Solution 1

Solution 1 (a)

Step 1: Identify characteristics of the dataspace

We are given 10 dimensional vectors where each element can be any real number ($x_i \in \mathbb{R}$):

$\therefore$ we can express the dataspace χ as: $\chi = \mathbb{R}^{10}$

Solution 1 (b)

Step 1: Identify characteristics of the dataspace

We are given 3 dimensional vectors where each element is zero or one ($x_i \in [0, 1]$):

$\therefore$ we can express the dataspace χ as: $\chi = [0, 1]^3$

Solution 2

Solution 2 (a)

Step 1: Define Euclidean distance (ℓ_2)

$$\ell_2 = \|p - q\|_2 = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

Step 2: Compute ℓ_2

Let $p = 1$ and $q = 10$

$$\begin{aligned}
\ell_2 &= \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \\
\ell_2 &= \sqrt{\sum_{i=1}^{1}(1 - 10)^2} \\
\ell_2 &= \sqrt{(-9)^2} \\
\ell_2 &= 9
\end{aligned}$$

$\therefore \ell_2 = 9$

Solution 2 (b)

Step 1: Define Euclidean distance (ℓ_2)

$$\ell_2 = \|p - q\|_2 = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

Step 2: Compute ℓ_2

Let $p = \begin{bmatrix} -1 \\ 12 \end{bmatrix}, q = \begin{bmatrix} 6 \\ -12 \end{bmatrix}$

$$\begin{aligned}
\ell_2 &= \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \\
\ell_2 &= \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2} \\
\ell_2 &= \sqrt{(-1 - 6)^2 + (12 - (-12))^2} \\
\ell_2 &= \sqrt{(-7)^2 + (24)^2} \\
\ell_2 &= \sqrt{625} \\
\ell_2 &= 25
\end{aligned}$$

$\therefore \ell_2 = 25$

Solution 2

Solution 2 (c)

Step 1: Define Euclidean distance (ℓ_2)

$$\ell_2 = \|p - q\|_2 = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

Step 2: Compute ℓ_2

Let $p = \begin{bmatrix} 1 \\ 5 \\ -1 \end{bmatrix}, q = \begin{bmatrix} 5 \\ 2 \\ 11 \end{bmatrix}$

$$\begin{aligned}
\ell_2 &= \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \\
\ell_2 &= \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2} \\
\ell_2 &= \sqrt{(1 - 5)^2 + (5 - 2)^2 + (-1 - 11)^2} \\
\ell_2 &= \sqrt{(-4)^2 + (3)^2 + (-12)^2} \\
\ell_2 &= \sqrt{169} \\
\ell_2 &= 13
\end{aligned}$$

$\therefore \ell_2 = 13$

Solution 3

Solution 3 (a)

Step 1: Normalize the vector x

Let $x = \begin{bmatrix} 10 \\ 15 \\ 25 \end{bmatrix}$

$$\sum_{i=1}^{3} x_i = x_1 + x_2 + x_3 = 10 + 15 + 25 = 50$$

Now, divide each entry by the total sum:

$$p = \frac{1}{50} \cdot x = \frac{1}{50} \begin{bmatrix} 10 \\ 15 \\ 25 \end{bmatrix} = \begin{bmatrix} 10/50 \\ 15/50 \\ 25/50 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.3 \\ 0.5 \end{bmatrix}$$

$\therefore$ the result (p) of scaling vertor x is the following:

$$p = \begin{bmatrix} 0.2 \\ 0.3 \\ 0.5 \end{bmatrix}$$

Solution 3 (b)

Step 1: Define dimension of the probability simplex

The dimension of vector p is 3 and $k = n - 1$ where k is the dimension of the probability simplex

$\therefore$ vector p lies in the probability simplex(Δ_2) for $k = 2$

Solution 4

Step 1: Define probability simplex Δ_2

For a point to be scalable to Δ_2, after scaling it must satisfy:

- All components must be non-negative
- The sum of components must equal 1

Step 2: Give example that violates one of the rules in Step 1

Let $x = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$

The second component of x violates the first rule, all components for a point must be non-negative Δ_2.

$\therefore$ the point $x = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$ cannot be scaled to lie in Δ_2

Solution 5

Visualizing the Simplex Δ_3 in 2D Projections

Here are the three 2D views of the probability simplex Δ_3. Each plot is a *shadow* of the 3D triangle, viewed along one of the principal axes.

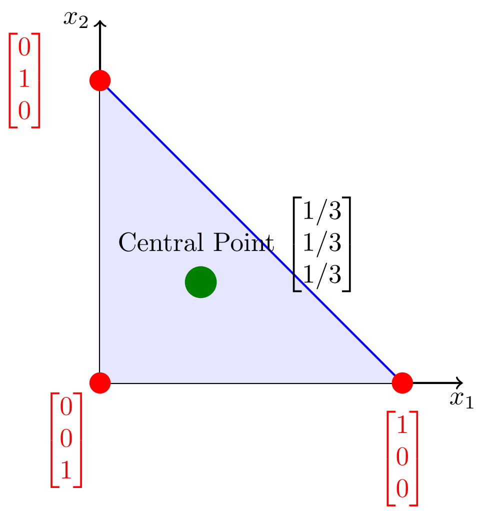

Figure 1: View 1: Projection onto the x_1-x_2 plane.

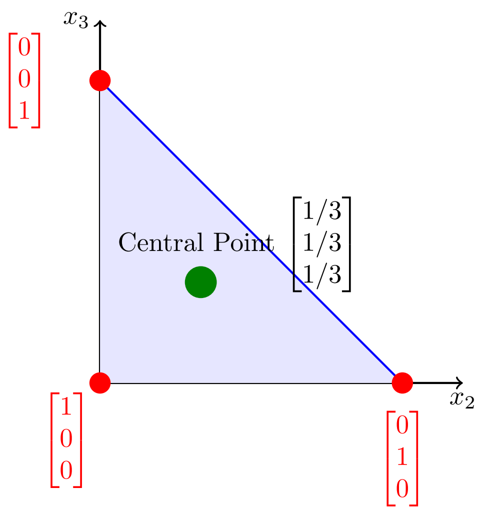

Figure 2: View 2: Projection onto the x_2-x_3 plane.

Solution 5

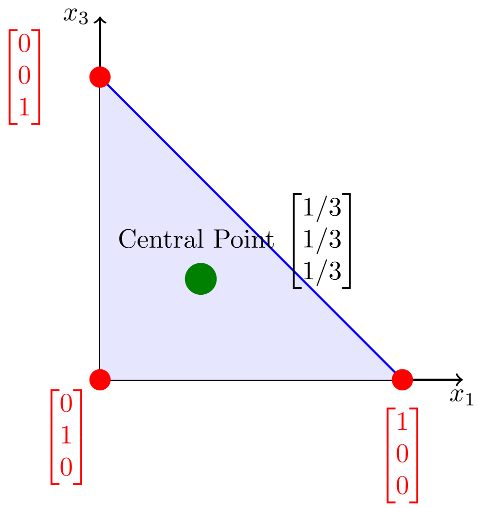

Figure 3: View 3: Projection onto the x_1-x_3 plane.

Solution 6

---

6 (a): ℓ_1 for p and q

The ℓ_1 distance between two vectors $p, q \in \mathbb{R}^n$ is given by:

$$\|p - q\|_1 = \sum_{i=1}^{n} |p_i - q_i|$$

Let $p = \begin{bmatrix} 1/2 \\ 1/4 \\ 1/8 \\ 1/8 \end{bmatrix}$ and $q = \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}$

$$\begin{aligned}
\|p - q\|_1 &= \left|\frac{1}{2} - \frac{1}{4}\right| + \left|\frac{1}{4} - \frac{1}{4}\right| + \left|\frac{1}{8} - \frac{1}{4}\right| + \left|\frac{1}{8} - \frac{1}{4}\right| \\
&= \left|\frac{2}{4} - \frac{1}{4}\right| + |0| + \left|\frac{1}{8} - \frac{2}{8}\right| + \left|\frac{1}{8} - \frac{2}{8}\right| \\
&= \frac{1}{4} + 0 + \left|-\frac{1}{8}\right| + \left|-\frac{1}{8}\right| \\
&= \frac{1}{4} + \frac{1}{8} + \frac{1}{8} = \frac{2}{8} + \frac{1}{8} + \frac{1}{8} = \frac{4}{8} = \frac{1}{2}
\end{aligned}$$

$\therefore \ell_1 = \frac{1}{2}$

Solution 6

---

6 (b): ℓ_1 for q and r

The ℓ_1 distance between two vectors $q, r \in \mathbb{R}^n$ is given by:

$$\|q - r\|_1 = \sum_{i=1}^{n} |q_i - r_i|$$

Let $q = \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}$ and $r = \begin{bmatrix} 1/2 \\ 0 \\ 1/4 \\ 1/4 \end{bmatrix}$

$$\begin{aligned}
\|q - r\|_1 &= \sum_{i=1}^{4} |q_i - r_i| \\
&= \left|\frac{1}{4} - \frac{1}{2}\right| + \left|\frac{1}{4} - 0\right| + \left|\frac{1}{4} - \frac{1}{4}\right| + \left|\frac{1}{4} - \frac{1}{4}\right| \\
&= \left|\frac{1}{4} - \frac{2}{4}\right| + \left|\frac{1}{4}\right| + |0| + |0| \\
&= \left|-\frac{1}{4}\right| + \frac{1}{4} + 0 + 0 \\
&= \frac{1}{4} + \frac{1}{4} \\
&= \frac{1}{2}
\end{aligned}$$

$\therefore \ell_1 = \frac{1}{2}$

Solution 6

---

6 (c): KL divergence $K(p,q)$

The Kullback-Leibler (KL) divergence from a distribution p to a distribution q is defined as:

$$K(p,q) = \sum_i p_i \ln \frac{p_i}{q_i}$$

Let $p = \begin{bmatrix} 1/2 \\ 1/4 \\ 1/8 \\ 1/8 \end{bmatrix}$ and $q = \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}$

$$\begin{aligned}
K(p,q) &= \sum_{i=1}^{4} p_i \ln\left(\frac{p_i}{q_i}\right) \\
&= p_1 \ln\left(\frac{p_1}{q_1}\right) + p_2 \ln\left(\frac{p_2}{q_2}\right) + p_3 \ln\left(\frac{p_3}{q_3}\right) + p_4 \ln\left(\frac{p_4}{q_4}\right) \\
&= \frac{1}{2} \ln\left(\frac{1/2}{1/4}\right) + \frac{1}{4} \ln\left(\frac{1/4}{1/4}\right) + \frac{1}{8} \ln\left(\frac{1/8}{1/4}\right) + \frac{1}{8} \ln\left(\frac{1/8}{1/4}\right) \\
&= \frac{1}{2} \ln(2) + \frac{1}{4} \ln(1) + \frac{1}{8} \ln\left(\frac{1}{2}\right) + \frac{1}{8} \ln\left(\frac{1}{2}\right) \\
&= \frac{1}{2} \ln(2) + \frac{1}{4}(0) - \frac{1}{8} \ln(2) - \frac{1}{8} \ln(2) \\
&= \frac{1}{2} \ln(2) - \frac{2}{8} \ln(2) \\
&= \left(\frac{1}{2} - \frac{1}{4}\right) \ln(2) \\
&= \frac{1}{4} \ln(2)
\end{aligned}$$

$\therefore K(p,q) = \frac{1}{4} \ln(2)$

Solution 6

---

6 (d): KL divergence $K(q, r)$

The Kullback-Leibler (KL) divergence from a distribution q to a distribution r is defined as:

$$K(q, r) = \sum_i q_i \ln \frac{q_i}{r_i}$$

Let $q = \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}$ and $r = \begin{bmatrix} 1/2 \\ 0 \\ 1/4 \\ 1/4 \end{bmatrix}$

Looking at the second component ($i = 2$). Here, $q_2 = \frac{1}{4} > 0$ while $r_2 = 0$. The corresponding term in the KL divergence sum, $q_2 \ln \left(\frac{q_2}{r_2}\right)$, involves division by zero.

Hence, the divergence will be infinite.

$\therefore K(q, r) = \infty$

Solution 7

Python Code

```python
import numpy as np
import matplotlib.pyplot as plt
from extract_feature import compute_or_load_features
from sklearn.neighbors import KNeighborsClassifier

def run_nearest_neighbor(x_train, y_train, x_test, y_test):
    # create classifier
    nn_classifier = KNeighborsClassifier(n_neighbors=1, algorithm='auto')

    # train
    nn_classifier.fit(x_train, y_train)

    # test and report accuracy
    test_acc = nn_classifier.score(x_test, y_test)

    print("Nearest neighbor accuracy on the test set: %f"%test_acc)

    return nn_classifier


def analyze_nn(classifier, x_train, y_train, x_test, y_test, x_test_features, N, model_type):
    """
    generalization to grab indices, predictions, and make plots
    """
    # list of image labels
    CIFAR_CLASSES = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

    # get predictions of classifier on test data
    y_pred = classifier.predict(x_test_features)

    # define bool condtion where classifier made correct prediction
    is_correct = (y_pred == y_test)

    # get indices for correct and incorrect samples
    correct_indices_test = np.where(is_correct)[0][:N]
    incorrect_indices_test = np.where(~is_correct)[0][:N]

    # make image plots
    def plot_pairs(title, test_indices):

        # check just in case of bad result.. nema nista
        if len(test_indices) == 0:
            return

        # Get the feature vectors for the selected test images (flattened)
        selected_test_features = x_test_features[test_indices]

        # get nearest neighbor
        # note: kneighbors -> (distances, indices).. only need the indices.
        _, nn_train_indices_2D = classifier.kneighbors(X=selected_test_features, n_neighbors=1)

        # 2D -> 1D
        nn_train_indices = nn_train_indices_2D.flatten()

        # get images and labels for the selected test points
        test_images_sample = x_test[test_indices]
        test_labels_sample = y_test[test_indices]

        #get the RAW neighbor image and label from the training points
        nn_images_sample = x_train[nn_train_indices]
        nn_labels_sample = y_train[nn_train_indices]

```

```
        # prediction is nearest neighbor label
        y_pred_sample = nn_labels_sample

        # reshape (N, C, H, W) to (N, H, W, C) for plot
        test_images_plt = test_images_sample.transpose(0, 2, 3, 1)
        nn_images_plt = nn_images_sample.transpose(0, 2, 3, 1)

        N_plot = len(test_indices)

        # just in case only 1 image is plotted (axes will be 1D instead of 2D)
        if N_plot == 1:
            fig, axes = plt.subplots(N_plot, 2, figsize=(6, 2))
            axes = axes[np.newaxis, :]
        else:
            fig, axes = plt.subplots(N_plot, 2, figsize=(6, 2 * N_plot))

        fig.suptitle(title, fontsize=14, y=1.02)

        for i in range(N_plot):
            is_correct = (test_labels_sample[i] == y_pred_sample[i])
            pred_color = 'green' if is_correct else 'red'

            # plot test image
            axes[i, 0].imshow(test_images_plt[i] / 255.0)
            axes[i, 0].set_title(f"Test ({CIFAR_CLASSES[test_labels_sample[i]]})",
                fontsize=10)
            axes[i, 0].axis('off')

            # plot nearest neighbor
            axes[i, 1].imshow(nn_images_plt[i] / 255.0)
            axes[i, 1].set_title(
                f"NN ({CIFAR_CLASSES[nn_labels_sample[i]]})",
                fontsize=10,
                color=pred_color
            )
            axes[i, 1].axis('off')

        plt.tight_layout(rect=[0, 0.03, 1, 0.98])
        plt.show()
        fig.savefig(f"{title.split(' ')[2].lower()}_{model_type}.png")

    # plot correct and incorrect cases
    plot_pairs(f"First {N} Correct Predictions ({model_type})", correct_indices_test)
    plot_pairs(f"First {N} Incorrect Predictions ({model_type})",
        incorrect_indices_test)

# raw pixel
raw_pixel_train_features, raw_pixel_test_features = compute_or_load_features(x_train,
    x_test, "raw_pixel")
raw_pixel_knn_classifier = run_nearest_neighbor(raw_pixel_train_features, y_train,
    raw_pixel_test_features, y_test)
analyze_nn(
    classifier=raw_pixel_knn_classifier,
    x_train=x_train,
    y_train=y_train,
    x_test=x_test,
    y_test=y_test,
    x_test_features=raw_pixel_test_features,
    N=5,
    model_type="raw_pixel"
)

# HoG
hog_train_features, hog_test_features = compute_or_load_features(x_train, x_test, "hog")
hog_knn_classifier = run_nearest_neighbor(hog_train_features, y_train,
    hog_test_features, y_test)
analyze_nn(
    classifier=hog_knn_classifier,
    x_train=x_train,
    y_train=y_train,
    x_test=x_test,
    y_test=y_test,
    x_test_features=hog_test_features,
```

```
    N=5,
    model_type="hog"
)

# vgg-last-fc
pretrained_cnn_last_fc_train_features, pretrained_cnn_last_fc_test_features =
    compute_or_load_features(x_train, x_test, "pretrained_cnn", "last_fc")
pretrained_cnn_last_fc_knn_classifier =
    run_nearest_neighbor(pretrained_cnn_last_fc_train_features, y_train,
    pretrained_cnn_last_fc_test_features, y_test)
analyze_nearest_neighbors_simple(
    classifier=pretrained_cnn_last_fc_knn_classifier,
    x_train=x_train,
    y_train=y_train,
    x_test=x_test,
    y_test=y_test,
    x_test_features=pretrained_cnn_last_fc_test_features,
    N=5,
    model_type='vgg_last_fc'
)
```

Solution 7

Part a: Dimensionality for each of the representations (raw pixel, HoG, VGG-last-fc, VGG-last-conv)

Feature Type	Dimensionality
Raw Pixel	3072
HoG	512
VGG-last-fc	4096
VGG-last-conv	512

Part b: Test accuracies for 1-nearest neighbor classification using the various representations (raw pixel, HoG, VGG-last-fc, VGG-last-conv, random-VGG-last-fc, random-VGG-last-conv).

Feature Type	1-NN test accuracy (%)
Raw Pixel	35.4
HoG	36.6
VGG-last-fc	92.1
VGG-last-conv	92.0
random VGG-last-fc	39.1
random VGG-last-conv	40.6

Solution 7

Part c: Raw Pixel correct/incorrect

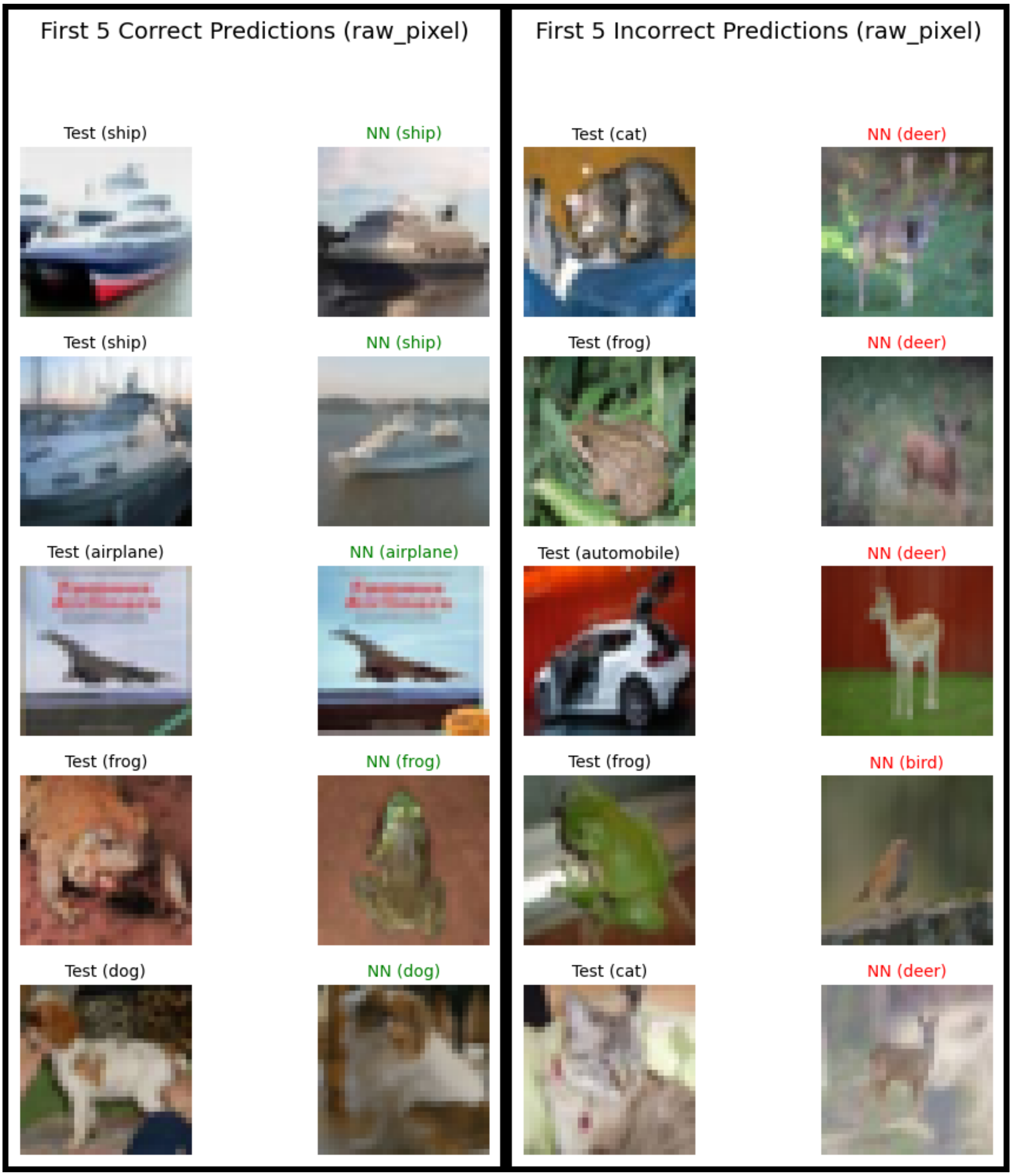

Figure 4: First five correct/incorrect images for Raw Pixel

Solution 7

Part c: HoG correct/incorrect

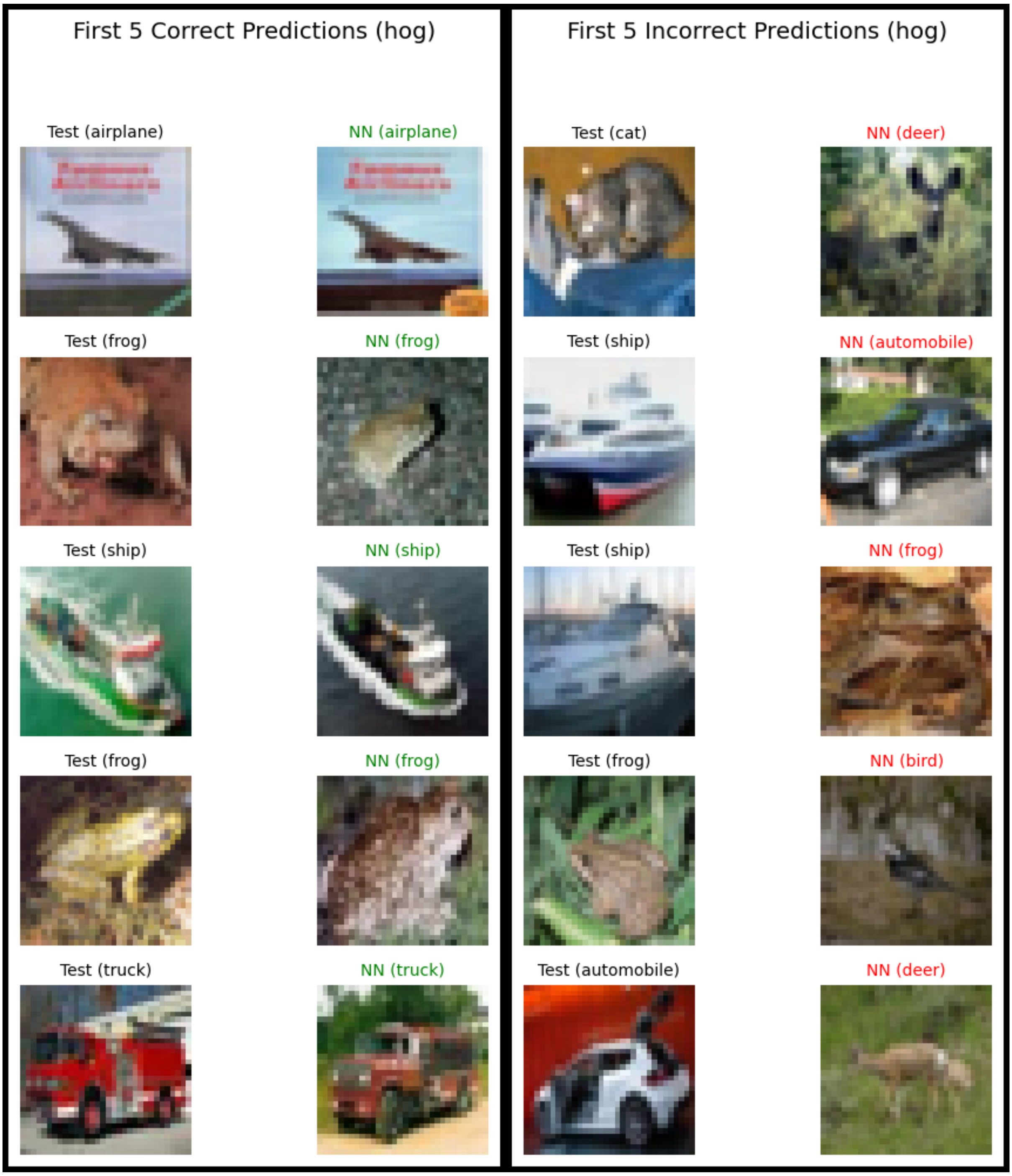

Figure 5: First five correct/incorrect images for HoG

Solution 7

Part c: VGG-last-fc correct/incorrect

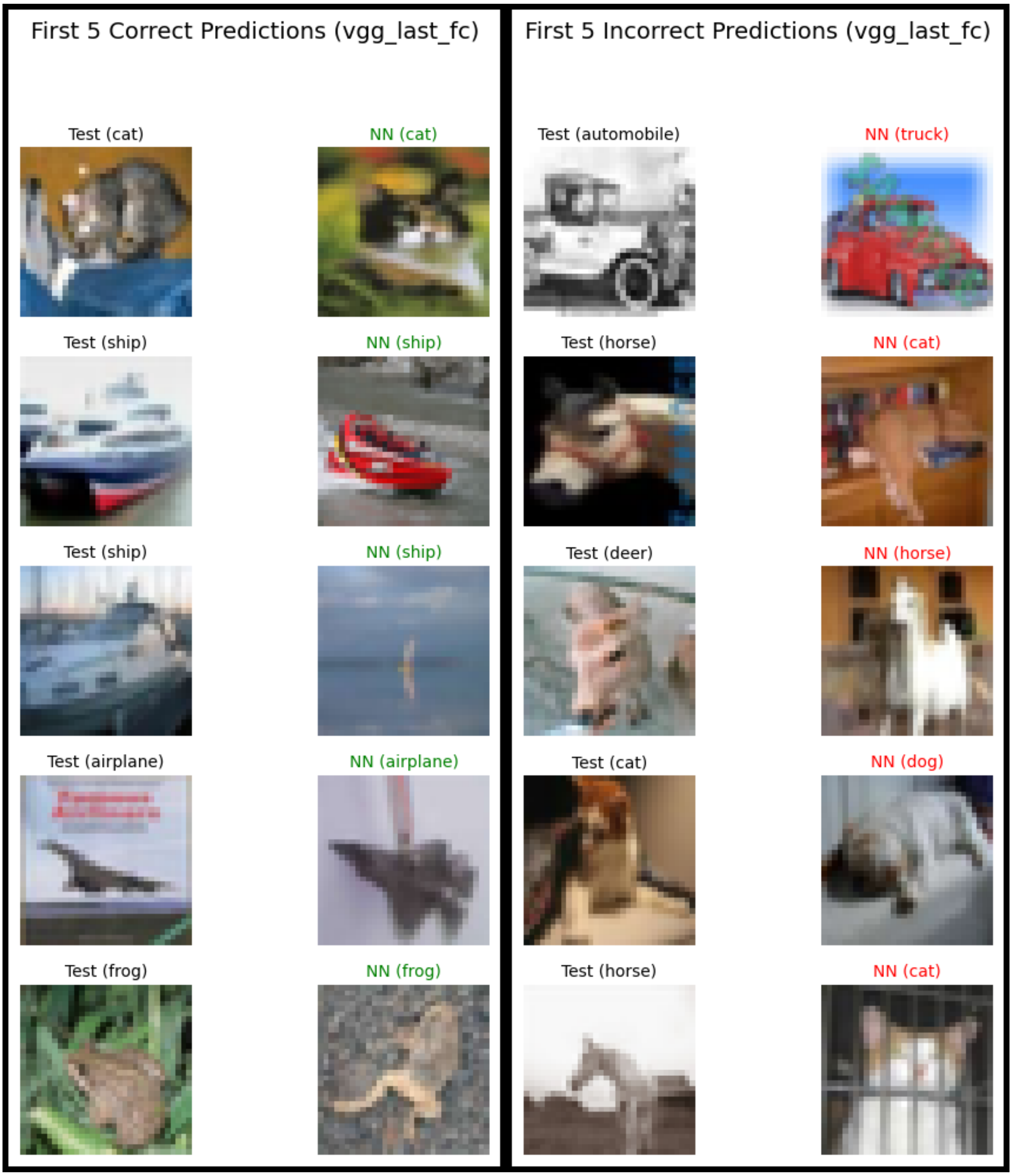

Figure 6: First five correct/incorrect images for VGG-last-fc

Solution 8

Python Code

```
import numpy as np
from sklearn.neighbors import NearestNeighbors

filename = 'glove.6B.300d.txt'
with open(filename) as f:
    content = f.read().splitlines()

# initialize vecs and words 'containers'
n = len(content)
vecs = np.zeros((n, 300))
words = ["" for i in range(n)]
for index, rawline in enumerate(content):
    line = rawline.split()
    words[index] = line[0]
    # need4numpy speed
    vecs[index] = np.array(line[1:], dtype=np.float32)

# make dict for access to word and index
word_to_index = {word: i for i, word in enumerate(words)}

# initialize target words
target_words = ['communism', 'africa', 'happy', 'sad', 'upset', 'computer', 'cat',
    'dollar']

# find 5 nearest neighbors (n).. remember k = n+1
n_neighbors = 6

# initialize  and fit the nn model
nn_model = NearestNeighbors(n_neighbors=n_neighbors, metric='euclidean',
    algorithm='auto')
nn_model.fit(vecs)

# gracefully check for typo
try:
    target_indices = [word_to_index[word] for word in target_words]
except KeyError as e:
    print(f" error: word '{e.args[0]}' was not found.")
    exit()

# extract the corresponding vectors for the target words from vecs
target_vectors = vecs[target_indices]

# find the nearest neighbors
distances, indices = nn_model.kneighbors(target_vectors)

# format and print results
results = {}
for i, word in enumerate(target_words):
    neighbor_indices = indices[i][1:]
    neighbor_words = [words[idx] for idx in neighbor_indices]
    results[word] = neighbor_words

print(f"5 nearest neighbors in {filename} for {target_words}")
print(results)
```

Solution 8

Word Vectors: 5 closest words

Target Word	Five Closest Words
communism	['fascism', 'capitalism', 'nazism', 'stalinism', 'socialism']
africa	['african', 'continent', 'south', 'africans', 'zimbabwe']
happy	['glad', 'pleased', 'always', 'everyone', 'sure']
sad	['sorry', 'tragic', 'happy', 'pathetic', 'awful']
upset	['upsetting', 'surprised', 'upsets', 'stunned', 'shocked']
computer	['computers', 'software', 'technology', 'laptop', 'computing']
cat	['cats', 'dog', 'pet', 'feline', 'dogs']
dollar	['currency', 'dollars', 'euro', 'multibillion', 'weaker']
