In [1]:

```
import pickle, os, json, skimage, ast, torch
from tqdm.notebook import tqdm
import IPython.display
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors

import numpy as np
import pandas as pd

os.environ["CUDA_VISIBLE_DEVICES"]="4"

pd.set_option('mode.chained_assignment', None)
%config InlineBackend.figure_format = 'retina'
%matplotlib inline

from clip_setup import *
from features_grouping import *
```

```
Model parameters: 151,277,313
Input resolution: 224
Context length: 77
Vocab size: 49408
/home/tul02009/.local/lib/python3.7/site-packages/torchvision/transforms/trans
forms.py:258: UserWarning: Argument interpolation should be of type Interpolat
ionMode instead of int. Please, use InterpolationMode enum.
  "Argument interpolation should be of type InterpolationMode instead of int.
"
```

In [2]:

```
EVAL_IDS, EMBEDDINGS = pickle.load(open( "evalids_embeddings.p", "rb" ) )
USE = torch.Tensor(np.load("cover_text_embed.npy")).cuda()
IMAGE_FEATURES = torch.tensor(np.load('../signed_card/clip/image_features.npy
```

In [3]:

```
config = {
    'img_folder': '../data/card_images',
    'ann_folder': '../signed_card/Data/Annotated Data', #where annotations.cs
    'duplicate_path': '../signed_card/near_duplicates/duplicates_and_empty.cs
    'image_feature_path': '../signed_card/clip/image_features.npy',
}

df = pd.read_csv(config["ann_folder"] + "/annotations.csv")
split = pd.read_csv("train_test_split.csv")
df = df.merge(split, on="card_id")

dup_df = pd.read_csv(config["duplicate_path"])
df = df.merge(dup_df, on="card_id")
del split, dup_df

COLS = ["holidays","special_occasions", "relationships", "messages"]
for COL in COLS:
    df[COL] = df[COL].map(ast.literal_eval)
    cl, ct = np.unique(df[COL].map(len), return_counts=True)
    print(f"{COL} with {cl[1]} labels, #samples={ct[1]}")

    df[COL] = df[COL].apply(lambda x: [i for i in x if i not in ["NONE", ""]]
    #df[COL] = df[COL].str[0] #use the first label only

# --------- Features labels preprocess --------- #
for s1, s2 in RM_FEATURES:
    df["features"] = df.features.str.replace(s1, s2)
df["features"] = df["features"].map(ast.literal_eval)
```

```
# remove label "None"
df["features"] = df.features.apply(lambda x: [i for i in x if i not in ["NONE

# create object-only features
df["obj_features"] = df.features.copy()
df["obj_features"] = df.obj_features.apply(lambda x: [i for i in x if i not i

# find unique labels
labels = np.concatenate(df['obj_features'])
labels, cts = np.unique(labels, return_counts=True)
print("Number of labels: ", len(labels))

# find unique labels which have count > thr
thr = 3
labels = np.unique(labels[np.where(cts>thr)])
print(f"Keep only those labels, which occur more than {thr} times: ", len(lab

df["obj_features"] =  df.obj_features.apply(lambda x: [i for i in x if i in l
```

