In [1]:
```
import os
import re
import pickle
import datetime
from datetime import datetime, timedelta
```

In [2]:
```
JOURNALS = {1: "Elsevier Pattern Recognition",
            2: "Elsevier Computer Vision and Image Understanding",
            3: "IEEE Transactions on Pattern Analysis and Machine Intelligence",
            4: "IEEE Transactions on Neural Networks and Learning Systems",
            5: "IEEE Transactions on Image Processing",
            6: "IET Computer Vision",
            7: "IET Image Processing",
            8: "IET Electronic Letters"}
```

In [3]:
```
files = [f for f in os.listdir('./collected_articles/') if os.path.isfile(os.path.join('./collected_articles/', f))]
```

In [4]:
```
files_filtered = files[2:]
```

In [5]:
```
all_articles = []

for file_name in files_filtered:
    with open("./collected_articles/"+file_name, 'rb') as file:
        collected_articles = pickle.load(file)

    date = file_name[:-4]

    for article in collected_articles:
        all_articles.append({"title": article[0], "url": article[1], "desc": article[2], "authors": article[3], "journal_id": int(article[4]), "d
```

Show Search Word Occurrences

In [156…
```
#search_words = ["infrared", "infra-red"]
```

In [6]:
```
def check_article_date(input_date, input_article):

    if input_article["date"]==input_date:
        return [input_article]
    else:
        return []
```

In [7]:
```
def check_article_for_search_words(input_search_words, input_article):
    out_article = []
    for search_word in input_search_words:
        if (search_word.lower() in input_article['title'].lower()) | (search_word.lower() in input_article['desc'].lower()):
            out_article = [input_article]
            break;
    return out_article
```

In [8]:
```
out = None

def search_word_histogram(start_date, end_date, input_search_words, journal_id=None):

    output_histogram = {}

    start = datetime.strptime(start_date, "%Y-%m")
    end = datetime.strptime(end_date, "%Y-%m")

    current = start

    # ITERATE THROUGH ALL DATES IN THE SPECIFIED RANGE

    while current <= end:

        # FILTER OUT ARTICLES ONLY AT GIVEN DATE

        articles_for_date = []

        date = current.strftime("%Y-%m")

        for article in all_articles:
            out = article
            checked_article = check_article_date(date, article)
            if len(checked_article) > 0:
                articles_for_date+=checked_article

        # CHECK ALL ARTICLES FOR PRESENCE OF SEARCH WORDS

        articles_for_search_words = []

        for article in articles_for_date:
            checked_article = check_article_for_search_words(input_search_words, article)
            if len(checked_article)>0:
                articles_for_search_words+=checked_article

        # COUNT THE NUMBER OF SEARCH WORD OCCURRENCES AND WRITE HISTOGRAM

        count = 0
        if (journal_id is None) | (not isinstance(journal_id, int)):
            count = len(articles_for_search_words)
        else:
            for article in articles_for_search_words:
```

```
                if article["journal_id"]==journal_id:
                    count+=1

        num_occurences = count
        output_histogram[date] = num_occurences


        if current.month == 12:
            current = current.replace(year=current.year + 1, month=1)
        else:
            current = current.replace(month=current.month + 1)

    return output_histogram


def total_articles_histogram(start_date, end_date, journal_id=None):

    output_histogram = {}

    start = datetime.strptime(start_date, "%Y-%m")
    end = datetime.strptime(end_date, "%Y-%m")

    current = start

    # ITERATE THROUGH ALL DATES IN THE SPECIFIED RANGE

    while current <= end:

        # FILTER OUT ARTICLES ONLY AT GIVEN DATE

        articles_for_date = []

        date = current.strftime("%Y-%m")

        for article in all_articles:
            out = article
            checked_article = check_article_date(date, article)
            if len(checked_article) > 0:
                articles_for_date+=checked_article

        # COUNT THE NUMBER OF ARTICLES AND WRITE HISTOGRAM

        count = 0
        if (journal_id is None) | (not isinstance(journal_id, int)):
            count = len(articles_for_date)
        else:
            for article in articles_for_search_words:
                if article["journal_id"]==journal_id:
                    count+=1

        num_occurences = count
        output_histogram[date] = num_occurences


        if current.month == 12:
            current = current.replace(year=current.year + 1, month=1)
        else:
            current = current.replace(month=current.month + 1)

    return output_histogram
```

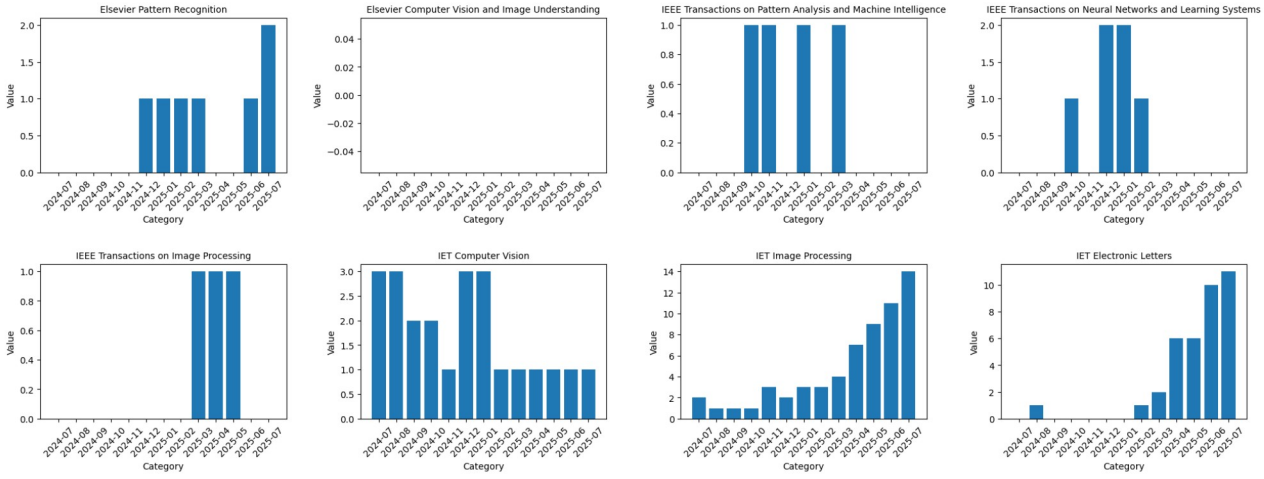
In [9]:

```
def show_search_words_hist(start_date, end_date, input_words):

    bar_data_list = [(i, search_word_histogram(start_date, end_date, input_words, journal_id=i)) for i in range(1,9)]

    fig, axes = plt.subplots(2, 4, figsize=(20, 8))
    axes = axes.flatten()  # flatten so you can index from 0 to 7

    for i, bar_data in enumerate(bar_data_list,0):
        keys = list(bar_data[1].keys())
        values = list(bar_data[1].values())

        ax = axes[i]
        ax.bar(keys, values)
        ax.set_title(f"{JOURNALS[1+i]}", fontsize=10)
        ax.set_xlabel("Category")
        ax.set_ylabel("Value")
        ax.tick_params(axis='x', rotation=45)

    # Layout adjustments
    plt.tight_layout(pad=3.0)
    plt.show()

def show_search_words_pie_chart(start_date, end_date, input_words, vis_angle=90):

    labels = JOURNALS.values()
    sizes = [sum(map(int,search_word_histogram(start_date, end_date, input_words, journal_id=i).values())) for i in range(1,9)]

    plt.figure(figsize=(6, 6))
    #plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90, pctdistance=1.0)
    patches, texts, autotexts = plt.pie(sizes, labels=None, autopct='%1.1f%%', startangle=vis_angle, pctdistance=1.1)
    plt.legend(patches, labels, loc='center left', bbox_to_anchor=(1.2, 0.5))
    plt.title(f'Journal Share from {start_date} to {end_date} for Search Terms \"{', '.join(input_words)}\"')
    plt.axis('equal')  # Makes the chart a circle
    plt.show()


def show_search_word_frequencies(start_date, end_date, input_words):

    labels = JOURNALS.values()
```

