HW1_CIFAR

September 25, 2022

1 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.

[165]:
```
!python -VV
```

```
Python 3.8.13 (default, Mar 28 2022, 06:16:26)
[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.

[166]:
```
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
```

[167]:
```
%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"
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

1.1 Preprocessing

1.1.1 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).

[169]:
```
%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",␣
 ↪data_path)
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_train_full, seed)
```

```
image_test_full,  label_test_full  = shuffle_data(image_test_full,␣
↪label_test_full, seed)

# 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_full,␣
↪cifar_class_list, 100)
image_test_uint,  label_test  = get_subset(image_test_full, label_test_full,␣
↪cifar_class_list, 25)

# 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.array(image_train_uint).astype(np.float32)/255.0
image_train = np.reshape(image_train, (-1, 3072))
image_test  = np.array(image_test_uint).astype(np.float32)/255.0
image_test  = np.reshape(image_test, (-1, 3072))
```

1.1.2 Data Visualization

[170]:
```
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(32, 32, 3))
    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]}")
```

1.2 KNN

1.2.1 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.

[171]:
```
from KNN_Model import KNN_Model

## TODO
model_cifar = KNN_Model(cifar_class_list, 9)
model_cifar.fit(image_train, label_train)
```

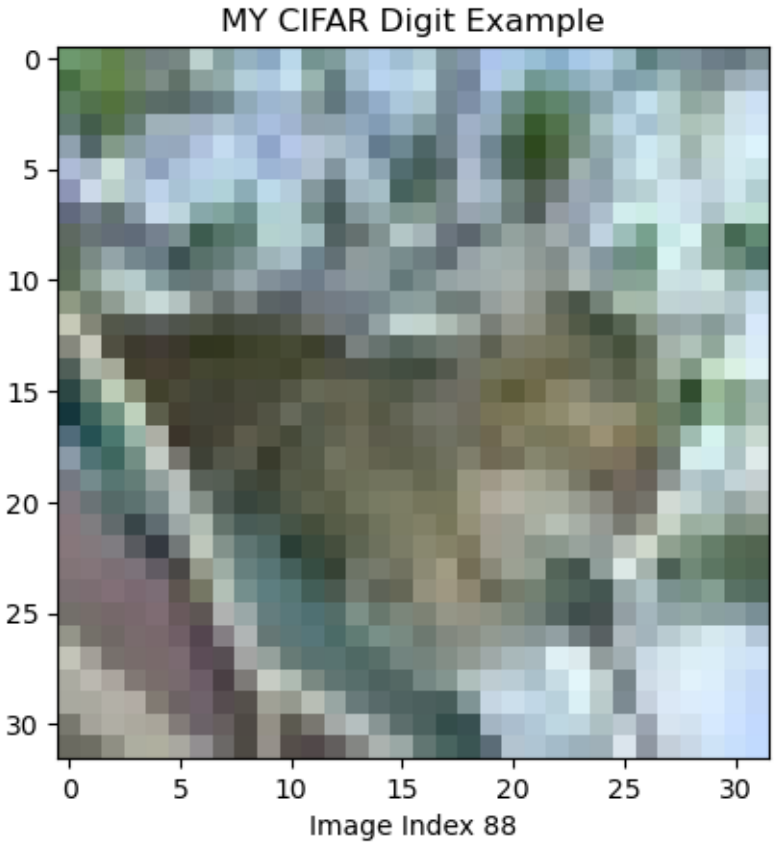
1.2.2 Model Visualization

Code Block #3: Interacting with the model

[172]:
```
## Pull in a specific image
sample_image = image_test[88].copy()
```

```
## TODO: Show what the image looks like using plt.imshow
## Make sure to title it using plt.title
plt.imshow(sample_image.reshape(32, 32, 3), cmap = "Greys")
plt.xlabel("Image Index 88")
plt.title("MY CIFAR Digit Example")
```

[172]:
```
Text(0.5, 1.0, 'MY CIFAR Digit Example')
```



[173]:
```
## TODO: Show what the image looks like using plt.imshow
## Make sure to title it using plt.title
```

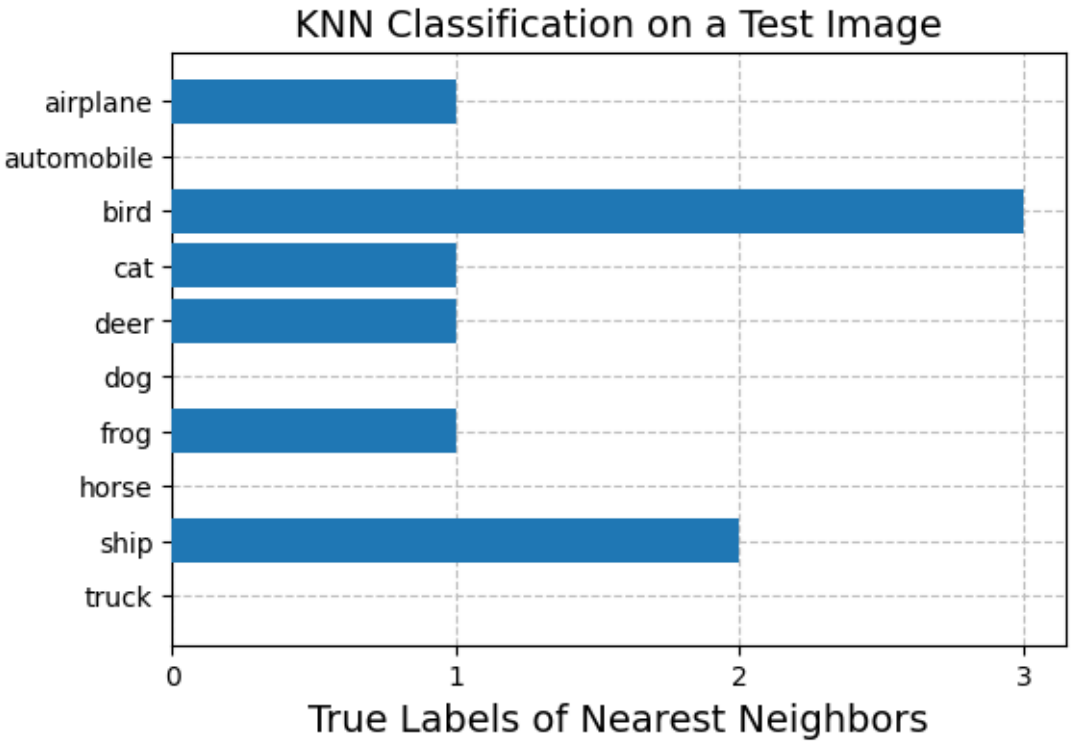
[174]:
```
## TODO: Figure out the closest k neighbors based on the model.
class_counts, nearest_indices = model_cifar.get_neighbor_counts(sample_image,␣
↪True)
```

[175]:
```
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)))
```

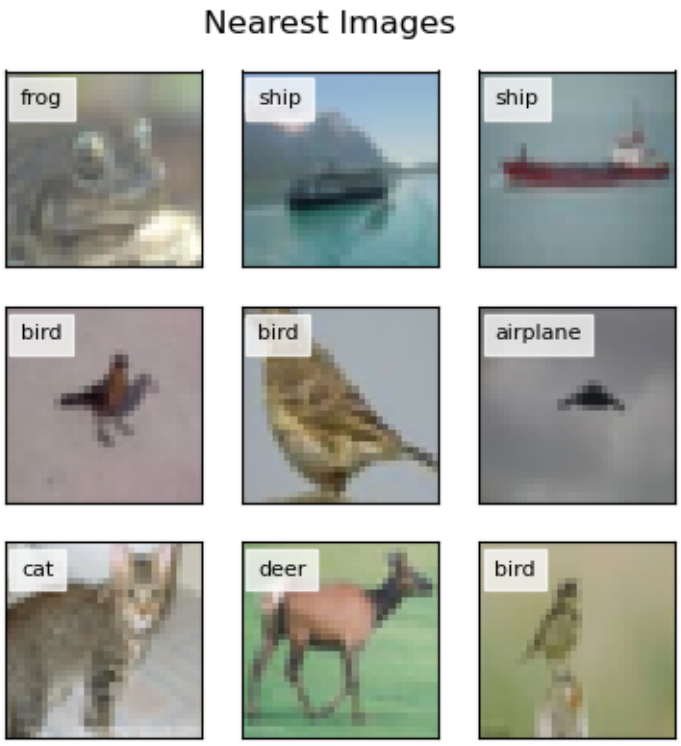
```
ax_knn.set_yticks(cifar_class_list)
ax_knn.set_title("KNN Classification on a Test Image", fontsize=14)
ax_knn.set_xlabel("True Labels of Nearest Neighbors", fontsize=14)
ax_knn.grid(linestyle="dashed", color="#bfbfbf", zorder=-100)
fig_knn.set_size_inches([6, 4])

## 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")
```



[176]:
```
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(32, 32, 3),␣
 ↪cmap="Greys")
    each_ax.tick_params(bottom=False, left=False, labelbottom=False,␣
 ↪labelleft=False)
    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)
```



1.3 Evaluation

It is time to evaluate the model.

1.3.1 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.

[177]:
```
%%time
## TODO: Get the accuracy on the test dataset
prediction_array = model_cifar.get_prediction_array(image_test)
prediction_acc = np.sum(label_test == prediction_array)/prediction_array.size
print(f"accuracy = {prediction_acc}")
```

```
progressing…
progressing…
progressing…
accuracy = 0.232
```

```
CPU times: user 1.5 s, sys: 237 ms, total: 1.74 s
Wall time: 1.74 s
```

1.3.2 Confusion Matrix

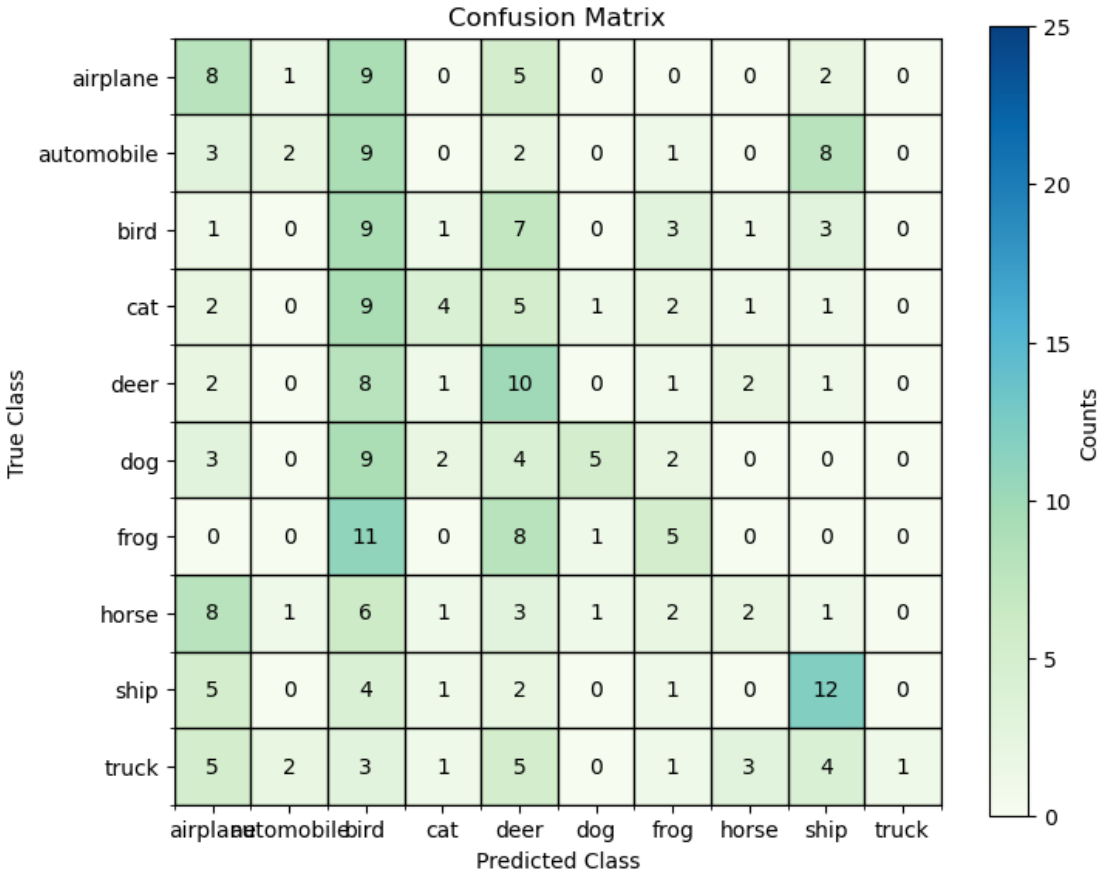
Code Block #5: Confusion matrix

[178]:
```
# TODO: Get the confusion matrix (hint: see KNN_ConfMtx)

confusion_mat = model_cifar.get_confusion_matrix(label_test, prediction_array)
print(confusion_mat)
```

```
[[ 8  1  9  0  5  0  0  0  2  0]
 [ 3  2  9  0  2  0  1  0  8  0]
 [ 1  0  9  1  7  0  3  1  3  0]
 [ 2  0  9  4  5  1  2  1  1  0]
 [ 2  0  8  1 10  0  1  2  1  0]
 [ 3  0  9  2  4  5  2  0  0  0]
 [ 0  0 11  0  8  1  5  0  0  0]
 [ 8  1  6  1  3  1  2  2  1  0]
 [ 5  0  4  1  2  0  1  0 12  0]
 [ 5  2  3  1  5  0  1  3  4  1]]
```

[179]:
```
fig_confusion1, ax_confusion1 = model_cifar.
↪visualize_confusion_matrix(confusion_mat)
```



1.4 Pretrained ResNet50

1.4.1 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`.

[180]:
```
%%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)
```

```
  0%|          | 0/100 [00:00<?, ?it/s]

  0%|          | 0/25 [00:00<?, ?it/s]

CPU times: user 7min 4s, sys: 2min 48s, total: 9min 53s
Wall time: 1min 48s
```

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.

[182]:
```
## TODO: Train the KNN Model on the ResNet embeddings of the trained data
model_resnet = KNN_Model(cifar_class_list, 9)
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.

[183]:
```
%%time
## TODO: Get testing accuracy on model working with ResNet embeddings
prediction_array2 = model_resnet.get_prediction_array(image_test_embeddings)
prediction_acc2 = np.sum(label_test == prediction_array2)/prediction_array2.size
print(f"accuracy = {prediction_acc2}")
```

```
progressing…
progressing…
progressing…
accuracy = 0.612
CPU times: user 1min 8s, sys: 1min 1s, total: 2min 10s
Wall time: 2min 10s
```

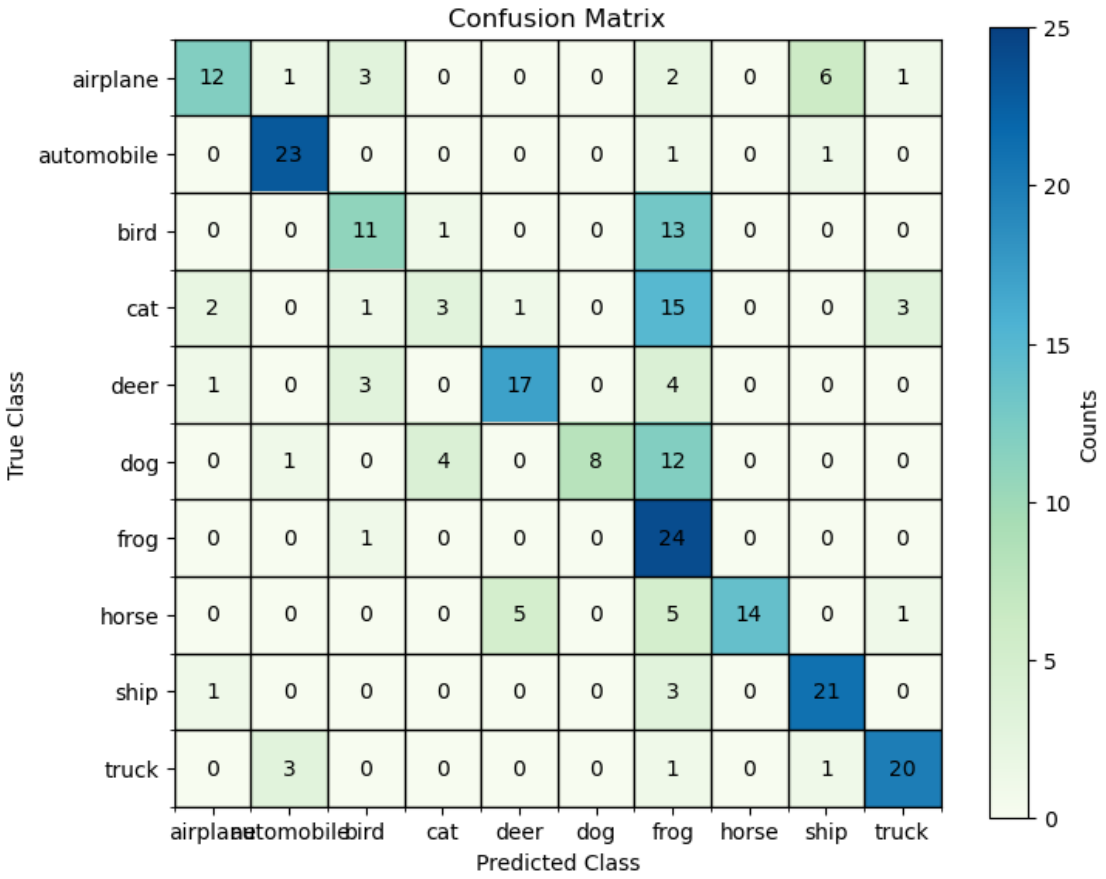
Code Block #9: Confusion matrix of KNN + ResNet

[184]:
```
## TODO: Compute the confusion matrix for this new model as before
confusion_mat2 = model_resnet.get_confusion_matrix(label_test,␣
 ↪prediction_array2)
print(confusion_mat2)
```

```
[[12  1  3  0  0  0  2  0  6  1]
 [ 0 23  0  0  0  0  1  0  1  0]
 [ 0  0 11  1  0  0 13  0  0  0]
 [ 2  0  1  3  1  0 15  0  0  3]
 [ 1  0  3  0 17  0  4  0  0  0]
```

```
[ 0  1  0  4  0  8 12  0  0  0]
[ 0  0  1  0  0  0 24  0  0  0]
[ 0  0  0  0  5  0  5 14  0  1]
[ 1  0  0  0  0  0  3  0 21  0]
[ 0  3  0  0  0  0  1  0  1 20]]
```

[185]:
```
fig_confusion2, ax_confusion2 = model_resnet.
↪visualize_confusion_matrix(confusion_mat2)
```



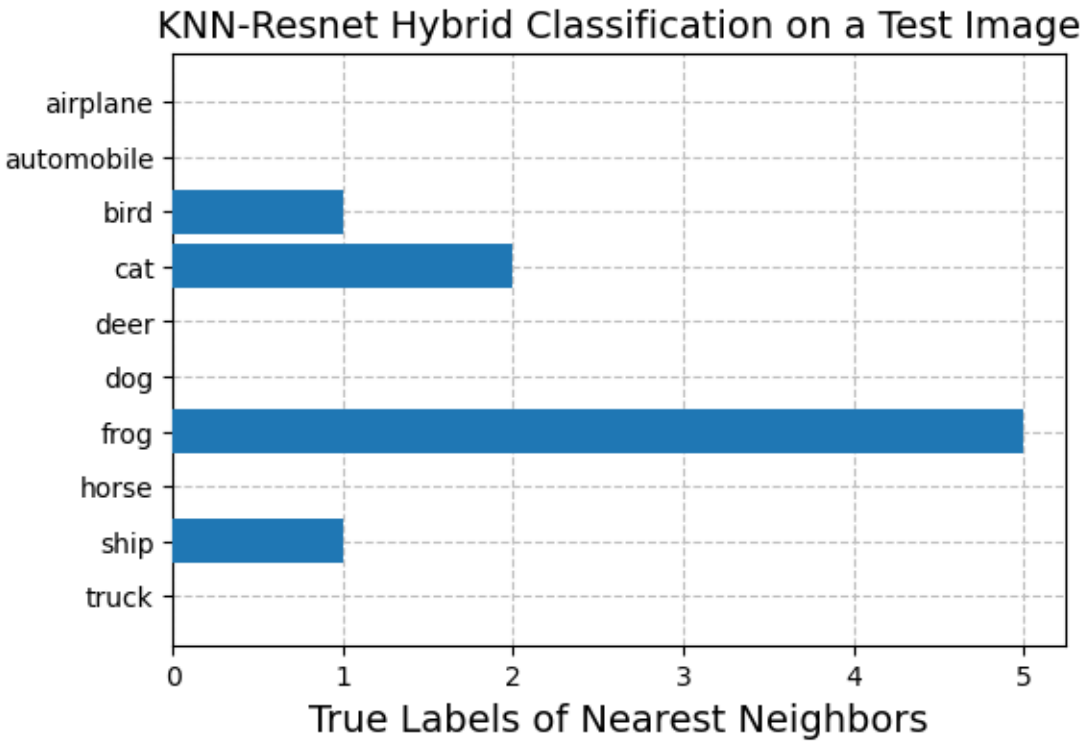
[186]:
```
class_counts, nearest_indices = model_resnet.
↪get_neighbor_counts(image_test_embeddings[88], return_indices=True)
print(class_counts)
```

```
[0, 0, 1, 2, 0, 0, 5, 0, 1, 0]
```

[187]:
```
fig_resnet, ax_resnet = plt.subplots()

class_counts, nearest_indices = model_resnet.
↪get_neighbor_counts(image_test_embeddings[88], return_indices=True)
```

```
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",␣
↪fontsize = 14)
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])
```



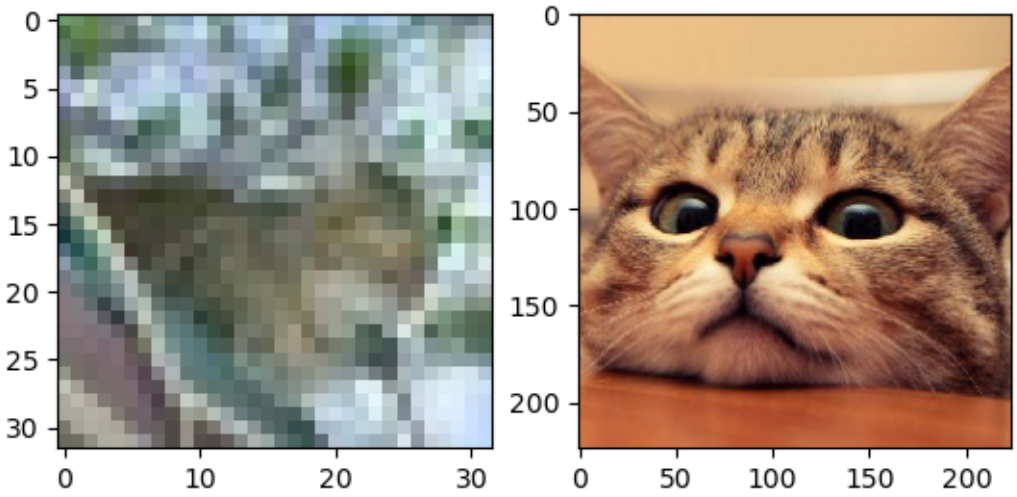
[188]:
```
fig_nearest2, ax_nearest2 = plt.subplots(3, 3, figsize=(4.5, 4.5))

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,␣
↪labelleft=False)
    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)
```

1.4.2 Full ResNet Model

[189]:
```
fig_compare, ax_compare = plt.subplots(1, 2)
ax_compare[0].imshow(image_test_uint[88])
ax_compare[1].imshow(plt.imread(kitten_path))
```

[189]: <matplotlib.image.AxesImage at 0x7fefaa0d3430>



Code Block #10: Full power of the ResNet50 model

[190]:
```
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
print(rs_wrapper.get_full_model_predictions(kitten_image_full))
```

```
[[('n02356798', 'fox_squirrel', 0.77195334), ('n02325366', 'wood_rabbit',
0.042624503), ('n02127052', 'lynx', 0.02329457), ('n02123045', 'tabby',
0.021774016), ('n02487347', 'macaque', 0.018253263)]]
[[('n02123045', 'tabby', 0.56698364), ('n02123394', 'Persian_cat', 0.17530513),
('n02123159', 'tiger_cat', 0.0835691), ('n02124075', 'Egyptian_cat',
0.024737587), ('n02883205', 'bow_tie', 0.023646023)]]
```

[ ]:
```

```