CS1470/2470 HW1: KNN (with CIFAR)

In this homework assignment, you will experience the overall machine learning process from start to end by implementing your own version of the **k-Nearest Neighbors** algorithm.

In [1]:
```
!python -VV
```

```
Python 3.9.12 (main, Apr  5 2022, 01:52:34)
[Clang 12.0.0 ]
```

If you are running the notebook on Colab, you need to mount your drive or repo. An example of these is provided here.

In [2]:
```
import os
import sys

## Path to data
data_path = "data"
kitten_path = "kitten.jpg"

## Make sure the data is downloaded appropriately
![ ! -d "$data_path" ] && cd .. && bash download.sh && cd code
```

In [3]:
```
%load_ext autoreload
%autoreload 1
%aimport KNN_Model, preprocess
from ResNetWrapper import ResNetWrapper

import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np

# ensures that we run only on cpu
# this environment variable is not permanent
# it is valid only for this session
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"
```

Preprocessing

Data Preparation

In a machine learning project, you need a separate train set and a test set. Sometimes, you also need a validation set to fit hyperparameters, but for this homework assignment, we are not going to use a validation set.

Code Block #1: Preprocessing

1. Unpickle the CIFAR files and load the full train and test datasets by using the function `get_data_CIFAR` in `preprocess.py`.
2. Shuffle the full datasets with your favorite random seed by using the function `shuffle_data` in `preprocess.py`. Please use the same random seed for both train and test sets.
3. Keep only a small subset of the full datasets by using the function `get_subset` in `preprocess.py`.
   - For the train set, keep only 100 images and labels for each class, so that your train image array should have the shape (1000, 32, 32, 3), and your train label array should have the shape (1000,)
   - For the test set, 25 images and labels for each class, so the shapes are (250, 32, 32, 3), and (250,).
   - The variable names of the test and train image arrays must be `image_train_uint` and `image_test_uint`.
4. Normalize the image arrays by dividing them with 255.0, convert the data type to np.float32, and flatten the images.
   - However, DO NOT throw away the np.uint8 images from TODO #3, because we need them for the ResNet.
   - The final train image array should have the shape (1000, 3072), and the final test image array (250, 3072).

In [4]:
```
%aimport preprocess
from preprocess import *

# TODO #1:
#   Unpickle the CIFAR files and load the full train and test datasets
#   by using the function unpickle_CIFAR in preprocess.py.
image_train_full, label_train_full, cifar_class_list = get_data_CIFAR('train
image_test_full,  label_test_full,  _  = get_data_CIFAR('test', data_path)

# TODO #2:
#   Shuffle the full datasets with your favorite random seed
#   by using the function shuffle_data in preprocess.py.
#   Please use the same random seed for both train and test sets.
seed = 12
image_train_full, label_train_full = shuffle_data(image_train_full, label_tr
image_test_full,  label_test_full  = shuffle_data(image_test_full, label_tes

# TODO #3:
#   Keep only a small subset of the full datasets by using the function
#     get_subset in preprocess.py.
#   For the train set, keep only 100 images and labels for each class,
#     so that your train image array should have the shape (1000, 32, 32, 3)
#     and your train label array should have the shape (1000,)
#   For the test set, 25 images and labels for each class,
#     so the shapes are (250, 32, 32, 3), and (250,)
#   The variable names of the test and train image arrays must be
#     "image_train_uint" and "image_test_uint"


# image_train_uint, label_train = get_subset(image_train_full, label_train_f
# image_test_uint,  label_test  = get_subset(image_test_full, label_test_ful

image_train_uint, label_train = image_train_full[:1000], label_train_full[:2
image_test_uint,  label_test  = image_test_full[:1000],  label_test_full[:25

    #I ran out of time and skipped the shuffling part and just selected the

# TODO #4:
#   Normalize the image arrays by dividing them with 255.0,
#     convert the data type to np.float32,
#     and flatten the images.
#   However, DO NOT throw away the np.uint8 images from TODO #3,
#     because we need them for the ResNet.
#   The final train image array should have the shape (1000, 3072),
#     and the final test image array (250, 3072).
image_train = np.float32(image_train_full[:1000,:,:,:]).transpose(0,3,2,1).r
image_test  = np.float32(image_test_full[:250]).transpose(0,3,2,1).reshape(-

print(image_train.shape)

label_train=label_train_full[:1000]
label_test=label_test_full[:250]
```

```
(1000, 3072)
```

Data Visualization

In [5]:
```
indices_to_inspect = range(0, 1000, 100)

fig, ax = plt.subplots(1, 10)
fig.set_size_inches(12, 1.2)

for i, each_image in enumerate(indices_to_inspect):
    ax[i].imshow(image_train[each_image].reshape(3,32,32).transpose(2,1,0))
    ax[i].tick_params(left=False)
    ax[i].tick_params(bottom=False)
    ax[i].tick_params(labelleft=False)
    ax[i].tick_params(labelbottom=False)
    ax[i].set_xlabel(f"{each_image}")
    ax[i].set_title(f"{label_train[each_image]}")
```

KNN

Model Building

Now it's time to make your own implementation of the k-Nearest Neighbors algorithm.