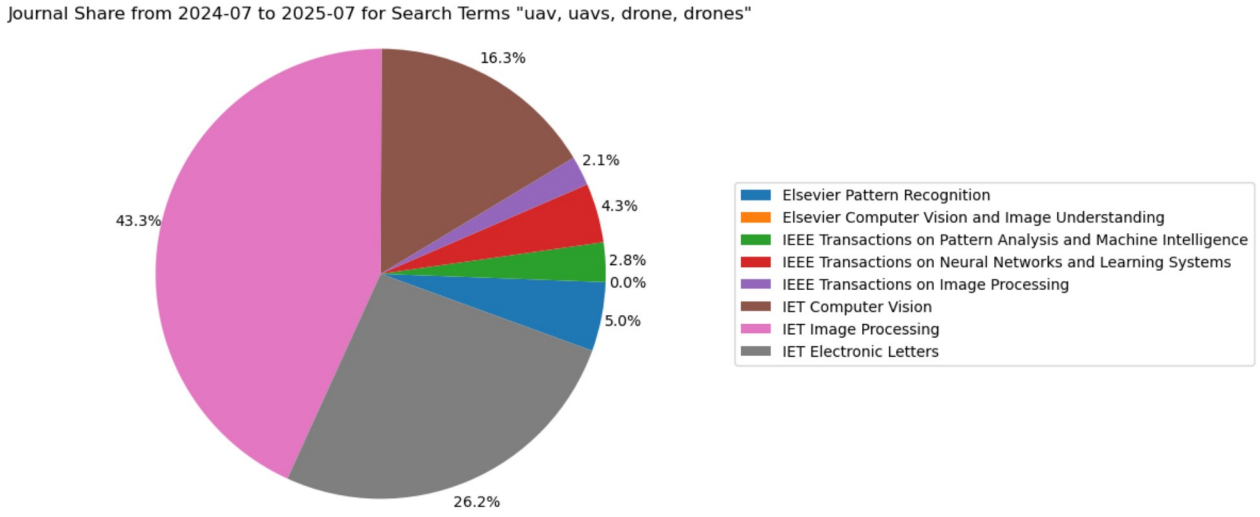
```
article_counts = [sum(map(int,search_word_histogram(start_date, end_date, input_words, journal_id=i).values())) for i in range(1,9)]
total_counts =  [sum(map(int,total_articles_histogram(start_date, end_date, journal_id=None).values())) for i in range(1,9)]
freqs = []
for a_count, tot_count in zip(article_counts, total_counts):
    freq = 0
    if tot_count!=0:
        freq = float(a_count)/float(tot_count)
    freqs.append(freq)

# Generate distinct colors (colormap or custom list)
colors = plt.cm.tab10(range(len(labels)))  # You can also use your own color list

# Create figure and axis
fig, ax = plt.subplots(figsize=(10, 4))

# Plot bars and save each bar object for the legend
bars = []
for i in range(len(labels)):
    bar = ax.bar(i, freqs[i], color=colors[i])
    bars.append(bar[0])  # bar() returns a container

# Set x-ticks and labels
ax.set_xticks(np.arange(len(labels)))
#ax.set_xticks([])
#ax.set_xticklabels(labels)

# Set logarithmic y-axis
ax.set_yscale('log')
ax.set_ylabel("Search Word Frequencies")
ax.set_title(f'Frequencies for "{", ".join(input_words)}" Across Journals')

# Add legend on the right
ax.legend(bars, labels, loc='center left', bbox_to_anchor=(1.0, 0.5))

# Optional layout tweak
plt.tight_layout()
plt.show()
```

SEARCH TERMS: UAV, UAVs, DRONE, DRONES

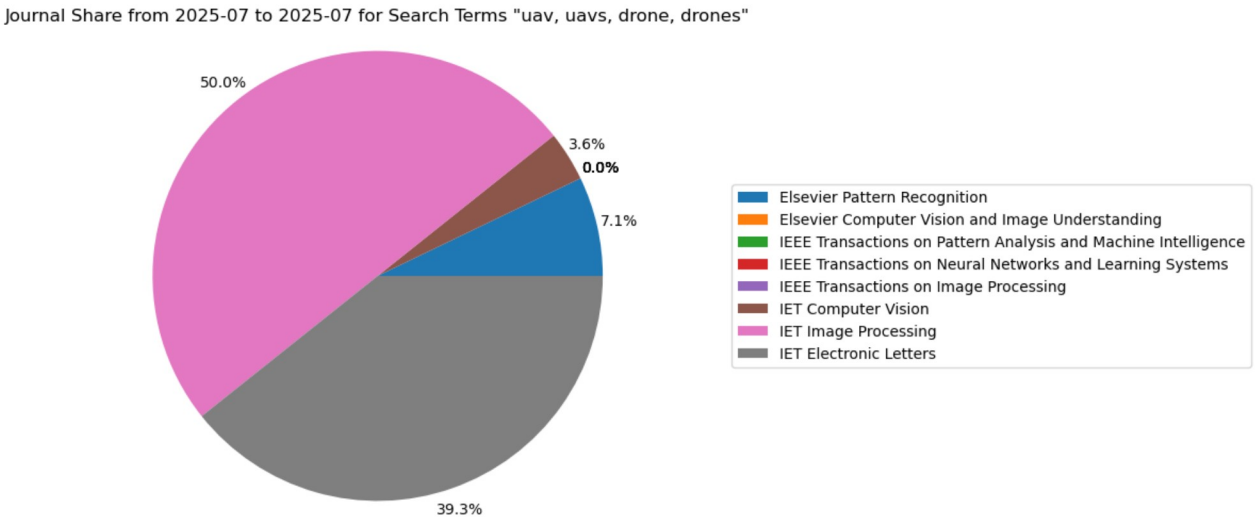
In [345…
```
show_search_words_hist("2024-07", "2025-07", ["uav", "uavs", "drone", "drones"])
```



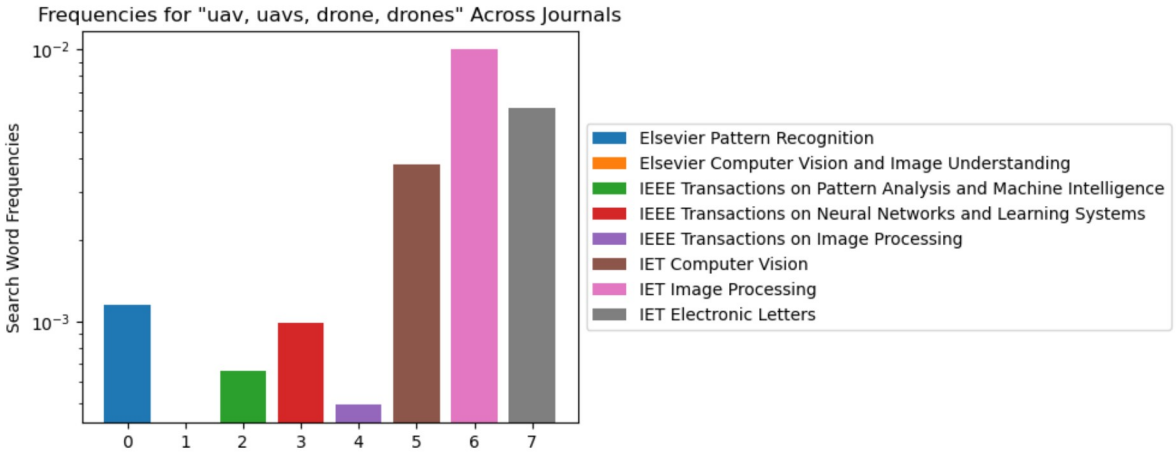
In [346…
```
show_search_words_pie_chart("2024-07", "2025-07", ["uav", "uavs", "drone", "drones"], -20)
```



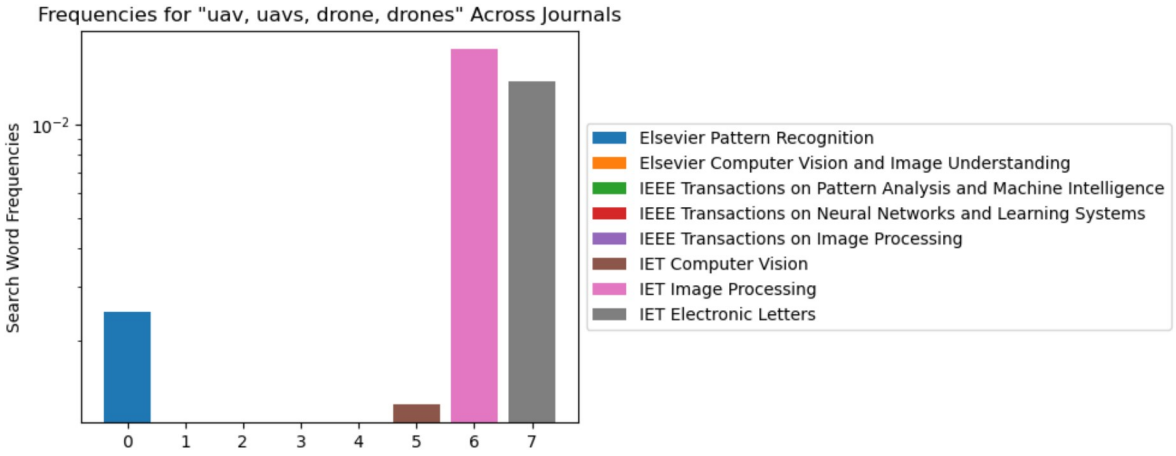
In [347…
```
show_search_words_pie_chart("2025-07", "2025-07", ["uav", "uavs", "drone", "drones"], 0)
```



In [348…
```
show_search_word_frequencies("2024-07", "2025-07", ["uav", "uavs", "drone", "drones"])
```

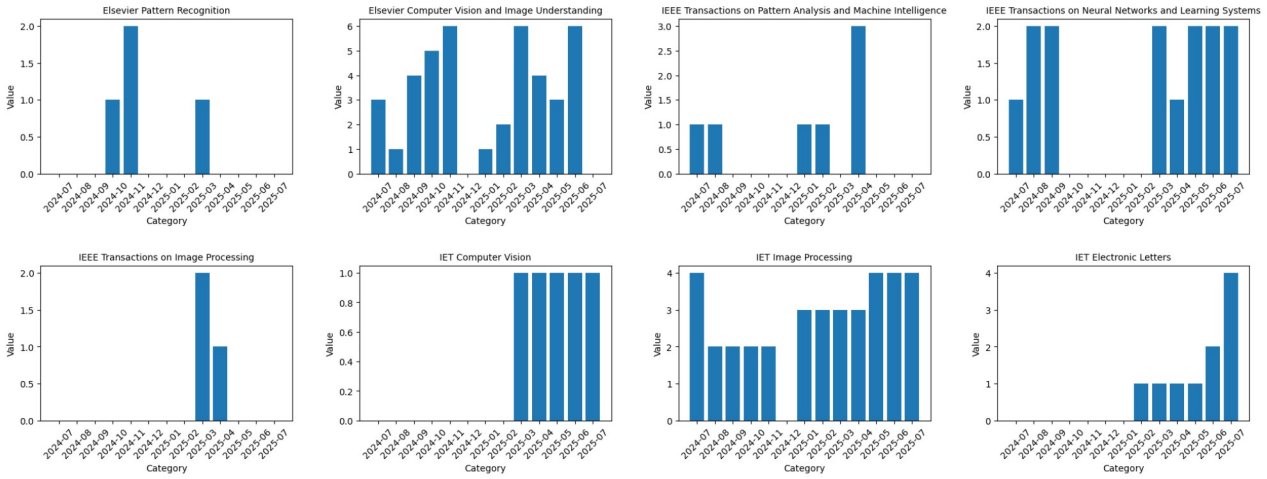


In [349…
```
show_search_word_frequencies("2025-07", "2025-07", ["uav", "uavs", "drone", "drones"])
```