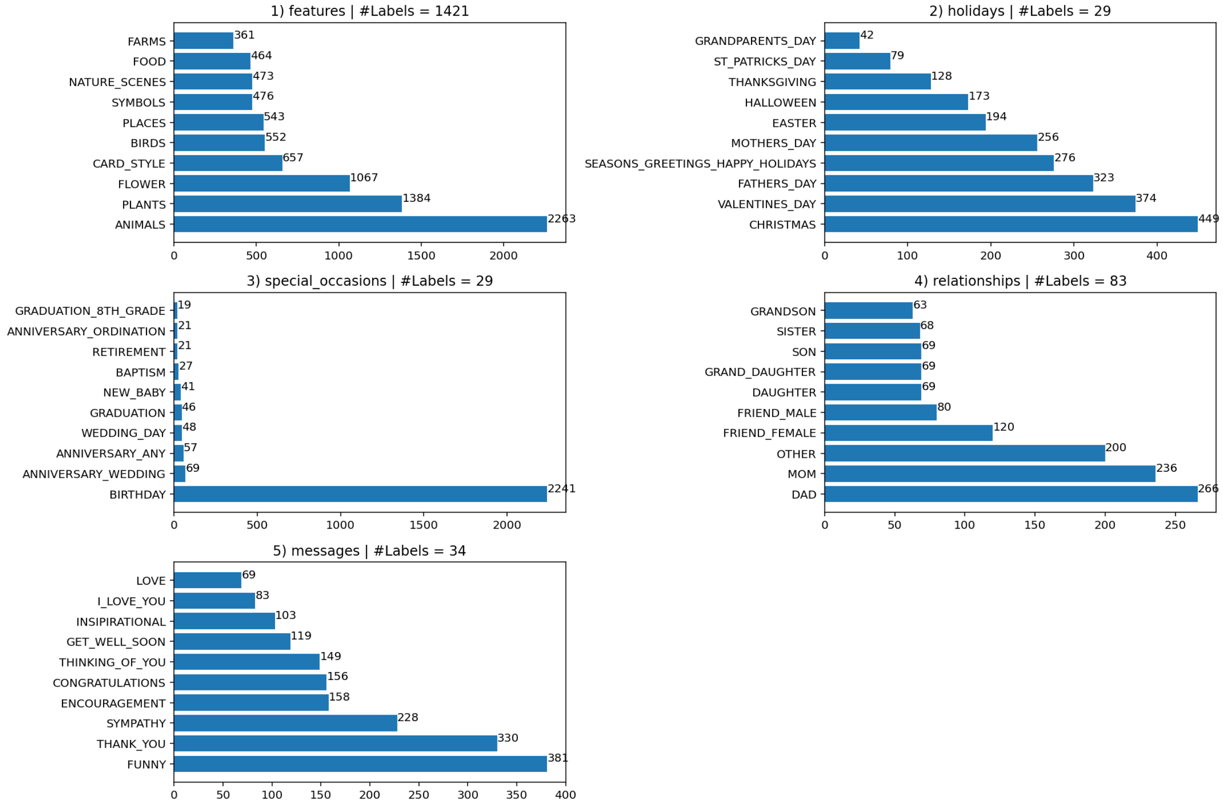
```
holidays with 2 labels, #samples=12
special_occasions with 2 labels, #samples=5
relationships with 2 labels, #samples=182
messages with 2 labels, #samples=77
Number of labels:  1334
Keep only those labels, which occur more than 3 times:  500
```

In [4]:

```
from cuml.manifold import TSNE
from sklearn.decomposition import PCA
colors = list(mcolors.TABLEAU_COLORS.keys())
```

In [49]:

```
plt.figure(figsize=(15, 10))
thr = 10
for i, COL in enumerate(["features", 'holidays', 'special_occasions', 'relati
    val, ct = np.unique(np.concatenate(df[COL]),return_counts=True)
    plt.subplot(3,2, i+1)
    plt.title(f"{i+1}) {COL} | #Labels = {len(val)}")
    idx = np.argsort(ct)[::-1]
    val = val[idx][:thr]
    ct = ct[idx][:thr]
    plt.barh(val,ct)
    for i in range(len(val)):
        plt.annotate(ct[i], xy=(ct[i],val[i]), ha='left', va='bottom')
plt.tight_layout()
plt.savefig("./figs/data.png", bbox_inches='tight')
```