Code Block #2: Building the model

Create a KNN model, or an instance of the class KNN_Model and fit it with the train dataset.

- Although, `k_neighbors` can be any integer in theory, keep `k_neighbors == 9` in this homework assignment.
- The name of the KNN_Model instance must be `model_cifar`, so that you can run the following Code Blocks without trouble.

In [6]:
```
from KNN_Model import KNN_Model

k_neighbors=9
class_list=np.array(cifar_class_list)
model_cifar = KNN_Model(class_list,k_neighbors)
model_cifar.fit(image_train,label_train)
```

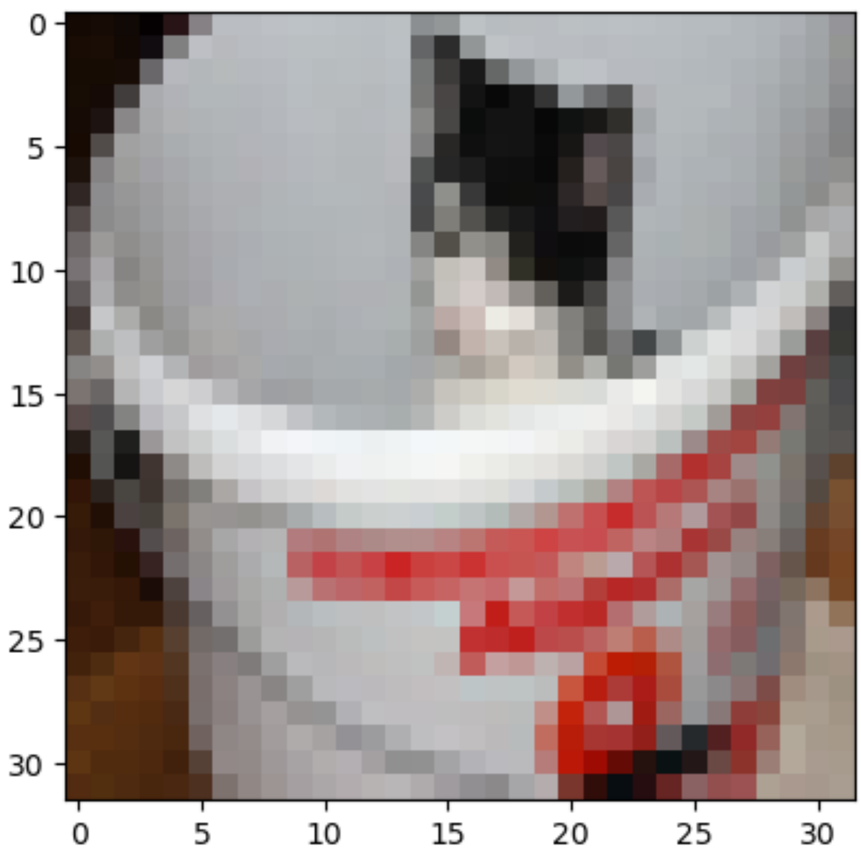
Model Visualization

Code Block #3: Interacting with the model

In [7]:
```
## Pull in a specific image
sample_image = image_test[88].copy().reshape(3,32,32).transpose(2,1,0)
## TODO: Show what the image looks like using plt.imshow
print(sample_image.shape)
plt.imshow(sample_image)
## Make sure to title it using plt.title
```

