_layers = 7
        num_out_classes = 10
        #       13 -          transfer learning            ))
        self.model = models.resnet50(weights="DEFAULT")
        self.model.fc = nn.Linear(2048, num_out_classes)

        for i, layer in enumerate(self.model.children()):
            if i < num_freeze_layers:
                for param in layer.parameters():
                    param.requires_grad = False

        self.model = self.model.to(device)

        self.loss_fn = torch.nn.CrossEntropyLoss()


        self.learning_rate = 1e-3
        #       9
        self.optimizer = torch.optim.Adam(self.model.parameters(), lr=self.
 ↪learning_rate)
```

```
  def save(self, name: str):
      PATH = '/content/drive/MyDrive/Colab Notebooks/Pretrained'
      torch.save(self.model, PATH + f'/{name}')

  def load(self, name: str):
      if name == 'best':
        url = f"https://drive.google.com/uc?
↪export=download&confirm=pbef&id=1-McLB3PGOGerrq1CfWWXhYH52NVj6VB8"
        # link = 'https://drive.google.com/file/d/
↪1-McLB3PGOGerrq1CfWWXhYH52NVj6VB8/view?usp=share_link'
        output = f'{name}'
        gdown.download(url, output, quiet=False)
        self.model = torch.load(f'{name}')
      else:
        PATH = '/content/drive/MyDrive/Colab Notebooks/Pretrained'
        self.model = torch.load(PATH + f'/{name}')
  #      4
  def load_from_epoch(self, name: str, n_epoch: int):
      PATH = '/content/drive/MyDrive/Colab Notebooks/Pretrained'
      self.model = torch.load(PATH + f'/{name}_epoch_{n_epoch}')


  def train(self, dataset: MyDataset, n_epoch: int, name='basic_model', ␣
↪additional_training=False, model_for_add='basic'):
      #      12
      if additional_training:
        self.load(model_for_add)

      train_size = int(len(dataset) * 0.8)
      val_size = len(dataset) - train_size
      #      2
      train_data, val_data = torch.utils.data.random_split(dataset,␣
↪[train_size, val_size])
      train_loader = torch.utils.data.DataLoader(dataset, batch_size=64,␣
↪shuffle=True)
      val_loader = torch.utils.data.DataLoader(val_data, batch_size=64,␣
↪shuffle=False)
      train_losses = []
      val_losses = []
      print(f'training started')
      for epoch in range(n_epoch):
        print("Epoch:", epoch+1)
        self.model.train(True)

        running_losses = []
```

```
        running_accuracies = []

        all_preds = []
        all_labels = []

        for i, batch in enumerate(train_loader):
            X_batch, y_batch = batch

            logits = self.model(X_batch.to(device))

            #
            loss = self.loss_fn(logits, y_batch.to(device))
            running_losses.append(loss.item())

            loss.backward() #
            self.optimizer.step() #
            self.optimizer.zero_grad() #

            model_answers = torch.argmax(logits, dim=1)
            train_accuracy = torch.sum(y_batch == model_answers.cpu()) /␣
↪len(y_batch)
            running_accuracies.append(train_accuracy)

            all_preds.extend(model_answers.cpu().numpy())
            all_labels.extend(y_batch.cpu().numpy())

            #      6 - j
            if (i+1) % 10 == 0:
              display.clear_output(wait=True)
              plt.plot(running_losses, label='Batch Loss')
              plt.title('Training Loss per Iteration')
              plt.xlabel('Iteration')
              plt.ylabel('Loss')
              plt.legend()
              plt.show()

            #       5
            if (i+1) % 50 == 0:
                print("    train     accuracy        50     :",
                      np.mean(running_losses), np.mean(running_accuracies),␣
↪end='\n')


        self.model.train(False)
        #      3
        self.save(f"{name}_epoch_{epoch}")
        #      1
```