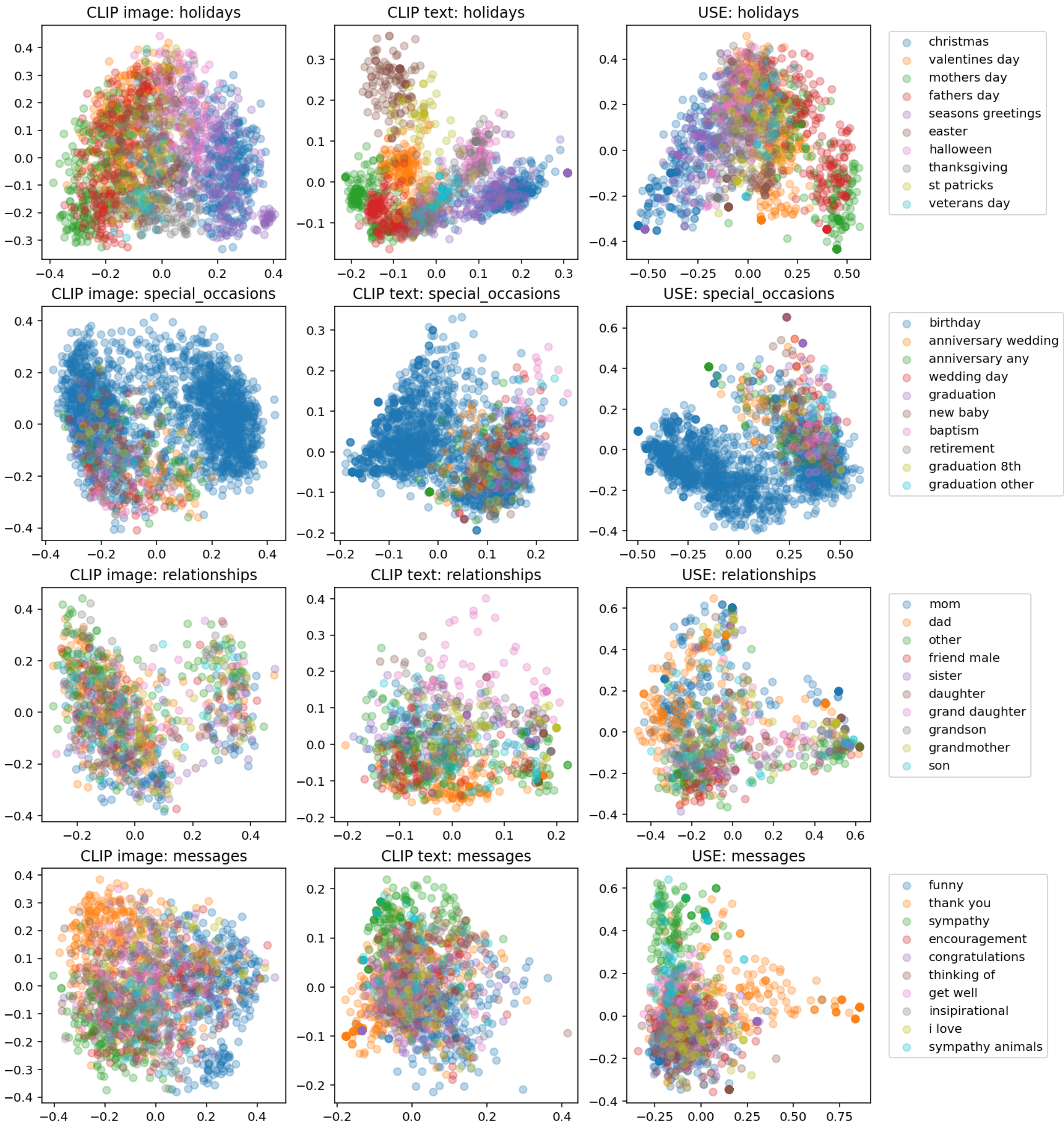
1. PCA

1.1 PCA on all features.

In [23]:

```
plt.figure(figsize=(12, 16))
print("Principal Component Analysis on features")
i = 0
col = 3
for COL in ['holidays', 'special_occasions', 'relationships', 'messages']:
    eval_ids = np.array(EVAL_IDS[COL])
    eval_ids = eval_ids[np.isin(eval_ids,EVAL_IDS["cover_text"])]
    gt = np.array(df.loc[eval_ids, COL].reset_index(drop=True).str[0])  #use
    vals, ct = np.unique(gt, return_counts=True)
    vals = vals[np.argsort(ct)][::-1][:len(colors)]

    for des, feat in [["CLIP image", IMAGE_FEATURES], ["CLIP text", EMBEDDING
        plt.subplot(4,col, i+1)
        pca = PCA(n_components=2)
        X = feat[eval_ids].cpu().numpy()
        principalComponents = pca.fit_transform(X)

