In [ ]:
```
%load_ext autoreload
%autoreload 2
```

In [ ]:
```
from MLP_classifier import MultiClassClassifier
from torch.utils.data import DataLoader
import sys
sys.path.append(".")
from dataset import *
import torch.nn as nn
import torch
sys.path.append("../tools")
from constants import *
from sklearn.model_selection import train_test_split
from torch.utils.data import random_split
import matplotlib.pyplot as plt
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsOneClassifier
sys.path.append("../tools")
from utils import plot_loss_history
```

```
/home/lsaland/micromamba/envs/clip/lib/python3.11/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See http
s://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
/home/lsaland/micromamba/envs/clip/lib/python3.11/site-packages/torch/cuda/__init__.py:619: UserWarning: Can't initialize NVML
  warnings.warn("Can't initialize NVML")
/home/lsaland/micromamba/envs/clip/lib/python3.11/site-packages/diffusers/models/transformers/transformer_2d.py:34: FutureWarning: `Transformer2DModelOutput` is d
eprecated and will be removed in version 1.0.0. Importing `Transformer2DModelOutput` from `diffusers.models.transformer_2d` is deprecated and this will be removed
in a future version. Please use `from diffusers.models.modeling_outputs import Transformer2DModelOutput`, instead.
  deprecate("Transformer2DModelOutput", "1.0.0", deprecation_message)
Loading pipeline components...: 100%|██████████| 7/7 [00:00<00:00,  8.66it/s]
```

Parameters

In [ ]:
```
device = "cuda:0"
model = MultiClassClassifier(n_classes=len(GEN_TO_INT.keys())).to(device=device)

lr = 1e-3
batch_size = 64
epochs = 5

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=lr)

data = DeepFakeDatasetFastLoad(PATH_TO_DATA4 + "df_34000_V2.pt")

rng = torch.Generator().manual_seed(SEED)
train_data, test_data, validation_data = random_split(data,[0.7,0.2,0.1],generator=rng)

train_loader = DataLoader(train_data,batch_size=batch_size,shuffle=True)
test_loader  = DataLoader(test_data,batch_size=len(test_data),shuffle=True)
val_loader  = DataLoader(validation_data,batch_size=len(validation_data),shuffle=True)

model.train()
```

Out[ ]:
```
MultiClassClassifier(
  (fc1): Linear(in_features=768, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=16, bias=True)
  (act): ReLU()
)
```

Train for multi-class classification

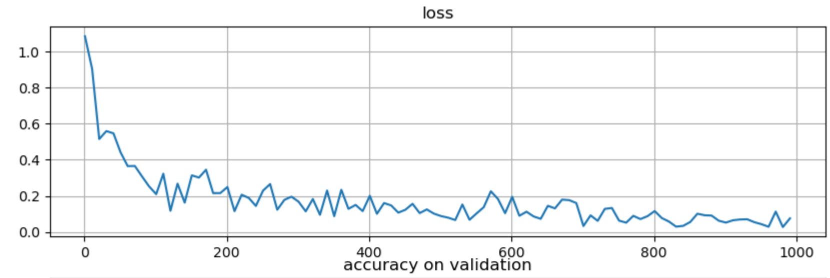
In [ ]:
```
model.set_generators_maps(gen_to_int=GEN_TO_INT,int_to_gen=INT_TO_GEN)
```

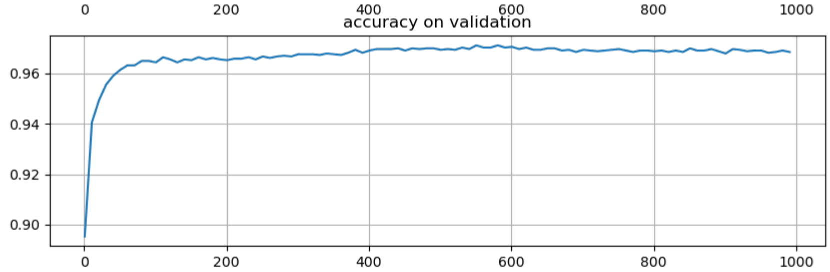
In [ ]:
```
n_epochs = 1000
val_accuracy = []
loss_history = []
for epoch in range(1,n_epochs+1):
    for idx, batch in enumerate(train_loader):
        # prediction and loss
        pred = model(batch["features"].cuda(device))
        loss = loss_fn(pred,batch["generator"].type(torch.LongTensor).cuda(device))

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

    loss, current = loss.item(), idx*batch_size + len(batch["features"])
    if epoch%10 == 0 and epoch > 0:
        loss_history.append(loss)
        for v in val_loader:
            val_accuracy.append(
                model.get_model_accuracy_binary(v["features"],
                                                v["label"],
                                                device))
        print(f"loss: {loss:>7f}  [{epoch:>5d}/{n_epochs:>5d}]")
```

```
loss: 1.083643  [   10/ 1000]
loss: 0.903609  [   20/ 1000]
loss: 0.515120  [   30/ 1000]
loss: 0.559094  [   40/ 1000]
loss: 0.546588  [   50/ 1000]
loss: 0.441140  [   60/ 1000]
loss: 0.365204  [   70/ 1000]
loss: 0.366031  [   80/ 1000]
loss: 0.308687  [   90/ 1000]
loss: 0.253033  [  100/ 1000]
loss: 0.210082  [  110/ 1000]
loss: 0.323321  [  120/ 1000]
loss: 0.117914  [  130/ 1000]
loss: 0.268384  [  140/ 1000]
loss: 0.163159  [  150/ 1000]
loss: 0.314148  [  160/ 1000]
loss: 0.301961  [  170/ 1000]
loss: 0.345350  [  180/ 1000]
loss: 0.215941  [  190/ 1000]
loss: 0.215817  [  200/ 1000]
loss: 0.249675  [  210/ 1000]
loss: 0.115867  [  220/ 1000]
loss: 0.207075  [  230/ 1000]
loss: 0.189053  [  240/ 1000]
loss: 0.144920  [  250/ 1000]
loss: 0.229414  [  260/ 1000]
loss: 0.266092  [  270/ 1000]
loss: 0.123982  [  280/ 1000]
loss: 0.178462  [  290/ 1000]
loss: 0.196176  [  300/ 1000]
loss: 0.168153  [  310/ 1000]
loss: 0.114833  [  320/ 1000]
loss: 0.183321  [  330/ 1000]
loss: 0.095777  [  340/ 1000]
loss: 0.230289  [  350/ 1000]
loss: 0.088249  [  360/ 1000]
loss: 0.234299  [  370/ 1000]
loss: 0.128863  [  380/ 1000]
loss: 0.150503  [  390/ 1000]
loss: 0.116159  [  400/ 1000]
loss: 0.201118  [  410/ 1000]
loss: 0.101444  [  420/ 1000]
loss: 0.160854  [  430/ 1000]
loss: 0.147623  [  440/ 1000]
loss: 0.107851  [  450/ 1000]
loss: 0.123872  [  460/ 1000]
loss: 0.156791  [  470/ 1000]
loss: 0.105354  [  480/ 1000]
loss: 0.125542  [  490/ 1000]
loss: 0.102317  [  500/ 1000]
loss: 0.088741  [  510/ 1000]
loss: 0.079975  [  520/ 1000]
loss: 0.066944  [  530/ 1000]
loss: 0.152889  [  540/ 1000]
loss: 0.068445  [  550/ 1000]
loss: 0.103755  [  560/ 1000]
loss: 0.138058  [  570/ 1000]
loss: 0.225921  [  580/ 1000]
loss: 0.183273  [  590/ 1000]
loss: 0.104766  [  600/ 1000]
loss: 0.195365  [  610/ 1000]
loss: 0.090388  [  620/ 1000]
loss: 0.112982  [  630/ 1000]
loss: 0.086784  [  640/ 1000]
loss: 0.073494  [  650/ 1000]
loss: 0.145628  [  660/ 1000]
loss: 0.131363  [  670/ 1000]
loss: 0.179785  [  680/ 1000]
loss: 0.176956  [  690/ 1000]
loss: 0.160835  [  700/ 1000]
loss: 0.033706  [  710/ 1000]
loss: 0.092893  [  720/ 1000]
loss: 0.062387  [  730/ 1000]
loss: 0.128669  [  740/ 1000]
loss: 0.133510  [  750/ 1000]
loss: 0.063594  [  760/ 1000]
loss: 0.052283  [  770/ 1000]
loss: 0.090075  [  780/ 1000]
loss: 0.072280  [  790/ 1000]
loss: 0.088315  [  800/ 1000]
loss: 0.116617  [  810/ 1000]
loss: 0.077982  [  820/ 1000]
loss: 0.058514  [  830/ 1000]
loss: 0.030378  [  840/ 1000]
loss: 0.034574  [  850/ 1000]
loss: 0.056091  [  860/ 1000]
loss: 0.101805  [  870/ 1000]
loss: 0.093489  [  880/ 1000]
loss: 0.091548  [  890/ 1000]
loss: 0.063140  [  900/ 1000]
loss: 0.052897  [  910/ 1000]
loss: 0.065542  [  920/ 1000]
loss: 0.070062  [  930/ 1000]
loss: 0.071158  [  940/ 1000]
loss: 0.054972  [  950/ 1000]
loss: 0.043856  [  960/ 1000]
loss: 0.029621  [  970/ 1000]
loss: 0.113798  [  980/ 1000]
loss: 0.028144  [  990/ 1000]
loss: 0.076223  [ 1000/ 1000]
```