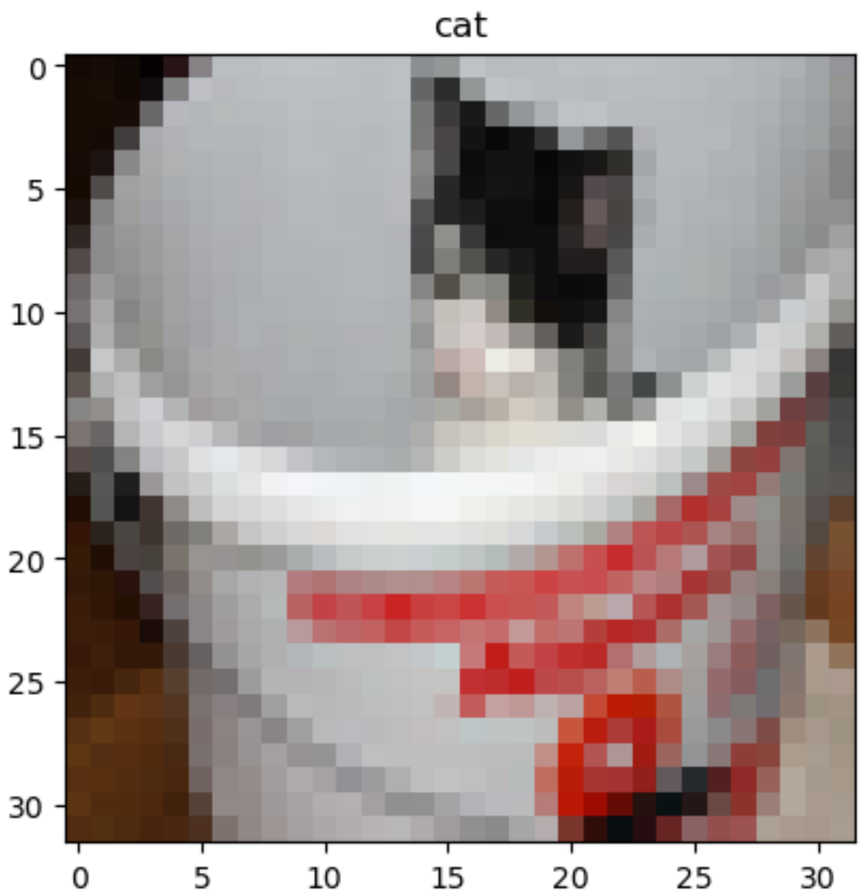
```
(32, 32, 3)
```

Out[7]:
```
<matplotlib.image.AxesImage at 0x16dfe2400>
```



In [8]:
```
## TODO: Show what the image looks like using plt.imshow
plt.imshow(sample_image)
plt.title(label_test[88])
## Make sure to title it using plt.title
```

Out[8]:
```
Text(0.5, 1.0, 'cat')
```



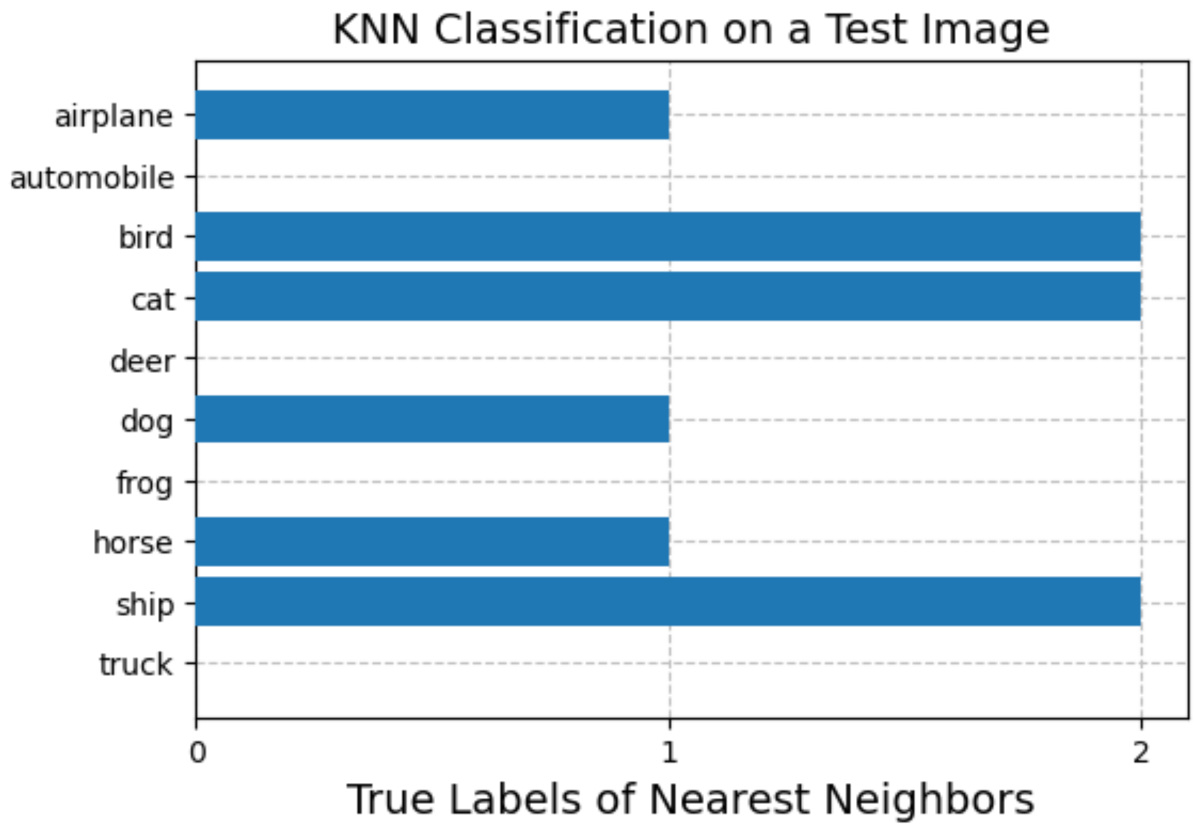
In [9]:
```
## TODO: Figure out the closest k neighbors based on the model.
class_counts, nearest_indices = model_cifar.get_neighbor_counts(image_test[8
print(class_counts)
```

```
[1 0 2 2 0 1 0 1 2 0]
```