        plt.title(des + ": " + COL)
        # temp = principalComponents[~np.isin(gt, viz[:,0])]
        # plt.scatter(temp[:,0], temp[:,1], label="other", c="gray", alpha=0.
        for val, c in zip(vals,colors[:len(vals)]):
            temp = principalComponents[gt==val]
            val = " ".join( val.split("_")[:2]).lower()
            plt.scatter(temp[:,0], temp[:,1], label=val, c=c, alpha=0.3)
        if (i+1)%col ==0:
            _ = plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
        i += 1

plt.savefig("./figs/pca.png", bbox_inches='tight')
```

```
Principal Component Analysis on features
```



Only 'holidays' column has apparent clusters. For the rest, they all seem to be mixed up.

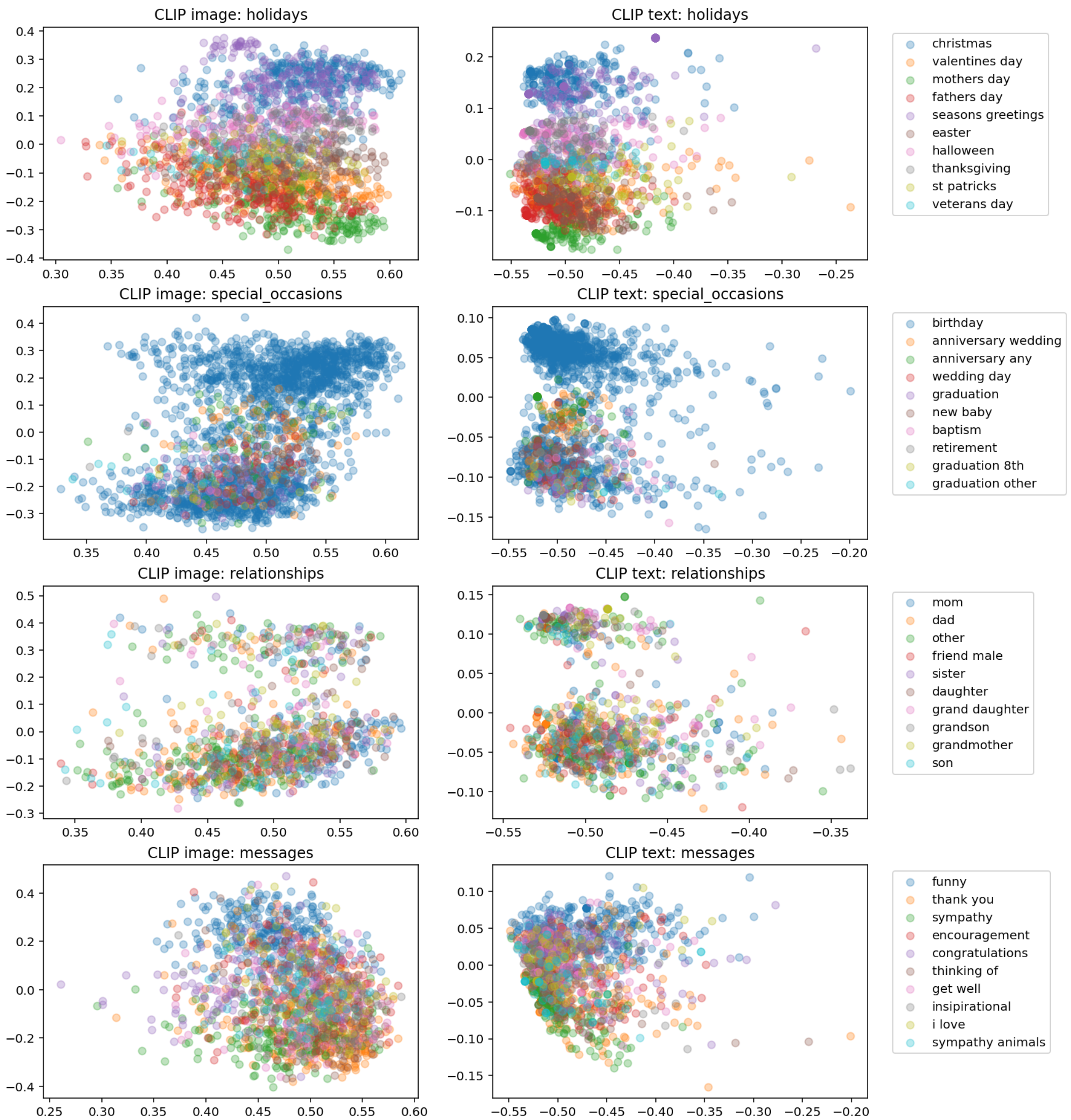
1.2 PCA on CLIP image and text features together