In [ ]:

```
fig, (ax1, ax2) = plt.subplots(2,1,figsize=(10,6))
ax1.set_title("loss")
ax2.set_title("accuracy on validation")
ax1.plot(range(1,n_epochs+1,10),loss_history)
ax2.plot(range(1,n_epochs+1,10),val_accuracy)
ax1.grid()
ax2.grid()
plt.show()
```





Test for binary classification

In [ ]:
```
import torch.types

with torch.no_grad():
    model.eval()
    for e in test_loader:
        accuracy = model.get_model_accuracy_binary(features=e["features"],
                                                   true_labels=e["label"],
                                                   device=device)
print(accuracy)
```

```
0.9763234853744507
```

Comparison with SVM

In [ ]:
```
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsOneClassifier

clf = OneVsOneClassifier(LinearSVC(dual="auto"))
```

In [ ]:
```
train_loader_all = DataLoader(train_data,batch_size=len(train_data))
for e in train_loader_all:
    X_train = e["features"]
    gen_train = e["generator"]
    label_train = e["label"]
for e in test_loader:
    X_test = e["features"]
    gen_test = e["generator"]
    label_test = e["label"]

clf.fit(X_train, gen_train) # train on multi-class classification
```

In [ ]:
```
import numpy as np
pred = data.class_to_label(clf.predict(X_test))
np.mean(label_test.numpy() == pred.numpy()) # binary classification performance
```

Test on multi-class classification

Neural Network

In [ ]:
```
import torch.types

with torch.no_grad():
    model.eval()
    for e in test_loader:
        accuracy = model.get_model_accuracy_multiclass(features=e["features"],
                                                true_classes=e["generator"],
                                                device="cuda:"+str(device))
print(accuracy)
```

SVM

In [ ]:
```
clf.fit(X_train,gen_train).score(X_test,gen_test)
```

Saving the model

In [ ]:
```
path = f"./checkpoints/multiclass_train_AID_df_34000_V2.pt"
torch.save(model.state_dict(),path)
```

Loading the model

In [ ]:
```
model = MultiClassClassifier(n_classes=len(GEN_TO_INT.keys()))
model.load_state_dict(torch.load(path))
```

Out[ ]:
```
<All keys matched successfully>
```

In [ ]:
```
model.eval().cuda(device)
with torch.no_grad():
    for e in test_loader:
        pred = torch.argmax(model(e["features"].cuda(device)),dim=1)
        acc = torch.mean(torch.eq(e["generator"].cuda(device),pred).float()).item()
        print(acc)
```

```
0.9085294008255005
```

Train on data3 test on SB

In [ ]:
```
from utils import load_synthbuster_balanced
X_sb, y_sb = load_synthbuster_balanced(PATH_TO_DATA4 + "synthbuster_test",
                                       binary_classification=True,
                                       balance_real_fake=True)
```

In [ ]:
```
model.get_model_accuracy_binary(torch.Tensor(X_sb).cuda(device),
                                torch.Tensor(y_sb).cuda(device),
                                device)
```

Test on OOD

In [ ]:
```
ood = OOD("../../data/ood",load_preprocessed=False,device="cuda:" + str(device))
```

In [ ]:
```
# ood.save("../../data/ood.pt")
```

In [ ]:
```
ood2 = OOD(PATH_TO_DATA4 + "ood.pt",load_preprocessed=True)
```

In [ ]:
```
loader_test = DataLoader(ood2, batch_size=len(ood2), shuffle=True)
```

In [ ]:
```
for e in loader_test:
    model.eval()
    with torch.no_grad():
        accuracy = model.get_model_accuracy_binary(e["features"],e["label"],device)

print(accuracy)
```

Training a binary classifier

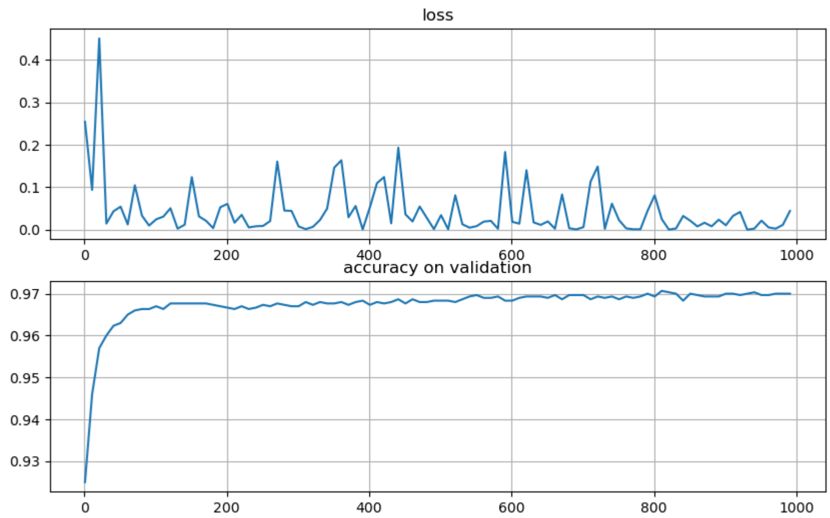
In [ ]:
```
device = "cuda:0"
clf_binary = MultiClassClassifier(n_classes=2).to(device)

lr = 1e-3
batch_size = 128
n_epochs = 1000

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(clf_binary.parameters(), lr=lr)

data = DeepFakeDatasetFastLoad(PATH_TO_DATA4 + "df_34000_V2.pt",
                               remove_blacklisted_gen=True)
clf_binary.set_generators_maps(gen_to_int=GEN_TO_INT,int_to_gen=INT_TO_GEN)


rng = torch.Generator().manual_seed(SEED)
train_data, test_data, validation_data = random_split(data,[0.7,0.2,0.1],generator=rng)

train_loader = DataLoader(train_data,batch_size=batch_size,shuffle=True)
test_loader  = DataLoader(test_data,batch_size=len(test_data),shuffle=True)
val_loader  = DataLoader(validation_data,batch_size=len(validation_data),shuffle=True)

clf_binary.train()
```

```
filtering: 100%|██████████| 12/12 [00:00<00:00, 69.99it/s]
```

Out[ ]:
```
MultiClassClassifier(
  (fc1): Linear(in_features=768, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=2, bias=True)
  (act): ReLU()
)
```

In [ ]:
```
val_accuracy = []
loss_history = []
for epoch in range(1,n_epochs+1):
    for idx, batch in enumerate(train_loader):
        # prediction and loss
        pred = clf_binary(batch["features"].to(device))
        loss = loss_fn(pred,batch["label"].type(torch.LongTensor).to(device))

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

    loss, current = loss.item(), idx*batch_size + len(batch["features"])
    if epoch%10 == 0 and epoch > 0:
        loss_history.append(loss)
        for v in val_loader:
            pred = torch.argmax(clf_binary(v["features"].to(device)),dim=1)
            val_accuracy.append(torch.mean(torch.eq(v["label"].to(device),pred).float()).item())
        print(f"loss: {loss:>7f}  [{epoch:>5d}/{n_epochs:>5d}]")
```

In [ ]:
```
fig, (ax1, ax2) = plt.subplots(2,1,figsize=(10,6))
ax1.set_title("loss")
ax2.set_title("accuracy on validation")
ax1.plot(range(1,n_epochs+1,10),loss_history)
ax2.plot(range(1,n_epochs+1,10),val_accuracy)
ax1.grid()
ax2.grid()
plt.show()
```



In [ ]:

```python
# path = f"./checkpoints/binary_epochs={n_epochs}_loss={loss}_filtered.pt"
torch.save(clf_binary.state_dict(),path)
```

Binary classifier Test VS MultiClass classifier

In [ ]:

```python
clf_binary.load_state_dict(torch.load("./checkpoints/binary_epochs=1000_loss=0.02289319783449173.pt"))
clf_binary.eval()
```

Out[ ]:

```
<All keys matched successfully>
```

In [ ]:

```python
model = MultiClassClassifier(n_classes=len(GEN_TO_INT.keys()))
model.eval()
model.load_state_dict(torch.load("./checkpoints/multiclass_epochs=1000_loss=0.0692238137125969.pt"))
model.set_generators_maps(GEN_TO_INT,INT_TO_GEN)
```