In [10]:
```
fig_knn, ax_knn = plt.subplots()

ax_knn.barh(y=cifar_class_list, width=class_counts, zorder=100)
ax_knn.invert_yaxis()
ax_knn.set_xticks(np.arange(1 + np.max(class_counts)))
ax_knn.set_yticks(cifar_class_list)
ax_knn.set_title("KNN Classification on a Test Image", fontsize=14)
ax_knn.set_xlabel("True Labels of Nearest Neighbors", fontsize=14)
ax_knn.grid(linestyle="dashed", color="#bfbfbf", zorder=-100)
fig_knn.set_size_inches([6, 4])

## You can also save the figure in the pdf, png, and svg formats
# fig.savefig(f"KNN_Test_Image_CIFAR.png", dpi=300, bbox_inches="tight")
```



In [11]:
```
fig_nearest, ax_nearest = plt.subplots(3, 3, figsize=(4.5, 4.5))

for each_ax, each_neighbor in zip(ax_nearest.flat, nearest_indices):
    each_ax.imshow(model_cifar.image_train[each_neighbor].reshape(3,32, 32).
    each_ax.tick_params(bottom=False, left=False, labelbottom=False, labelle
    each_ax.text(2, 5, model_cifar.label_train[each_neighbor],
        fontsize=8, bbox = dict(color="White", alpha=0.75))
    fig_nearest.suptitle("Nearest Images", y = 0.95)
```

Evaluation

It is time to evaluate the model.

Overall Accuracy

Code Block #4: Overall accuracy

1. Get predictions on every image in the test dataset.
2. Calculate and print out the overall accuracy of the model, which is defined as the number of correct predictions divided by the number of all predictions.

In [12]:
```
%%time
## TODO: Get the accuracy on the test dataset
prediction_array=[]
Correct_Count=0
for i in range(label_test.shape[0]):
    prediction_array.append(model_cifar.predict(image_test[i]))
    if prediction_array[i]==label_test[i]:
        Correct_Count+=1


prediction_acc = Correct_Count/label_test.shape[0]

print(f"accuracy = {prediction_acc}")
```

```
accuracy = 0.232
CPU times: user 335 ms, sys: 2.28 ms, total: 337 ms
Wall time: 336 ms
```

Confusion Matrix

Code Block #5: Confusion matrix

In [13]:
```
# TODO: Get the confusion matrix (hint: see KNN_ConfMtx)
confusion_mat = model_cifar.get_confusion_matrix(label_test, prediction_arra
print(confusion_mat)
```

```
[[10  0  7  0  9  0  0  0  7  1]
 [ 2  1  4  1  4  0  1  0  9  1]
 [ 1  0  7  1  8  0  1  1  2  0]
 [ 1  0 10  2 10  0  4  0  1  2]
 [ 2  0  7  0 14  0  1  3  1  0]
 [ 2  0  6  1  5  2  4  1  0  0]
 [ 0  0  5  0  7  0  4  1  0  0]
 [ 6  0  3  1 10  0  2  2  1  1]
 [ 1  0  3  1  2  0  2  0 12  0]
 [ 5  0  2  2  6  0  3  4  3  4]]
```