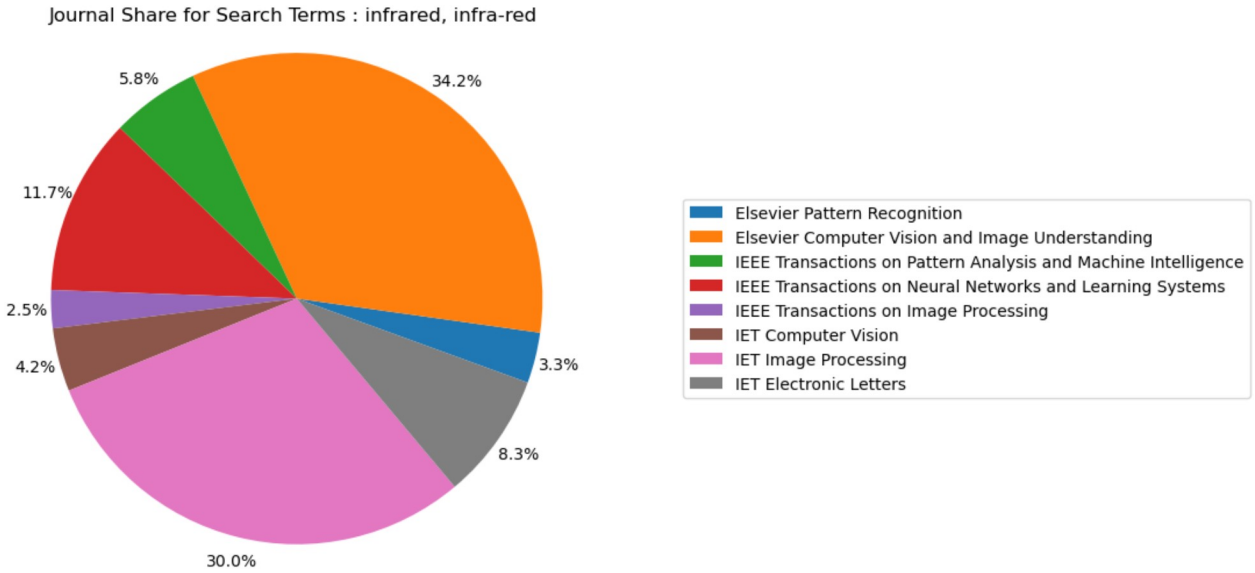


SEARCH WORD: INFRARED, INFRA-RED

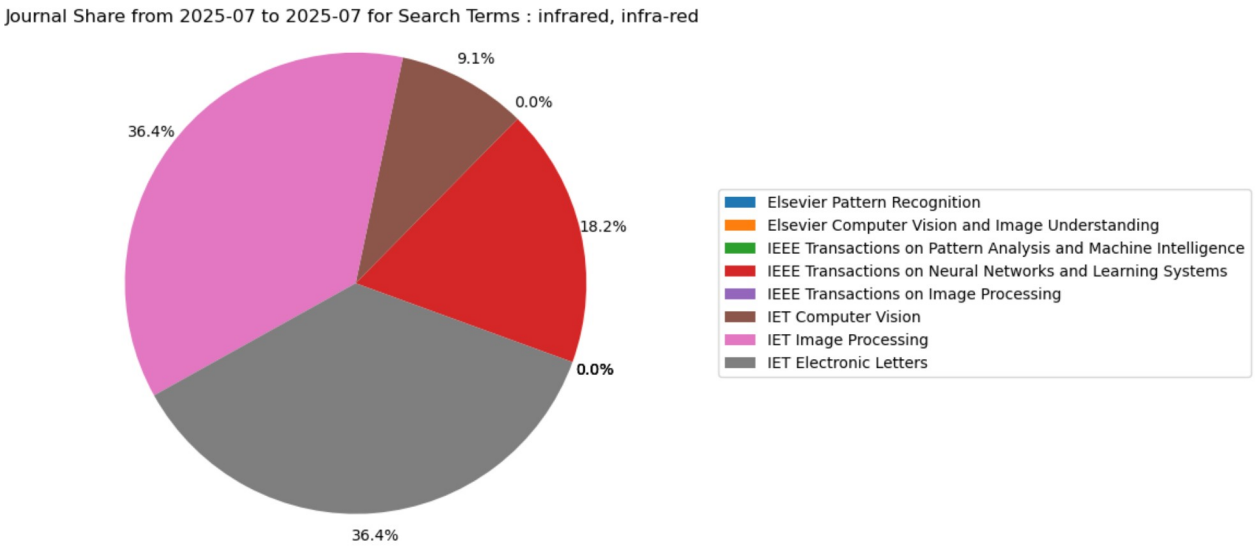
In [225…
```
show_search_words_hist("2024-07", "2025-07", ["infrared", "infra-red"])
```



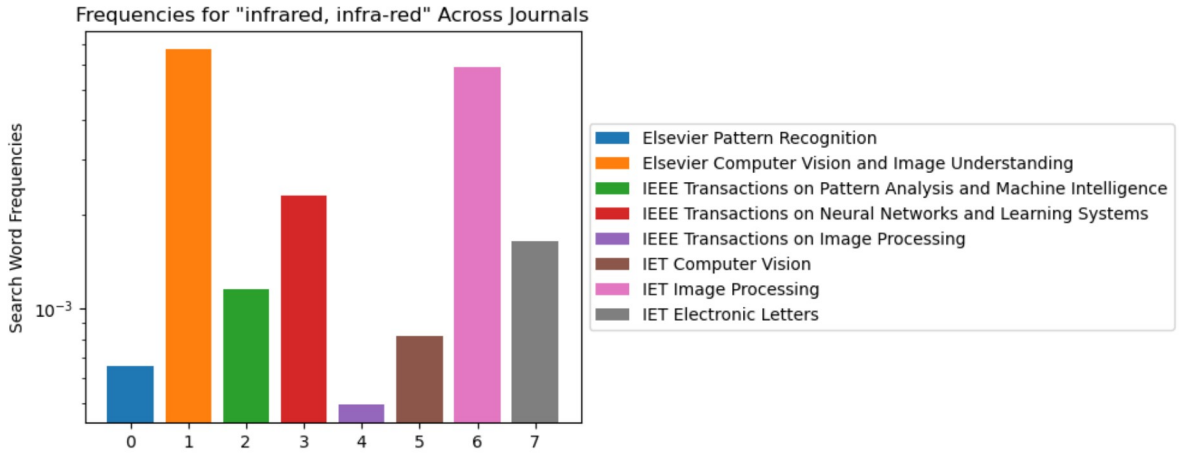
In [289…
```
show_search_words_pie_chart("2024-07", "2025-07", ["infrared", "infra-red"], -20)
```



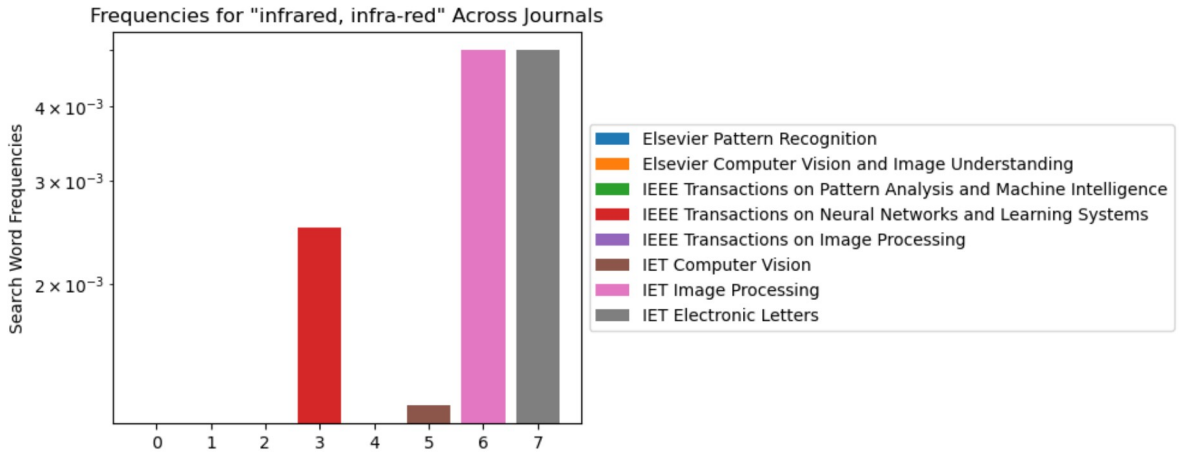
In [300…
```
show_search_words_pie_chart("2025-07", "2025-07", ["infrared", "infra-red"], -20)
```



In [350…
```
show_search_word_frequencies("2024-07", "2025-07", ["infrared", "infra-red"])
```