In [ ]:

```python
for e in test_loader:
    with torch.no_grad():
        print("Accuracy with binary classifier:     ",clf_binary.get_model_accuracy_binary(e["features"],e["label"],device,binary_model=True))
        print("Accuracy with multi-class classifier:",model.get_model_accuracy_binary(e["features"],e["label"],"cpu",binary_model=False))
```

```
Accuracy with binary classifier:      0.9754411578178406
Accuracy with multi-class classifier: 0.9752941131591797
```

Test on AID_TEST

In [ ]:

```python
data_test = DeepFakeTest(PATH_TO_DATA4 + "df_test_60000.pt",
                         load_from_disk=True,
                         remove_blacklisted_gen=True)

with torch.no_grad():

    print("Accuracy with binary classifier:     ",
          clf_binary.get_model_accuracy_binary(data_test.features,
                                              data_test.label,
                                              device,
                                              binary_model=True))
    model.set_generators_maps(gen_to_int=GEN_TO_INT,int_to_gen=INT_TO_GEN)
    print("Accuracy with multi-class classifier:",
          model.get_model_accuracy_binary(data_test.features.to(device),
                                         data_test.label.to(device),
                                         device,
                                         binary_model=False))
```

```
filtering: 100%|██████████| 3/3 [00:00<00:00, 2833.99it/s]
Accuracy with binary classifier:      0.7097500562667847
Accuracy with multi-class classifier: 0.7320000529289246
```

Accuracy per class

In [ ]:

```python
d_AID = {}
d_AID_TEST = {}

for e in test_loader:
    features = e["features"]
    label    = e["label"]
    gen      = e["generator"]

for key in GEN_TO_GEN:
    d_AID[key] = {
        "features": features[gen == GEN_TO_INT[key]],
        "label": label[gen == GEN_TO_INT[key]]
    }
    d_AID_TEST[key] = {
        "features": data_test.features[data_test.gen == GEN_TO_INT[key]],
        "label"   : data_test.label[data_test.gen == GEN_TO_INT[key]]}
```

In [ ]:

```python
torch.bincount(data.gen)
```

Out[ ]:

```
tensor([17000,  3000,  2000,  1000,  1000,  1000,  1000,  1000,  1000,  1000,
         1000,  1000,  2000,     0,     0,  1000])
```

In [ ]:

```python
torch.bincount(data_test.gen.int())
```

Out[ ]:

```
tensor([30000,  6000,  3000,  2000,  2000,  1000,  1000,  1000,  1000,  1000,
         1000,  3000,  4000,  1000,  1000,  2000], device='cuda:0')
```

```
In [ ]: accuracy_AID = {"binary":{},"multi":{}} # accuracy on the test data from the same folder as training data
accuracy_AID_TEST = {"binary":{},"multi":{}} # accuracy on data from AID_TEST (can be seen as OOD)
for g in d_AID:
    with torch.no_grad():
        accuracy_AID["binary"][g] = \
            model.get_model_accuracy_binary(d_AID[g]["features"],
                                            d_AID[g]["label"],
                                            "cpu",
                                            binary_model=False)

        accuracy_AID["multi"][g] = \
            model.get_model_accuracy_multiclass(
                d_AID[g]["features"],
                torch.ones_like(d_AID[g]["label"]) * GEN_TO_INT[g],
                "cpu")

        accuracy_AID_TEST["binary"][g] = \
            model.get_model_accuracy_binary(d_AID_TEST[g]["features"],
                                            d_AID_TEST[g]["label"],
                                            "cpu",
                                            binary_model=False)

        accuracy_AID_TEST["multi"][g] = \
            model.get_model_accuracy_multiclass(
                d_AID_TEST[g]["features"],
                torch.ones_like(d_AID_TEST[g]["label"]) * GEN_TO_INT[g],
                "cpu")

display("binary accuracy on test from AID:",accuracy_AID["binary"])
display("binary accuracy on AID_TEST:     ",accuracy_AID_TEST["binary"])
display("multi-class accuracy on test from AID:",accuracy_AID["multi"])
display("multi-class accuracy on AID_TEST:",accuracy_AID_TEST["multi"])
```

```
'binary accuracy on test from AID:'
{'null': 0.9822364449501038,
 'Stable diffusion': 0.9640522599220276,
 'Kandisky': 0.9897435903549194,
 'DF_XL': 0.9519230723381042,
 'dreamlike': 0.9402984976768494,
 'gigaGan': 0.8612716794013977,
 'GlideUP': 0.9902912378311157,
 'LafitteUP': 0.9952830076217651,
 'LCM_Dreamshaper': 0.994535505771637,
 'megaDallEUP': 0.9892473220825195,
 'miniDallEUP': 0.9948186278343201,
 'pixart': 0.9947368502616882,
 'playground': 0.9410377144813538,
 'styleGan2': nan,
 'styleGan3': nan,
 'animagineXL': 0.9893617033958435}
'binary accuracy on AID_TEST:     '
{'null': 0.562666654586792,
 'Stable diffusion': 0.9446666836738586,
 'Kandisky': 0.8793333172798157,
 'DF_XL': 0.8870000243186951,
 'dreamlike': 0.7275000214576721,
 'gigaGan': 0.8100000023841858,
 'GlideUP': 0.9940000176429749,
 'LafitteUP': 0.9980000257492065,
 'LCM_Dreamshaper': 0.9800000190734863,
 'megaDallEUP': 0.9909999966621399,
 'miniDallEUP': 0.9929999709129333,
 'pixart': 0.890666663646698,
 'playground': 0.8812500238418579,
 'styleGan2': 0.7480000257492065,
 'styleGan3': 0.6819999814033508,
 'animagineXL': 0.9884999990463257}
'multi-class accuracy on test from AID:'
{'null': 0.9822364449501038,
 'Stable diffusion': 0.8235294222831726,
 'Kandisky': 0.8487179279327393,
 'DF_XL': 0.870192289352417,
 'dreamlike': 0.6368159055709839,
 'gigaGan': 0.7398843765258789,
 'GlideUP': 0.9368932247161865,
 'LafitteUP': 0.9905660152435303,
 'LCM_Dreamshaper': 0.9234972596168518,
 'megaDallEUP': 0.8655914068222046,
 'miniDallEUP': 0.9430052042007446,
 'pixart': 0.699999988079071,
 'playground': 0.7429245114326477,
 'styleGan2': nan,
 'styleGan3': nan,
 'animagineXL': 0.9042553305625916}
'multi-class accuracy on AID_TEST:'
{'null': 0.562666654586792,
 'Stable diffusion': 0.6356666684150696,
 'Kandisky': 0.6086666584014893,
 'DF_XL': 0.07699999958276749,
 'dreamlike': 0.09099999815225601,
 'gigaGan': 0.2809999883174896,
 'GlideUP': 0.9430000185966492,
 'LafitteUP': 0.9950000047683716,
 'LCM_Dreamshaper': 0.8920000195503235,
 'megaDallEUP': 0.9100000262260437,
 'miniDallEUP': 0.9169999957084656,
 'pixart': 0.07599999755620956,
 'playground': 0.10724999755620956,
 'styleGan2': 0.0,
 'styleGan3': 0.0,
 'animagineXL': 0.4230000078678131}
```

Binary classifier

Performance of binary classifier on pair dataset

In [ ]:
```
device = "cuda:0"

model = MultiClassClassifier(n_classes=2).to(device=device)
model.eval()
model.load_state_dict(torch.load("./checkpoints/binary_epochs=1000_loss=0.04426886886358261_filtered.pt"))

lr = 1e-3
batch_size = 64
epochs = 5

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=lr)

data = RealFakePairs(load_from_disk=True,path="/data4/saland/data/real_fake_pairs_1090.pt",device="cuda:0")

with torch.no_grad():
    acc = model.get_model_accuracy_binary(features=data.features,
                                          true_labels=data.label,
                                          device=device,
                                          binary_model=True)
print(acc)
```

```
0.899082601070404
```

Train on 800 pairs and test on 200

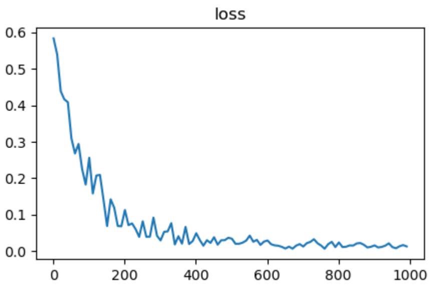
In [ ]:
```
device = "cuda:1"
model = MultiClassClassifier(n_classes=2,n_features=CLIP_FEATURE_DIM).to(device=device)
model_double = MultiClassClassifier(n_classes=2,n_features=CLIP_FEATURE_DIM*2).to(device=device)
model_diff = MultiClassClassifier(n_classes=2,n_features=CLIP_FEATURE_DIM).to(device=device)
model.train()
model_double.train()
model_diff.train()
# model.load_state_dict(torch.load("./checkpoints/binary_epochs=1000_loss=0.02289319783449173.pt"))

lr = 1e-3
batch_size = 64
n_epochs = 1000

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=lr)
optimizer2 = torch.optim.SGD(model_double.parameters(), lr=lr)
optimizer3 = torch.optim.SGD(model_diff.parameters(), lr=lr)

rng = torch.Generator().manual_seed(SEED)

data = DoubleCLIP(load_from_disk=True,path_to_datset="/data4/saland/data/Double_CLIP_2000.pt")
train_data, test_data = random_split(data,[0.2,0.8],generator=rng)

train_loader = DataLoader(train_data,batch_size=64,shuffle=True)
test_loader = DataLoader(test_data,batch_size=len(test_data))
```