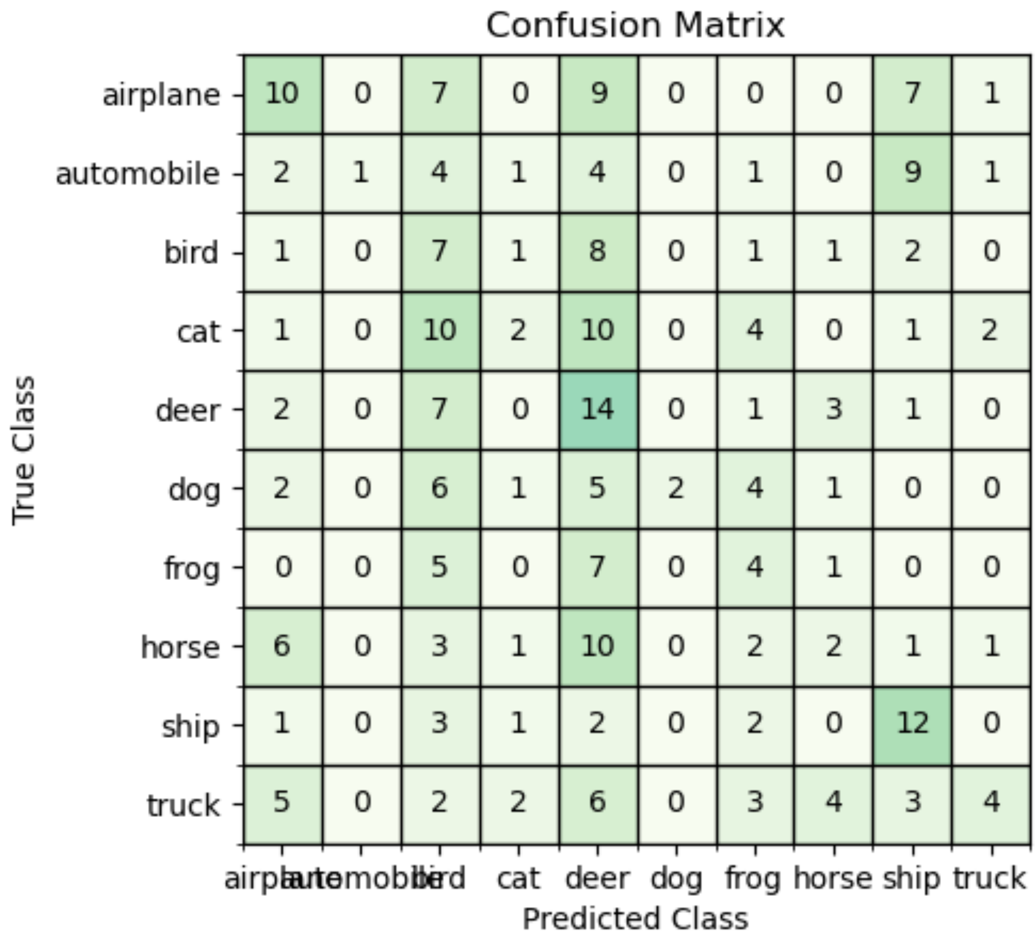
In [14]:
```
fig_confusion1, ax_confusion1 = model_cifar.visualize_confusion_matrix(confu
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [14], line 1
----> 1 fig_confusion1, ax_confusion1 = model_cifar.visualize_confusion_matr
ix(confusion_mat)

File ~/Documents/GitHub/hw1-knn-JungKyle/hw1/code/KNN_ConfMtx.py:72, in KNN_
ConfMtx.visualize_confusion_matrix(self, confusion_mat)
     64 for each_true_index, each_pred_index \
     65     in itertools.product(range(num_classes), range(num_classes)):
     66     ax_confusion.text(each_pred_index,
     67                       each_true_index,
     68                       confusion_mat[each_true_index, each_pred_index
],
     69                       ha = "center",
     70                       va = "center")
---> 72 fig_confusion.colorbar(
     73     mpl.cm.ScalarMappable(norm=norm, cmap=cmap),
     74     orientation="vertical",
     75     label="Counts",
     76     shrink = 0.83)
     78 fig_confusion.set_size_inches([8, 8])
     80 return fig_confusion, ax_confusion

File /opt/miniconda3/envs/tensorflow/lib/python3.9/site-packages/matplotlib/
figure.py:1256, in FigureBase.colorbar(self, mappable, cax, ax, use_gridspec
, **kwargs)
   1254 if cax is None:
   1255     if ax is None:
-> 1256         raise ValueError(
   1257             'Unable to determine Axes to steal space for Colorbar. '
   1258             'Either provide the *cax* argument to use as the Axes fo
r '
   1259             'the Colorbar, provide the *ax* argument to steal space 
'
   1260             'from it, or add *mappable* to an Axes.')
   1261     current_ax = self.gca()
   1262     userax = False

ValueError: Unable to determine Axes to steal space for Colorbar. Either pro
vide the *cax* argument to use as the Axes for the Colorbar, provide the *ax
* argument to steal space from it, or add *mappable* to an Axes.
```



Pretrained ResNet50

KNN on ResNet Embeddings

Code Block #6: More preprocessing for ResNet50

From now on, use `image_train_embeddings` instead of `image_train`, and use `image_test_embeddings` instead of `image_test`.

In [15]:
```
%%time

rs_wrapper = ResNetWrapper()

image_train_resnet = rs_wrapper.preprocess_image(image_train_uint)
image_test_resnet  = rs_wrapper.preprocess_image(image_test_uint)

image_train_embeddings = rs_wrapper.get_resnet_embeddings(image_train_resnet
image_test_embeddings  = rs_wrapper.get_resnet_embeddings(image_test_resnet)
```

```
Metal device set to: Apple M1 Pro
```

```
2022-09-29 22:46:43.678636: I tensorflow/core/common_runtime/pluggable_devic
e/pluggable_device_factory.cc:306] Could not identify NUMA node of platform
GPU ID 0, defaulting to 0. Your kernel may not have been built with NUMA sup
port.
2022-09-29 22:46:43.678763: I tensorflow/core/common_runtime/pluggable_devic
e/pluggable_device_factory.cc:272] Created TensorFlow device (/job:localhost
/replica:0/task:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice
(device: 0, name: METAL, pci bus id: <undefined>)
  0%|          | 0/100 [00:00<?, ?it/s]
  0%|          | 0/100 [00:00<?, ?it/s]
CPU times: user 8.23 s, sys: 1.81 s, total: 10 s
Wall time: 13 s
```

Code Block #7: Building the model again, because ResNet

Create a KNN model, or an instance of the class KNN_Model and fit it with the train dataset.

- Although, `k_neighbors` can be any integer in theory, keep `k_neighbors == 9` in this homework assignment.
- The name of the KNN_Model instance must be `model_resnet`, so that you can run the following Code Blocks without trouble.

In [16]:
```
## TODO: Train the KNN Model on the ResNet embeddings of the trained data
from KNN_Model import KNN_Model

k_neighbors=9
class_list=np.array(cifar_class_list)
model_resnet = KNN_Model(class_list,k_neighbors)
model_resnet.fit(image_train_embeddings,label_train)
```

Code Block #8: Overall accuracy of KNN + ResNet

1. Get predictions on every image embeddings in the test dataset.
2. Calculate and print out the overall accuracy of the model, which is defined as the number of correct predictions divided by the number of all predictions.