```
        val_accuracy, val_loss = evaluate(self.model, val_loader,␣
↪loss_fn=self.loss_fn)
        #      5
        print("    {}/{}: val      accuracy:".format(epoch+1, n_epoch,),
                    val_loss, val_accuracy, end='\n')

        train_losses.append(np.mean(running_losses))
        val_losses.append(val_loss)


      print(f'training done')
      #      6
      plt.plot(train_losses, label='Train Loss ')
      plt.plot(val_losses, label='Validation Loss')
      plt.title('Training and Validation Loss')
      plt.xlabel('Epoch')
      plt.ylabel('Loss')
      plt.legend()
      plt.show()

      #       8     1
      sensitivity = recall_score(all_labels, all_preds, average='weighted')
      specificity = accuracy_score(all_labels, all_preds)
      print("          (Sensitivity):", sensitivity)
      print("        (Specificity):", specificity)

      #      10
      self.save(f"{name}_complited")
      #      8     2
      plot_confusion_matrix(all_labels, all_preds, classes=[f'class_{i}' for␣
↪i in range(9)])




  def test_on_dataset(self, dataset: Dataset, limit=None):
      test_loader = torch.utils.data.DataLoader(dataset, batch_size=64,␣
↪shuffle=False)
      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):

      resnet_transforms = transforms.Compose([
```

```
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
        ])
        logit = self.model(resnet_transforms(img).unsqueeze(0).to(device)).cpu()
        return torch.nn.functional.softmax(logit, dim=-1).detach().numpy().
↪argmax()
```

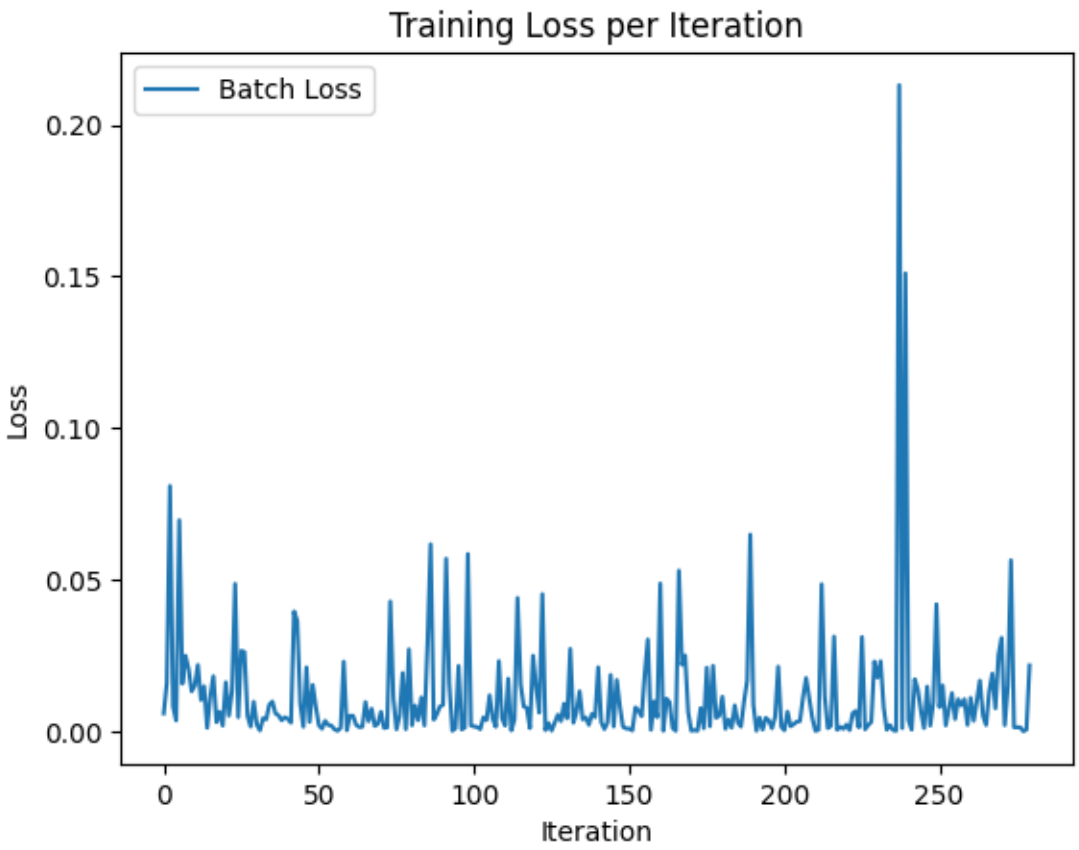
0.0.5

. . . ‘train_small’ ‘test_small’.

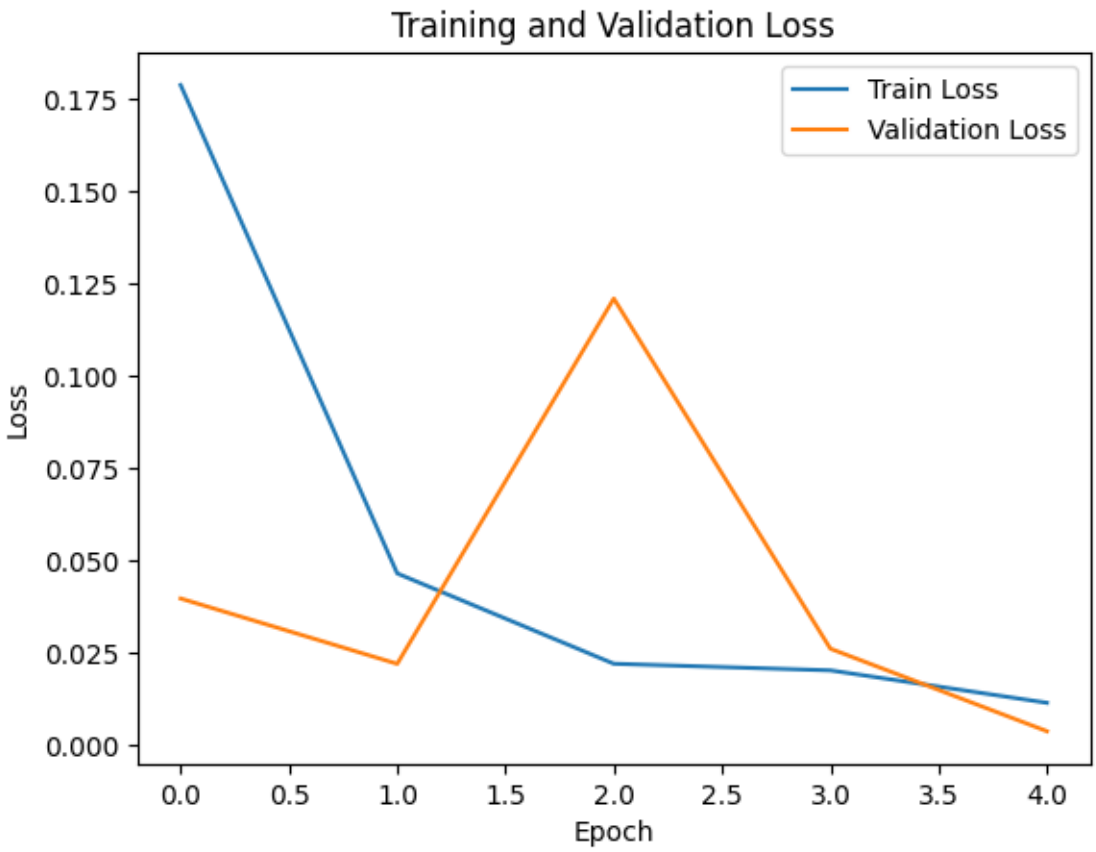
[ ]:
```
d_train = MyDataset('train')
d_test = MyDataset('test')
```

```
Downloading…
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1_IGG3T6R9plow
38LTFBCf-2HHDZNSXXo
To: /content/train.npz
100%|      | 2.10G/2.10G [00:19<00:00, 105MB/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=11YvaW-
tT6GtSN5uX6IukmBl1nZR88eev
To: /content/test.npz
100%|      | 525M/525M [00:02<00:00, 251MB/s]