Performs PCA on both image and text together. Visualize separately. Both text & features' components can share the same dimension and same eigenvector.

In [24]:

```
plt.figure(figsize=(12, 16))
i = 0
col = 2
for COL in ['holidays', 'special_occasions', 'relationships', 'messages']:
    eval_ids = np.array(EVAL_IDS[COL])
    eval_ids = eval_ids[np.isin(eval_ids,EVAL_IDS["cover_text"])]
    gt = np.array(df.loc[eval_ids, COL].reset_index(drop=True).str[0])  #use

    vals, ct = np.unique(gt, return_counts=True)
    vals = vals[np.argsort(ct)][::-1][:len(colors)]

    pca = PCA(n_components=2)
    X = torch.cat((IMAGE_FEATURES[eval_ids], EMBEDDINGS["cover_text"][eval_id
    principalComponents = pca.fit_transform(X)
```

```
    for j, des in enumerate(["CLIP image", "CLIP text"]):
        plt.subplot(4,col, i+1)
        plt.title(des + ": " + COL)
        for val, c in zip(vals,colors[:len(vals)]):
            temp = principalComponents[j*len(eval_ids): (j+1)*len(eval_ids)]
            temp = temp[gt==val]
            val = " ".join( val.split("_")[:2]).lower()
            plt.scatter(temp[:,0], temp[:,1], label=val, c=c, alpha=0.3)
#             if j == 0:
#                 x0, x1 = plt.xlim()
#                 y0, y1 = plt.ylim()
#             else:
#                 plt.xlim((x0,x1))
#                 plt.ylim((y0,y1))
        if (i+1)%col ==0:
            _ = plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
        i+=1
plt.savefig("./figs/pca2.png", bbox_inches='tight')
```