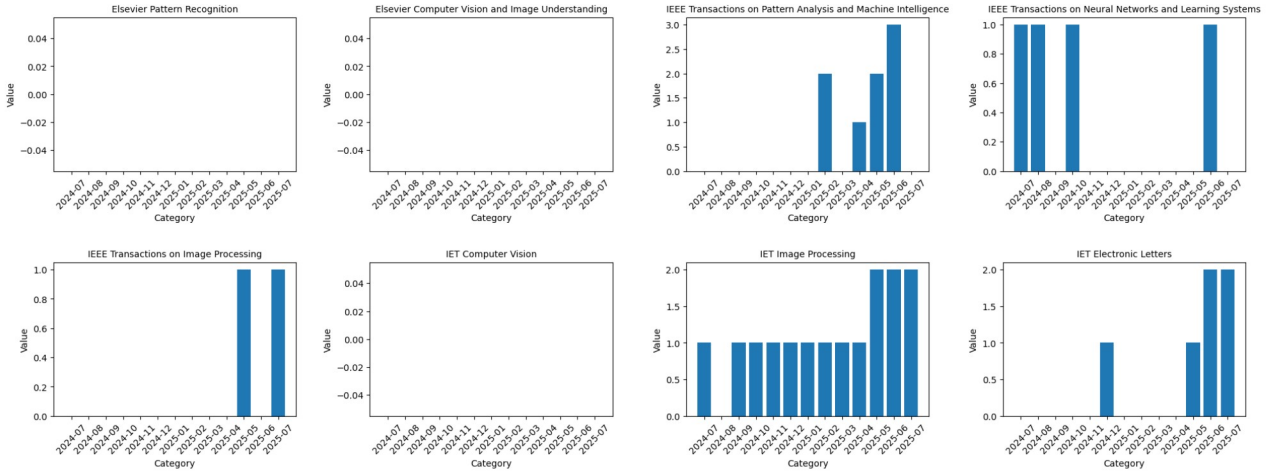


In [351…
```
show_search_word_frequencies("2025-07", "2025-07", ["infrared", "infra-red"])
```

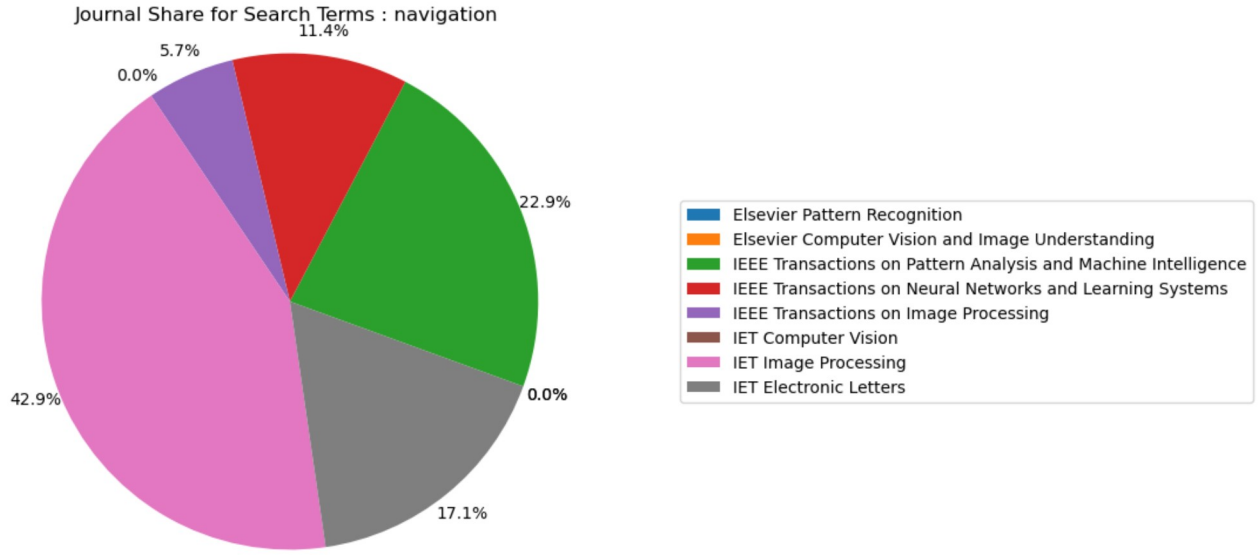


SEARCH WORD: NAVIGATION

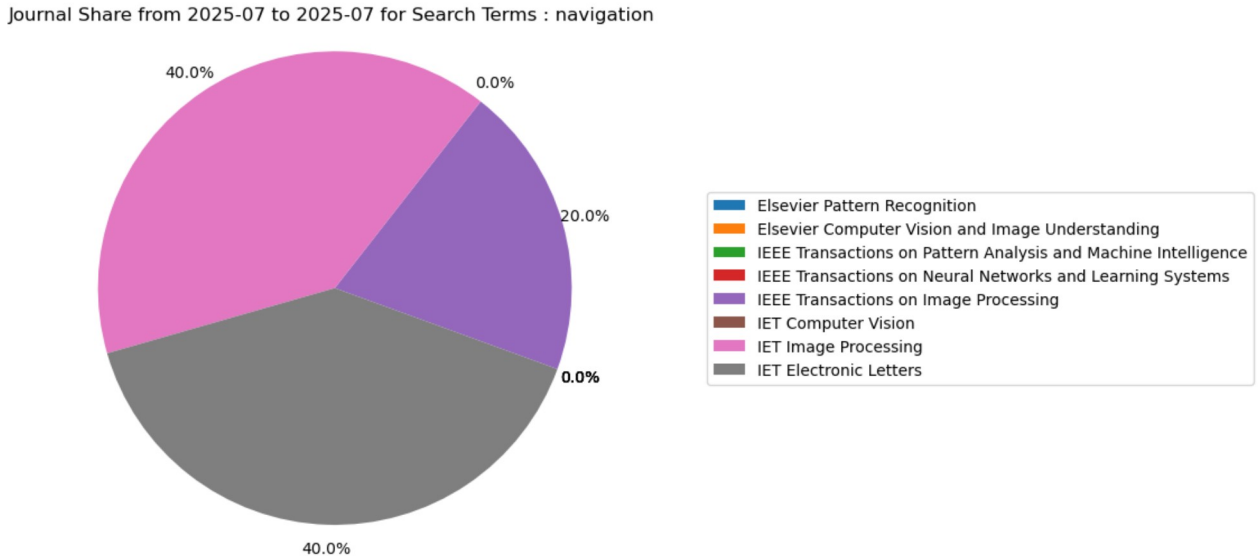
In [226…
```
show_search_words_hist("2024-07", "2025-07", ["navigation"])
```



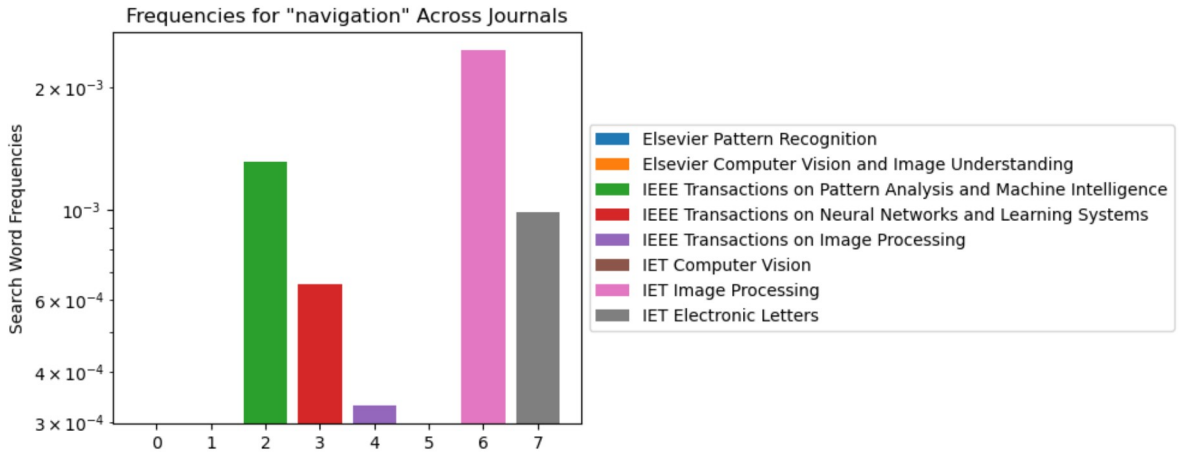
In [290…
```
show_search_words_pie_chart("2024-07", "2025-07", ["navigation"], -20)
```



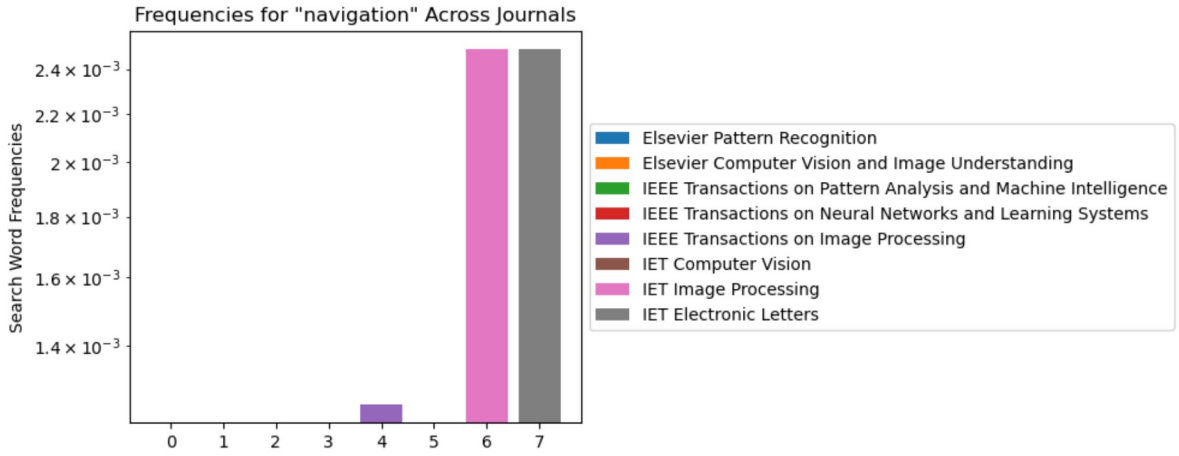
In [301…
```
show_search_words_pie_chart("2025-07", "2025-07", ["navigation"], -20)
```



