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

In [ ]:
```
import open_clip
import torch
from sklearn.svm import LinearSVC
import numpy as np
from datasets import load_from_disk
from tqdm import tqdm
import sys
sys.path.append("../tools")
from utils import load_data_split
from sklearn.model_selection import cross_val_score
import warnings
import pandas as pd
```

In [ ]:
```
model, _, preprocess = open_clip.create_model_and_transforms('hf-hub:laion/CLIP-ViT-L-14-DataComp.XL-s13B-b90K',device="cuda")

model.eval()
```

Out[ ]:
```
CLIP(
  (visual): VisionTransformer(
    (conv1): Conv2d(3, 1024, kernel_size=(14, 14), stride=(14, 14), bias=False)
    (patch_dropout): Identity()
    (ln_pre): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
    (transformer): Transformer(
      (resblocks): ModuleList(
        (0-23): 24 x ResidualAttentionBlock(
          (ln_1): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
          (attn): MultiheadAttention(
            (out_proj): NonDynamicallyQuantizableLinear(in_features=1024, out_features=1024, bias=True)
          )
          (ls_1): Identity()
          (ln_2): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
          (mlp): Sequential(
            (c_fc): Linear(in_features=1024, out_features=4096, bias=True)
            (gelu): GELU(approximate='none')
            (c_proj): Linear(in_features=4096, out_features=1024, bias=True)
          )
          (ls_2): Identity()
        )
      )
    )
    (ln_post): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
  )
  (transformer): Transformer(
    (resblocks): ModuleList(
      (0-11): 12 x ResidualAttentionBlock(
        (ln_1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=768, out_features=768, bias=True)
        )
        (ls_1): Identity()
        (ln_2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (mlp): Sequential(
          (c_fc): Linear(in_features=768, out_features=3072, bias=True)
          (gelu): GELU(approximate='none')
          (c_proj): Linear(in_features=3072, out_features=768, bias=True)
        )
        (ls_2): Identity()
      )
    )
  )
  (token_embedding): Embedding(49408, 768)
  (ln_final): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
)
```

Loading the local version of ELSA

In [ ]:
```
dataset_path = "../../data/big/"
```

In [ ]:
```
ds_train = load_from_disk(dataset_path+"train")
```

SVM Binary Classification

Feature extraction with CLIP

In [ ]:
```
def preprocess_img(X):
    with torch.no_grad():
        for i, img in enumerate(X):
            X[i] = model.encode_image(preprocess(img).unsqueeze(0).cuda()).detach().cpu().numpy()
        return X
```

In [ ]:
```
X_train, y_train = preprocess_img(ds_train["image"]), ds_train["label"]
```

```
/home/lsaland/micromamba/envs/clip/lib/python3.11/site-packages/PIL/Image.py:1000: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(
```

In [ ]:
```
X_train = np.array([x.flatten() for x in X_train])
```

In [ ]:
```
clf = LinearSVC()

clf.fit(X_train,y_train)
```

```
/home/lsaland/micromamba/envs/clip/lib/python3.11/site-packages/sklearn/svm/_classes.py:31: FutureWarning: The default value of `dual` will change from `True` to ``'auto'`` in 1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
/home/lsaland/micromamba/envs/clip/lib/python3.11/site-packages/sklearn/svm/_base.py:1237: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

Out[ ]:
```
LinearSVC()
```

In [ ]:
```
ds_test = load_from_disk(dataset_path+"test")
```

In [ ]:
```
X_test, y_test = preprocess_img(ds_test["image"]), ds_test["label"]
```

```
/home/lsaland/micromamba/envs/clip/lib/python3.11/site-packages/PIL/Image.py:1000: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(
```

In [ ]:
```
X_test = np.array([x.flatten() for x in X_test])
```

In [ ]:
```
mean_accuracy_test = clf.score(X_test,y_test)
print("Mean acccuracy on test set: ",mean_accuracy_test)
```

```
Mean acccuracy on test set:  0.9819
```

In [ ]:
```
X = np.vstack((X_train,X_test))
y = np.hstack((y_train,y_test))
```

In [ ]:
```
warnings.filterwarnings("ignore", category=Warning)

cv = cross_val_score(clf,X,y,cv=10)
print("Cross validation score: ",np.mean(cv))
```

```
Cross validation score:  0.9846
```

SVM on JPEG QF 40 data with training on any img format (training set is not JPEG only)

In [ ]:
```
X_train, y_train = load_data_split("../../data/big",
                                   split="train",
                                   model=model,
                                   preprocess=preprocess,
                                   device="cuda")
```

In [ ]:
```
X_test, y_test = load_data_split("../../data/big_QF_40",
                                 split="test",
                                 model=model,
                                 preprocess=preprocess,device="cuda")
```

In [ ]:
```
clf = LinearSVC()
clf.fit(X_train,y_train)
clf.score(X_test,y_test)
```

Out[ ]:
```
0.7381
```

SVM on JPEG QF 40 data with training on JPEG QF 40

In [ ]:
```
X_train, y_train = load_data_split("../../data/big_QF_40",
                                   split="train",
                                   model=model,
                                   preprocess=preprocess,
                                   device="cuda")
```

In [ ]:
```
clf.fit(X_train,y_train)
clf.score(X_test,y_test)
```

Out[ ]:
```
0.9811
```

SVM trained on JPEG QF 40 and tested on ELSA

In [ ]:
```
X_test, y_test = load_data_split("../../data/big",
                                 split="test",
                                 model=model,
                                 preprocess=preprocess,
                                 device="cuda")
```

