In [1]:
```
import os, json, skimage, ast, torch
os.environ["CUDA_VISIBLE_DEVICES"]="4"
from tqdm.notebook import tqdm
from collections import OrderedDict

import numpy as np
import pandas as pd
# %matplotlib widget
import matplotlib.pyplot as plt

from torchvision.transforms import Compose, Resize, CenterCrop, ToTensor, Nor
import IPython.display

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

# my customized CLIP setup code
from clip_setup import *
from features_grouping import *
```

```
Model parameters: 151,277,313
Input resolution: 224
Context length: 77
Vocab size: 49408
/home/tul02009/miniconda3/lib/python3.8/site-packages/torchvision/transforms/t
ransforms.py:257: UserWarning: Argument interpolation should be of type Interp
olationMode instead of int. Please, use InterpolationMode enum.
  warnings.warn(
```

1. Data preprocessing

1.1 Preprocess target columns

In [2]:
```
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
<ipython-input-2-b74cc9dbefdb>:27: FutureWarning: The default value of regex w
ill change from True to False in a future version. In addition, single charact
er regular expressions will*not* be treated as literal strings when regex=Tru
e.
  df["features"] = df.features.str.replace(s1, s2)
Number of labels:  1334
Keep only those labels, which occur more than 3 times:  500
```

1.2 Preprocess cover text and inside text

- Since CLIP has limit on the sentence length, crop texts if needed
- convert none to ""
- lower case
- add sentence beginning "This is an image of" helps.

In [3]:

```
def preprocess_text(series, max_len=350):
    series = series.str.lower().replace("[\'none\']", "").replace('[^A-Za-z\s
    series[series.str.len()>0] = series[series.str.len()>0].apply(lambda x: "

    n_words = 15
    while True:
        series[series.str.len()>max_len] = series[series.str.len()>max_len].a
        if len(series[series.str.len()>max_len]) == 0:
            break
        else:
            n_words -= 1
    return series

# --------- cover/inside text -----------
df["inside_text"] = preprocess_text( df["inside_text"])
df["cover_text"] = preprocess_text(df["cover_text"])
```

In [4]:

```
COLS = ["holidays", "special_occasions", "relationships", "messages", "obj_fe
EVAL_IDS = {}
for COL in COLS:
```

```
    not_none_ids = list(df[(df[COL].str.len()>0) & (df["keep"])].index)
    EVAL_IDS[COL] = not_none_ids
    temp = np.sum(df.loc[not_none_ids, "train_test"] == "test")
    print(f"{COL:<20} | not_none/duplicate samples={len(not_none_ids):,} | te
```

```
holidays             | not_none/duplicate samples=2,427 | test_samples=418
special_occasions    | not_none/duplicate samples=2,735 | test_samples=521
relationships        | not_none/duplicate samples=1,847 | test_samples=334
messages             | not_none/duplicate samples=2,196 | test_samples=404
obj_features         | not_none/duplicate samples=6,699 | test_samples=1,203
cover_text           | not_none/duplicate samples=6,668 | test_samples=1,217
inside_text          | not_none/duplicate samples=5,058 | test_samples=913
```

In [5]:

```
evals = []
COLS = ["holidays", "special_occasions", "relationships", "messages", "obj_fe
for COL in COLS:
    non_none_ids = EVAL_IDS[COL]
    row = {"Category": COL, "N":len(non_none_ids), "# Unique Labels": len(np.
    evals.append(row)
pd.DataFrame(evals)
```

Out[5]:

	Category	N	# Unique Labels
0	holidays	2427	29
1	special_occasions	2735	29
2	relationships	1847	83
3	messages	2196	34
4	obj_features	6699	500

In [6]:

```
USE = torch.Tensor(np.load("cover_text_embed.npy")).cuda()
IMAGE_FEATURES = torch.tensor(np.load('../signed_card/clip/image_features.npy
```

2. Obtain embeddings

2.1 target categories

Get a set of text embeddings for each target columns

In [7]:

```
COLS = ["holidays", "special_occasions", "relationships", "messages", "obj_fe
EMBEDDINGS = {}
LABELS = {}
SENTENCES = {}
obj_groups = [i for i in OBJECTS.keys() if i != "OTHERS"]

for COL in tqdm(COLS):
    # Get unique labels
    labels = np.unique(np.concatenate(list(df.loc[EVAL_IDS[COL], COL])))
    labels = [l for l in labels if l not in ["NONE", "OTHER", ""]]

    if COL == "obj_features":
        # not including group labels, eg: animal
        labels = [l for l in labels if l not in OBJECTS.keys()]
        # there is only 1 tool, an special artist. thus create label for them
        labels += ["TOOLS", "ARTISTS"]

    # Create a sentence for each labels
```

```
    sentences = []
    for t in labels:
        group = None
        if COL == "obj_features":
            for g in obj_groups:
                if t in OBJECTS[g]:
                    group = g
                    break

        elif COL not in ["special_occasions", "relationships"]:
            group = COL

        t = t.lower().replace("_", " ").replace("/", " or ")
        if group is not None:
            group = group.lower().replace("_", " ").replace("/", " or ")
            sentences.append(f"This is an image of {t}, a type of {group}")
        else:
            sentences.append(f"This is an image of {t}")


    sentences = np.array(sentences)
    labels = np.array(labels)

    # Tokenize all texts
    tokenizer = SimpleTokenizer()
    text_tokens = [tokenizer.encode(desc) for desc in sentences]

    sot_token = tokenizer.encoder['<|startoftext|>']
    eot_token = tokenizer.encoder['<|endoftext|>']

    text_input = torch.zeros(len(text_tokens), model.context_length, dtype=to
    for i, tokens in enumerate(text_tokens):
        tokens = [sot_token] + tokens + [eot_token]
        text_input[i, :len(tokens)] = torch.tensor(tokens)

    # text_input = text_input.cuda()

    # Generate embeddings for each sentence
    with torch.no_grad():
        text_features = model.encode_text(text_input.cuda()).float()
        text_features /= text_features.norm(dim=-1, keepdim=True)

        EMBEDDINGS[COL] = text_features
        LABELS[COL] = labels
        SENTENCES[COL] = sentences
```

2.2 cover text

In [8]:

```
EMBEDDINGS["cover_text"] = torch.zeros_like(IMAGE_FEATURES)

batch_size = 200
n_batch = (len(EVAL_IDS["cover_text"])//batch_size)+1

for batch in tqdm(range(n_batch)):
    ids = np.arange(batch*batch_size, min((batch+1)*batch_size, len(EVAL_IDS[
    ids = np.array(EVAL_IDS["cover_text"])[ids]

    sentences = list(df.loc[ids,"cover_text"])
    tokenizer = SimpleTokenizer()
    text_tokens = [tokenizer.encode(sentence) for sentence in sentences]

    sot_token = tokenizer.encoder['<|startoftext|>']
```

```
    eot_token = tokenizer.encoder['<|endoftext|>']

    text_input = torch.zeros(len(text_tokens), model.context_length, dtype=to
    for i, tokens in enumerate(text_tokens):
        tokens = [sot_token] + tokens + [eot_token]
        text_input[i, :len(tokens)] = torch.tensor(tokens)

    # Generate embeddings for each sentence
    with torch.no_grad():
        ctext_features = model.encode_text(text_input.cuda()).float()
        ctext_features /= ctext_features.norm(dim=-1, keepdim=True)
        EMBEDDINGS["cover_text"][ids] = ctext_features
```