In [ ]:
```
loss_history = []
for epoch in range(1,n_epochs+1):
    for idx, batch in enumerate(train_loader):
        # prediction and loss
        pred = model(batch["features"][:,:CLIP_FEATURE_DIM].to(device))
        loss = loss_fn(pred,batch["label"].type(torch.LongTensor).to(device))

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

    loss, current = loss.item(), idx*batch_size + len(batch["features"])
    if epoch%10 == 0 and epoch > 0:
        loss_history.append(loss)
        print(f"loss: {loss:>7f}  [{epoch:>5d}/{n_epochs:>5d}]")
```

```
loss: 0.583756  [   10/ 1000]
loss: 0.539579  [   20/ 1000]
loss: 0.438721  [   30/ 1000]
loss: 0.416519  [   40/ 1000]
loss: 0.408712  [   50/ 1000]
loss: 0.309370  [   60/ 1000]
loss: 0.267908  [   70/ 1000]
loss: 0.294130  [   80/ 1000]
loss: 0.225140  [   90/ 1000]
loss: 0.182545  [  100/ 1000]
loss: 0.256157  [  110/ 1000]
loss: 0.158156  [  120/ 1000]
loss: 0.207466  [  130/ 1000]
loss: 0.209256  [  140/ 1000]
loss: 0.140768  [  150/ 1000]
loss: 0.068818  [  160/ 1000]
loss: 0.142247  [  170/ 1000]
loss: 0.119114  [  180/ 1000]
loss: 0.068986  [  190/ 1000]
loss: 0.068263  [  200/ 1000]
loss: 0.112906  [  210/ 1000]
loss: 0.071339  [  220/ 1000]
loss: 0.076113  [  230/ 1000]
loss: 0.060360  [  240/ 1000]
loss: 0.039138  [  250/ 1000]
loss: 0.081971  [  260/ 1000]
loss: 0.039457  [  270/ 1000]
loss: 0.039661  [  280/ 1000]
loss: 0.092106  [  290/ 1000]
loss: 0.042148  [  300/ 1000]
loss: 0.029439  [  310/ 1000]
loss: 0.053061  [  320/ 1000]
loss: 0.054157  [  330/ 1000]
loss: 0.076776  [  340/ 1000]
loss: 0.018342  [  350/ 1000]
loss: 0.040988  [  360/ 1000]
loss: 0.020522  [  370/ 1000]
loss: 0.066546  [  380/ 1000]
loss: 0.019537  [  390/ 1000]
loss: 0.027696  [  400/ 1000]
loss: 0.049153  [  410/ 1000]
loss: 0.030431  [  420/ 1000]
loss: 0.014990  [  430/ 1000]
loss: 0.030045  [  440/ 1000]
loss: 0.022594  [  450/ 1000]
loss: 0.038213  [  460/ 1000]
loss: 0.017799  [  470/ 1000]
loss: 0.029900  [  480/ 1000]
loss: 0.030164  [  490/ 1000]
loss: 0.036720  [  500/ 1000]
loss: 0.034103  [  510/ 1000]
loss: 0.020227  [  520/ 1000]
loss: 0.020237  [  530/ 1000]
loss: 0.023274  [  540/ 1000]
loss: 0.028910  [  550/ 1000]
loss: 0.042629  [  560/ 1000]
loss: 0.025856  [  570/ 1000]
loss: 0.030912  [  580/ 1000]
loss: 0.017213  [  590/ 1000]
loss: 0.026246  [  600/ 1000]
loss: 0.029192  [  610/ 1000]
loss: 0.019026  [  620/ 1000]
loss: 0.015804  [  630/ 1000]
loss: 0.014834  [  640/ 1000]
loss: 0.011729  [  650/ 1000]
loss: 0.007162  [  660/ 1000]
loss: 0.012345  [  670/ 1000]
loss: 0.006837  [  680/ 1000]
loss: 0.015054  [  690/ 1000]
loss: 0.019335  [  700/ 1000]
loss: 0.012731  [  710/ 1000]
loss: 0.021826  [  720/ 1000]
loss: 0.025399  [  730/ 1000]
loss: 0.032711  [  740/ 1000]
loss: 0.021752  [  750/ 1000]
loss: 0.015446  [  760/ 1000]
loss: 0.006741  [  770/ 1000]
loss: 0.019202  [  780/ 1000]
loss: 0.025521  [  790/ 1000]
loss: 0.011189  [  800/ 1000]
loss: 0.023951  [  810/ 1000]
loss: 0.011295  [  820/ 1000]
loss: 0.012027  [  830/ 1000]
loss: 0.015748  [  840/ 1000]
loss: 0.015332  [  850/ 1000]
loss: 0.021128  [  860/ 1000]
loss: 0.022317  [  870/ 1000]
loss: 0.017479  [  880/ 1000]
loss: 0.010099  [  890/ 1000]
loss: 0.012004  [  900/ 1000]
loss: 0.015814  [  910/ 1000]
loss: 0.010053  [  920/ 1000]
loss: 0.011627  [  930/ 1000]
loss: 0.015007  [  940/ 1000]
loss: 0.021446  [  950/ 1000]
loss: 0.011118  [  960/ 1000]
loss: 0.007925  [  970/ 1000]
loss: 0.013390  [  980/ 1000]
loss: 0.016757  [  990/ 1000]
loss: 0.012903  [ 1000/ 1000]
```

In [ ]:

```
fig= plt.figure(figsize=(5,3))
plt.title("loss")
plt.plot(range(1,n_epochs+1,10),loss_history)
plt.show()
```

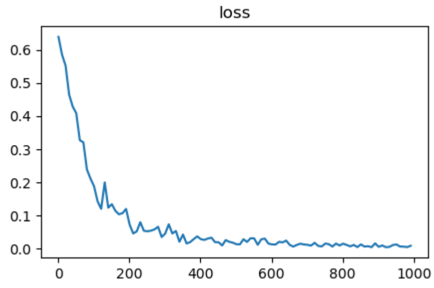


In [ ]:
```
model.eval()
with torch.no_grad():
    for e in test_loader:
        print("n_train:",len(train_data))
        print("n_test:",len(test_data))
        acc = model.get_model_accuracy_binary(e["features"][:,:CLIP_FEATURE_DIM],e["label"],device,True)

print(acc)
accuracy_768_features = acc
```

```
n_train: 400
n_test: 1600
0.9906249642372131
```

Train on concatenated version of CLIP features

In [ ]:
```
fig= plt.figure(figsize=(5,3))
plt.title("loss")
plt.plot(range(1,n_epochs+1,10),loss_history)
plt.show()
```



In [ ]:
```
model.eval()
with torch.no_grad():
    for e in test_loader:
        acc = model_double.get_model_accuracy_binary(e["features"],e["label"],device,True)

print("n_train:",len(train_data))
print("n_test:",len(test_data))
print("Accuracy 768 CLIP features:   ",accuracy_768_features)
print("Accuracy double CLIP features:",acc)
```

```
n_train: 400
n_test: 1600
Accuracy 768 CLIP features:    0.9906249642372131
Accuracy double CLIP features: 0.9899999499320984
```

Looking at the weights associated to each CLIP features (original and generated image)

In [ ]:
```
torch.sum(torch.abs(model_double.fc1.weight[:768])).item()
```

Out[ ]:
```
10058.380859375
```

In [ ]:
```
torch.sum(torch.abs(model_double.fc1.weight[:,768:])).item()
```

Out[ ]:
```
5020.86669921875
```

Train on the difference between the 2 CLIP vectors

In [ ]:
```
loss_history = []
for epoch in range(1,n_epochs+1):
    for idx, batch in enumerate(train_loader):
        # prediction and loss
        pred = model_diff((batch["features"][:,:CLIP_FEATURE_DIM] - batch["features"][:,CLIP_FEATURE_DIM:]).to(device))
        loss = loss_fn(pred,batch["label"].type(torch.LongTensor).to(device))

        # backpropagation
        loss.backward()
        optimizer3.step()
        optimizer3.zero_grad()

    loss, current = loss.item(), idx*batch_size + len(batch["features"])
    if epoch%10 == 0 and epoch > 0:
        loss_history.append(loss)
        print(f"loss: {loss:>7f}  [{epoch:>5d}/{n_epochs:>5d}]")
```