In [17]:
```
%%time
## TODO: Get testing accuracy on model working with ResNet embeddings
prediction_array2=[]
Correct_Count2=0
for k in range(label_test.shape[0]):
    prediction_array2.append(model_resnet.predict(image_test_embeddings[k]))
    if prediction_array2[k]==label_test[k]:
        Correct_Count2+=1







prediction_acc2 = Correct_Count2/label_test.shape[0]

print(f"accuracy = {prediction_acc2}")
```

```
accuracy = 0.58
CPU times: user 11.1 s, sys: 7.91 s, total: 19 s
Wall time: 19 s
```

Code Block #9: Confusion matrix of KNN + ResNet

In [18]:
```
## TODO: Compute the confusion matrix for this new model as before
confusion_mat2 = model_cifar.get_confusion_matrix(label_test, prediction_arr
print(confusion_mat2)
```

```
[[12  0  3  0  0  0  5  0  9  5]
 [ 0 20  0  0  0  0  1  0  0  2]
 [ 0  0  9  0  1  0 11  0  0  0]
 [ 0  0  0  3  2  0 22  0  0  3]
 [ 1  0  1  0 22  0  4  0  0  0]
 [ 0  1  0  1  1  7 11  0  0  0]
 [ 0  0  0  0  0  0 16  0  0  1]
 [ 0  0  0  0  5  0  5 14  1  1]
 [ 0  0  0  0  0  0  3  0 18  0]
 [ 0  3  0  0  0  0  2  0  0 24]]
```

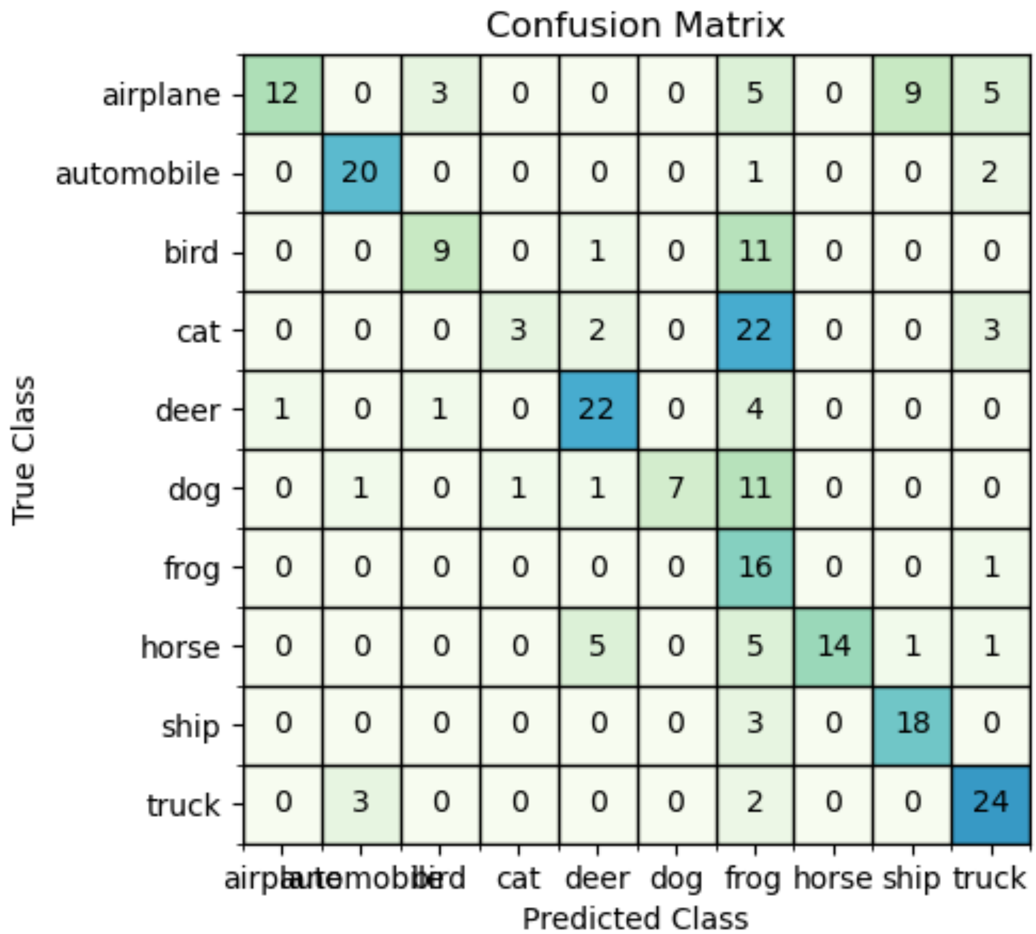
In [19]:
```
fig_confusion2, ax_confusion2 = model_resnet.visualize_confusion_matrix(conf
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [19], line 1
----> 1 fig_confusion2, ax_confusion2 = model_resnet.visualize_confusion_mat
rix(confusion_mat2)

File ~/Documents/GitHub/hw1-knn-JungKyle/hw1/code/KNN_ConfMtx.py:72, in KNN_
ConfMtx.visualize_confusion_matrix(self, confusion_mat)
     64 for each_true_index, each_pred_index \
     65     in itertools.product(range(num_classes), range(num_classes)):
     66     ax_confusion.text(each_pred_index,
     67                       each_true_index,
     68                       confusion_mat[each_true_index, each_pred_index
],
     69                       ha = "center",
     70                       va = "center")
---> 72 fig_confusion.colorbar(
     73     mpl.cm.ScalarMappable(norm=norm, cmap=cmap),
     74     orientation="vertical",
     75     label="Counts",
     76     shrink = 0.83)
     78 fig_confusion.set_size_inches([8, 8])
     80 return fig_confusion, ax_confusion

File /opt/miniconda3/envs/tensorflow/lib/python3.9/site-packages/matplotlib/
figure.py:1256, in FigureBase.colorbar(self, mappable, cax, ax, use_gridspec
, **kwargs)
   1254 if cax is None:
   1255     if ax is None:
-> 1256         raise ValueError(
   1257             'Unable to determine Axes to steal space for Colorbar. '
   1258             'Either provide the *cax* argument to use as the Axes fo
r '
   1259             'the Colorbar, provide the *ax* argument to steal space
'
   1260             'from it, or add *mappable* to an Axes.')
   1261     current_ax = self.gca()
   1262     userax = False

ValueError: Unable to determine Axes to steal space for Colorbar. Either pro
vide the *cax* argument to use as the Axes for the Colorbar, provide the *ax
* argument to steal space from it, or add *mappable* to an Axes.
```