2.3 Encode image features

In [9]:

```
#np.save("CLIP_preds_for_object_features.npy", LABELS["obj_features"][PREDS["
```

3 Classification

In [10]:

```
batch_size = 200
n_batch = (len(df)//batch_size)+1
evals = []
print("Overall Accuracy (ignoring samples have no target label and no cover t
for des in ["image features", "text features", "image & text features",]: #"t
    PREDS_both = {}
    PROBS_both = {}
    for COL in COLS:
        PREDS_both[COL] = []
        PROBS_both[COL] = []

    for batch in tqdm(range(n_batch)):
        ids = np.arange(batch*batch_size, min((batch+1)*batch_size, len(df)))
        with torch.no_grad():
            agg = "NA"
            if "image" in des:
                image_features = IMAGE_FEATURES[ids]
            if "text" in des:
                ctext_features = EMBEDDINGS["cover_text"][ids]

            if "image" not in des:
                image_features = ctext_features
            if "&" in des:
                agg = "Average"
                ctext_ids = torch.all(ctext_features != 0, axis=-1)
                image_features[ctext_ids] = (0.5 * image_features + 0.5 * cte

            for COL in COLS:
                label_features = EMBEDDINGS[COL]
                similarity = label_features.cpu().numpy() @ image_features.cp
                text_probs = (100.0 * image_features @ label_features.T).softm
                top_probs, top_labels = text_probs.cpu().topk(5, dim=-1)
                PREDS_both[COL].append(top_labels)
                PROBS_both[COL].append(top_probs)

    for COL in COLS:
        PREDS_both[COL] = np.concatenate(PREDS_both[COL], axis=0)
        PROBS_both[COL] = np.concatenate(PROBS_both[COL], axis=0)

    row = {"Input Features": des, "Aggreagator": agg}
```

```
    for COL in ["holidays", "special_occasions",  'relationships', 'messages'
        pred_ids = PREDS_both[COL]
        preds = LABELS[COL][pred_ids]
        acc1, acc5 = 0, 0

        eval_ids = EVAL_IDS[COL]
        eval_ids = [i for i in eval_ids if i in EVAL_IDS["cover_text"]]

        for i in eval_ids:
            gt =  df[COL][i]

            # if the top-1 prediction is one of the gt label
            if preds[i][0] in gt:
                acc1 +=1

            # if any of the top-5 prediction is one of the gt label
            for p in preds[i]:
                if p in gt:
                    acc5 +=1
                    break
        row[f"{COL}"] = acc1/len(eval_ids)
        row[f"{COL}_T5"] = acc5/len(eval_ids)
    evals.append(row)

pd.DataFrame(evals).round(3)
# i = ["holidays", "special_occasions",  'relationships', 'messages']
# temp = ["Input Features","Aggreagator"] + i + [f"{j}_T5" for j in i ]
# evals[temp].round(3)
```

```
Overall Accuracy (ignoring samples have no target label and no cover text)
```

Out[10]:

	Input Features	Aggreagator	holidays	holidays_T5	special_occasions	special_occasions_T5	rel
0	image features	NA	0.632	0.950	0.531	0.877	
1	text features	NA	0.776	0.891	0.595	0.942	
2	image & text features	Average	0.811	0.969	0.657	0.964	

4. Object Dection

4. 1 Post-process obj_features and Evaluate

In [11]:

```
G = {
    "BIRDS":["EASTER_CHICKS",],
    "ANIMALS": ['CAT', 'DOGS', 'BIRDS', "EASTER_CHICKS","EASTER_BUNNY"],
    "PEOPLE": ['ARITISTS','FAMOUS_PEOPLE',"ANIMATION_CHARACTERS", "KINGS", "Q
    "PLANTS": ['FLOWERS', "FRUITS", "GARDENING", "WREATHS", "GARDENS"],
    "FOOD": ["DRINKS", "FRUITS"],
    "FLOWER": ["FLORAL_PRINT"],
    "VINEYARD": ["WINE"],
    "SKELETON": ["SKULLS"],
    "STUFFED_ANIMALS": ["EASTER_CHICKS", "EASTER_BUNNY", "TEDDY_BEAR"],
    "HOUSES": ['HAUNTED_HOUSES', 'BIRDHOUSES', 'LIGHTHOUSES'],
```

```
    "PLANES":["JETS"],
}

obj_groups = [i for i in OBJECTS.keys() if i != "OTHERS"]

def extend_preds(old):
    # if predicted objects under a category, then the category is added to pr
    old = np.array(old)
    new = np.copy(old)

    while True:
        for p in old:
            # if p is in one of the OBJECTS group
            if p not in OBJECTS["OTHERS"] + obj_groups:
                for g in obj_groups:
                    if p in OBJECTS[g]:
                        new = np.append(new, g)
                        break

            # if p is in one of the G group
            for g in G.keys():
                if p in G[g]:
                    new = np.append(new, g)
                    break

        new = np.unique(new)
        if len(new) == len(old):
            break
        else:
            old = np.copy(new)

    return new
```

In [12]:

```
batch_size = 200
n_batch = (len(df)//batch_size)+1

PREDS = {}
PROBS = {}
for COL in COLS:
    PREDS[COL] = []
    PROBS[COL] = []

for batch in tqdm(range(n_batch)):
    ids = np.arange(batch*batch_size, min((batch+1)*batch_size, len(df)))

    with torch.no_grad():
        image_features = IMAGE_FEATURES[ids]

        for COL in COLS:
            text_features = EMBEDDINGS[COL]
            similarity = text_features.cpu().numpy() @ image_features.cpu().n

            text_probs = (100.0 * image_features @ text_features.T).softmax(d
            top_probs, top_labels = text_probs.cpu().topk(5, dim=-1)
            PREDS[COL].append(top_labels)
            PROBS[COL].append(top_probs)

for COL in COLS:
    PREDS[COL] = np.concatenate(PREDS[COL], axis=0)
    PROBS[COL] = np.concatenate(PROBS[COL], axis=0)
```