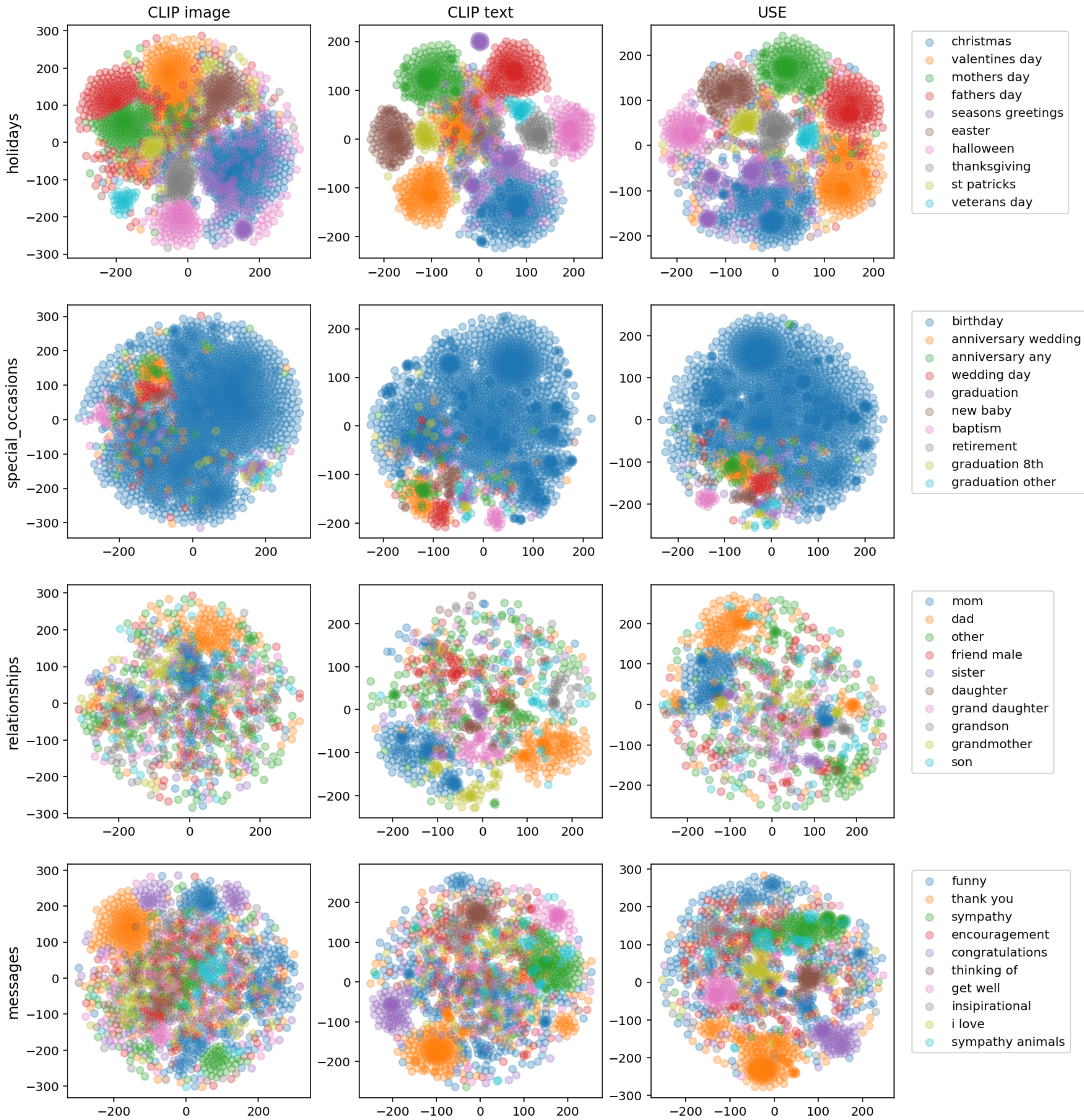


2. TSNE

2.1 Plot dots

https://medium.com/rapids-ai/tsne-with-gpus-hours-to-seconds-9d9c17c941db
TSNE is a nonlinear dimensionality reduction algorithm, whilst Principal Component Analysis is linear. This mean's PCA's components often have some meaning, while TSNE's are no longer ordered by importance, or at all interpretable outside of the neighborhoods they create.