In [ ]:
```
model.eval()
with torch.no_grad():
    for e in test_loader:
        acc_diff = model_diff.get_model_accuracy_binary((e["features"][:,:CLIP_FEATURE_DIM] - e["features"][:,CLIP_FEATURE_DIM:]),e["label"],device,True)

print("n_train:",len(train_data))
print("n_test:",len(test_data))
print("Accuracy 768 CLIP features:       ",accuracy_768_features)
print("Accuracy double CLIP features:    ",acc)
print("Accuracy difference CLIP features:",acc_diff)
```

```
n_train: 400
n_test: 1600
Accuracy 768 CLIP features:        0.9906249642372131
Accuracy double CLIP features:     0.9899999499320984
Accuracy difference CLIP features: 0.9300000071525574
```

Test on OOD of model trained on AID and fine-tuned on real/fake pairs

In [ ]:
```
device = "cuda:1"
path_no_fine_tune = "checkpoints/binary_epochs=1000_loss=0.04426886886358261_filtered.pt"
model = MultiClassClassifier(n_classes=2).to(device)
model.load_state_dict(torch.load(path_no_fine_tune))
model.eval()
```

```
Loading pipeline components...: 100%|██████████| 7/7 [00:02<00:00,  2.95it/s]
```

Out[ ]:
```
MultiClassClassifier(
  (fc1): Linear(in_features=768, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=2, bias=True)
  (act): ReLU()
)
```

In [ ]:
```python
data_test = OOD("/data4/saland/data/ood.pt",
                load_preprocessed=True,
                device=device,
                remove_blacklisted_gen=True)

acc = model.get_model_accuracy_binary(data_test.features,data_test.label,device,binary_model=True)
print("accuracy of model on OOD before fine-tuning on real/fake pairs:",acc)
```

```
filtering: 100%|██████████| 5/5 [00:00<00:00, 2982.72it/s]
accuracy of model on OOD before fine-tuning on real/fake pairs: 0.8280000686645508
```

In [ ]:
```python
torch.bincount(data_test.gen.int())
```

Out[ ]:
```
tensor([100, 100, 100, 100, 100], device='cuda:1')
```

In [ ]:
```python
accuracy = {}
for gen in GEN_TO_INT_OOD:
    mask = data_test.gen == GEN_TO_INT_OOD[gen]
    accuracy[gen] = model.get_model_accuracy_binary(data_test.features[mask],data_test.label[mask],device=device, binary_model=True)
display(accuracy)
```

```
{'null': 0.85999995470047,
 'Lexica': 0.699999988079071,
 'Ideogram': 0.7899999618530273,
 'Leonardo': 0.8100000023841858,
 'Copilot': 0.9799999594688416,
 'img2img_SD1.5': nan,
 'Photoshop_generativemagnification': nan,
 'Photoshop_generativefill': nan}
```

Fine-tuning on real/fake pairs

In [ ]:
```python
lr = 1e-3
batch_size = 32
n_epochs = 1000
n_imgs = 100

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=lr)

model.train()
rng = torch.Generator().manual_seed(SEED)
data = RealFakePairs(load_from_disk=True,path="/data4/saland/data/real_fake_pairs_1090.pt")
data_shuffled = shuffle_data(data,in_place=False,seed=SEED)
data_slice = select_slice(data_shuffled,n_imgs)
train_loader = DataLoader(data_slice,batch_size=batch_size,shuffle=True,generator=rng)
```

```
shuffling features
shuffling label
```

In [ ]:
```python
loss_history = []
for epoch in range(1,n_epochs+1):
    for idx, batch in enumerate(train_loader):
        # prediction and loss
        pred = model((batch["features"]).to(device))
        loss = loss_fn(pred,batch["label"].type(torch.LongTensor).to(device))

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

    loss, current = loss.item(), idx*batch_size + len(batch["features"])
    if epoch%10 == 0 and epoch > 0:
        loss_history.append(loss)
        print(f"loss: {loss:>7f}  [{epoch:>5d}/{n_epochs:>5d}]")
```

```
loss: 0.648602  [   10/ 1000]
loss: 0.568120  [   20/ 1000]
loss: 0.567317  [   30/ 1000]
loss: 0.617613  [   40/ 1000]
loss: 0.420520  [   50/ 1000]
loss: 0.367580  [   60/ 1000]
loss: 0.376093  [   70/ 1000]
loss: 0.382430  [   80/ 1000]
loss: 0.233814  [   90/ 1000]
loss: 0.345064  [  100/ 1000]
loss: 0.265198  [  110/ 1000]
loss: 0.183815  [  120/ 1000]
loss: 0.158704  [  130/ 1000]
loss: 0.241281  [  140/ 1000]
loss: 0.223197  [  150/ 1000]
loss: 0.236249  [  160/ 1000]
loss: 0.166535  [  170/ 1000]
loss: 0.147699  [  180/ 1000]
loss: 0.159505  [  190/ 1000]
loss: 0.104365  [  200/ 1000]
loss: 0.119927  [  210/ 1000]
loss: 0.132870  [  220/ 1000]
loss: 0.085604  [  230/ 1000]
loss: 0.102017  [  240/ 1000]
loss: 0.054877  [  250/ 1000]
loss: 0.060638  [  260/ 1000]
loss: 0.076238  [  270/ 1000]
loss: 0.093681  [  280/ 1000]
loss: 0.071183  [  290/ 1000]
loss: 0.049570  [  300/ 1000]
loss: 0.077595  [  310/ 1000]
loss: 0.142943  [  320/ 1000]
loss: 0.031797  [  330/ 1000]
loss: 0.053163  [  340/ 1000]
loss: 0.062460  [  350/ 1000]
loss: 0.073654  [  360/ 1000]
loss: 0.030702  [  370/ 1000]
loss: 0.031895  [  380/ 1000]
loss: 0.048693  [  390/ 1000]
loss: 0.028523  [  400/ 1000]
loss: 0.055153  [  410/ 1000]
loss: 0.034302  [  420/ 1000]
loss: 0.033053  [  430/ 1000]
loss: 0.037271  [  440/ 1000]
loss: 0.036634  [  450/ 1000]
loss: 0.068536  [  460/ 1000]
loss: 0.038766  [  470/ 1000]
loss: 0.019831  [  480/ 1000]
loss: 0.055756  [  490/ 1000]
loss: 0.041419  [  500/ 1000]
loss: 0.032802  [  510/ 1000]
loss: 0.022648  [  520/ 1000]
loss: 0.015947  [  530/ 1000]
loss: 0.056358  [  540/ 1000]
loss: 0.051124  [  550/ 1000]
loss: 0.027787  [  560/ 1000]
loss: 0.024908  [  570/ 1000]
loss: 0.063908  [  580/ 1000]
loss: 0.015396  [  590/ 1000]
loss: 0.039129  [  600/ 1000]
loss: 0.011725  [  610/ 1000]
loss: 0.027839  [  620/ 1000]
loss: 0.046101  [  630/ 1000]
loss: 0.046641  [  640/ 1000]
loss: 0.052139  [  650/ 1000]
loss: 0.034220  [  660/ 1000]
loss: 0.027134  [  670/ 1000]
loss: 0.022907  [  680/ 1000]
loss: 0.010623  [  690/ 1000]
loss: 0.035484  [  700/ 1000]
loss: 0.013988  [  710/ 1000]
loss: 0.011439  [  720/ 1000]
loss: 0.008745  [  730/ 1000]
loss: 0.038446  [  740/ 1000]
loss: 0.011897  [  750/ 1000]
loss: 0.009231  [  760/ 1000]
loss: 0.026995  [  770/ 1000]
loss: 0.009816  [  780/ 1000]
loss: 0.017916  [  790/ 1000]
loss: 0.027710  [  800/ 1000]
loss: 0.007724  [  810/ 1000]
loss: 0.016254  [  820/ 1000]
loss: 0.015668  [  830/ 1000]
loss: 0.019503  [  840/ 1000]
loss: 0.005144  [  850/ 1000]
loss: 0.013432  [  860/ 1000]
loss: 0.014530  [  870/ 1000]
loss: 0.003654  [  880/ 1000]
loss: 0.039450  [  890/ 1000]
loss: 0.007473  [  900/ 1000]
loss: 0.012986  [  910/ 1000]
loss: 0.015741  [  920/ 1000]
loss: 0.023200  [  930/ 1000]
loss: 0.016599  [  940/ 1000]
loss: 0.006802  [  950/ 1000]
loss: 0.011170  [  960/ 1000]
loss: 0.006867  [  970/ 1000]
loss: 0.008278  [  980/ 1000]
loss: 0.018037  [  990/ 1000]
loss: 0.019782  [ 1000/ 1000]
```