Loading dataset test from npz.
Done. Dataset test consists of 4500 images.
```

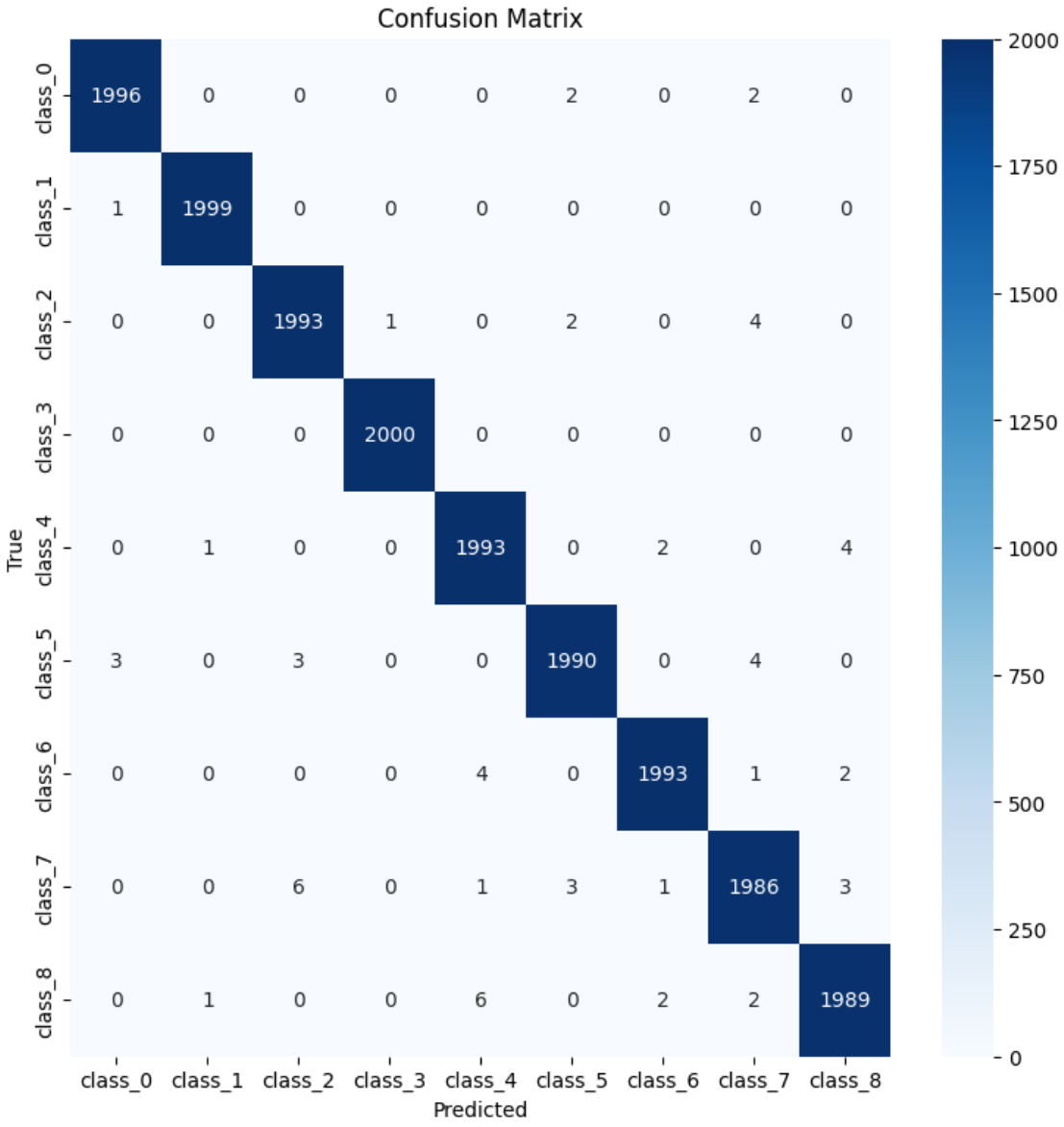
[ ]:
```
EVALUATE_ONLY = False
model = Model()
if not EVALUATE_ONLY:
    model.train(d_train, 5,'best')
    model.save('best')
else:
    model.load('best')
```