```
/home/lsaland/micromamba/envs/clip/lib/python3.11/site-packages/PIL/Image.py:1000: UserWarning: Palette images with Transparency expressed in bytes should be conv
erted to RGBA images
  warnings.warn(
```

In [ ]:
```
clf.score(X_test,y_test)
```

Out[ ]:
```
0.9515
```

SVM performance on different training size (train jpeg 40, test ELSA 10 000)

Train size: 100

In [ ]:
```
X_train_100, y_train_100 = load_data_split("../../data/small_QF_40",
                                           split="train",
                                           model=model,
                                           preprocess=preprocess,
                                           device="cuda")
```

In [ ]:
```
warnings.filterwarnings("ignore",category=FutureWarning)

clf = LinearSVC()
print("mean accuracy for train set of size 100: ",clf.fit(X_train_100,y_train_100).score(X_test,y_test))
```

```
mean accuracy for train set of size 100:  0.8805
```

Train size: 1000

In [ ]:
```
X_train_1000, y_train_1000 = load_data_split("../../data/medium_QF_40",
                                             split="train",
                                             model=model,
                                             preprocess=preprocess,
                                             device="cuda")
```

In [ ]:
```
from sklearn.exceptions import ConvergenceWarning
warnings.filterwarnings("ignore",category=ConvergenceWarning)

clf = LinearSVC()
print("mean accuracy for train set of size 1000: ",clf.fit(X_train_1000,y_train_1000).score(X_test,y_test))
```

```
mean accuracy for train set of size 1000:  0.9391
```

Train size: 10 000

In [ ]:
```
X_train_10000, y_train_10000 = load_data_split("../../data/big_QF_40",
                                               split="train",
                                               model=model,
                                               preprocess=preprocess,
                                               device="cuda")
```

In [ ]:
```
from sklearn.exceptions import ConvergenceWarning
warnings.filterwarnings("ignore",category=ConvergenceWarning)

clf = LinearSVC()
print("mean accuracy for train set of size 10 000: ",clf.fit(X_train_10000,y_train_10000).score(X_test,y_test))
```

```
mean accuracy for train set of size 10 000:  0.9513
```

Training and testing for various quality factors

In [ ]:
```
results = np.zeros((3,3))
k = 0

load_dataset_q = lambda q, split : load_data_split("../../data/big_QF_" + str(q),
                                                   split=split,
                                                   model=model,
                                                   preprocess=preprocess,
                                                   device="cuda",
                                                   show_progress_bar=True)

train_data = [load_dataset_q(40,"train"),
              load_dataset_q(65,"train"),
              load_dataset_q(90,"train")]

test_data  = [load_dataset_q(40,"test"),
              load_dataset_q(65,"test"),
              load_dataset_q(90,"test")]
```

```
100%|██████████| 10000/10000 [02:58<00:00, 56.12it/s]
100%|██████████| 10000/10000 [02:58<00:00, 55.96it/s]
100%|██████████| 10000/10000 [02:58<00:00, 56.01it/s]
100%|██████████| 10000/10000 [02:58<00:00, 56.12it/s]
100%|██████████| 10000/10000 [02:58<00:00, 56.12it/s]
100%|██████████| 10000/10000 [02:58<00:00, 56.05it/s]
```

In [ ]:
```
warnings.filterwarnings("ignore",category=Warning)
clf = LinearSVC()
k = 0
with tqdm(total=9) as bar:
    for i in range(3):
        X_train, y_train = train_data[i]
        for j in range(3):
            X_test, y_test = test_data[j]
            results[i,j] = clf.fit(X_train,y_train).score(X_test,y_test)

            k += 1
            bar.n = k
            bar.refresh()
```

```
  0%|          | 0/9 [00:00<?, ?it/s]100%|██████████| 9/9 [00:06<00:00,  1.47it/s]
```

In [ ]:
```
results
```

Out[ ]:
```
array([[0.9813, 0.973 , 0.9797],
       [0.945 , 0.9825, 0.983 ],
       [0.7744, 0.8581, 0.9925]])
```

In [ ]:
```
qf = (40,65,90)
d = {"q_train" : [], "q_test": [], "accuracy": []}

for i in range(3):
    for j in range(3):
        d["q_train"].append(qf[i])
        d["q_test"].append(qf[j])
        d["accuracy"].append(results[i,j])
```

In [ ]:
```
pd.DataFrame.from_records(results).to_csv("../../docs/CLIP_jpg.csv")
```

Out[ ]:

	0	1	2
0	0.9813	0.9730	0.9797
1	0.9450	0.9825	0.9830
2	0.7744	0.8581	0.9925