In [16]:
```
colors = list(mcolors.TABLEAU_COLORS.keys())
col = 3
fig, axes = plt.subplots(4, col, figsize=(12, 16))

for row, COL in enumerate(['holidays', 'special_occasions', 'relationships',
    eval_ids = np.array(EVAL_IDS[COL])
    eval_ids = eval_ids[np.isin(eval_ids,EVAL_IDS["cover_text"])]
    gt = np.array(df.loc[eval_ids, COL].reset_index(drop=True).str[0])  #use
    vals, ct = np.unique(gt, return_counts=True)
    vals = vals[np.argsort(ct)][::-1][:len(colors)]
    axes[row, 0].set_ylabel(COL, size='large')
    for col, (des, feat) in enumerate([["CLIP image", IMAGE_FEATURES], ["CLIP
        axes[0, col].set_title(des)

        tsne = TSNE(n_components = 2)
        X = feat[eval_ids].cpu().numpy()
        principalComponents = tsne.fit_transform(X)

        for val, c in zip(vals,colors[:len(vals)]):
            temp = principalComponents[gt==val]
            val = " ".join( val.split("_")[:2]).lower()
            axes[row, col].scatter(temp[:,0], temp[:,1], label=val, c=c, alph

        if col ==2:
            _ = axes[row, col].legend(bbox_to_anchor=(1.05, 1), loc='upper le

plt.savefig("./figs/tsne.png", bbox_inches='tight')
```



Similar to the findings from PCA, 'holidays' column has apparent clusters. 'messages' column also has some separated clusters.