```
    5/5: val      accuracy: 0.0038461954407051835 0.9994444
training done
```




```
 (Sensitivity): 0.9966111111111111
(Specificity): 0.9966111111111111
```



:

[ ]:
```
# evaluating model on 10% of test dataset
pred_1 = model.test_on_dataset(d_test, limit=0.1)
Metrics.print_all(d_test.labels[:len(pred_1)], pred_1, '10% of test')
```

```
  0%|          | 0/450 [00:00<?, ?it/s]

metrics for 10% of test:
        accuracy 0.9978:
        balanced accuracy 0.9978:

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:2184:
```

```
UserWarning: y_pred contains classes not in y_true
  warnings.warn("y_pred contains classes not in y_true")
```

:

```
[ ]: # evaluating model on full test dataset (may take time)
TEST_ON_LARGE_DATASET = True
if TEST_ON_LARGE_DATASET:
    pred_2 = model.test_on_dataset(d_test)
    Metrics.print_all(d_test.labels, pred_2, 'test')
```

```
  0%|          | 0/4500 [00:00<?, ?it/s]

metrics for test:
         accuracy 0.9833:
         balanced accuracy 0.9833:
```

.

, .. ,

..

pdf (->)

pdf .

0.0.6

. , test_tiny,

(2%) test. ,

.

, .

```
[ ]: final_model = Model()
final_model.load('best')
d_test_tiny = MyDataset('test_tiny')
pred = model.test_on_dataset(d_test_tiny)
Metrics.print_all(d_test_tiny.labels, pred, 'test-tiny')
```

```
Downloading…
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1-McLB3PG0Gerr
q1CfWWXhYH52NVj6VB8
To: /content/best
100%|      | 94.4M/94.4M [00:00<00:00, 208MB/s]
Downloading…
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1ek3WB-
xOa_-q5dhSGMYI8HNbgcZoRZCy
To: /content/test_tiny.npz
100%|      | 10.6M/10.6M [00:00<00:00, 149MB/s]

Loading dataset test_tiny from npz.
Done. Dataset test_tiny consists of 90 images.
```

```
  0%|          | 0/90 [00:00<?, ?it/s]
metrics for test-tiny:
         accuracy 0.9889:
         balanced accuracy 0.9889:
```

Google Drive.

```
[ ]: drive.flush_and_unmount()
```

---

1 “ ”

, .

1.0.1

- timeit :

```
[ ]: import timeit

def factorial(n):
    res = 1
    for i in range(1, n + 1):
        res *= i
    return res


def f():
    return factorial(n=1000)

n_runs = 128
print(f'Function f is caluclated {n_runs} times in {timeit.timeit(f,␣
 ↪number=n_runs)}s.')
```

1.0.2 Scikit-learn

“ ” scikit-learn (https://scikit-learn.org/stable/). MNIST SVM:

```
[ ]: # Standard scientific Python imports
import matplotlib.pyplot as plt

# Import datasets, classifiers and performance metrics
from sklearn import datasets, svm, metrics
from sklearn.model_selection import train_test_split
```

```
# The digits dataset
digits = datasets.load_digits()

# The data that we are interested in is made of 8x8 images of digits, let's
# have a look at the first 4 images, stored in the `images` attribute of the
# dataset.  If we were working from image files, we could load them using
# matplotlib.pyplot.imread.  Note that each image must have the same size. For␣
 ↪these
# images, we know which digit they represent: it is given in the 'target' of
# the dataset.
_, axes = plt.subplots(2, 4)
images_and_labels = list(zip(digits.images, digits.target))
for ax, (image, label) in zip(axes[0, :], images_and_labels[:4]):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    ax.set_title('Training: %i' % label)

# To apply a classifier on this data, we need to flatten the image, to
# turn the data in a (samples, feature) matrix:
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

# Create a classifier: a support vector classifier
classifier = svm.SVC(gamma=0.001)

# Split data into train and test subsets
X_train, X_test, y_train, y_test = train_test_split(
    data, digits.target, test_size=0.5, shuffle=False)

# We learn the digits on the first half of the digits
classifier.fit(X_train, y_train)

# Now predict the value of the digit on the second half:
predicted = classifier.predict(X_test)

images_and_predictions = list(zip(digits.images[n_samples // 2:], predicted))
for ax, (image, prediction) in zip(axes[1, :], images_and_predictions[:4]):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    ax.set_title('Prediction: %i' % prediction)

print("Classification report for classifier %s:\n%s\n"
      % (classifier, metrics.classification_report(y_test, predicted)))
disp = metrics.plot_confusion_matrix(classifier, X_test, y_test)
disp.figure_.suptitle("Confusion Matrix")
print("Confusion matrix:\n%s" % disp.confusion_matrix)
```

```
plt.show()
```

1.0.3 Scikit-image

, numpy,
, , scikit-image (https://scikit-image.org/). Canny edge detector.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage as ndi

from skimage import feature


# Generate noisy image of a square
im = np.zeros((128, 128))
im[32:-32, 32:-32] = 1

im = ndi.rotate(im, 15, mode='constant')
im = ndi.gaussian_filter(im, 4)
im += 0.2 * np.random.random(im.shape)

# Compute the Canny filter for two values of sigma
edges1 = feature.canny(im)
edges2 = feature.canny(im, sigma=3)

# display results
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(8, 3),
                                    sharex=True, sharey=True)

ax1.imshow(im, cmap=plt.cm.gray)
ax1.axis('off')
ax1.set_title('noisy image', fontsize=20)

ax2.imshow(edges1, cmap=plt.cm.gray)
ax2.axis('off')
ax2.set_title(r'Canny filter, $\sigma=1$', fontsize=20)

ax3.imshow(edges2, cmap=plt.cm.gray)
ax3.axis('off')
ax3.set_title(r'Canny filter, $\sigma=3$', fontsize=20)

fig.tight_layout()

plt.show()
```

1.0.4 Tensorflow 2

Tensorflow 2.
, MNIST.

[ ]:
```
# Install TensorFlow

import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

model.evaluate(x_test,  y_test, verbose=2)
```

, Google Colab
GPU TPU. “ ” -> “ ”.

Tensorflow 2
https://www.tensorflow.org/tutorials?hl=ru.

, Tensorflow 2. ,
, TensorFlow 2. , (
), : https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly.

1.0.5 Numba

, ,
for python , JIT- Numba
(https://numba.pydata.org/). Numba Google Colab : 1.
https://colab.research.google.com/github/cbernet/maldives/blob/master/numba/numba_cuda.ipynb
2. https://colab.research.google.com/github/evaneschneider/parallel-programming/blob/master/COMPASS_gpu_intro.ipynb

, Numba , , ,

. Numba .

1.0.6 zip Google Drive

zip . ,
zip . ,
Google Drive.

2 , tmp PROJECT_DIR, tmp tmp.zip.

[ ]:
```
PROJECT_DIR = "/dev/prak_nn_1/"
arr1 = np.random.rand(100, 100, 3) * 255
arr2 = np.random.rand(100, 100, 3) * 255

img1 = Image.fromarray(arr1.astype('uint8'))
img2 = Image.fromarray(arr2.astype('uint8'))

p = "/content/drive/MyDrive/" + PROJECT_DIR

if not (Path(p) / 'tmp').exists():
    (Path(p) / 'tmp').mkdir()

img1.save(str(Path(p) / 'tmp' / 'img1.png'))
img2.save(str(Path(p) / 'tmp' / 'img2.png'))

%cd $p
!zip -r "tmp.zip" "tmp"
```

tmp.zip tmp2 PROJECT_DIR. tmp2 tmp,
2 .

[ ]:
```
p = "/content/drive/MyDrive/" + PROJECT_DIR
%cd $p
!unzip -uq "tmp.zip" -d "tmp2"
```