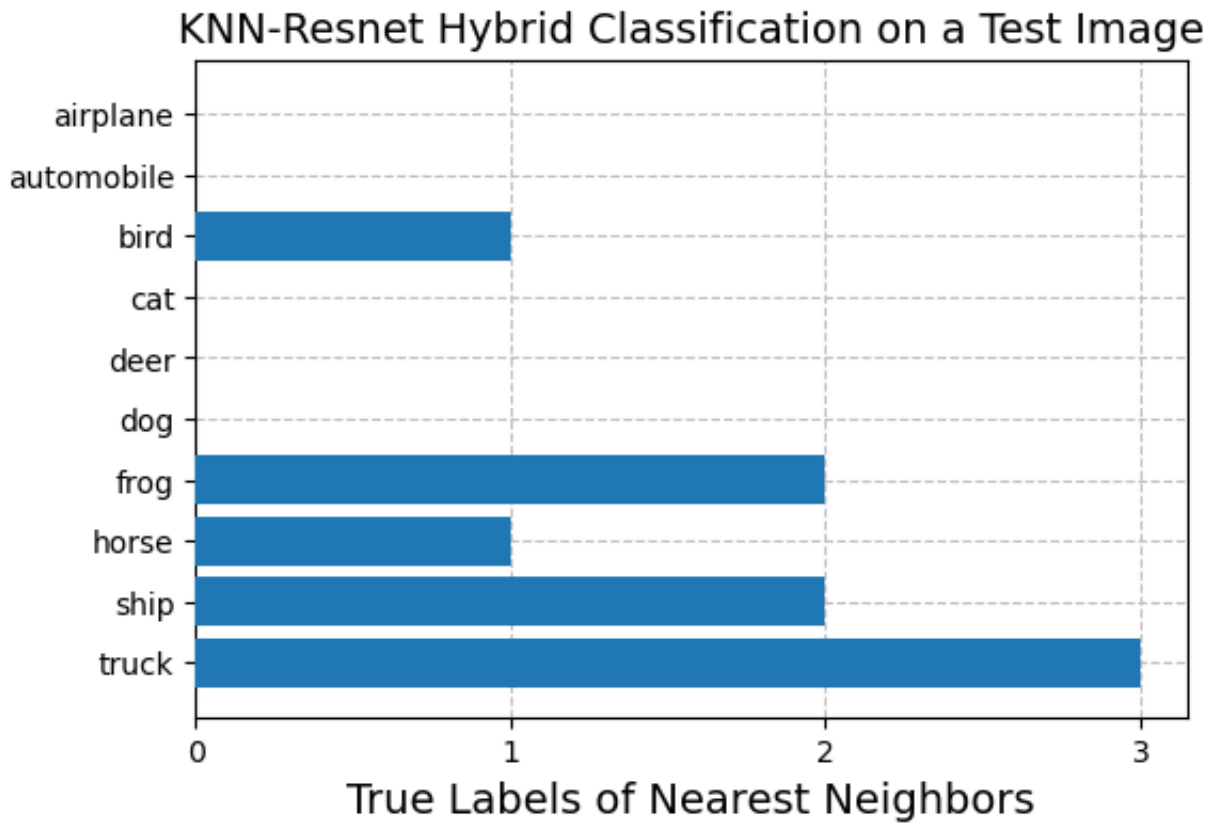


In [20]:
```
class_counts, nearest_indices = model_resnet.get_neighbor_counts(image_test_
print(class_counts)
```

```
[0 0 1 0 0 0 2 1 2 3]
```