In [13]:

```
COL = "obj_features"
labels, labels_cts = np.unique(np.concatenate(df[COL]), return_counts=True)
correct_labels_cts = np.zeros(len(labels_cts))

pred_ids = PREDS[COL]
preds = LABELS[COL][pred_ids]
acc5_ = 0
avg_label_acc, avg_label_acc_ = [], []

for i in tqdm(EVAL_IDS[COL]):
    gt =  df[COL][i]
    pred_labels = extend_preds(preds[i])

    label_acc, label_acc_ = 0,0
    for p in pred_labels:
        if p in gt:
            if p in preds[i]:
                label_acc +=1
            label_acc_ +=1
            correct_labels_cts[np.where(labels == p)[0][0]] += 1

    avg_label_acc.append(min(label_acc, 5)/min(len(gt), 5))
    avg_label_acc_.append(min(label_acc_, 5)/min(len(gt), 5))

    # if any of predictions is one of the gt label
    for p in pred_labels:
        if p in gt:
            acc5_ +=1
            break

print(f"# samples = {len(EVAL_IDS[COL])}")
print(f"Top-5 accuracy:  before post-processing={acc5/len(EVAL_IDS[COL]):.3f}
print(f"Avg. label acc. : for each sample, calculate the average accuracy. th
print(f"Avg. label acc: before post-processing={np.mean(avg_label_acc):.3f}, (
```

```
# samples = 6699
Top-5 accuracy:  before post-processing=0.184, after post-processing=0.811
Avg. label acc. : for each sample, calculate the average accuracy. then, avera
ge for all samples.
Avg. label acc: before post-processing=0.341, after post-processing=0.615
```

In [14]:

```
COL = "obj_features"
print("labels with 0 correct. (ct is the number of label occurence in the who
print("ct    | label                | sentence")
idxs = np.where(correct_labels_cts==0)[0]
for i in idxs:
    l = labels[i]
    if l in LABELS[COL]:
        print(f"{labels_cts[i]: <5}| {labels[i]:<20}|",
              SENTENCES[COL][np.where(LABELS[COL] == l)[0][0]])
    else:
        print("**", labels_cts[i], labels[i])
```

```
labels with 0 correct. (ct is the number of label occurence in the whole datas
et)
ct    | label                | sentence
9     | ARTISTS              | This is an image of artists
4     | BEARDS               | This is an image of beards
5     | BRUNETTES            | This is an image of brunettes
4     | BUBBLES              | This is an image of bubbles
4     | CUPID                | This is an image of cupid, a type of mythic creat
ures
11    | DOLLS                | This is an image of dolls, a type of toys
7     | JEWELRY              | This is an image of jewelry
```

```
6     | KINGS                | This is an image of kings
4     | MEDICINE             | This is an image of medicine
6     | PAISLEY              | This is an image of paisley, a type of textiles
7     | PARTIES_HATS         | This is an image of parties hats
5     | PINE_CONES           | This is an image of pine cones
4     | RIBBONS              | This is an image of ribbons
5     | SKATEBOARDS          | This is an image of skateboards
6     | TOILET_PAPER         | This is an image of toilet paper
4     | TOMATOES             | This is an image of tomatoes, a type of vegetable
s
5     | WATER                | This is an image of water
4     | WEREWOLVES           | This is an image of werewolves, a type of mythic
creatures
4     | WILDLIFE             | This is an image of wildlife
4     | WORMS                | This is an image of worms, a type of animals
```

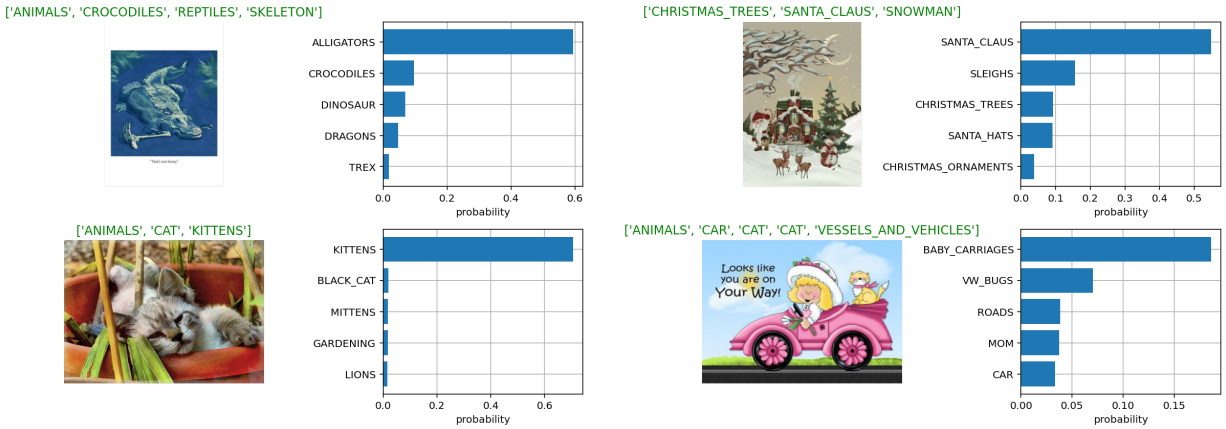
In [15]:
```
# [i for i in LABELS[COL] if "EASTER" in i]
```

4.2 Visualize

In [16]:
```
COL = "obj_features"
N = 4
ids = np.random.choice(EVAL_IDS[COL],N)
# ids = list(df[df[COL].apply(",".join) != ""].index)
#ids = list(df[df[COL].apply(",".join).str.contains("B")].index)
# ids = list(df[df["special_occasions"].apply(",".join).str.contains("BIRTHDA
# ids = np.random.choice(ids, min(10, len(ids)))
preprocess2 = Compose([
    Resize(input_resolution, interpolation=Image.BICUBIC),
    ToTensor()
])
images = []
for i in ids:
    image = Image.open(os.path.join(config['img_folder'], f'{df["card_id"][i]
    image = preprocess2(image.convert("RGB"))
    images.append(image)

targets = np.array(df[COL][ids])
preds = LABELS[COL][PREDS[COL][ids]]
probs = PROBS[COL][ids]

plt.figure(figsize=(16, int(N*1.5)))
for i, image in enumerate(images):
    plt.subplot(N//2, 4, 2 * i + 1)
    plt.imshow(image.permute(1, 2, 0))

    c = "red"
    for t in targets[i]:
        if t in preds[i]:
            c = "green"
            break
    plt.title(targets[i], c=c)

    plt.axis("off")

    plt.subplot(N//2, 4, 2 * i + 2)
    y = np.arange(probs[i].shape[-1])
    plt.grid()
    plt.barh(y, probs[i])
    plt.gca().invert_yaxis()
    plt.gca().set_axisbelow(True)
    plt.yticks(y, preds[i])
    plt.xlabel("probability")
```

```
plt.subplots_adjust(wspace=0.5)
plt.tight_layout()
plt.savefig("./figs/retrieval.png", bbox_inches='tight')
plt.show()
```

```
/home/tul02009/miniconda3/lib/python3.8/site-packages/torchvision/transforms/t
ransforms.py:257: UserWarning: Argument interpolation should be of type Interp
olationMode instead of int. Please, use InterpolationMode enum.
  warnings.warn(
```



In [17]:

```
EMBEDDINGS["cover_text"] = torch.zeros_like(IMAGE_FEATURES)

PREDS_ctext = {}
PROBS_ctext = {}
for COL in COLS:
    PREDS_ctext[COL] = []
    PROBS_ctext[COL] = []

batch_size = 200
print("len not_none_ids",len(EVAL_IDS["cover_text"]))
n_batch = (len(EVAL_IDS["cover_text"])//batch_size)+1

for batch in tqdm(range(n_batch)):
    images = []
    ids = np.arange(batch*batch_size, min((batch+1)*batch_size, len(EVAL_IDS[
    ids = np.array(EVAL_IDS["cover_text"])[ids]

    sentences = list(df.loc[ids,"cover_text"])
    tokenizer = SimpleTokenizer()
    text_tokens = [tokenizer.encode(sentence) for sentence in sentences]

    sot_token = tokenizer.encoder['<|startoftext|>']
    eot_token = tokenizer.encoder['<|endoftext|>']

    text_input = torch.zeros(len(text_tokens), model.context_length, dtype=to
    for i, tokens in enumerate(text_tokens):
        tokens = [sot_token] + tokens + [eot_token]
        text_input[i, :len(tokens)] = torch.tensor(tokens)

    # Generate embeddings for each sentence
    with torch.no_grad():
        ctext_features = model.encode_text(text_input.cuda()).float()
        ctext_features /= ctext_features.norm(dim=-1, keepdim=True)
        EMBEDDINGS["cover_text"][ids] = ctext_features

        for COL in ["holidays", "special_occasions",  'relationships', 'messa
            text_features = EMBEDDINGS[COL]
            similarity = text_features.cpu().numpy() @ ctext_features.cpu().n
```

```
            text_probs = (100.0 * ctext_features @ text_features.T).softmax(d
            top_probs, top_labels = text_probs.cpu().topk(5, dim=-1)
            PREDS_ctext[COL].append(top_labels)
            PROBS_ctext[COL].append(top_probs)

for COL in ["holidays", "special_occasions",  'relationships', 'messages']:
    PREDS_ctext[COL] = np.concatenate(PREDS_ctext[COL], axis=0)
    PROBS_ctext[COL] = np.concatenate(PROBS_ctext[COL], axis=0)
```

```
len not_none_ids 6668
```

In [18]:

```
import pickle
pickle.dump( (EVAL_IDS, EMBEDDINGS), open( "evalids_embeddings.p", "wb" ) )
```

In [19]:

```
print("Overall Accuracy (Ignoring None)")
temp = df.loc[EVAL_IDS["cover_text"]].reset_index()

for COL in ["holidays", "special_occasions",  'relationships', 'messages']:
    non_none_ids = list(temp[temp[COL].apply(",".join) != ""].index)
    pred_ids = PREDS_ctext[COL][:, 0]

    acc1 = np.mean(temp[COL][non_none_ids] == LABELS[COL][pred_ids][non_none_

    pred_ids = PREDS_ctext[COL]
    preds = LABELS[COL][pred_ids]

    acc1 = 0
    acc5 = 0
    for i in non_none_ids:
        gt =  temp[COL][i]

        # if the top-1 prediction is one of the gt label
        if preds[i][0] in gt:
            acc1 +=1

        # if any of the top-5 prediction is one of the gt label
        for p in preds[i]:
            if p in gt:
                acc5 +=1
                break

    print(f"{COL: <20} | # samples = {len(non_none_ids)}, top 1 accuracy = {a
```

```
Overall Accuracy (Ignoring None)
holidays             | # samples = 2193, top 1 accuracy = 0.776, top-5 accurac
y = 0.891
special_occasions    | # samples = 2449, top 1 accuracy = 0.595, top-5 accurac
y = 0.942
relationships        | # samples = 1773, top 1 accuracy = 0.450, top-5 accurac
y = 0.734
messages             | # samples = 2022, top 1 accuracy = 0.336, top-5 accurac
y = 0.509
```

In [20]:

```
ids = np.random.choice(np.arange(len(EVAL_IDS["cover_text"])),5)
rows = []
for i in ids:
    row = {"sentence": temp.loc[i, "cover_text"][20:]}
    for COL in ["holidays", "special_occasions",  'relationships', 'messages'
        row[COL] = ",".join(temp.loc[i,COL])
        row["pred_"+COL] = LABELS[COL][PREDS_ctext[COL][i, 0]]
    rows.append(row)
```

```
print("random sampled cover text and their predictions:")
pd.DataFrame(rows)
```

```
random sampled cover text and their predictions:
```

Out[20]:

	sentence	holidays	pred_holidays	special_occasions	pred_special_occasions	
0	youve earned em thank you so much		CHRISTMAS		BIRTHDAY_BELATED	
1	the cats behind me isnt he		NEW_YEARS		BIRTHDAY_BELATED	
2	one so dear one so loved one so missed		SUMMER		BIRTHDAY_BELATED	
3	happy birthday cuz		NEW_YEARS	BIRTHDAY	BIRTHDAY	COUSI
4	happy mothers day	MOTHERS_DAY	MOTHERS_DAY		BIRTHDAY_BELATED	

6. Retrieval: text queries

Text --> Images

6.1 Demonstrate with random input text and retrieve 10 images

In [21]:

```
print("image_features shape: ", IMAGE_FEATURES.shape)

temp = pd.read_csv(config["ann_folder"] + "/annotations.csv")
gt_cols = ["holidays", "special_occasions",  'relationships', 'messages', "fe
gt_df = temp[gt_cols].apply(",".join, axis=1).str.upper()

test_dict = {
    "BLUE": "This is an image with blue color",
    "ROMANTIC": "This is an romantic image",
    "CHINESE": "This is an image with chinese",
    "SUMMER": "This is an image about summer",
    "BON_VOYAGE": "This is an image about bon voyage",
    "MANLY": "This is an manly image",
    "MOM": "This is an image for mom",
    "GRANDFATHER": "This is an image for grandfather man",
    "APOLOGY": "This is an image for apology",
    "ALIENS": "This is an image of aliens"
}
```

```
image_features shape:  torch.Size([8033, 512])
```

In [22]:

```
labels = ["ROMANTIC", "CHINESE", "BON_VOYAGE"]
_probs = {}
```

```
_images = {}
K=5

for label in labels:
    sentence = test_dict[label]
    ids = list(df[df["features"].apply(",".join).str.contains(label)].index)

    tokenizer = SimpleTokenizer()
    text_tokens = [tokenizer.encode(sentence)]

    sot_token = tokenizer.encoder['<|startoftext|>']
    eot_token = tokenizer.encoder['<|endoftext|>']

    text_input = torch.zeros(len(text_tokens), model.context_length, dtype=to
    for i, tokens in enumerate(text_tokens):
        tokens = [sot_token] + tokens + [eot_token]
        text_input[i, :len(tokens)] = torch.tensor(tokens)

    # Generate embeddings for each sentence
    with torch.no_grad():
        text_features = model.encode_text(text_input.cuda()).float()
        text_features /= text_features.norm(dim=-1, keepdim=True)

        image_probs = IMAGE_FEATURES @ text_features.T
        top_probs, top_images = image_probs.cpu().topk(K, dim=0)
        _images[label] = np.concatenate(list(top_images))
        _probs[label] = np.concatenate(top_probs.numpy())
```

In [23]:

```
#plt.figure(figsize=(16, 8))
fig, axes = plt.subplots(len(labels), K+1, figsize=(16, 8))

# for ax, row in zip(axes[:,0], labels):
#     ax.set_ylabel(row, rotation=0, size='large')

for j, label in enumerate(labels):
    _ = axes[j,0].text(0.5, 0.5, test_dict[label], ha="center", verticalalign
    axes[j,0].axis("off")
    for i in range(K):
        image_idx = _images[label]
        #plt.subplot(len(labels), K, i+1)
        image = Image.open(os.path.join(config['img_folder'], f'{df["card_id"
        axes[j,i+1].imshow(image)
        axes[j,i+1].set_title(f"[{i+1}] p={_probs[label][i]:.3f}")
        axes[j,i+1].axis("off")
plt.tight_layout()
plt.savefig("./figs/abstract_retrieval.png", bbox_inches='tight')
```

6.2 Eval: retrieval using text

Use text as the query and check the accuracy for top1, top10, and top20.

In [24]:

```
for temp, text in enumerate(["GT = its column labels", "GT = all column label
    print("\n" + text)
    eval_r = {}
    for COL in ["holidays", "special_occasions",  'relationships', 'messages'
        eval_r[COL] = {}
        for top in [1,10,20]:
            eval_r[COL][top] = {}

        for i in range(len(LABELS[COL])):
            label = LABELS[COL][i]

            for top in [1,10,20]:
                eval_r[COL][top][label] = 0

            text_features = torch.unsqueeze(EMBEDDINGS[COL][i],0)
            image_probs = IMAGE_FEATURES @ text_features.T
            top_probs, top_images = image_probs.cpu().topk(20, dim=0)
            top_images = np.concatenate(list(top_images))

            for top_i in range(20):
                if temp ==0:
                    gt = df.loc[top_images[top_i],COL]
                else:
                    gt = gt_df[top_images[top_i]]
                if label in gt:
                    for top in [1,10,20]:
                        if top_i < top:
                            eval_r[COL][top][label] += 1

            for top in [1,10,20]:
                eval_r[COL][top][label] /= top

        s = f"{COL: <20} | #unique = {len(LABELS[COL]):<4} |"
        for top in [1,10,20]:
            s += f" top{top} = {np.mean(list(eval_r[COL][top].values())):.3f}
        print(s)
```

```
GT = its column labels
holidays             | #unique = 29   | top1 = 0.828 | top10 = 0.600 | top20 =
```

```
0.493 |
special_occasions     | #unique = 29   | top1 = 0.655 | top10 = 0.393 | top20 =
0.286 |
relationships         | #unique = 82   | top1 = 0.329 | top10 = 0.187 | top20 =
0.136 |
messages              | #unique = 34   | top1 = 0.441 | top10 = 0.276 | top20 =
0.235 |
obj_features          | #unique = 482  | top1 = 0.490 | top10 = 0.272 | top20 =
0.203 |

GT = all column labels
holidays              | #unique = 29   | top1 = 0.897 | top10 = 0.652 | top20 =
0.547 |
special_occasions     | #unique = 29   | top1 = 0.655 | top10 = 0.434 | top20 =
0.326 |
relationships         | #unique = 82   | top1 = 0.378 | top10 = 0.215 | top20 =
0.157 |
messages              | #unique = 34   | top1 = 0.500 | top10 = 0.324 | top20 =
0.279 |
obj_features          | #unique = 482  | top1 = 0.525 | top10 = 0.289 | top20 =
0.218 |
```

In [25]:

```
rows = []
n = 7
for COL in ["holidays", "special_occasions",  'relationships', 'messages', "ol
    temp = np.array(list(eval_r[COL][20].items()))
    temp = temp[np.argsort(temp[:,1])]

    row = {"column": COL, "des": "Min"}
    for i, val in enumerate(temp[:n]):
        row[i] = val
    rows.append(row)

    row = {"column": COL, "des": "Max"}
    for i, val in enumerate(temp[-n:][::-1]):
        row[i] = val
    rows.append(row)

print("best and worst labels in each category: ")
pd.DataFrame(rows)
```

```
best and worst labels in each category:
```

Out[25]:

	column	des	0	1	
0	holidays	Min	[GROUNDHOG_DAY, 0.05]	[BELTANE, 0.1]	[SUMMER
1	holidays	Max	[MOTHERS_DAY, 1.0]	[HALLOWEEN, 1.0]	[VETERANS_
2	special_occasions	Min	[HONEYMOON, 0.0]	[BIRTHDAY_BELATED, 0.0]	[EXPECTING_A_E
3	special_occasions	Max	[BIRTHDAY, 0.95]	[ANNIVERSARY_ORDINATION, 0.75]	[ORDINATION,
4	relationships	Min	[GREAT_NEPHEW, 0.0]	[STEPDAUGHTER, 0.0]	[FIANCEE
5	relationships	Max	[DAD, 0.9]	[MOM, 0.85]	[DAUGHTER,
6	messages	Min	[YOU_WILL_BE_MISSED, 0.0]	[WISHING_YOU, 0.0]	[SORRY_IM_I
7	messages	Max	[GET_WELL_SOON, 1.0]	[THINKING_OF_YOU, 0.95]	[THANK_YOU

	column	des	0	1	
8	obj_features	Min	[ALIENS, 0.0]	[MEDICINE, 0.0]	[CARNATIONS
9	obj_features	Max	[HORSES, 0.95]	[BUTTERFLIES, 0.95]	[SKELETON,

7. Retrieval: image queries

recall = TP/ (TP + FN)

7.1 Eval on test set

In [26]:
```
#gt = np.array(df.loc[eval_ids, COL].reset_index(drop=True).str[0])  #use the
```

In [27]:
```
unique = {}
for COL in ["holidays", "special_occasions",  'relationships', 'messages', "o
    val, cts = np.unique(np.concatenate(df[COL]), return_counts=True)
    unique[COL] = {}
    for v,c in zip(val,cts):
        unique[COL][v] = c
```

In [28]:
```
test_ids = list(df[df["train_test"] == "test"].index)
```

In [29]:
```
print("Retrieval Top-k Accuracy:")
retrieval_stat = []

for des in [ "USE (Universal Sentense Encoder)",  "cover text", "image", "ima
    print(f"*** Retrieval using {des} features ***")
    retrieval_pred = {}
    retrieval_prob = {}

    row = {"Features": des}
    for COL in ["holidays", "special_occasions",  'relationships', 'messages'
        retrieval_pred[COL], retrieval_prob[COL] = [], []
        eval_ids = EVAL_IDS[COL]
        eval_ids = [i for i in eval_ids if i in test_ids]
        _ids = np.arange(len(df))[eval_ids]

        # Input Features
        if "USE" in des:
            F = USE
        elif "cover text" in des:
            F = EMBEDDINGS["cover_text"]
        elif des == "image":
            F = IMAGE_FEATURES

        if "image and" in des:
            F = 0.5 * F + 0.5 * IMAGE_FEATURES
        if des == "all":
            F = 0.5 * IMAGE_FEATURES + 0.25 * USE + 0.25 * EMBEDDINGS["cover_
        _F = F[eval_ids]

        # Get predictions
        with torch.no_grad():
```

```
            for batch in range(n_batch):
                ids = np.arange(batch*batch_size, min((batch+1)*batch_size, l
                query_features = _F[ids]
                probs = (100.0 * query_features @ _F.T)
                probs[np.arange(len(ids)), ids] = 0 #set the probability for
                probs = probs.softmax(dim=-1)
                top_probs, top_labels = probs.cpu().topk(20, dim=-1)

                retrieval_pred[COL] += [_ids[l] for l in top_labels]
                retrieval_prob[COL].append(top_probs)

            retrieval_pred[COL] = np.array(retrieval_pred[COL])
            retrieval_prob[COL] = np.concatenate(retrieval_prob[COL])

        preds = retrieval_pred[COL]
        R = {1:0, 5:0, 10:0}
        N = {1:0, 5:0, 10:0}
        for i in range(len(preds)):
            pred = df.loc[preds[i], COL]
            gt = df.loc[eval_ids[i], COL]

            tp = list(pred.apply(lambda x: x[0] in gt))
            for k in [1, 5, 10]:
                R[k] += sum(tp[:k])
                N[k] += min(k, unique[COL][gt[0]])

        for k in [1, 5, 10]:
            row[f"{COL}_{k}"] = R[k]/ N[k]
    retrieval_stat.append(row)
```

```
Retrieval Top-k Accuracy:
*** Retrieval using USE (Universal Sentense Encoder) features ***
*** Retrieval using cover text features ***
*** Retrieval using image features ***
*** Retrieval using image and USE features ***
*** Retrieval using image and cover text features ***
*** Retrieval using all features ***
```

In [30]:

```
pd.DataFrame(retrieval_stat).round(3).T
```

Out[30]:

	0	1	2	3	4	5
Features	USE (Universal Sentense Encoder)	cover text	image	image and USE	image and cover text	all
holidays_1	0.679	0.687	0.711	0.77	0.785	0.782
holidays_5	0.648	0.69	0.636	0.717	0.741	0.736
holidays_10	0.605	0.647	0.566	0.66	0.702	0.683
special_occasions_1	0.889	0.871	0.835	0.877	0.866	0.877
special_occasions_5	0.839	0.844	0.804	0.828	0.828	0.824
special_occasions_10	0.822	0.832	0.799	0.819	0.814	0.813
relationships_1	0.437	0.479	0.278	0.407	0.365	0.386
relationships_5	0.343	0.348	0.205	0.302	0.285	0.283
relationships_10	0.268	0.291	0.172	0.244	0.241	0.23
messages_1	0.502	0.478	0.347	0.5	0.485	0.485
messages_5	0.466	0.414	0.316	0.462	0.406	0.433
messages_10	0.415	0.368	0.28	0.417	0.374	0.387

	0	1	2	3	4	5
obj_features_1	0.229	0.233	0.463	0.372	0.451	0.438
obj_features_5	0.221	0.211	0.391	0.306	0.367	0.354
obj_features_10	0.203	0.205	0.37	0.287	0.344	0.333

Retrieval Findings:

- In terms of text encodings, for holidays and messages USE is better than CLIP for top-1, but worse for top-10 & top-20.
- In terms of text encodings, for special occasions and relationships CLIP is better than USE.
- In terms of text + image, USE is better than clip for all measures.
- The best performance comes forom USE + CLIP image features. It is even better then using all features: 0.5 image + 0.25 USE + 0.25 CLIP text

7.2 Visualize

In [32]:
```
# COL = "relationships"
# i = np.random.choice(np.arange(len(EVAL_IDS[COL])))
# preds = retrieval_pred[COL][i]
# probs = retrieval_prob[COL][i]

# plt.figure(figsize=(3, 3))
# card_id = df.loc[EVAL_IDS[COL][i], "card_id"]
# gtlabel = df.loc[EVAL_IDS[COL][i], COL]
# image = Image.open(os.path.join(config['img_folder'], f'{card_id}.jpg' ))
# plt.imshow(image)
# plt.title(gtlabel)

# plt.figure(figsize=(16, 8))
# for i, image_idx in enumerate(preds):
#     plt.subplot(4, 5, i+1)
#     image = Image.open(os.path.join(config['img_folder'], f'{df["card_id"][
#     plt.imshow(image)
#     label = df[COL][image_idx]
#     c = "red"
#     for l in label:
#         if l in gtlabel:
#             c = "green"
#     plt.title( f"{label} {probs[i]:.2f}", c=c, )
#     plt.axis('off')
```

8. Abstract concepts

In [33]:
```
ABSTRACT_CONCEPTS = {
"REGION":  ["IRELAND", "AMERICANA", "UNITED_STATES", "AFRICA", "MEXICO", "USA
"CHINESE_THEMED", "IRISH_THEMED", "HAWAIIAN_THEMED", "HEBREW", "SIAMESE"],


"LANGUAGE":  ["CHINESE_LANGUAGE", "SPANISH_LANGUAGE"],
"COLORS" :   ["BLACK", "GREEN", "PINK", "BLUE", "YELLOW", "PURPLE", "RED", "W

"SEASONS" : ["FALL", "WINTER", "SPRING", "SUMMER"],
```

```
"CARD_STYLE": [ "CLEAN", "COOL", "CUTE",  "GIRLY", "MANLY", "ART", "ROMANTIC"


"RELIGIOUS": [ "CHRISTIANITY",  "JUDAISM", "CATHOLICISM","CHRISTIAN_THEMED",
}
```

In [35]:

```
EMBEDDINGS_A = {}
SENTENCES_A = {}

for group in tqdm(ABSTRACT_CONCEPTS.keys()):
    # Get unique labels
    labels = ABSTRACT_CONCEPTS[group]

    # Create a sentence for each labels
    sentences = []
    for t in labels:
        t = t.lower().replace("_", " ").replace("/", " or ")
        sentence = f"This is an image of {t}"

        if group == "LANGUAGE":
            sentence = f"This is an image with {t} words"
        elif group == "COLORS":
            sentence = f"This is an {t} image"
        elif group == "SEASONS":
            sentence = f"This is an image of the {t} season"
        elif group == "CARD_STYLE":
            sentence = f"This is an image with a {t} style"
        elif group == "RELIGIOUS":
            sentence = f"This is an religious image about {t}"
        else:
            sentence= f"This is an image of {t}"
        sentences.append(sentence)

    sentences = np.array(sentences)
    labels = np.array(labels)

    # Tokenize all texts
    tokenizer = SimpleTokenizer()
    text_tokens = [tokenizer.encode(desc) for desc in sentences]

    sot_token = tokenizer.encoder['<|startoftext|>']
    eot_token = tokenizer.encoder['<|endoftext|>']

    text_input = torch.zeros(len(text_tokens), model.context_length, dtype=to
    for i, tokens in enumerate(text_tokens):
        tokens = [sot_token] + tokens + [eot_token]
        text_input[i, :len(tokens)] = torch.tensor(tokens)

    # text_input = text_input.cuda()

    # Generate embeddings for each sentence
    with torch.no_grad():
        text_features = model.encode_text(text_input.cuda()).float()
        text_features /= text_features.norm(dim=-1, keepdim=True)

        EMBEDDINGS_A[group] = text_features
        SENTENCES_A[group] = sentences
```

In [36]:

```
batch_size = 200
n_batch = (len(df)//batch_size)+1
PREDS_abstract = {}
```

```
for group in tqdm(ABSTRACT_CONCEPTS.keys()):
    PREDS_abstract[group] = {}
    temp = df[df["keep"]].loc[df.features.apply(lambda x: bool(set(x) & set(A
    temp = temp.apply(lambda x: [i for i in x if i in ABSTRACT_CONCEPTS[group
    ids = temp.index

    with torch.no_grad():
        image_features = IMAGE_FEATURES[ids]
        text_features = EMBEDDINGS_A[group]
        similarity = text_features.cpu().numpy() @ image_features.cpu().numpy

        text_probs = (100.0 * image_features @ text_features.T).softmax(dim=-
        top_probs, top_labels = text_probs.cpu().topk(min(5, len(ABSTRACT_CON
        PREDS_abstract[group]["top_labels"] = top_labels
        PREDS_abstract[group]["ids"] = ids
        PREDS_abstract[group]["gt"] = temp.values
```

8.1 Eval

In [37]:

```
rows = []
for group in ABSTRACT_CONCEPTS.keys():
    acc=0
    preds = np.array(ABSTRACT_CONCEPTS[group])[PREDS_abstract[group]["top_lab
    for i in range(len(preds)):
        if preds[i] in PREDS_abstract[group]["gt"]:
            acc+=1
    rows.append({"group":group.lower(), "num. samples": len(preds), "num. uni
print("classification accuracy")
pd.DataFrame(rows)
```

```
classification accuracy
```

Out[37]:

	group	num. samples	num. unique labels	acc
0	region	435	18	0.788506
1	language	12	2	1.000000
2	colors	219	8	1.000000
3	seasons	255	4	0.972549
4	card_style	1122	28	0.892157
5	religious	249	5	1.000000

Only evaluated on the samples with the abstract concept tags </br> All having high accuracy.

8.2 visualize

Retrieve images with an abstract-concept tag

In [38]:

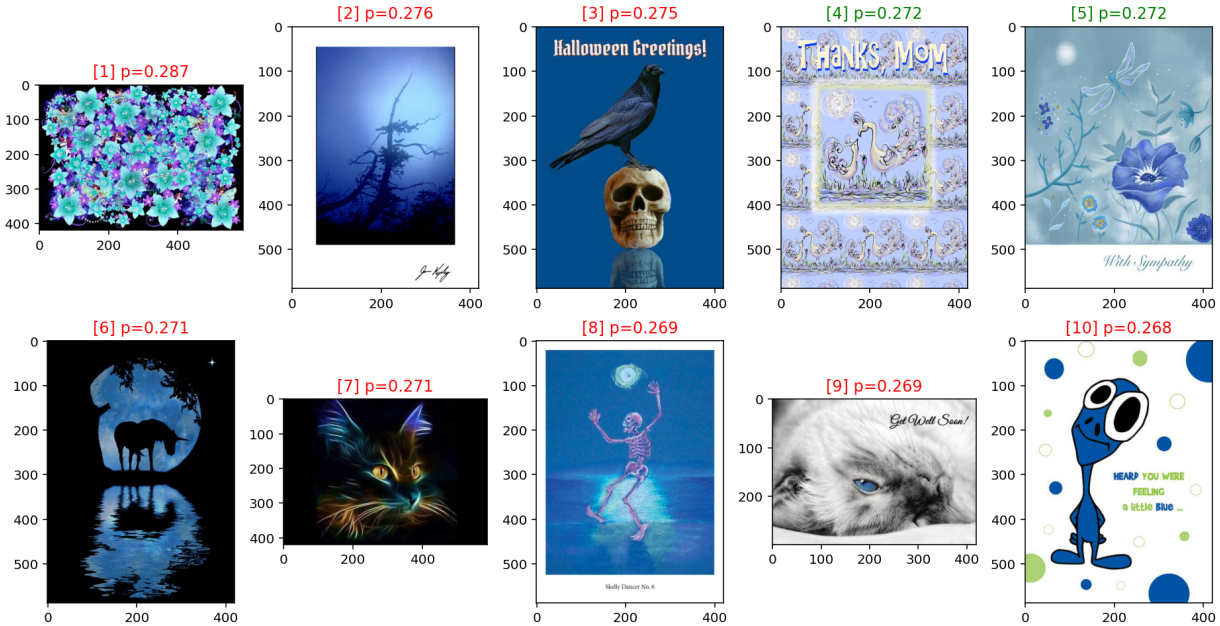
```
# ABSTRACT_CONCEPTS["REGION"]
```

In [40]:

```
group = "COLORS"
keyword = "BLUE"
idx = ABSTRACT_CONCEPTS[group].index(keyword)
```

```
text_features = EMBEDDINGS_A[group][idx]
text_features = torch.unsqueeze(text_features,0)

image_probs = IMAGE_FEATURES @ text_features.T
top_probs, top_images = image_probs.cpu().topk(10, dim=0)
top_images = np.concatenate(list(top_images))
top_probs = np.concatenate(top_probs.numpy())

print(SENTENCES_A[group][idx])
plt.figure(figsize=(16, 8))
for i, image_idx in enumerate(top_images):
    plt.subplot(2, 5, i+1)
    image = Image.open(os.path.join(config['img_folder'], f'{df["card_id"][ima
    plt.imshow(image)
    if keyword in df["features"][image_idx]:
        c = "g"
    else:
        c = "r"
    plt.title(f"[{i+1}] p={top_probs[i]:.3f}", c=c)
    #plt.title(gt, c=c)
```

```
This is an blue image
```



9 Features analysis

K-NN

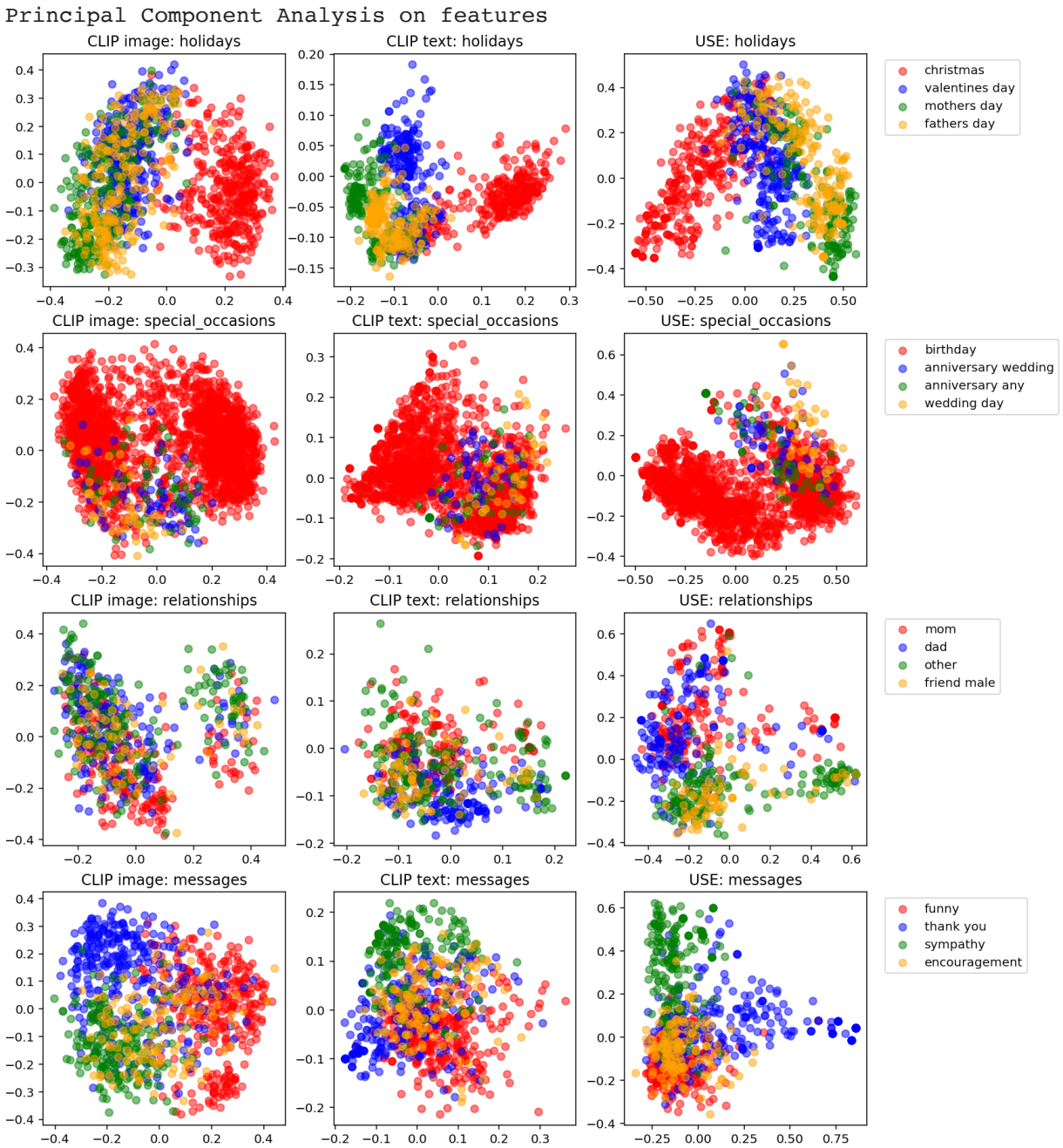
In [41]:

```
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
```

In [42]:

```
colors = ["r", "b", "g", "orange"]
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][:4]
```

```
    for des, feat in [["CLIP image", IMAGE_FEATURES], ["CLIP text", EMBEDDINGS
        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):
            temp = principalComponents[gt==val]
            val = " ".join( val.split("_")[:2]).lower()
            plt.scatter(temp[:,0], temp[:,1], label=val, c=c, alpha=0.5)
        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')
```



In [43]:

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

```
principalComponents = tsne.fit_transform(X)
```

In [ ]:

```
colors = ["r", "b", "g", "orange"]
plt.figure(figsize=(12, 16))
print("TSNE 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][:4]

    for des, feat in [["CLIP image", IMAGE_FEATURES], ["CLIP text", EMBEDDINGS
        plt.subplot(4,col, i+1)
        tsne = TSNE(n_components=2)
        X = feat[eval_ids].cpu().numpy()
        principalComponents = tsne.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):
            temp = principalComponents[gt==val]
            val = " ".join( val.split("_")[:2]).lower()
            plt.scatter(temp[:,0], temp[:,1], label=val, c=c, alpha=0.5)
        if (i+1)%col ==0:
            _ = plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
        i += 1

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

```
TSNE on features
```

Future steps for improving aggregators

Goal v1: Merge CLIP features and text features, with or without further training, for better prediction results
Goal v2: Enhance the current features, by training a new features.
or with few-shot learning. Goal v3:

Motivation: CLIP is trained on a diverse dataset which I don't have the computer resources to train it within a short time. It already captures a diverse range of concepts in its image embeddings and text embeddings. To improve it on our specialized tasks, we may either fine-tune the CLIP model or train a new model to enhance it.

1. Fine-Tuning. It looks like it may take a large amount of computer resources. I don't know, should be tested.
2. New small model:
   i. Merge the existing embeddings
   - concatenation [tested]
   - addition

   ii. Merge the existing embeddings and train a new model for the domain adapatation.
   - Textual features residual + image features. Trained new layers for weighting and merging. [Gated and residual features][Sidra is trying]

```
 - x1 * text feat. + x2 * image feat. Train x1 & x2.
 - concatenation & graph convolution on nearest neighbors
 - concatenation + RESNET's features
 - graph NN. aggregate neighbor images. However I think it is only
benefitial if neighborhood images provides significant different
info.
 - train a new model to analysis the image again, create a residual
features and merge with the zero-shot features.
     - the residual features should has less weight than the
original
     - residual models: ResNet
```

Similar structure to CLIP: Scaling Up Visual and Vision-Language Representation Learning With Noisy Text Supervision

1. Deside which subset to train on
2. Get imagenet image features