In [352…
```
show_search_word_frequencies("2024-07", "2025-07", ["navigation"])
```

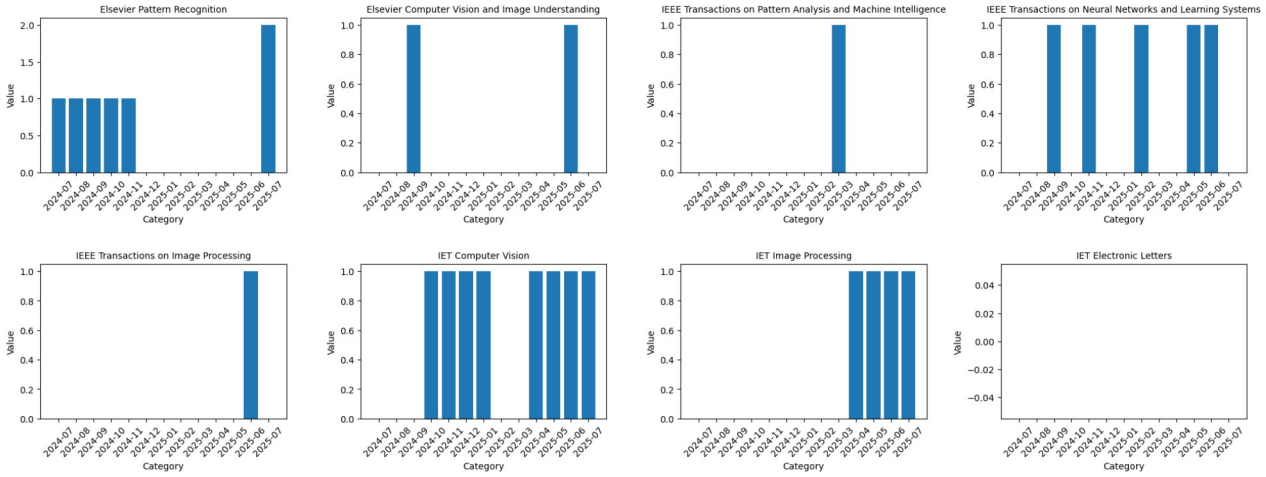


In [353…
```
show_search_word_frequencies("2025-07", "2025-07", ["navigation"])
```

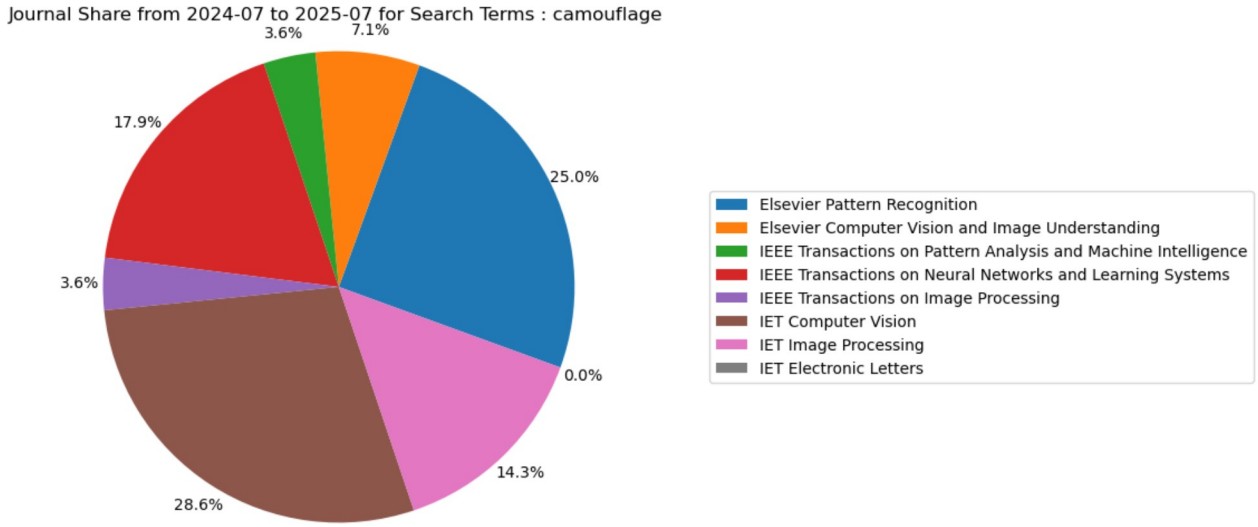


SEARCH WORD: CAMOUFLAGE

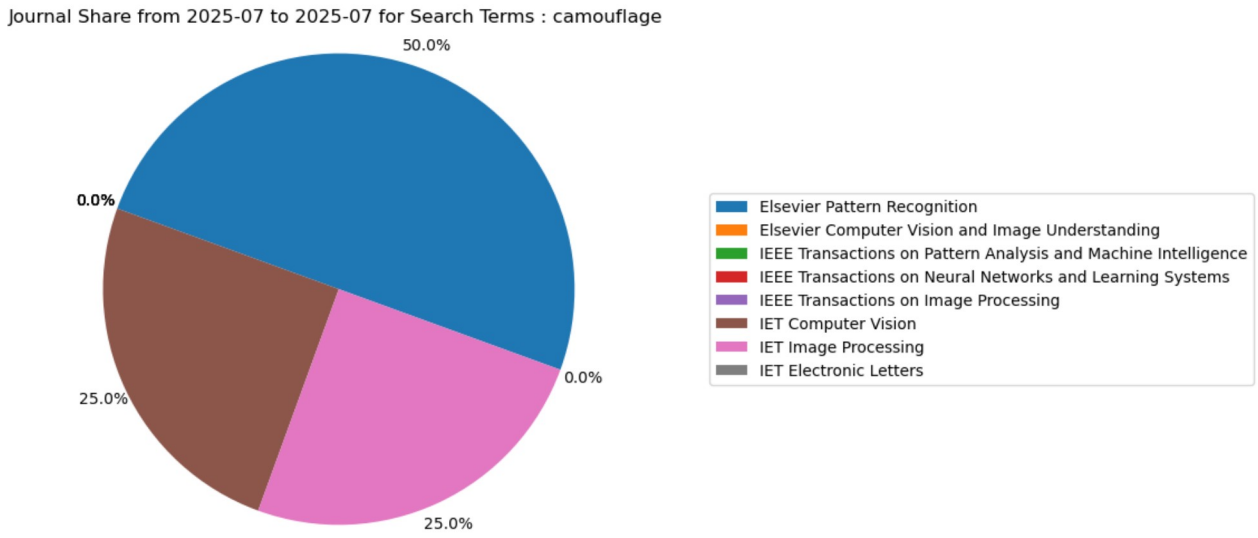
In [303…
```
show_search_words_hist("2024-07", "2025-07", ["camouflage"])
```



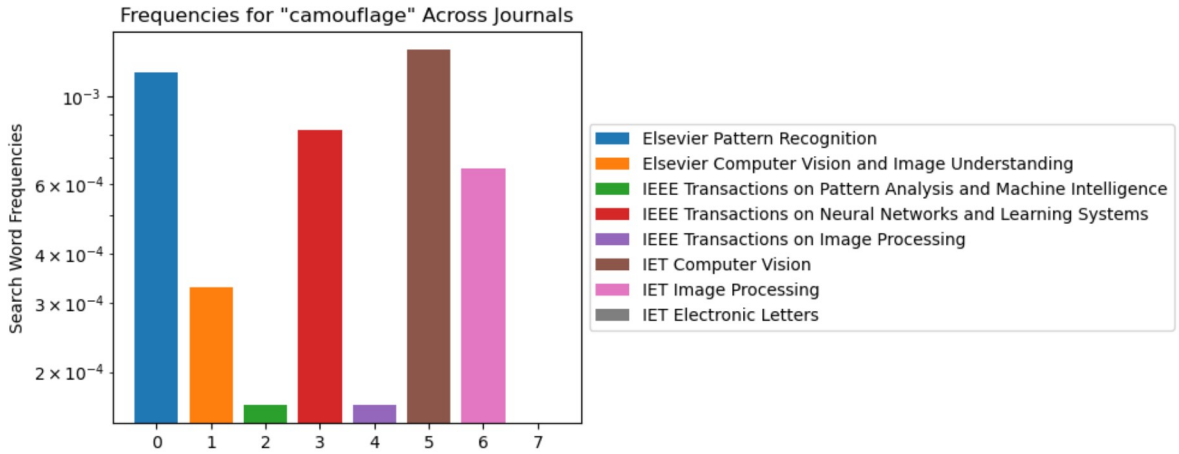
In [304…
```
show_search_words_pie_chart("2024-07", "2025-07", ["camouflage"], -20)
```



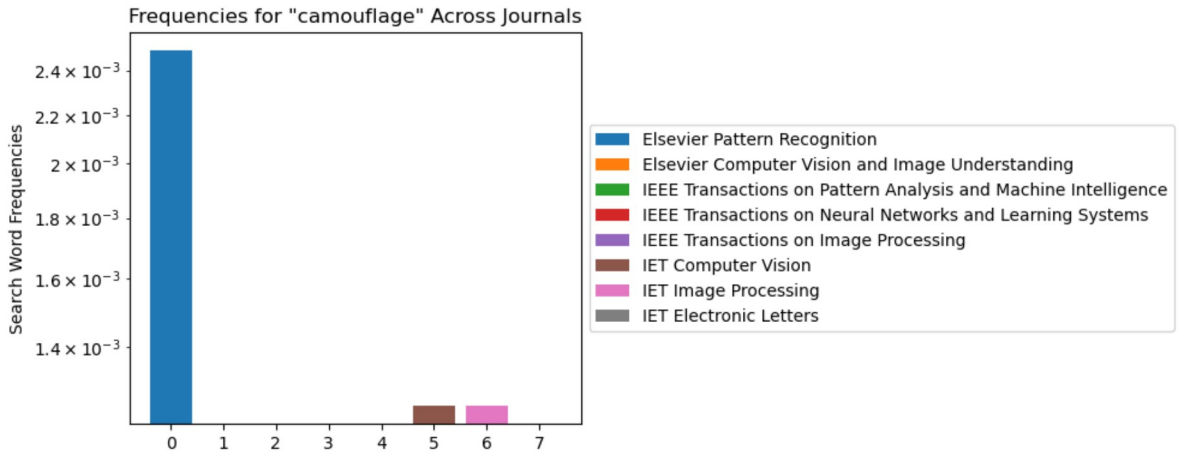
In [305…
```
show_search_words_pie_chart("2025-07", "2025-07", ["camouflage"], -20)
```