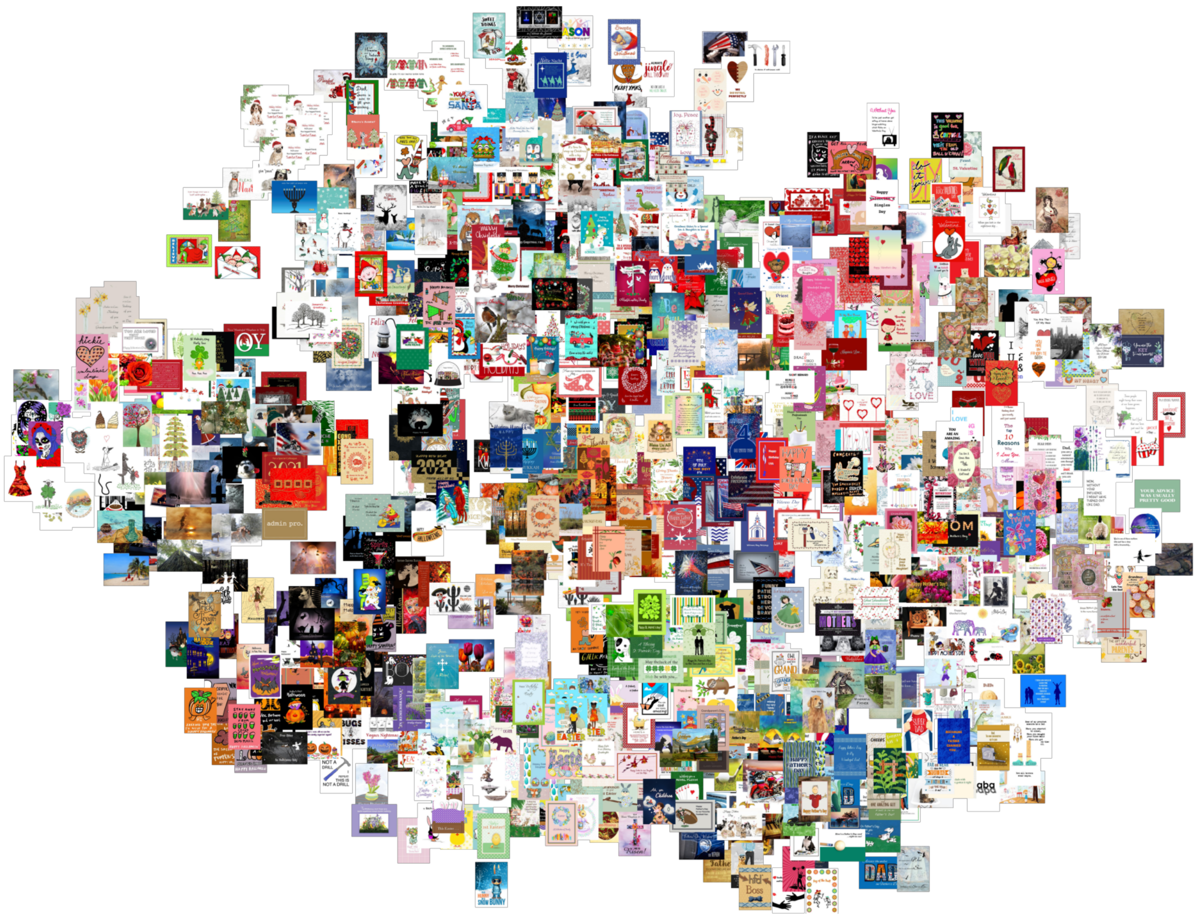
2.2 Plot actual images

2.2.1 TSNE with CLIP image +USE features for holidays cards

In [26]:

```
width = 4000
height = 3000
max_dim = 150

full_image = Image.new('RGBA', (width, height))

COL = "holidays"
eval_ids = np.array(EVAL_IDS[COL])
eval_ids = eval_ids[np.isin(eval_ids,EVAL_IDS[COL])]

tsne = TSNE(n_components = 2)
X = (IMAGE_FEATURES[eval_ids] + USE[eval_ids]).cpu().numpy()
principalComponents = tsne.fit_transform(X)

tx, ty = principalComponents[:,0], principalComponents[:,1]
```

```
tx = (tx-np.min(tx)) / (np.max(tx) - np.min(tx))
ty = (ty-np.min(ty)) / (np.max(ty) - np.min(ty))

i = 0
for image_idx, x, y in zip(eval_ids, tx, ty):
    tile = Image.open(os.path.join(config['img_folder'], f'{df["card_id"][ima
    rs = max(1, tile.width/max_dim, tile.height/max_dim)
    tile = tile.resize((int(tile.width/rs), int(tile.height/rs)), Image.ANTIA
    full_image.paste(tile, (int((width-max_dim)*x), int((height-max_dim)*y)),
    i+=1
    if i == 1000:
        break
plt.figure(figsize = (16,12))
plt.axis('off')
plt.imshow(full_image)
plt.savefig(f"./figs/tsne_imfeat_{COL}.png", bbox_inches='tight')
```



2.2.2 TSNE with CLIP text features for holidays cards

In [23]:

```
full_image = Image.new('RGBA', (width, height))

tsne = TSNE(n_components = 2)
X = EMBEDDINGS["cover_text"][eval_ids].cpu().numpy()
principalComponents = tsne.fit_transform(X)

tx, ty = principalComponents[:,0], principalComponents[:,1]
tx = (tx-np.min(tx)) / (np.max(tx) - np.min(tx))
ty = (ty-np.min(ty)) / (np.max(ty) - np.min(ty))

i = 0
for image_idx, x, y in zip(eval_ids, tx, ty):
    tile = Image.open(os.path.join(config['img_folder'], f'{df["card_id"][ima
    rs = max(1, tile.width/max_dim, tile.height/max_dim)
    tile = tile.resize((int(tile.width/rs), int(tile.height/rs)), Image.ANTIA
```

```
    full_image.paste(tile, (int((width-max_dim)*x), int((height-max_dim)*y)),
    i+=1
    if i == 1000:
        break

plt.figure(figsize = (16,12))
plt.axis('off')
plt.imshow(full_image)
plt.savefig(f"./figs/tsne_textfeat_{COL}.png", bbox_inches='tight')
```