In [21]:
```
fig_resnet, ax_resnet = plt.subplots()

class_counts, nearest_indices = model_resnet.get_neighbor_counts(image_test_
ax_resnet.barh(y=cifar_class_list, width=class_counts, zorder=100)
ax_resnet.invert_yaxis()
ax_resnet.set_xticks(np.arange(1 + np.max(class_counts)))
ax_resnet.set_yticks(cifar_class_list)
ax_resnet.set_title("KNN-Resnet Hybrid Classification on a Test Image", font
ax_resnet.set_xlabel("True Labels of Nearest Neighbors", fontsize = 14)
ax_resnet.grid(linestyle="dashed", color="#bfbfbf", zorder= -100)
fig_resnet.set_size_inches([6, 4])
```



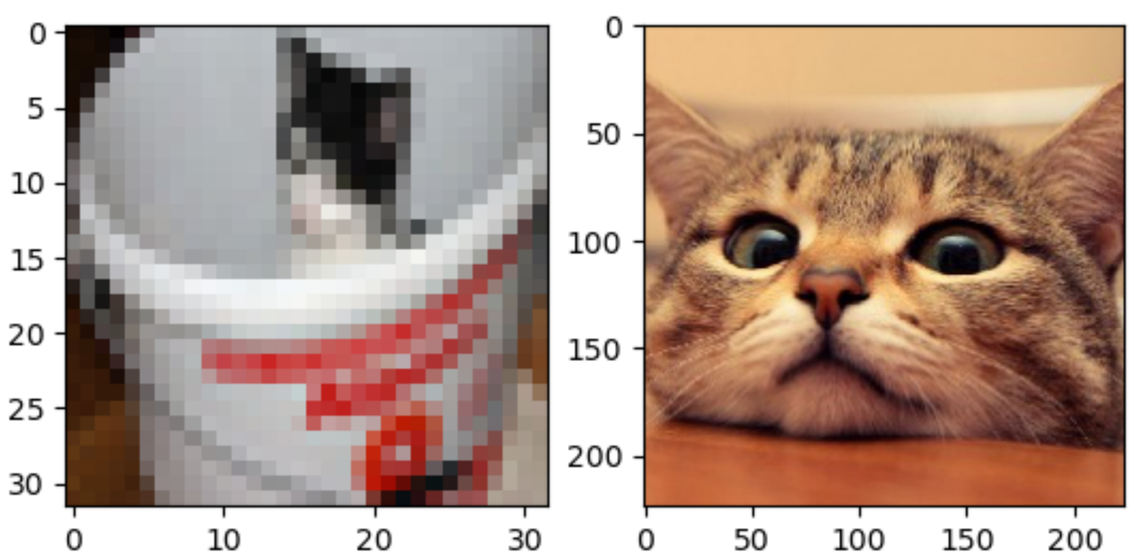
In [22]:
```
fig_nearest2, ax_nearest2 = plt.subplots(3, 3, figsize=(4.5, 4.5))

for each_ax, each_neighbor in zip(ax_nearest2.flat, nearest_indices):
    each_ax.imshow(image_train_uint[each_neighbor], cmap="Greys")
    each_ax.tick_params(bottom=False, left=False, labelbottom=False, labelle
    each_ax.text(2, 5, label_train[each_neighbor],
        fontsize=8, bbox = dict(color="White", alpha=0.75))
    fig_nearest2.suptitle("Nearest Images", y = 0.95)
```

Full ResNet Model

In [23]:
```
fig_compare, ax_compare = plt.subplots(1, 2)
ax_compare[0].imshow(image_test_uint[88])
ax_compare[1].imshow(plt.imread(kitten_path))
```

Out[23]: <matplotlib.image.AxesImage at 0x28dfd4520>



Code Block #10: Full power of the ResNet50 model

In [24]:
```
print(rs_wrapper.get_full_model_predictions(image_test_resnet[88]))

kitten_image_full = plt.imread(kitten_path)
kitten_image_full = np.array(kitten_image_full)

## TODO: Get the ResNet model predictions on the kitten image above
rs_wrapper.get_full_model_predictions(kitten_image_full)
```

```
[[('n03843555', 'oil_filter', 0.25348353), ('n02909870', 'bucket', 0.2143275
6), ('n02951585', 'can_opener', 0.19816446), ('n03764736', 'milk_can', 0.135
79011), ('n04579145', 'whiskey_jug', 0.06520222)]]
```

Out[24]:
```
[[('n02123045', 'tabby', 0.5669847),
  ('n02123394', 'Persian_cat', 0.17530411),
  ('n02123159', 'tiger_cat', 0.08356924),
  ('n02124075', 'Egyptian_cat', 0.024737658),
  ('n02883205', 'bow_tie', 0.023645932)]]
```