In [354…
```
show_search_word_frequencies("2024-07", "2025-07", ["camouflage"])
```

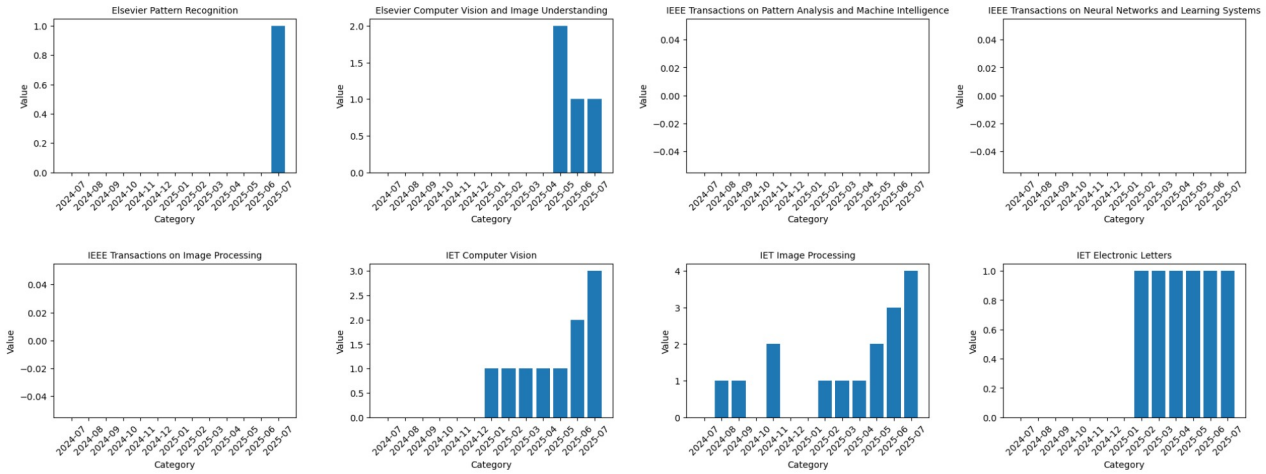


In [355…
```
show_search_word_frequencies("2025-07", "2025-07", ["camouflage"])
```

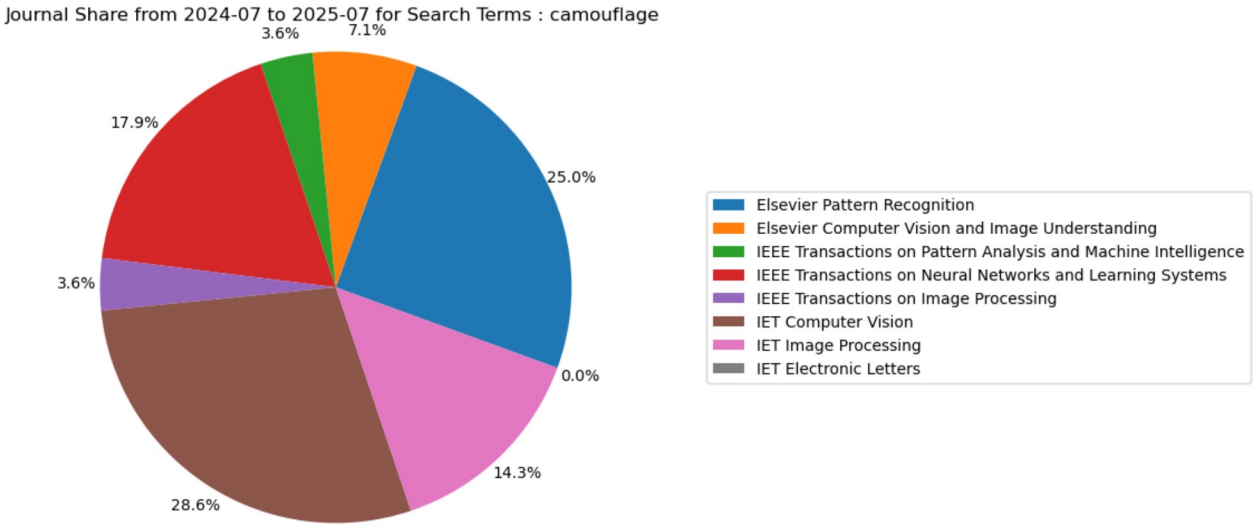


SEARCH WORD: MAMBA

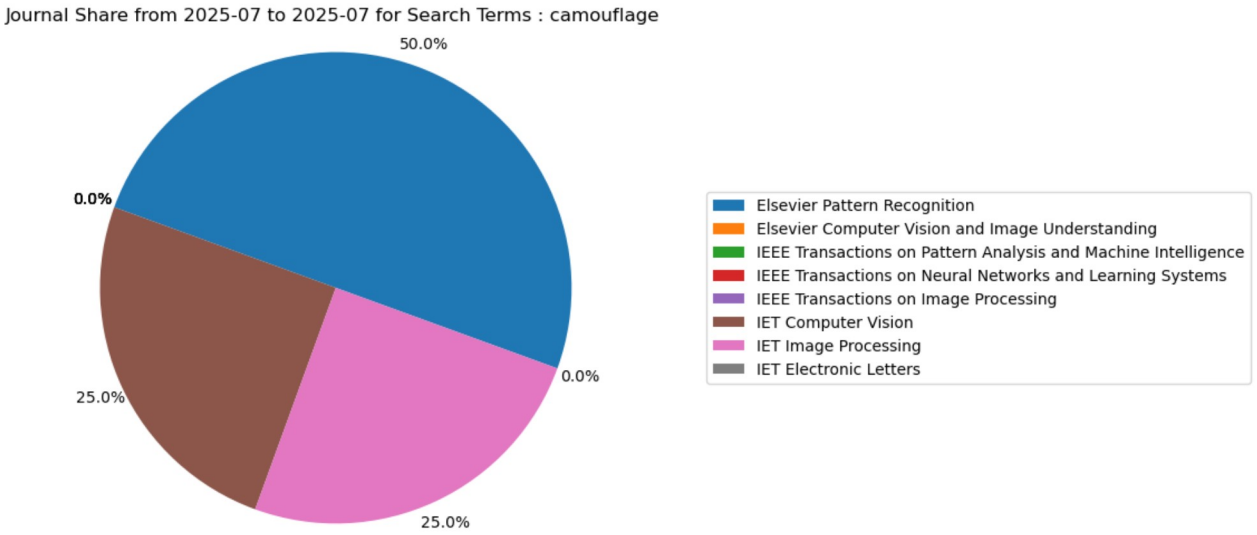
In [306…
```
show_search_words_hist("2024-07", "2025-07", ["mamba"])
```



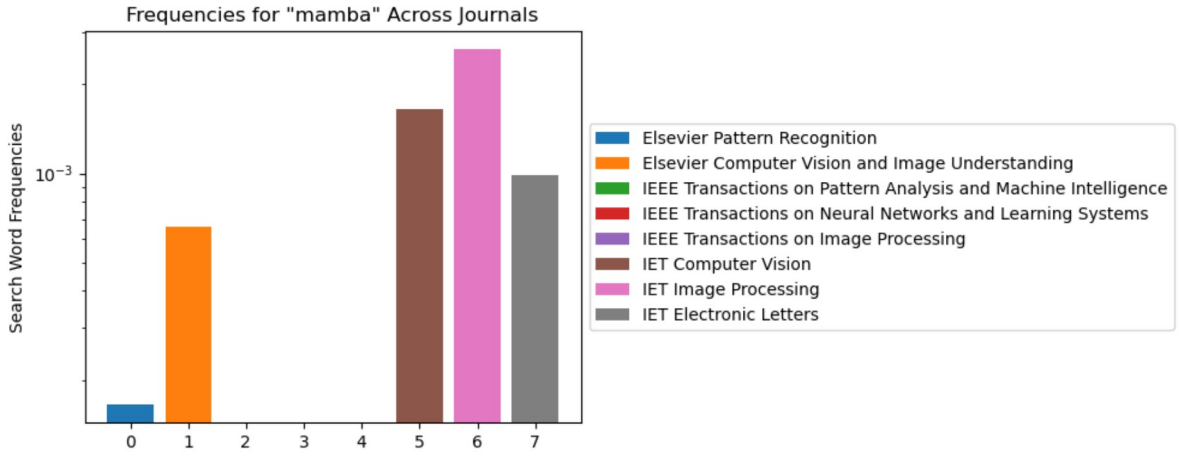
In [307…
```
show_search_words_pie_chart("2024-07", "2025-07", ["camouflage"], -20)
```