In [ ]:
```
model.eval()
acc = model.get_model_accuracy_binary(data_test.features,data_test.label,device,True)
print("accuracy of the model on OOD after fine-tuning:",acc)
```

```
accuracy of the model on OOD after fine-tuning: 0.7520000338554382
```

In [ ]:
```
path_fine_tune = f"./checkpoints/binary_train_AID_fine_tune_pairs_loss={loss}.pt"
# torch.save(model.state_dict(),path_fine_tune)
```

In [ ]:
```
model_fine_tuned = MultiClassClassifier(n_classes=2).to(device)
model_fine_tuned.load_state_dict(torch.load("./checkpoints/binary_train_AID_fine_tune_pairs_loss=0.000141876342240721.pt"))
model_fine_tuned.eval()
acc = model_fine_tuned.get_model_accuracy_binary(data_test.features,data_test.label,device,True)
print("accuracy of the model on OOD after fine-tuning (from checkpoint):",acc)
```

```
accuracy of the model on OOD after fine-tuning (from checkpoint): 0.800000011920929
```

In [ ]:
```
model.get_model_accuracy_binary_per_gen(data_test.features,
                                         data_test.label,
                                         data_test.gen,
                                         device,
                                         True,
                                         GEN_TO_INT_OOD)
```

Out[ ]:
```
{'null': 0.9399999976158142,
 'Lexica': 0.6200000047683716,
 'Ideogram': 0.7599999904632568,
 'Leonardo': 0.75,
 'Copilot': 0.9699999690055847,
 'img2img_SD1.5': nan,
 'Photoshop_generativemagnification': nan,
 'Photoshop_generativefill': nan}
```

Test sur long caption avec et sans fine-tune

In [ ]:
```
path_fine_tune = "./checkpoints/binary_train_AID_fine_tune_pairs_loss=0.000141876342240721.pt"
model = MultiClassClassifier(n_classes=2).to(device)
model.load_state_dict(torch.load(path_no_fine_tune))

model_fine_tuned = MultiClassClassifier(n_classes=2).to(device)
model_fine_tuned.load_state_dict(torch.load(path_fine_tune))

model.eval()
model_fine_tuned.eval()

long_caption = LongCaption(path="/data4/saland/data/LongCaption.pt", load_from_disk=True)

print("Binary accuracy on long caption images:   ",
      model.get_model_accuracy_binary(long_caption.features,
                                      long_caption.label,
                                      device=device,
                                      binary_model=True))

print("Fine-tune accuracy on long caption images:",
      model_fine_tuned.get_model_accuracy_binary(long_caption.features,
                                                 long_caption.label,
                                                 device=device,
                                                 binary_model=True))
```

```
Binary accuracy on long caption images:    0.8497109413146973
Fine-tune accuracy on long caption images: 0.8323699235916138
```

binary train on 2k real and 2k fake

In [ ]:
```
device = "cuda:0"
model = MultiClassClassifier(n_classes=2).to(device)
model.train()

lr = 1e-3
batch_size = 32
n_epochs = 1000
n_imgs = 100

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=lr)

model.train()
rng = torch.Generator().manual_seed(SEED)
data = FlickrAndPairs(path="/data4/saland/data/flickr_and_pairs_DinoV2.pt",load_from_disk=True)
train_loader = DataLoader(data,batch_size=batch_size,shuffle=True,generator=rng)
```

In [ ]:
```
loss_history = []
for epoch in range(1,n_epochs+1):
    for idx, batch in enumerate(train_loader):
        # prediction and loss
        pred = model((batch["features"]).to(device))
        loss = loss_fn(pred,batch["label"].type(torch.LongTensor).to(device))

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

    loss, current = loss.item(), idx*batch_size + len(batch["features"])
    if epoch%10 == 0 and epoch > 0:
        loss_history.append(loss)
        print(f"loss: {loss:>7f}  [{epoch:>5d}/{n_epochs:>5d}]")
```

```
loss: 0.502425  [   10/ 1000]
loss: 0.271259  [   20/ 1000]
loss: 0.216291  [   30/ 1000]
loss: 0.204625  [   40/ 1000]
loss: 0.148112  [   50/ 1000]
loss: 0.041347  [   60/ 1000]
loss: 0.137004  [   70/ 1000]
loss: 0.064819  [   80/ 1000]
loss: 0.039667  [   90/ 1000]
loss: 0.022802  [  100/ 1000]
loss: 0.044861  [  110/ 1000]
loss: 0.023949  [  120/ 1000]
loss: 0.026538  [  130/ 1000]
loss: 0.027375  [  140/ 1000]
loss: 0.017287  [  150/ 1000]
loss: 0.023578  [  160/ 1000]
loss: 0.025720  [  170/ 1000]
loss: 0.018888  [  180/ 1000]
loss: 0.017412  [  190/ 1000]
loss: 0.021926  [  200/ 1000]
loss: 0.014409  [  210/ 1000]
loss: 0.011642  [  220/ 1000]
loss: 0.008274  [  230/ 1000]
loss: 0.013666  [  240/ 1000]
loss: 0.012596  [  250/ 1000]
loss: 0.008794  [  260/ 1000]
loss: 0.010618  [  270/ 1000]
loss: 0.005700  [  280/ 1000]
loss: 0.005616  [  290/ 1000]
loss: 0.007401  [  300/ 1000]
loss: 0.005720  [  310/ 1000]
loss: 0.008593  [  320/ 1000]
loss: 0.006102  [  330/ 1000]
loss: 0.006715  [  340/ 1000]
loss: 0.006789  [  350/ 1000]
loss: 0.003347  [  360/ 1000]
loss: 0.005897  [  370/ 1000]
loss: 0.006217  [  380/ 1000]
loss: 0.004787  [  390/ 1000]
loss: 0.002686  [  400/ 1000]
loss: 0.004223  [  410/ 1000]
loss: 0.003616  [  420/ 1000]
loss: 0.003971  [  430/ 1000]
loss: 0.005222  [  440/ 1000]
loss: 0.004513  [  450/ 1000]
loss: 0.004684  [  460/ 1000]
loss: 0.004796  [  470/ 1000]
loss: 0.001684  [  480/ 1000]
loss: 0.001545  [  490/ 1000]
loss: 0.002868  [  500/ 1000]
loss: 0.004363  [  510/ 1000]
loss: 0.003061  [  520/ 1000]
loss: 0.003216  [  530/ 1000]
loss: 0.003732  [  540/ 1000]
loss: 0.003420  [  550/ 1000]
loss: 0.003265  [  560/ 1000]
loss: 0.003799  [  570/ 1000]
loss: 0.002587  [  580/ 1000]
loss: 0.001751  [  590/ 1000]
loss: 0.002847  [  600/ 1000]
loss: 0.002927  [  610/ 1000]
loss: 0.002885  [  620/ 1000]
loss: 0.002998  [  630/ 1000]
loss: 0.001743  [  640/ 1000]
loss: 0.003599  [  650/ 1000]
loss: 0.002107  [  660/ 1000]
loss: 0.002721  [  670/ 1000]
loss: 0.002432  [  680/ 1000]
loss: 0.002017  [  690/ 1000]
loss: 0.002379  [  700/ 1000]
loss: 0.001127  [  710/ 1000]
loss: 0.001688  [  720/ 1000]
loss: 0.002390  [  730/ 1000]
loss: 0.002208  [  740/ 1000]
loss: 0.001799  [  750/ 1000]
loss: 0.001913  [  760/ 1000]
loss: 0.002023  [  770/ 1000]
loss: 0.002446  [  780/ 1000]
loss: 0.001294  [  790/ 1000]
loss: 0.001492  [  800/ 1000]
loss: 0.001349  [  810/ 1000]
loss: 0.001153  [  820/ 1000]
loss: 0.001820  [  830/ 1000]
loss: 0.001981  [  840/ 1000]
loss: 0.001939  [  850/ 1000]
loss: 0.001444  [  860/ 1000]
loss: 0.001696  [  870/ 1000]
loss: 0.001516  [  880/ 1000]
loss: 0.001166  [  890/ 1000]
loss: 0.001613  [  900/ 1000]
loss: 0.001376  [  910/ 1000]
loss: 0.001137  [  920/ 1000]
loss: 0.001084  [  930/ 1000]
loss: 0.001947  [  940/ 1000]
loss: 0.001247  [  950/ 1000]
loss: 0.001719  [  960/ 1000]
loss: 0.001191  [  970/ 1000]
loss: 0.001666  [  980/ 1000]
loss: 0.001745  [  990/ 1000]
loss: 0.001453  [ 1000/ 1000]
```

In [ ]:

```
torch.save(model.state_dict(),"./checkpoints/binary_train_real2k_fake1999_dinoV2.pt")
```

In [ ]:

```
model = MultiClassClassifier(n_classes=2).to(device)
model.load_state_dict(torch.load("checkpoints/binary_train_real2k_fake1999.pt"))
# data_test = DeepFakeTest(path_to_data="/data4/saland/data/df_test_60000.pt", load_from_disk=True)
data_test = OOD(path_to_data="/data4/saland/data/ood.pt", load_preprocessed=True, remove_blacklisted_gen=True)
model.eval()
model.get_model_accuracy_binary(data_test.features,
                                data_test.label,device,
                                binary_model=True)
```

```
filtering:   0%|          | 0/5 [00:00<?, ?it/s]filtering: 100%|██████████| 5/5 [00:00<00:00, 129.62it/s]
```

Out[ ]:

```
0.28200000524520874
```

In [ ]:

```
meta = TestMeta(path="/data4/saland/data/test_meta.pt",
                load_from_disk=True)
```

Fine tuning on Test Meta

In [ ]:
```
device = "cpu"
model = MultiClassClassifier(n_classes=2).to(device)
model.load_state_dict(torch.load("./checkpoints/binary_train_real2k_fake1999.pt"))
model.eval()

rng = torch.Generator().manual_seed(SEED)

data = TestMeta(path="/data4/saland/data/test_meta.pt",load_from_disk=True)
train_data, test_data = random_split(data, [0.8, 0.2],generator=rng)

train_loader = DataLoader(train_data, batch_size=64,shuffle=True,generator=rng)
test_loader  = DataLoader(test_data,batch_size=len(test_data))
```

```
Loading pipeline components...: 100%|██████████| 7/7 [00:00<00:00, 12.02it/s]
```

Accuracy on meta test before fine tuning

In [ ]:
```
generators = list(set(data.gen_original_name))
d = {gen : {"features":torch.empty((0,CLIP_FEATURE_DIM)),
            "label":torch.empty(0)} for gen in generators}

for e in test_loader:
    for i, features in enumerate(e["features"]):
        gen = e["gen_original_name"][i]
        label = e["label"][i]
        d[gen]["features"] = torch.cat((d[gen]["features"],features.unsqueeze(0)),dim=0)
        d[gen]["label"] = torch.cat((d[gen]["label"],label.unsqueeze(0)),dim=0)

acc = {}
for gen in d:
    acc[gen] = model.get_model_accuracy_binary(d[gen]["features"],
                                               d[gen]["label"],
                                               binary_model=True,
                                               device = device)

display(acc)
```

```
{'Source_2_animagineXL3': 0.800000011920929,
 'Source_29_SDXL': 0.5076923370361328,
 'Source_7_gigaGan': 0.34567901492118835,
 'Copilot_images': 0.9605262875556946,
 'Source_32_Dreamshaper': 0.921875,
 'Source_9_10_kandinsky': 0.7843137383460999,
 'LongCaptions': 0.8068181872367859,
 'Source_25_SD1_5': 0.3650793731212616,
 'Lexica_images': 0.9145299196243286,
 'Source_6_dreamlike': 0.6379310488700867,
 'MidJourneyV6': 0.8947368264198303,
 'Leonardo_images': 0.72826087474823,
 'Source_27_SD2_1': 0.4893617033958435,
 'Source_30_31_stylegan2_3': 0.12087912112474442,
 'Source_17_pixartAlpha': 0.8309859037399292,
 'null': 0.9373825788497925,
 'Source_21_playground': 0.859375,
 'Source_18_pixartSigma': 0.78125,
 'Source_4_DF-XL': 0.1358024626970291,
 'Ideogram_images': 0.8051947951316833}
```

fine-tuning

In [ ]:
```
lr = 1e-3
batch_size = 64
n_epochs = 200

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=lr)

model.train()
```

Out[ ]:
```
MultiClassClassifier(
  (fc1): Linear(in_features=768, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=2, bias=True)
  (act): ReLU()
)
```

In [ ]:
```
loss_history = []
for epoch in range(1,n_epochs+1):
    for idx, batch in enumerate(train_loader):
        # prediction and loss
        pred = model((batch["features"]).to(device))
        loss = loss_fn(pred,batch["label"].type(torch.LongTensor).to(device))

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

    loss, current = loss.item(), idx*batch_size + len(batch["features"])
    if epoch%10 == 0 and epoch > 0:
        loss_history.append(loss)
        print(f"loss: {loss:>7f}  [{epoch:>5d}/{n_epochs:>5d}]")
```

```
loss: 0.041811  [   10/  200]
loss: 0.222908  [   20/  200]
loss: 0.012555  [   30/  200]
loss: 0.120002  [   40/  200]
loss: 0.015896  [   50/  200]
loss: 0.020531  [   60/  200]
loss: 0.045851  [   70/  200]
loss: 0.045070  [   80/  200]
loss: 0.032931  [   90/  200]
loss: 0.005113  [  100/  200]
loss: 0.052948  [  110/  200]
loss: 0.527596  [  120/  200]
loss: 0.073503  [  130/  200]
loss: 0.001805  [  140/  200]
loss: 0.019284  [  150/  200]
loss: 0.090728  [  160/  200]
loss: 0.013563  [  170/  200]
loss: 0.025673  [  180/  200]
loss: 0.094159  [  190/  200]
loss: 0.106813  [  200/  200]
```

Accuracy on Test Meta after fine-tuning

```
In [ ]: d = {gen : {"features":torch.empty((0,CLIP_FEATURE_DIM)),
                    "label":torch.empty(0)} for gen in generators}

        for e in test_loader:
            for i, features in enumerate(e["features"]):
                gen = e["gen_original_name"][i]
                label = e["label"][i]
                d[gen]["features"] = torch.cat((d[gen]["features"],features.unsqueeze(0)),dim=0)
                d[gen]["label"] = torch.cat((d[gen]["label"],label.unsqueeze(0)),dim=0)

        acc = {}
        for gen in d:
            acc[gen] = model.get_model_accuracy_binary(d[gen]["features"],
                                                       d[gen]["label"],
                                                       binary_model=True,
                                                       device = device)

        display(acc)
```

```
{'Source_2_animagineXL3': 1.0,
 'Source_29_SDXL': 0.9384615421295166,
 'Source_7_gigaGan': 0.8888888955116272,
 'Copilot_images': 1.0,
 'Source_32_Dreamshaper': 0.96875,
 'Source_9_10_kandinsky': 1.0,
 'LongCaptions': 0.9431818127632141,
 'Source_25_SD1_5': 0.920634925365448,
 'Lexica_images': 0.9743589758872986,
 'Source_6_dreamlike': 0.982758641242981,
 'MidJourneyV6': 0.9736841917037964,
 'Leonardo_images': 0.8804348111152649,
 'Source_27_SD2_1': 0.957446813583374,
 'Source_30_31_stylegan2_3': 0.9890109896659851,
 'Source_17_pixartAlpha': 0.98591548204422,
 'null': 0.9768315553665161,
 'Source_21_playground': 0.953125,
 'Source_18_pixartSigma': 1.0,
 'Source_4_DF-XL': 0.9506173133850098,
 'Ideogram_images': 0.9740259647369385}
```

```
In [ ]: torch.save(model.state_dict(),"./checkpoints/binary_train_real_fake_2k_fine_tune_meta_test.pt")
```

test fine-tuned on OOD

```
In [ ]: model = MultiClassClassifier(n_classes=2)
        model.load_state_dict(torch.load("./checkpoints/binary_train_real2k_fake1999.pt"))
        model.eval()

        model_fine_tuned = MultiClassClassifier(n_classes=2)
        model_fine_tuned.load_state_dict(
            torch.load("./checkpoints/binary_train_real_fake_2k_fine_tune_meta_test.pt"))
        model_fine_tuned.eval()

        data_test_OOD = OOD("/data4/saland/data/ood.pt",
                        load_preprocessed=True,
                        device=device,
                        remove_blacklisted_gen=True)

        acc = model.get_model_accuracy_binary(data_test_OOD.features,
                                              data_test_OOD.label,
                                              device,
                                              binary_model=True)
        acc_fine_tuned = model_fine_tuned.get_model_accuracy_binary(data_test_OOD.features,
                                                                    data_test_OOD.label,
                                                                    device,
                                                                    binary_model=True)

        for e in test_loader:
            acc_test_meta = model.get_model_accuracy_binary(e["features"],
                                                            e["label"],
                                                            device,
                                                            binary_model=True)
            acc_test_meta_fine_tuned = model_fine_tuned.get_model_accuracy_binary(e["features"],
                                                                                  e["label"],
                                                                                  device,
                                                                                  True)

        print("accuracy on OOD before fine-tuning on meta test:",acc)
        print("accuracy on OOD after fine-tuning on meta test: ",acc_fine_tuned)
        print("accuracy on test meta before fine-tuning on test meta:",acc_test_meta)
        print("accuracy on test meta after fine-tuning on test meta: ",acc_test_meta_fine_tuned)
```

```
filtering: 100%|██████████| 5/5 [00:00<00:00, 4355.46it/s]
accuracy on OOD before fine-tuning on meta test: 0.8339999914169312
accuracy on OOD after fine-tuning on meta test:  0.906000018119812
accuracy on test meta before fine-tuning on test meta: 0.785647451877594
accuracy on test meta after fine-tuning on test meta:  0.9706708192825317
```

Train FlickrAndPairs CLIP + DINOV2 -> fine tune test meta CLIP + DINO_V2

```
In [ ]: fp_clip = FlickrAndPairs(path="/data4/saland/data/flickr_and_pairs.pt",load_from_disk=True)
        fp_dino = FlickrAndPairs(path="/data4/saland/data/flickr_and_pairs_DinoV2.pt",load_from_disk=True)
        train_data = SimpleDataset(torch.cat((fp_clip.features,fp_dino.features),dim=1),fp_clip.label)

        tm_clip = TestMeta(path="/data4/saland/data/test_meta.pt",load_from_disk=True)
        tm_dino = TestMeta(path="/data4/saland/data/test_meta_DinoV2.pt",load_from_disk=True)
        ft_data = SimpleDataset(torch.cat((tm_clip.features,tm_dino.features),dim=1),tm_clip.label)
```

In [ ]:
```
device = "cuda:0"
model = MultiClassClassifier(n_features=CLIP_FEATURE_DIM+DINO_FEATURE_DIM,n_classes=2).to(device)

lr = 1e-3
batch_size = 64
n_epochs = 1000

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=lr)
rng = torch.Generator().manual_seed(SEED)

model.train()

train_loader = DataLoader(train_data,batch_size=batch_size, generator=rng)

for epoch in range(1,n_epochs+1):
    for idx, batch in enumerate(train_loader):
        # prediction and loss
        pred = model((batch["features"]).to(device))
        loss = loss_fn(pred,batch["label"].type(torch.LongTensor).to(device))

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

    loss, current = loss.item(), idx*batch_size + len(batch["features"])
    if epoch%10 == 0 and epoch > 0:
        loss_history.append(loss)
        print(f"loss: {loss:>7f}  [{epoch:>5d}/{n_epochs:>5d}]")
```

```
loss: 0.278704  [   10/ 1000]
loss: 0.157806  [   20/ 1000]
loss: 0.106904  [   30/ 1000]
loss: 0.079469  [   40/ 1000]
loss: 0.061921  [   50/ 1000]
loss: 0.049769  [   60/ 1000]
loss: 0.041108  [   70/ 1000]
loss: 0.034674  [   80/ 1000]
loss: 0.029796  [   90/ 1000]
loss: 0.025970  [  100/ 1000]
loss: 0.022908  [  110/ 1000]
loss: 0.020427  [  120/ 1000]
loss: 0.018344  [  130/ 1000]
loss: 0.016594  [  140/ 1000]
loss: 0.015098  [  150/ 1000]
loss: 0.013800  [  160/ 1000]
loss: 0.012680  [  170/ 1000]
loss: 0.011692  [  180/ 1000]
loss: 0.010821  [  190/ 1000]
loss: 0.010048  [  200/ 1000]
loss: 0.009359  [  210/ 1000]
loss: 0.008739  [  220/ 1000]
loss: 0.008190  [  230/ 1000]
loss: 0.007688  [  240/ 1000]
loss: 0.007234  [  250/ 1000]
loss: 0.006822  [  260/ 1000]
loss: 0.006449  [  270/ 1000]
loss: 0.006107  [  280/ 1000]
loss: 0.005793  [  290/ 1000]
loss: 0.005508  [  300/ 1000]
loss: 0.005243  [  310/ 1000]
loss: 0.004999  [  320/ 1000]
loss: 0.004774  [  330/ 1000]
loss: 0.004564  [  340/ 1000]
loss: 0.004371  [  350/ 1000]
loss: 0.004191  [  360/ 1000]
loss: 0.004023  [  370/ 1000]
loss: 0.003866  [  380/ 1000]
loss: 0.003719  [  390/ 1000]
loss: 0.003582  [  400/ 1000]
loss: 0.003452  [  410/ 1000]
loss: 0.003330  [  420/ 1000]
loss: 0.003215  [  430/ 1000]
loss: 0.003107  [  440/ 1000]
loss: 0.003004  [  450/ 1000]
loss: 0.002908  [  460/ 1000]
loss: 0.002817  [  470/ 1000]
loss: 0.002730  [  480/ 1000]
loss: 0.002649  [  490/ 1000]
loss: 0.002571  [  500/ 1000]
loss: 0.002496  [  510/ 1000]
loss: 0.002426  [  520/ 1000]
loss: 0.002359  [  530/ 1000]
loss: 0.002296  [  540/ 1000]
loss: 0.002235  [  550/ 1000]
loss: 0.002177  [  560/ 1000]
loss: 0.002121  [  570/ 1000]
loss: 0.002068  [  580/ 1000]
loss: 0.002018  [  590/ 1000]
loss: 0.001969  [  600/ 1000]
loss: 0.001922  [  610/ 1000]
loss: 0.001878  [  620/ 1000]
loss: 0.001835  [  630/ 1000]
loss: 0.001794  [  640/ 1000]
loss: 0.001754  [  650/ 1000]
loss: 0.001716  [  660/ 1000]
loss: 0.001679  [  670/ 1000]
loss: 0.001644  [  680/ 1000]
loss: 0.001610  [  690/ 1000]
loss: 0.001577  [  700/ 1000]
loss: 0.001546  [  710/ 1000]
loss: 0.001515  [  720/ 1000]
loss: 0.001485  [  730/ 1000]
loss: 0.001457  [  740/ 1000]
loss: 0.001429  [  750/ 1000]
loss: 0.001402  [  760/ 1000]
loss: 0.001377  [  770/ 1000]
loss: 0.001352  [  780/ 1000]
loss: 0.001327  [  790/ 1000]
loss: 0.001304  [  800/ 1000]
loss: 0.001281  [  810/ 1000]
loss: 0.001259  [  820/ 1000]
loss: 0.001238  [  830/ 1000]
loss: 0.001217  [  840/ 1000]
loss: 0.001197  [  850/ 1000]
loss: 0.001177  [  860/ 1000]
loss: 0.001158  [  870/ 1000]
loss: 0.001140  [  880/ 1000]
loss: 0.001122  [  890/ 1000]
loss: 0.001105  [  900/ 1000]
loss: 0.001088  [  910/ 1000]
loss: 0.001071  [  920/ 1000]
loss: 0.001055  [  930/ 1000]
loss: 0.001039  [  940/ 1000]
loss: 0.001024  [  950/ 1000]
loss: 0.001009  [  960/ 1000]
loss: 0.000995  [  970/ 1000]
loss: 0.000981  [  980/ 1000]
loss: 0.000967  [  990/ 1000]
loss: 0.000954  [ 1000/ 1000]
```

In [ ]:

```python
torch.save(model.state_dict(),"../model/checkpoints/binary_train_real2k_fake1999_clip_dino.pt")
```

In [ ]:

```
device = "cuda:0"
model = MultiClassClassifier(n_features=CLIP_FEATURE_DIM+DINO_FEATURE_DIM,n_classes=2).to(device)
model.load_state_dict(torch.load("../model/checkpoints/binary_train_real2k_fake1999_clip_dino.pt"))

lr = 1e-3
batch_size = 64
n_epochs = 200

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=lr)
rng = torch.Generator().manual_seed(SEED)

model.train()

train_loader = DataLoader(ft_data,batch_size=batch_size, generator=rng)

for epoch in range(1,n_epochs+1):
    for idx, batch in enumerate(train_loader):
        # prediction and loss
        pred = model((batch["features"]).to(device))
        loss = loss_fn(pred,batch["label"].type(torch.LongTensor).to(device))

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

    loss, current = loss.item(), idx*batch_size + len(batch["features"])
    if epoch%10 == 0 and epoch > 0:
        loss_history.append(loss)
        print(f"loss: {loss:>7f}  [{epoch:>5d}/{n_epochs:>5d}]")
```

```
loss: 0.002120  [   10/  200]
loss: 0.003366  [   20/  200]
loss: 0.003721  [   30/  200]
loss: 0.003728  [   40/  200]
loss: 0.003650  [   50/  200]
loss: 0.003516  [   60/  200]
loss: 0.003355  [   70/  200]
loss: 0.003177  [   80/  200]
loss: 0.002988  [   90/  200]
loss: 0.002800  [  100/  200]
loss: 0.002614  [  110/  200]
loss: 0.002438  [  120/  200]
loss: 0.002271  [  130/  200]
loss: 0.002119  [  140/  200]
loss: 0.001982  [  150/  200]
loss: 0.001857  [  160/  200]
loss: 0.001742  [  170/  200]
loss: 0.001639  [  180/  200]
loss: 0.001544  [  190/  200]
loss: 0.001458  [  200/  200]
```

In [ ]:

```
torch.save(model.state_dict(),"../model/checkpoints/binary_train_real_fake_2k_fine_tune_meta_test_clip_dino.pt")
```