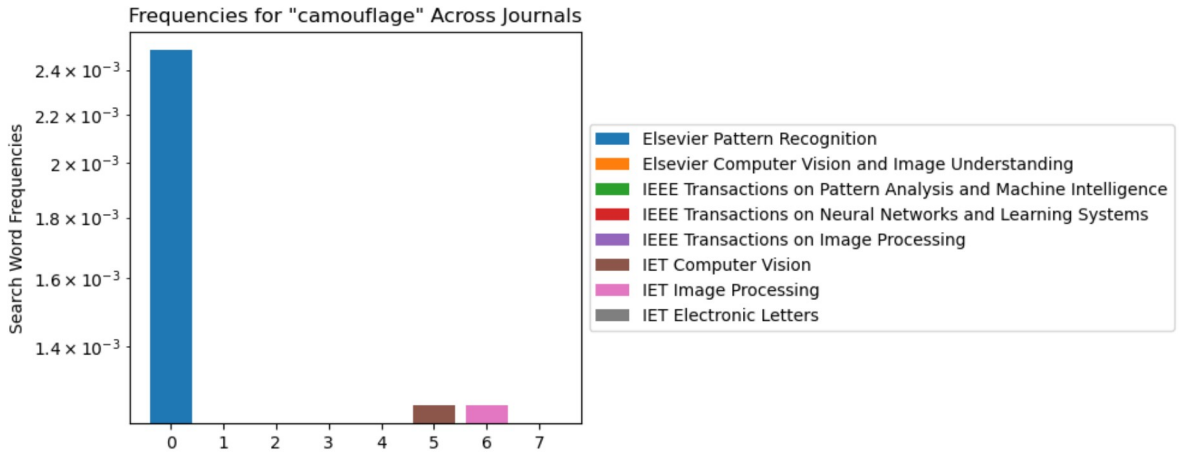
In [308…
```
show_search_words_pie_chart("2025-07", "2025-07", ["camouflage"], -20)
```



In [356…
```
show_search_word_frequencies("2024-07", "2025-07", ["mamba"])
```



In [357…
```
show_search_word_frequencies("2025-07", "2025-07", ["camouflage"])
```



TRENDING TOPICS ANALYSIS

In [10]:
```
text = ' '.join([re.sub(r"<[^>]*>", "", article['title']) for article in all_articles]).replace(":","").replace(",","").replace(".","")
```

In [11]:
```
from nltk import ngrams, FreqDist
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

stop_words = set(stopwords.words('english'))
stop_words.add('via')
stop_words.add('based')

JOURNAL_STOPWORDS = {'via', 'based', 'for', 'network', 'learning', 'video', 'image', 'model', 'feature', 'toward',
                     'towards', 'efficient', 'using', 'method', 'images', 'detection'}

stop_words = stop_words.union(JOURNAL_STOPWORDS)

word_tokens = word_tokenize(text.lower())
# converts the words in word_tokens to lower case and then checks whether
#they are present in stop_words or not
filtered_sentence = [w for w in word_tokens if not w.lower() in stop_words]
#with no lower case conversion
filtered_sentence = []

for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)

all_counts = dict()
for size in 1, 2, 3, 4, 5:
    all_counts[size] = FreqDist(ngrams(filtered_sentence, size))
```

In [15]:
```
def frequence_dictionary_for_date(input_date, n_gram_size=2):

    out_dict = {}

    articles_for_date = []

    for article in all_articles:
        out = article
        checked_article = check_article_date(input_date, article)
        if len(checked_article) > 0:
            articles_for_date+=checked_article

    text = ' '.join([re.sub(r"<[^>]*>", "", article['title']) for article in articles_for_date]).replace(":","").replace(",","").replace(".","")

    word_tokens = word_tokenize(text.lower())
    # converts the words in word_tokens to lower case and then checks whether
```

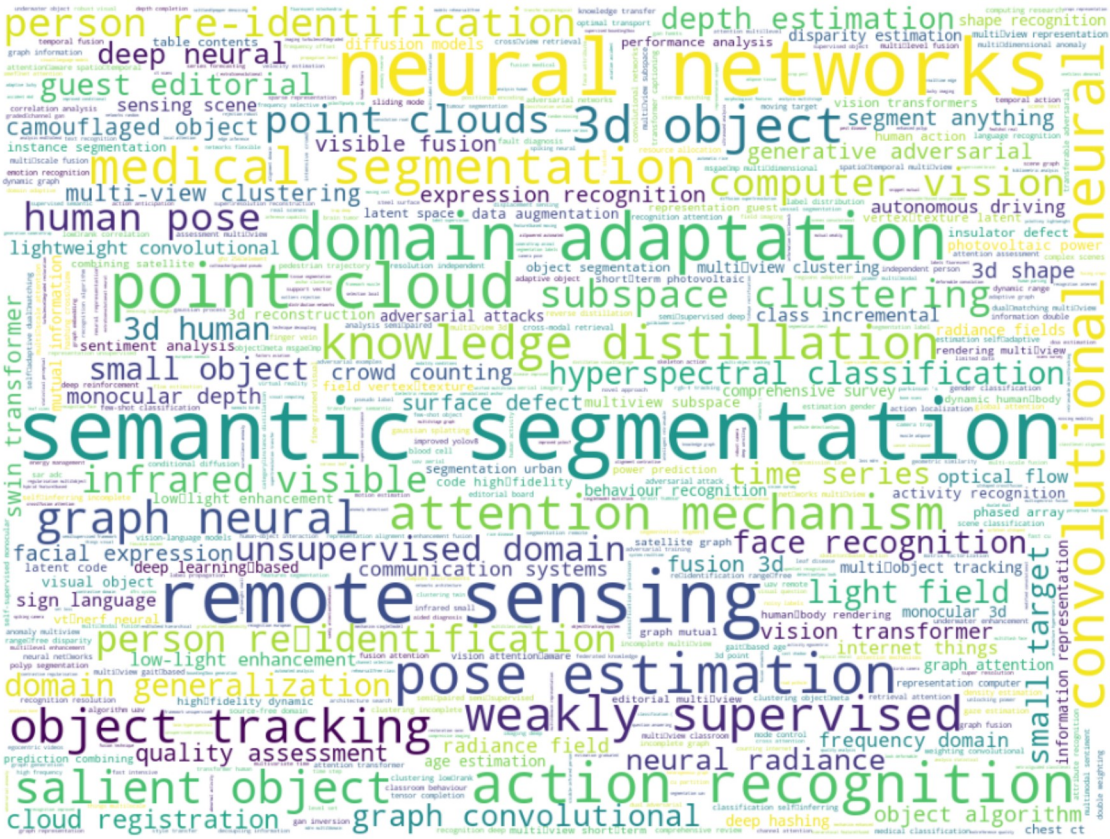
```
    #they are present in stop_words or not
    filtered_sentence = [w for w in word_tokens if not w.lower() in stop_words]
    #with no lower case conversion
    filtered_sentence = []

    for w in word_tokens:
        if w not in stop_words:
            filtered_sentence.append(w)

    all_counts = FreqDist(ngrams(filtered_sentence, n_gram_size)).most_common()

    for count in all_counts:
        out_dict[' '.join(count[0])] = count[1]

    return out_dict

def frequency_dictionaries_for_date_interval(start_date, end_date, n_gram_size=2):

    freq_date_dict = {}

    start = datetime.strptime(start_date, "%Y-%m")
    end = datetime.strptime(end_date, "%Y-%m")

    current = start

    # ITERATE THROUGH ALL DATES IN THE SPECIFIED RANGE

    while current <= end:

        date = current.strftime("%Y-%m")

        freq_dict = frequence_dictionary_for_date(date, n_gram_size)
        freq_date_dict[date] = freq_dict

        if current.month == 12:
            current = current.replace(year=current.year + 1, month=1)
        else:
            current = current.replace(month=current.month + 1)

    return freq_date_dict
```

In [400…

```
N_GRAM_SIZE = 2

word_with_freqs = {" ".join(n_gram[0]): n_gram[1] for n_gram in all_counts[N_GRAM_SIZE].most_common()}

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from matplotlib import pyplot as plt

#JOURNAL_STOPWORDS = {}
NEW_STOPWORDS = STOPWORDS.union(JOURNAL_STOPWORDS)
wc = WordCloud(stopwords=NEW_STOPWORDS, width=1024, height=768, max_words=3000, background_color='white')
wc = wc.generate_from_frequencies(word_with_freqs)
plt.figure(figsize = (12,9))
plt.axis("off")
plt.imshow(wc)
```

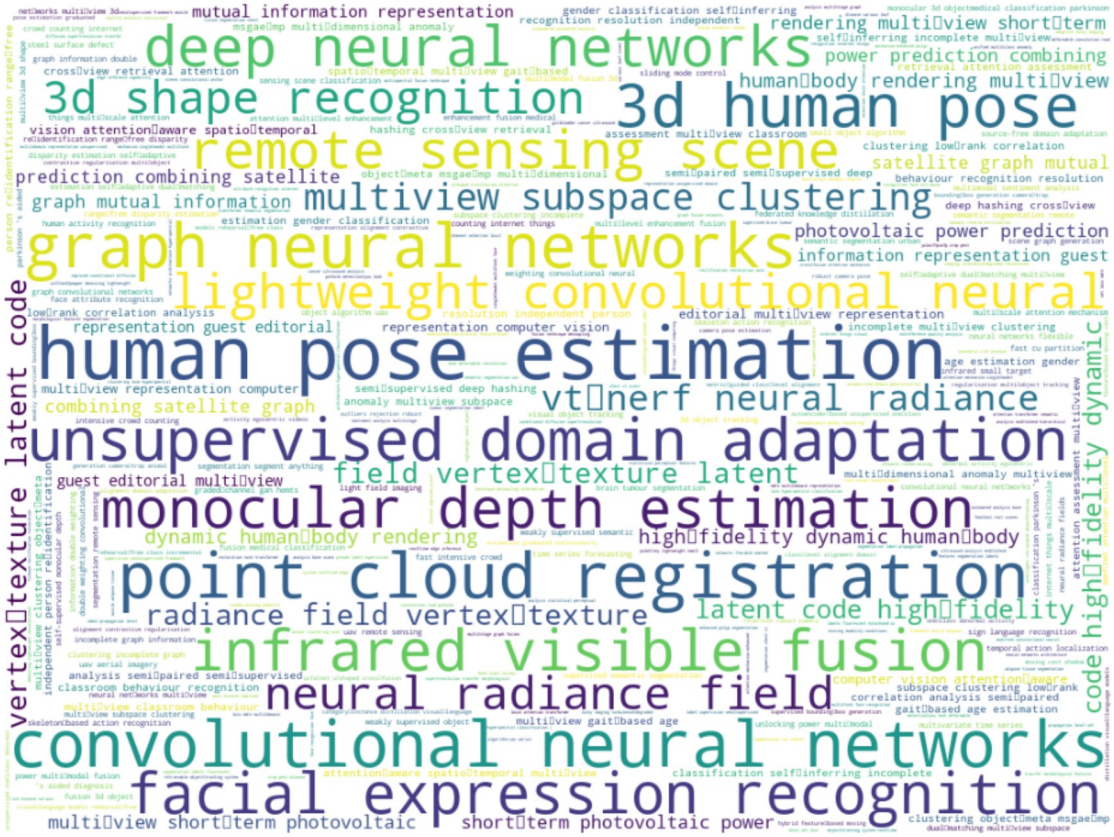
Out[400…

```
<matplotlib.image.AxesImage at 0x202103c5250>
```



In [401…
```
N_GRAM_SIZE = 3

word_with_freqs = {" ".join(n_gram[0]): n_gram[1] for n_gram in all_counts[N_GRAM_SIZE].most_common()}

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from matplotlib import pyplot as plt

#JOURNAL_STOPWORDS = {}
NEW_STOPWORDS = STOPWORDS.union(JOURNAL_STOPWORDS)
wc = WordCloud(stopwords=NEW_STOPWORDS, width=1024, height=768, max_words=3000, background_color='white')
wc = wc.generate_from_frequencies(word_with_freqs)
plt.figure(figsize = (12,9))
plt.axis("off")
plt.imshow(wc)
```

Out[401…
```
<matplotlib.image.AxesImage at 0x2021a6806b0>
```



In [12]:
```
topics_2gram = [' '.join(article_count[0]) for article_count in  all_counts[2].most_common()]
```

In [13]:
```
topics_over_time = {topic: [] for topic in topics_2gram}
```

In [16]:
```
freq_dict_date = frequency_dictionaries_for_date_interval("2024-07", "2025-07", 2)
```

In [17]:
```
start_date = "2024-07"
end_date = "2025-07"


start = datetime.strptime(start_date, "%Y-%m")
end = datetime.strptime(end_date, "%Y-%m")

current = start

# ITERATE THROUGH ALL DATES IN THE SPECIFIED RANGE

while current <= end:

    date = current.strftime("%Y-%m")

    for topic in topics_over_time.keys():
        if topic in freq_dict_date[date].keys():
            topics_over_time[topic].append(freq_dict_date[date][topic])
        else:
            topics_over_time[topic].append(0)

    if current.month == 12:
        current = current.replace(year=current.year + 1, month=1)
    else:
        current = current.replace(month=current.month + 1)
```

In [18]:
```
import numpy as np
strongest_growing_topics = {k: np.median(v) for k,v in topics_over_time.items()}

fastest_growing_topics = {k: np.median(np.clip(np.diff(v), 0, None)) for k,v in topics_over_time.items()}

def top_n_keys_by_value(d, n=5):
    return sorted(d.items(), key=lambda item: item[1], reverse=True)[:n]
```

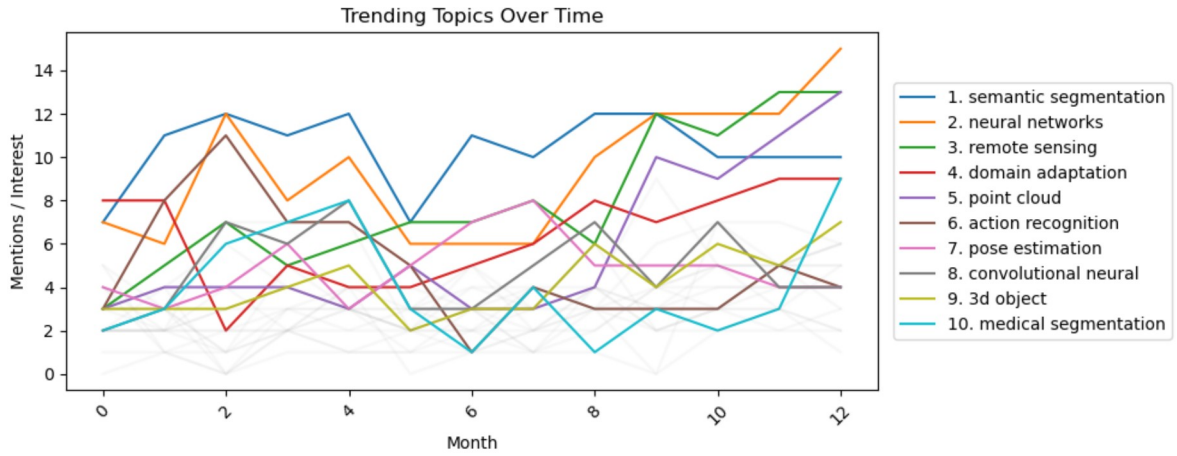
In [19]:
```
top_n_keys_by_value(strongest_growing_topics, 10)
```

Out[19]:
```
[('semantic segmentation', 11.0),
 ('neural networks', 10.0),
 ('remote sensing', 7.0),
 ('domain adaptation', 7.0),
 ('pose estimation', 5.0),
 ('point cloud', 4.0),
 ('action recognition', 4.0),
 ('convolutional neural', 4.0),
 ('3d object', 4.0),
 ('weakly supervised', 4.0)]
```

In [20]:
```
top_n_keys_by_value(fastest_growing_topics, 10)
```

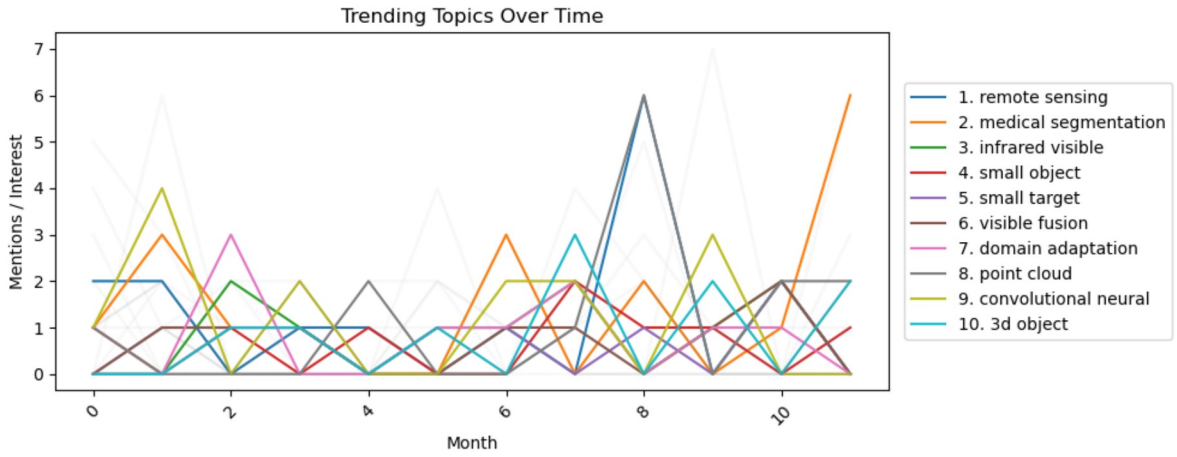
Out[20]:
```
[('remote sensing', 1.0),
 ('medical segmentation', 1.0),
 ('infrared visible', 1.0),
 ('small object', 1.0),
 ('small target', 1.0),
 ('visible fusion', 1.0),
 ('domain adaptation', 0.5),
 ('point cloud', 0.5),
 ('convolutional neural', 0.5),
 ('3d object', 0.5)]
```

In [36]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 4))
for e, (topic, values) in enumerate(topics_over_time.items()):
    if topic in list(topics_over_time.keys())[:10]:
        plt.plot(np.arange(len(values)), values, label=f"{1+e}. {topic}")
    if topic in list(topics_over_time.keys())[10:26]:
        plt.plot(np.arange(len(values)), values, color='grey', alpha=0.05)

plt.title("Trending Topics Over Time")
plt.xlabel("Month")
plt.ylabel("Mentions / Interest")
plt.xticks(rotation=45)
plt.legend(loc='center left', bbox_to_anchor=(1.02, 0.5), borderaxespad=0.)
plt.tight_layout()
plt.show()
```



In [47]:
```
fastest_growing_accs = {k:np.clip(np.diff(topics_over_time[k]),0, None) for k,v in top_n_keys_by_value(fastest_growing_topics, 100)}
```

In [48]:
```
plt.figure(figsize=(10, 4))
for e, (topic, values) in enumerate(fastest_growing_accs.items()):
    if topic in list(fastest_growing_accs.keys())[:10]:
        plt.plot(np.arange(len(values)), values, label=f"{1+e}. {topic}")
    if topic in list(fastest_growing_accs.keys())[10:26]:
        plt.plot(np.arange(len(values)), values, color='grey', alpha=0.05)

plt.title("Trending Topics Over Time")
plt.xlabel("Month")
plt.ylabel("Mentions / Interest")
plt.xticks(rotation=45)
plt.legend(loc='center left', bbox_to_anchor=(1.02, 0.5), borderaxespad=0.)
plt.tight_layout()
plt.show()
```



In [ ]: