csc421_Fall_2023_assignment4

December 8, 2023

1 CSC421 Fall 2023 Assignment 4

1.0.1 Author: George Tzanetakis

This notebook is based on the topics covered in **Chapter 18 Learning** and **Chapter 21 Deep Learning from the book *Artificial Intelligence: A Modern Approach.* You are welcome and actually it can be educational to look at the code at the aima-code repository as well as other code resources you can find on the web. However, make sure you understand any code that you incoporate.

The assignment structure is as follows - each item is worth 1 point:

1. Create mini CIFAR-10 (Basic)

2. SVM classification of CIFAR-10 (Basic)
3. Naive Bayes Gaussian (Expected)
4. Sort classes by prediction accuracy (Basic)
5. Show misclassification examples (Expected)
6. Compare raw image, histogram-of-gradients, and principal component analysis of hogs (Expected)
7. Change batch size and optimizer and compare (Basic)
8. Add noise to test images (Expected)

9. Generate synthetic dataset (4 colors, 4 shapes, 4 sizes, 4 x positions, 4 y positions) (Advanced)
10. Deep learning classification of synthetic dataset (Advanced)

2 Question 1 (Basic) Create mini CIFAR-10

Re-use the code from the deep learning notebook to load the CIFAR-10 training and test datasets. Create a mini CIFAR-10 dataset with 5000 instances for training and 5000 instances for testing. The examples in CIFAR-10 are randomly shuffled so you can simply take the first 5000 examples of each dataset. Print the shape of the resulting training set and test set.

[8]:
```
# pip3 install torch torchvision torchaudio --index-url https://download.
↪pytorch.org/whl/cpu --no-cache-dir
# pip3 install tabulate
# pip3 install torchsummary
import torch
import torchvision
import torchvision.transforms as transforms
```

```python
from tabulate import tabulate

transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

batch_size = 32

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainset.data = trainset.data[:5000]
trainset.targets = trainset.targets[:5000]
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=4)
print("Trainset is modified")

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testset.data = testset.data[:5000]
testset.targets = testset.targets[:5000]
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=4)
print("Testset is modified")
print(f'Training set sample shape: {trainset.data.shape}')
print(f'Labels: {len(trainset.targets)}')
print(f'Testing set sample shape: {testset.data.shape}')
print(f'Labels: {len(testset.targets)}')
print('\nClasses\n')
print(tabulate(
    list(trainset.class_to_idx.items()), headers=['Name', 'Index'],
    tablefmt='orgtbl'
))
```

```
Files already downloaded and verified
Trainset is modified
Files already downloaded and verified
Testset is modified
Training set sample shape: (5000, 32, 32, 3)
Labels: 5000
Testing set sample shape: (5000, 32, 32, 3)
Labels: 5000

Classes

| Name       |   Index |
|------------+---------|
| airplane   |       0 |
```

```
| automobile |        1 |
| bird       |        2 |
| cat        |        3 |
| deer       |        4 |
| dog        |        5 |
| frog       |        6 |
| horse      |        7 |
| ship       |        8 |
| truck      |        9 |
```

2.1 Question 2 (Basic) - SVM

Train a SVM classifier on PCA dimensionality reduced Histogram of Oriented Gradients features. You can re-use the code from the deep learning notebook but instead of using the full training and testing sets use the mini-CIFAR-10 datset you created in question 1. Report the classification accuracy and confusion matrix.

[2]:
```python
# Your code goes here
import matplotlib.pyplot as plt
from sklearn import datasets, metrics, svm
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report
from skimage.feature import hog
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import numpy as np

x_train = trainset.data
x_test = testset.data

xtrain_hog = []
for i in range(len(x_train)):
    fd  = hog(x_train[i] , orientations=9 , pixels_per_cell = (8,8),
                     cells_per_block = (2,2) , visualize = False,␣
 ↪channel_axis=-1)
    xtrain_hog.append(fd)
    if ((i % 500) == 0):
        print(i)
xtrain_hog = np.array(xtrain_hog)
print('Done calculating HOGs for training')
print(xtrain_hog.shape)

xtest_hog = []
for i in range(len(x_test)):
    fd = hog(x_test[i] , orientations=9 , pixels_per_cell = (8,8),
                     cells_per_block = (2,2) , visualize = False,␣
 ↪channel_axis=-1)
    xtest_hog.append(fd)
```

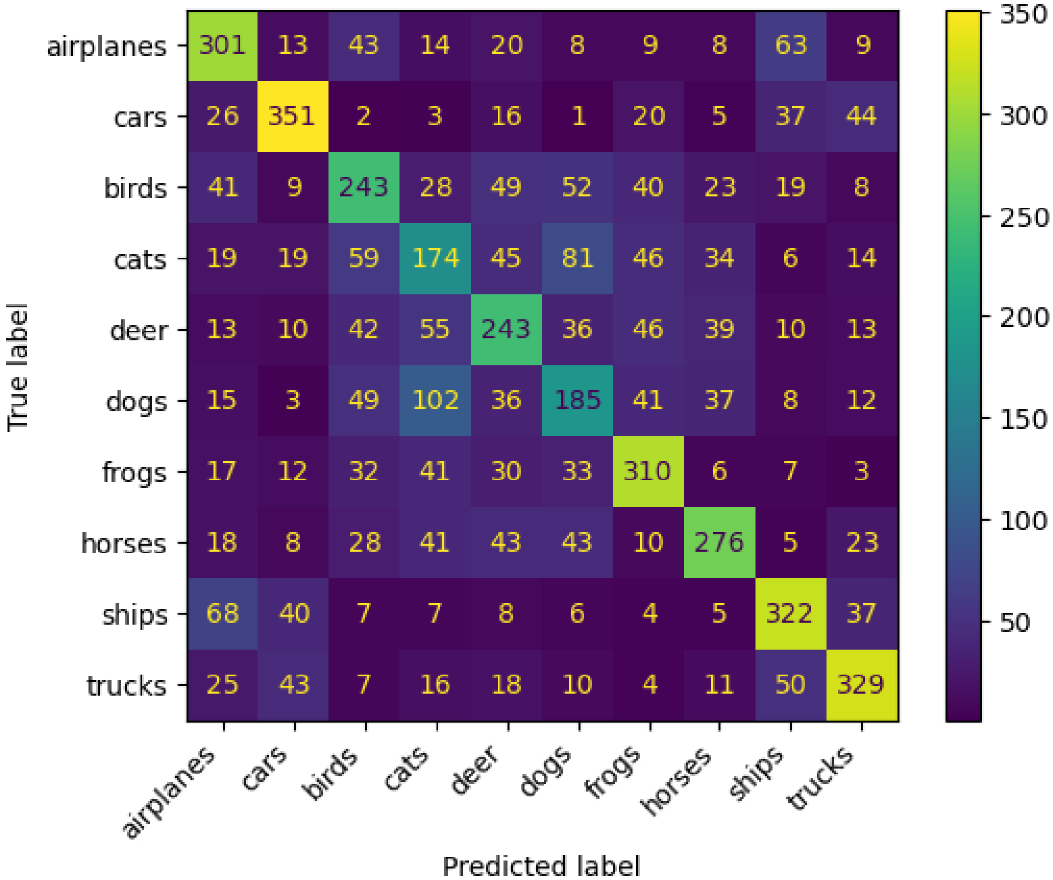
```
    if ((i % 500) == 0):
        print(i)

xtest_hog = np.array(xtest_hog)
print('Done calculating HOGs for testing')
print(xtest_hog.shape)

print("Principal component analysis PCA")
pca = PCA(0.8)
y_train = trainset.targets
y_test = testset.targets

xtrain_pca = pca.fit_transform(xtrain_hog)
xtest_pca  = pca.transform(xtest_hog)
print(xtrain_pca.shape)
print(xtest_pca.shape)
```

```
0
500
1000
1500
2000
2500
3000
3500
4000
4500
Done calculating HOGs for training
(5000, 324)
0
500
1000
1500
2000
2500
3000
3500
4000
4500
Done calculating HOGs for testing
(5000, 324)
Principal component analysis PCA
(5000, 63)
(5000, 63)
```

[3]:
```
# Create a classifier: a support vector classifier
clf = svm.SVC(C=10, cache_size=5000)
```

```
clf.fit(xtrain_pca, y_train)

ytest_predict_svc  = clf.predict(xtest_pca)
print(classification_report(y_test, ytest_predict_svc))

color = 'white'
cm_svc = confusion_matrix(y_test, ytest_predict_svc)
disp = ConfusionMatrixDisplay(cm_svc, display_labels=['airplanes', 'cars',␣
 ↪'birds', 'cats', 'deer', 'dogs', 'frogs', 'horses', 'ships', 'trucks'])
disp.plot()
plt.xticks(rotation=45, ha='right')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.55      0.62      0.58       488
           1       0.69      0.70      0.69       505
           2       0.47      0.47      0.47       512
           3       0.36      0.35      0.36       497
           4       0.48      0.48      0.48       507
           5       0.41      0.38      0.39       488
           6       0.58      0.63      0.61       491
           7       0.62      0.56      0.59       495
           8       0.61      0.64      0.62       504
           9       0.67      0.64      0.65       513

    accuracy                           0.55      5000
   macro avg       0.55      0.55      0.55      5000
weighted avg       0.55      0.55      0.55      5000
```



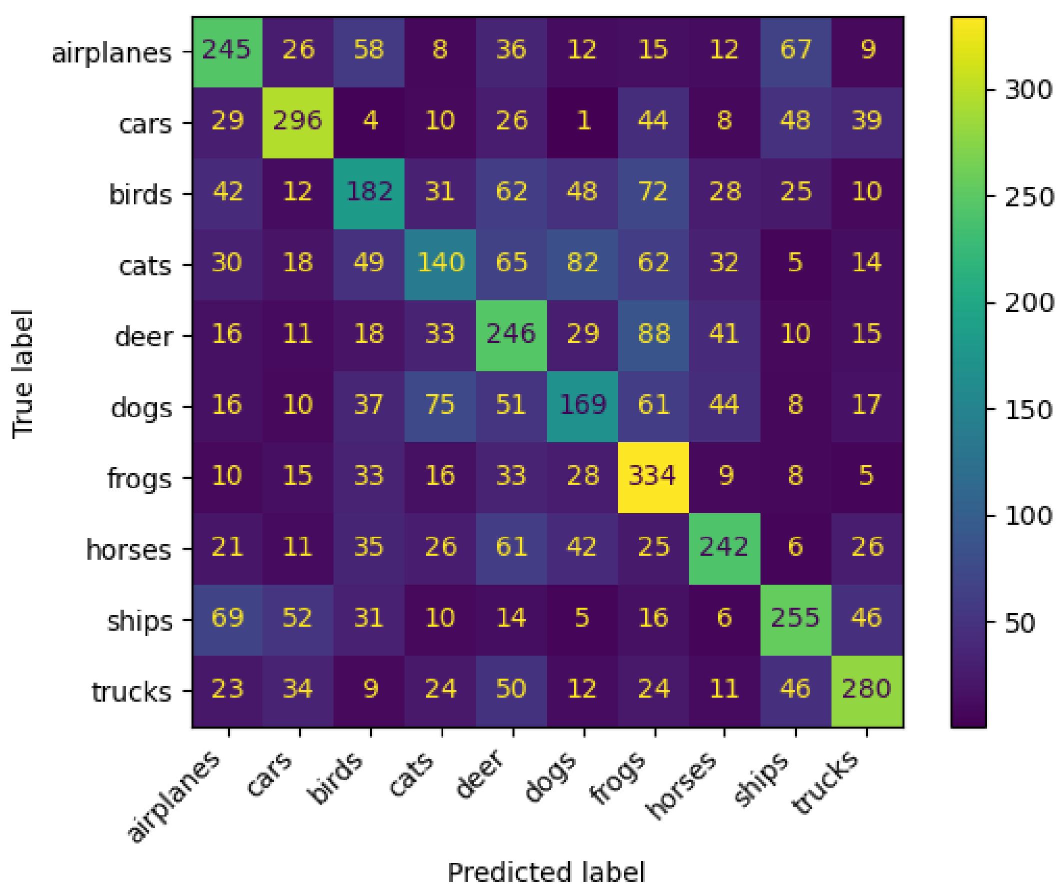
3 Question 3 (Expected) - Gaussian Naive Bayes Classifier

Repeat the training and evaluation of the mini-CIFAR-10 dataset using the Gaussian Naive Bayes classifier from scikit-learn: Gaussian Naive Bayes Similarly report on the classification accuracy and confusion matrix.

[4]:
```
# YOUR CODE GOES HERE
# Create a Gaussian Naive Bayes Classifier
gnb = GaussianNB()
gnb.fit(xtrain_pca, y_train)
ytest_predict_gnb  = gnb.predict(xtest_pca)
print(classification_report(y_test, ytest_predict_gnb))

color = 'white'
cm_gnb = confusion_matrix(y_test, ytest_predict_gnb)
disp = ConfusionMatrixDisplay(cm_gnb, display_labels=['airplanes', 'cars',␣
 ↪'birds', 'cats', 'deer', 'dogs', 'frogs', 'horses', 'ships', 'trucks'])
disp.plot()
```

```
plt.xticks(rotation=45, ha='right')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.49      0.50      0.50       488
           1       0.61      0.59      0.60       505
           2       0.40      0.36      0.38       512
           3       0.38      0.28      0.32       497
           4       0.38      0.49      0.43       507
           5       0.39      0.35      0.37       488
           6       0.45      0.68      0.54       491
           7       0.56      0.49      0.52       495
           8       0.53      0.51      0.52       504
           9       0.61      0.55      0.57       513

    accuracy                           0.48      5000
   macro avg       0.48      0.48      0.47      5000
weighted avg       0.48      0.48      0.47      5000
```



Question 4 (Expected) - Sort classes by prediction accuracy

Write a function that takes as input the computed confusion matrix and returns a list of classes sorted by classification accuracy. Each item in the list should be a tuple of the form (class, accuracy). Show the outpput for the SVM and Gaussian NB classifiers for the mini CIFAR-10 dataset.

[5]:
```
# Your answer goes here
# we will calculate FP and FN to get the accuracy, Acc = N-FP-FN/N
def class_acc(con_matrix):
    columns = sum(con_matrix) #columns of the matrix
    class_num = [i for i in range(len(con_matrix[0]))] #get labels
    tp = [con_matrix[i][i] for i in class_num] #list of true positives for each␣
 ↪class
    fn = [sum(con_matrix[i])-tp[i] for i in class_num] #list of false negatives␣
 ↪for each class
    fp = [columns[i]-tp[i] for i in class_num] #list of false positives for␣
 ↪each class
    acc = [round((sum(columns) - fn[i] - fp[i]) / sum(columns), 2) for i in␣
 ↪class_num] #accuracy for eacc class
    matrix_acc = [(i, acc[i]) for i in class_num]
    return matrix_acc

#SVC Confusion Matrix accuracy for each class
svc_acc = class_acc(cm_svc)
print("SVC confusion matrix report: ", svc_acc)
gnb_acc = class_acc(cm_gnb)
print("GNB confusion matrix report: ", gnb_acc)
```

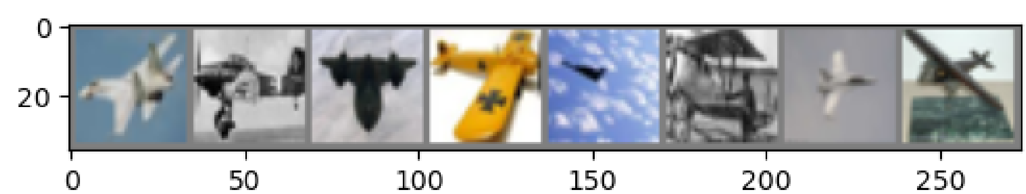
```
SVC confusion matrix report:  [(0, 0.91), (1, 0.94), (2, 0.89), (3, 0.87), (4,
0.89), (5, 0.89), (6, 0.92), (7, 0.92), (8, 0.92), (9, 0.93)]
GNB confusion matrix report:  [(0, 0.9), (1, 0.92), (2, 0.88), (3, 0.88), (4,
0.87), (5, 0.88), (6, 0.89), (7, 0.91), (8, 0.91), (9, 0.92)]
```

4 Question 5 (Expected) - Show misclassification examples

Write a function that takes as input a particular class (for example dog) and shows an array of images (similar to the functions showing images in the deep learning notebook) in which each row contains 10 example images from another class that were misclassified. The resulting grid will have 9 rows (one for each class other than the input class) and 10 examples. For example the row for truck would have images of trucks that were misclassified as dogs. Show the output of this function for the SVM classifier and the class horse.

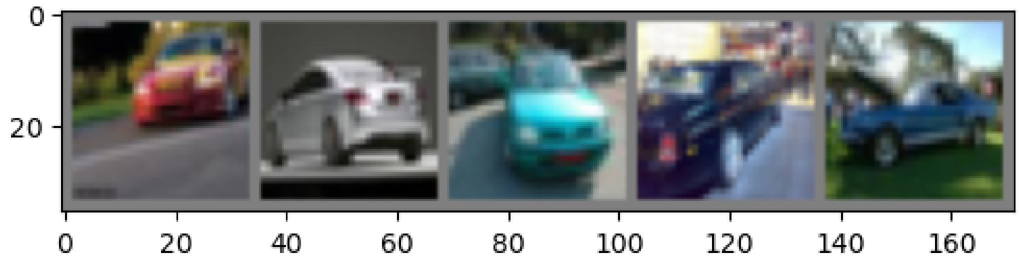
[6]:
```
# Your answer goes here
import numpy as np
classes = ['airplane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

```
def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.figure(figsize=(10,10))
    plt.show()

def misclass_image(pred_output, aclass): #takes a classifier's predicted␣
 ↪output, correct ouput, and the class/label (integer)
    exp_index = [i for i in range(len(testset)) if testset[i][1] == aclass]␣
 ↪#gather the selected class's expected indices
    mis_index = [i for i in range(len(pred_output)) if pred_output[i] == aclass␣
 ↪and i not in exp_index] #gather the misclassified image's indices
    #start sorting
    misclass_dict = {}
    for i in range(len(list(testset.class_to_idx.items()))): #make each class a␣
 ↪key, and assign an empty list to each key
        if i != aclass:
            misclass_dict[i] = []
    for index in mis_index: #append the misclassified image index to their␣
 ↪corresponding list
        if len(misclass_dict[testset[index][1]]) < 10:
            misclass_dict[testset[index][1]].append(testset[index][0])
    #display image
    for key in misclass_dict.keys():
        imshow(torchvision.utils.make_grid(misclass_dict[key], nrow=10)) #make␣
 ↪sure the misclassified images belong to the right class
        print(classes[key])

misclass_image(ytest_predict_svc, 7) #SVM classifier output, check for horse␣
 ↪class misclassified images
```




```
<Figure size 1000x1000 with 0 Axes>
```

```
airplane
```



<Figure size 1000x1000 with 0 Axes>

car

<Figure size 1000x1000 with 0 Axes>

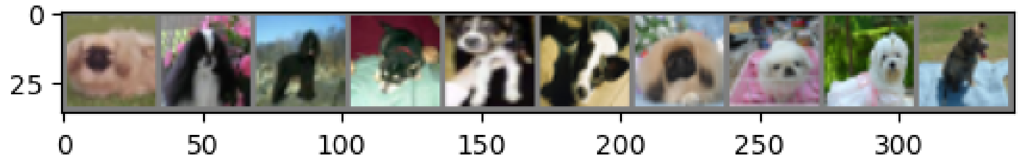
bird

<Figure size 1000x1000 with 0 Axes>

cat

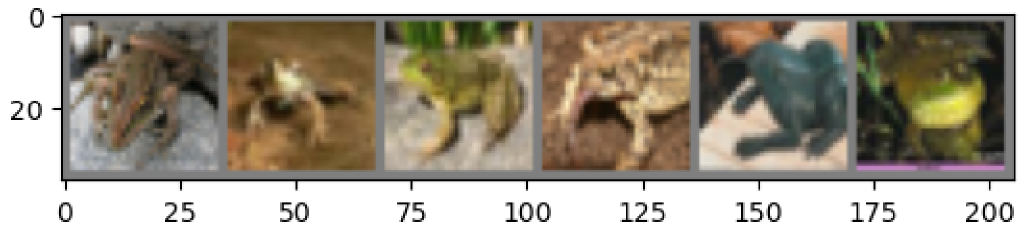
<Figure size 1000x1000 with 0 Axes>

deer



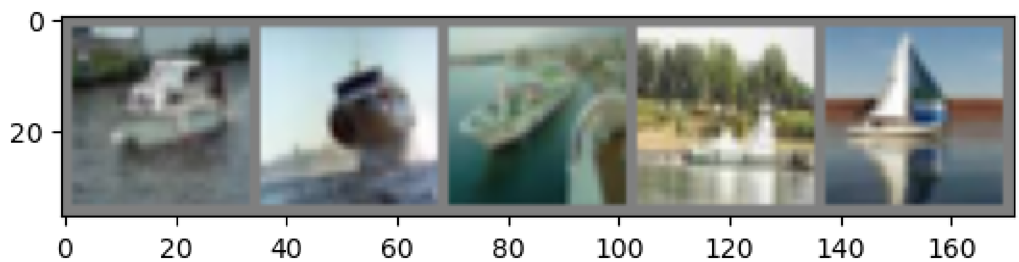
<Figure size 1000x1000 with 0 Axes>

dog



<Figure size 1000x1000 with 0 Axes>

frog



<Figure size 1000x1000 with 0 Axes>

ship

```
<Figure size 1000x1000 with 0 Axes>
```

```
truck
```

5 Question 6 (Expected) - Comparison of different features

In the deep learning notebook the CIFAR-10 classification code using SVM utilizes a histogram of oriented gradients features followed by a PCA transformation for dimensionality reduction. Using the mini-CIFAR-10 dataset compare the following three feature front-ends using SVM classificaiton (use the same parameters as the deep learning notebook):

1. Flatten the training images to a single (32 * 32 * 3) vector
2. Compute the Histogram of Gradients
3. Computer the Histogram of Gradients followed by PCA (as done in the deep learning notebook).

Compare these three feature front ends by showing the corresponding classification accurarcy and confusion matrices for each one.

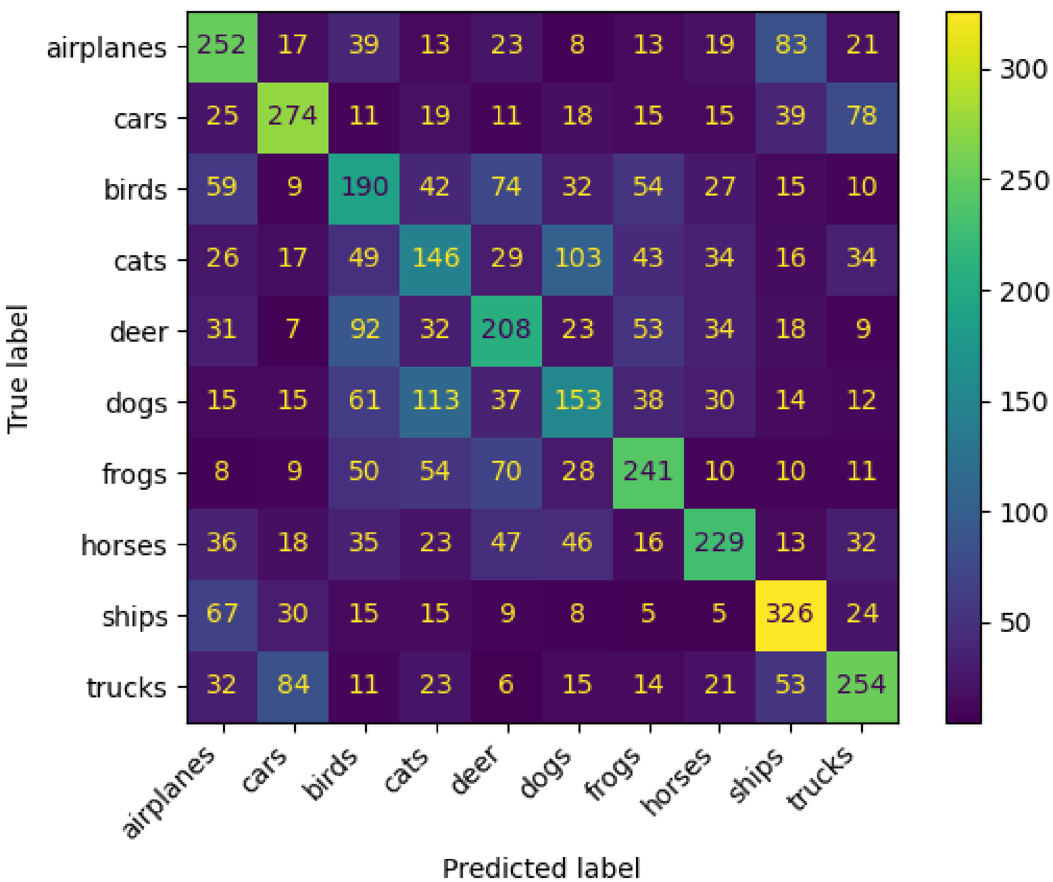
5.0.1 Flatten the training images to a single (32 * 32 * 3) vector

[7]:
```
# Your code goes here
# flatten training images to a single (32*32*3) vector
x_train_flat = x_train.reshape(-1, 32 * 32 * 3) #flattened shape
x_test_flat = x_test.reshape(-1, 32 * 32 * 3)
print("Images haven been flattened to a single (32*32*3) vector, proceeding to␣
↪classification report and confusion matrix")
```

```
Images haven been flattened to a single (32*32*3) vector, proceeding to
classification report and confusion matrix
```

[8]:
```
#show classification report and confusion matrix when images are flattened
#It may take some time to train SVM
clf_flat = svm.SVC(C=10, cache_size=5000)
clf_flat.fit(x_train_flat, y_train)
ytest_predict_svc_flat  = clf_flat.predict(x_test_flat)

print(classification_report(y_test, ytest_predict_svc_flat))

color = 'white'
cm_svc_flat = confusion_matrix(y_test, ytest_predict_svc_flat)
disp = ConfusionMatrixDisplay(cm_svc_flat, display_labels=['airplanes', 'cars',␣
↪'birds', 'cats', 'deer', 'dogs', 'frogs', 'horses', 'ships', 'trucks'])
disp.plot()
plt.xticks(rotation=45, ha='right')
plt.show()
```

	precision	recall	f1-score	support
0	0.46	0.52	0.49	488
1	0.57	0.54	0.56	505
2	0.34	0.37	0.36	512
3	0.30	0.29	0.30	497
4	0.40	0.41	0.41	507
5	0.35	0.31	0.33	488
6	0.49	0.49	0.49	491
7	0.54	0.46	0.50	495
8	0.56	0.65	0.60	504
9	0.52	0.50	0.51	513
accuracy			0.45	5000
macro avg	0.45	0.45	0.45	5000
weighted avg	0.45	0.45	0.45	5000



5.0.2 Compute the Histogram of Gradients

[9]:
```
# Compute the Histogram of Gradients
xtrain_hog = []
for i in range(len(x_train)):
    fd  = hog(x_train[i] , orientations=9 , pixels_per_cell = (8,8),
                     cells_per_block = (2,2) , visualize = False,␣
↪channel_axis=-1)
    xtrain_hog.append(fd)
    if ((i % 500) == 0):
        print(i)
xtrain_hog = np.array(xtrain_hog)
print('Finished HOG on training data')
print(xtrain_hog.shape)

xtest_hog = []
for i in range(len(x_test)):
    fd = hog(x_test[i] , orientations=9 , pixels_per_cell = (8,8),
                     cells_per_block = (2,2) , visualize = False,␣
↪channel_axis=-1)
    xtest_hog.append(fd)
    if ((i % 500) == 0):
        print(i)
xtest_hog = np.array(xtest_hog)
print('Finished HOG on testing data')
print(xtest_hog.shape)
```

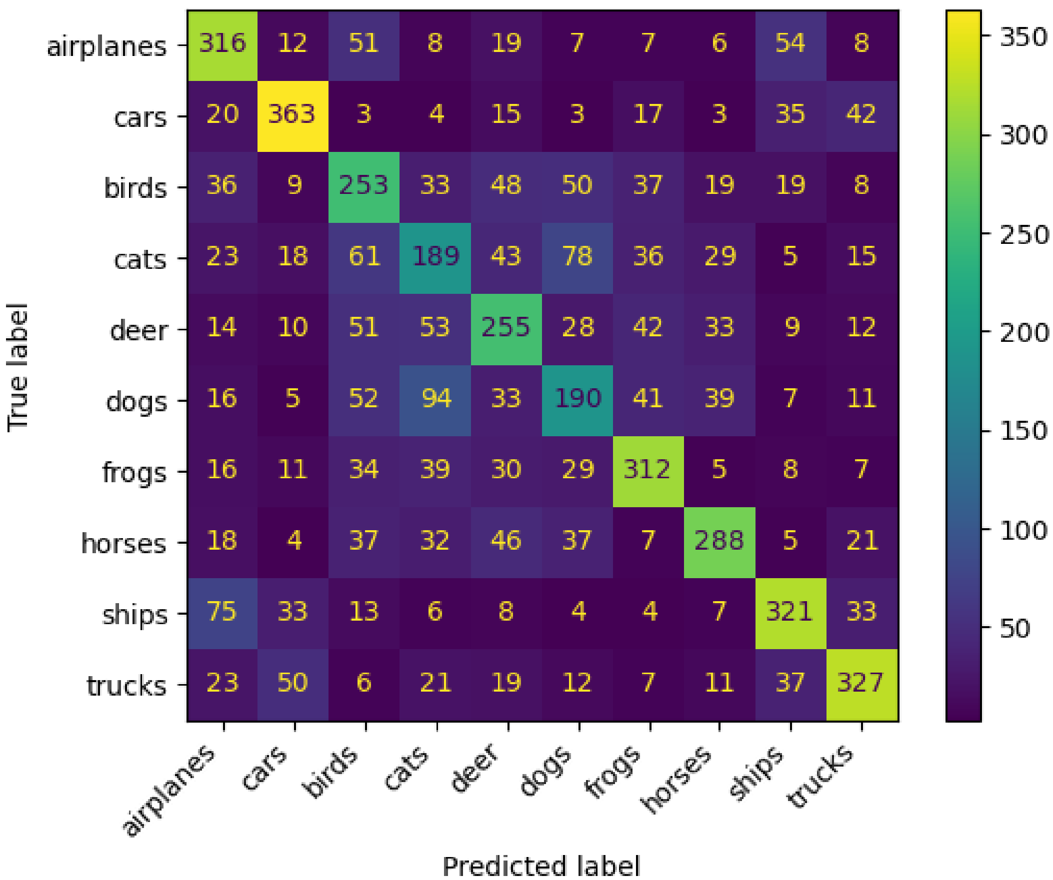
```
0
500
1000
1500
2000
2500
3000
3500
4000
4500
Finished HOG on training data
(5000, 324)
0
500
1000
1500
2000
2500
3000
3500
4000
```

```
4500
Finished HOG on testing data
(5000, 324)
```

[10]:

```python
#Feed it to SVM
clf_hog = svm.SVC(C=10, cache_size=5000)
clf_hog.fit(xtrain_hog, y_train)
ytest_predict_svc_hog  = clf_hog.predict(xtest_hog)
print(classification_report(y_test, ytest_predict_svc_hog)) #show␣
↪classification report and confusion matrix when HOG is applied

color = 'white'
cm_svc_hog = confusion_matrix(y_test, ytest_predict_svc_hog)
disp = ConfusionMatrixDisplay(cm_svc_hog, display_labels=['airplanes', 'cars',␣
↪'birds', 'cats', 'deer', 'dogs', 'frogs', 'horses', 'ships', 'trucks'])
disp.plot()
plt.xticks(rotation=45, ha='right')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.57      0.65      0.60       488
           1       0.70      0.72      0.71       505
           2       0.45      0.49      0.47       512
           3       0.39      0.38      0.39       497
           4       0.49      0.50      0.50       507
           5       0.43      0.39      0.41       488
           6       0.61      0.64      0.62       491
           7       0.65      0.58      0.62       495
           8       0.64      0.64      0.64       504
           9       0.68      0.64      0.66       513

    accuracy                           0.56      5000
   macro avg       0.56      0.56      0.56      5000
weighted avg       0.56      0.56      0.56      5000
```



5.0.3 Computer the Histogram of Gradients followed by PCA

[11]:
```
xtrain_hog = []
for i in range(len(x_train)):
    fd  = hog(x_train[i] , orientations=9 , pixels_per_cell = (8,8),
                     cells_per_block = (2,2) , visualize = False,␣
 ↪channel_axis=-1)
    xtrain_hog.append(fd)
xtrain_hog = np.array(xtrain_hog)
print('Finished HOG on training data')
print(xtrain_hog.shape)

xtest_hog = []
for i in range(len(x_test)):
    fd = hog(x_test[i] , orientations=9 , pixels_per_cell = (8,8),
                     cells_per_block = (2,2) , visualize = False,␣
 ↪channel_axis=-1)
    xtest_hog.append(fd)
```

```
xtest_hog = np.array(xtest_hog)
print('Finished HOG on testing data')
print(xtest_hog.shape)
#PCA
pca = PCA(0.8)
print("PCA starts")
xtrain_pca = pca.fit_transform(xtrain_hog)
xtest_pca  = pca.transform(xtest_hog)
print(xtrain_pca.shape)
print(xtest_pca.shape)
print("HOG followed by PCA completed, proceed to the classification report and␣
↪confusion matrix")
```

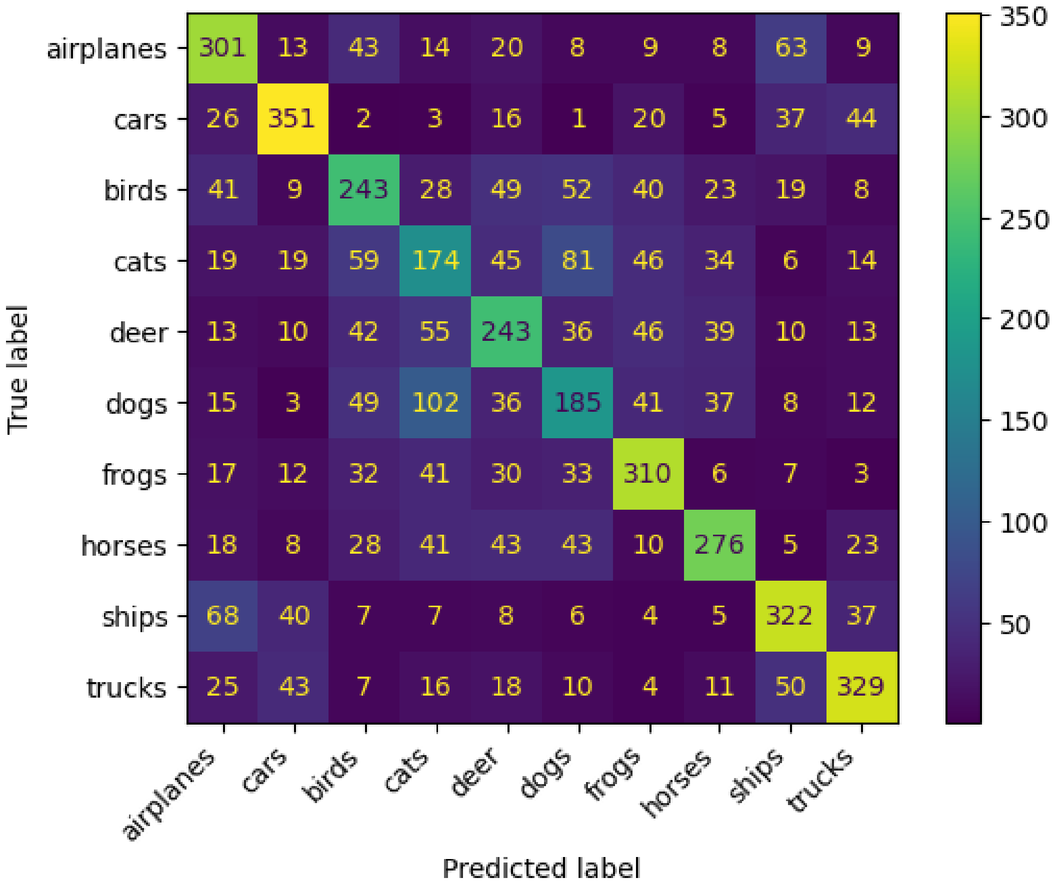
```
Finished HOG on training data
(5000, 324)
Finished HOG on testing data
(5000, 324)
PCA starts
(5000, 63)
(5000, 63)
HOG followed by PCA completed, proceed to the classification report and
confusion matrix
```

[12]:

```
clf = svm.SVC(C=10, cache_size=10000)
clf.fit(xtrain_pca, y_train)

ytest_predict  = clf.predict(xtest_pca)
print(classification_report(y_test, ytest_predict))

color = 'white'
cm = confusion_matrix(y_test, ytest_predict)
disp = ConfusionMatrixDisplay(cm, display_labels=['airplanes', 'cars', 'birds',␣
↪'cats', 'deer', 'dogs', 'frogs', 'horses', 'ships', 'trucks'])
disp.plot()
plt.xticks(rotation=45, ha='right')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.55      0.62      0.58       488
           1       0.69      0.70      0.69       505
           2       0.47      0.47      0.47       512
           3       0.36      0.35      0.36       497
           4       0.48      0.48      0.48       507
           5       0.41      0.38      0.39       488
           6       0.58      0.63      0.61       491
           7       0.62      0.56      0.59       495
```

8	0.61	0.64	0.62	504
9	0.67	0.64	0.65	513
accuracy			0.55	5000
macro avg	0.55	0.55	0.55	5000
weighted avg	0.55	0.55	0.55	5000



6 QUESTION 7 (Basic) - Deep learning classification

Retrain the deep neural network specified in the deep learning notebook. You will need to install PyTorch for your system. You don't need to use the GPU unless you have one and can set it up. Your training time will depend on your hardware on my laptop with CPU it takes about 4 minutes and with GPU about 2 minutes. It should not be more than 30 minutes even on an old slow laptop. Another option is to use Google Colab.

Once you have trained and evaluated the network and got numbers similar to the deep learning notebook change the batch size to 16. Repeat the training and report on how the accuracy and training time changed.

[4]:
```python
# Your code goes here
import torch
import torchvision
import torchvision.transforms as transforms
torch.cuda.is_available()
# print(torch.cuda.get_device_name(0)) #don't have a Nvidia gpu unfortunately

if torch.cuda.is_available():
    dev = "cuda:0"
else:
    dev = "cpu"

#dev = "cpu"
device = torch.device(dev)
dev = "cuda"
print(device)
```

```
cpu
```

[5]:
```python
#If ModuleNotFoundError is found, run "pip install torchsummary" or "pip␣
↪install torchsummary --no-cache-dir"
import torch.nn as nn
import torch.nn.functional as F
from torchsummary import summary

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3,32,5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 32, 5)
        self.fc1 = nn.Linear(32 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
net.to(device)
```

```python
print(net)

summary(net, (3,32,32), batch_size=32, device=dev)
```

```
Net(
  (conv1): Conv2d(3, 32, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv2): Conv2d(32, 32, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=800, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [32, 32, 28, 28]           2,432
         MaxPool2d-2           [32, 32, 14, 14]               0
            Conv2d-3           [32, 32, 10, 10]          25,632
         MaxPool2d-4             [32, 32, 5, 5]               0
            Linear-5                  [32, 120]          96,120
            Linear-6                   [32, 84]          10,164
            Linear-7                   [32, 10]             850
================================================================
Total params: 135,198
Trainable params: 135,198
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.38
Forward/backward pass size (MB): 8.69
Params size (MB): 0.52
Estimated Total Size (MB): 9.58
----------------------------------------------------------------
```

[6]:
```python
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

[13]:
```python
import time
start_time = time.time()
for epoch in range(20):  # loop over the dataset multiple times, it will take␣
 ↪roughly 30 min

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
```

```
        inputs, labels = data
        inputs, labels = inputs.to(device,non_blocking=True), labels.to(device,␣
 ↪non_blocking=True)
        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

        if i % 100 == 99:    # print every 100 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 1500))
            running_loss = 0.0
finished_time = time.time()
print('Finished Training for 32 batch size')
print("Execution time: ", finished_time-start_time)
```

```
[1,   100] loss: 0.057
[2,   100] loss: 0.056
[3,   100] loss: 0.052
[4,   100] loss: 0.052
[5,   100] loss: 0.050
[6,   100] loss: 0.046
[7,   100] loss: 0.047
[8,   100] loss: 0.042
[9,   100] loss: 0.042
[10,   100] loss: 0.038
[11,   100] loss: 0.036
[12,   100] loss: 0.035
[13,   100] loss: 0.031
[14,   100] loss: 0.033
[15,   100] loss: 0.027
[16,   100] loss: 0.027
[17,   100] loss: 0.025
[18,   100] loss: 0.021
[19,   100] loss: 0.021
[20,   100] loss: 0.019
Finished Training for 32 batch size
Execution time:  1898.6060237884521
```

[14]:
```
def test_accuracy(net, testloader, device):
    correct = 0

    # since we're not training, we don't need to calculate the gradients for␣
↪our outputs
    with torch.no_grad():
        net.eval()
        for images, labels in testloader:
            images, labels = images.to(device), labels.to(device)

            # calculate outputs by running images through the network
            outputs = net(images)

            # the class with the highest energy is what we choose as prediction
            predicted = torch.max(outputs.data, 1)[1]

            correct += (predicted == labels).sum().item()

    return correct / len(testloader.dataset)


def test_accuracy_per_class(net, testloader, device):
    correct_pred = {classname: 0 for classname in trainset.classes}
    total_pred = {classname: 0 for classname in trainset.classes}

    with torch.no_grad():
        net.eval()
        for images, labels in testloader:
            images, labels = images.to(device), labels.to(device)

            outputs = net(images)
            predicted = torch.max(outputs.data, 1)[1]

            # collect the correct predictions for each class
            for label, prediction in zip(labels, predicted):
                if label == prediction:
                    correct_pred[trainset.classes[label]] += 1
                total_pred[trainset.classes[label]] += 1

    accuracy_per_class = {classname: 0 for classname in trainset.classes}
    for classname, correct_count in correct_pred.items():
        accuracy = (100 * float(correct_count)) / total_pred[classname]
        accuracy_per_class[classname] = accuracy

    return accuracy_per_class
```

```
test_acc = test_accuracy(net, testloader, 'cpu')
print(f'Best trial test set accuracy: {test_acc}')

overall_accuracy = test_accuracy(net, testloader, 'cpu')

print(
    'Overall accuracy of the network  '
    f'{(overall_accuracy * 100):.2f} %\n'
    'on the 5000 test images'
)

accuracy_per_class = test_accuracy_per_class(net, testloader, 'cpu')

print('Accuracy per class\n')
for classname, accuracy in accuracy_per_class.items():
    print(f'{classname:12s} {accuracy:.2f} %')
```

```
Best trial test set accuracy: 0.5396
Overall accuracy of the network  53.96 %
on the 5000 test images
Accuracy per class

airplane     54.30 %
automobile   72.48 %
bird         37.70 %
cat          47.28 %
deer         43.20 %
dog          39.14 %
frog         63.95 %
horse        52.53 %
ship         72.82 %
truck        56.14 %
```

Start to train it in 16 batch size

[15]:
```
trainloader_16 = torch.utils.data.DataLoader(trainset, batch_size=16,
                                          shuffle=True, num_workers=4)
testloader_16 = torch.utils.data.DataLoader(testset, batch_size=16,
                                         shuffle=False, num_workers=4)
net_16 = Net()
net_16.to(device)
print(net_16)

summary(net, (3,32,32), batch_size=16, device=dev)
optimizer_16 = optim.SGD(net_16.parameters(), lr=0.001, momentum=0.9)
```

```
Net(
  (conv1): Conv2d(3, 32, kernel_size=(5, 5), stride=(1, 1))
```

```
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv2): Conv2d(32, 32, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=800, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [16, 32, 28, 28]           2,432
         MaxPool2d-2           [16, 32, 14, 14]               0
            Conv2d-3           [16, 32, 10, 10]          25,632
         MaxPool2d-4             [16, 32, 5, 5]               0
            Linear-5                  [16, 120]          96,120
            Linear-6                   [16, 84]          10,164
            Linear-7                   [16, 10]             850
================================================================
Total params: 135,198
Trainable params: 135,198
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.19
Forward/backward pass size (MB): 4.34
Params size (MB): 0.52
Estimated Total Size (MB): 5.05
----------------------------------------------------------------
```

[16]:
```python
start_time = time.time()
for epoch in range(20):  # loop over the dataset multiple times, it should take␣
 ↪around 30 min

    running_loss = 0.0
    for i, data in enumerate(trainloader_16, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        inputs, labels = inputs.to(device,non_blocking=True), labels.to(device,␣
 ↪non_blocking=True)
        # zero the parameter gradients
        optimizer_16.zero_grad()

        # forward + backward + optimize
        outputs = net_16(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer_16.step()
```

```python
        # print statistics
        running_loss += loss.item()

        if i % 100 == 99:    # print every 100 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 1500))
            running_loss = 0.0
finished_time = time.time()
print('Finished Training for 16 batch size')
print("Execution time: ", finished_time-start_time)
```

```
[1,   100] loss: 0.154
[1,   200] loss: 0.154
[1,   300] loss: 0.153
[2,   100] loss: 0.153
[2,   200] loss: 0.152
[2,   300] loss: 0.151
[3,   100] loss: 0.148
[3,   200] loss: 0.146
[3,   300] loss: 0.142
[4,   100] loss: 0.137
[4,   200] loss: 0.134
[4,   300] loss: 0.133
[5,   100] loss: 0.130
[5,   200] loss: 0.129
[5,   300] loss: 0.125
[6,   100] loss: 0.125
[6,   200] loss: 0.122
[6,   300] loss: 0.119
[7,   100] loss: 0.118
[7,   200] loss: 0.117
[7,   300] loss: 0.110
[8,   100] loss: 0.112
[8,   200] loss: 0.111
[8,   300] loss: 0.107
[9,   100] loss: 0.105
[9,   200] loss: 0.108
[9,   300] loss: 0.106
[10,   100] loss: 0.102
[10,   200] loss: 0.104
[10,   300] loss: 0.103
[11,   100] loss: 0.101
[11,   200] loss: 0.097
[11,   300] loss: 0.102
[12,   100] loss: 0.098
[12,   200] loss: 0.098
[12,   300] loss: 0.096
```

```
[13,   100] loss: 0.096
[13,   200] loss: 0.095
[13,   300] loss: 0.091
[14,   100] loss: 0.092
[14,   200] loss: 0.091
[14,   300] loss: 0.092
[15,   100] loss: 0.088
[15,   200] loss: 0.092
[15,   300] loss: 0.088
[16,   100] loss: 0.085
[16,   200] loss: 0.087
[16,   300] loss: 0.086
[17,   100] loss: 0.082
[17,   200] loss: 0.082
[17,   300] loss: 0.083
[18,   100] loss: 0.081
[18,   200] loss: 0.078
[18,   300] loss: 0.084
[19,   100] loss: 0.079
[19,   200] loss: 0.078
[19,   300] loss: 0.078
[20,   100] loss: 0.074
[20,   200] loss: 0.076
[20,   300] loss: 0.075
Finished Training for 16 batch size
Execution time:  2702.878611087799
```

[17]:
```python
test_acc = test_accuracy(net_16, testloader_16, 'cpu')
print(f'Best trial test set accuracy: {test_acc}')

overall_accuracy = test_accuracy(net_16, testloader_16, 'cpu')

print(
    'Overall accuracy of the network  '
    f'{(overall_accuracy * 1):.2f} %\n'
    'on the 5000 test images'
)

accuracy_per_class = test_accuracy_per_class(net_16, testloader_16, 'cpu')

print('Accuracy per class\n')
for classname, accuracy in accuracy_per_class.items():
    print(f'{classname:12s} {accuracy:.2f} %')
```

```
Best trial test set accuracy: 0.527
Overall accuracy of the network  52.70 %
on the 5000 test images
Accuracy per class
```

```
airplane     55.74 %
automobile   67.72 %
bird         43.55 %
cat          34.41 %
deer         39.64 %
dog          37.30 %
frog         62.73 %
horse        68.89 %
ship         67.46 %
truck        49.71 %
```

We can see that the classification accuracy is slightly decreased and the training time is increased.

7 QUESTION 8 (EXPECTED) - Deep learning classification of noisy images

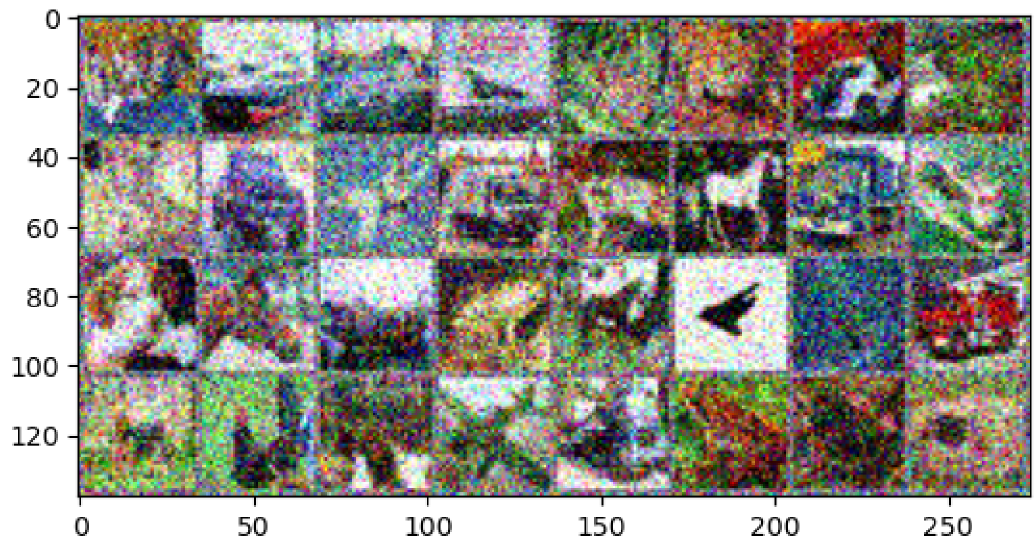
In this question you will explore the effect of adding noise to the classification of the CIFAR-10 dataset. You can add random noise with a mean of 0 and standard deviation of 1 to a tensor using the *torch.randn*. For example: x = x + torch.randn(x.shape) will add noise to the tensor x. Add noise with a mean of 0 and a standard deviation of 0.2 to the images of the CIFAR-10 dataset. First see how the images with the added noise will look by adding the noise in the *imshow* function. Then check how the classification accuracy on the test set is affected if you add noise to the test but NOT the training set.

[86]:
```python
# Your code goes here
import matplotlib.pyplot as plt
import numpy as np
def imshow_noise(img): #adding noise to image to see effect
    img = (img / 2 + 0.5) + 0.2 * torch.randn(img.shape)     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()
dataiter = iter(testloader)
images_noise, labels = next(dataiter)
imshow_noise(torchvision.utils.make_grid(images_noise))

#One way to add noise is through modifying the testing accuracy function
def test_accuracy(net, testloader, device):
    correct = 0

    # since we're not training, we don't need to calculate the gradients for␣
↪our outputs
    with torch.no_grad():
        net.eval()
        for images, labels in testloader:
```

```
            images, labels = (images+torch.rand(images.shape)*0.2).to(device),␣
 ↪labels.to(device)

            # calculate outputs by running images through the network
            outputs = net(images)

            # the class with the highest energy is what we choose as prediction
            predicted = torch.max(outputs.data, 1)[1]

            correct += (predicted == labels).sum().item()

    return correct / len(testloader.dataset)


def test_accuracy_per_class(net, testloader, device):
    correct_pred = {classname: 0 for classname in trainset.classes}
    total_pred = {classname: 0 for classname in trainset.classes}

    with torch.no_grad():
        net.eval()
        for images, labels in testloader:
            images, labels = (images+torch.rand(images.shape)*0.2).to(device),␣
 ↪labels.to(device)

            outputs = net(images)
            predicted = torch.max(outputs.data, 1)[1]

            # collect the correct predictions for each class
            for label, prediction in zip(labels, predicted):
                if label == prediction:
                    correct_pred[trainset.classes[label]] += 1
                total_pred[trainset.classes[label]] += 1

    accuracy_per_class = {classname: 0 for classname in trainset.classes}
    for classname, correct_count in correct_pred.items():
        accuracy = (100 * float(correct_count)) / total_pred[classname]
        accuracy_per_class[classname] = accuracy

    return accuracy_per_class


test_acc = test_accuracy(net, testloader, 'cpu')
print(f'Best trial test set accuracy: {test_acc}')

overall_accuracy = test_accuracy(net, testloader, 'cpu')

print(
```

```
    'Overall accuracy of the network  '
    f'{(overall_accuracy * 100):.2f} %\n'
    'on the 5000 test images'
)

accuracy_per_class = test_accuracy_per_class(net, testloader, 'cpu')

print('Accuracy per class\n')
for classname, accuracy in accuracy_per_class.items():
    print(f'{classname:12s} {accuracy:.2f} %')
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```



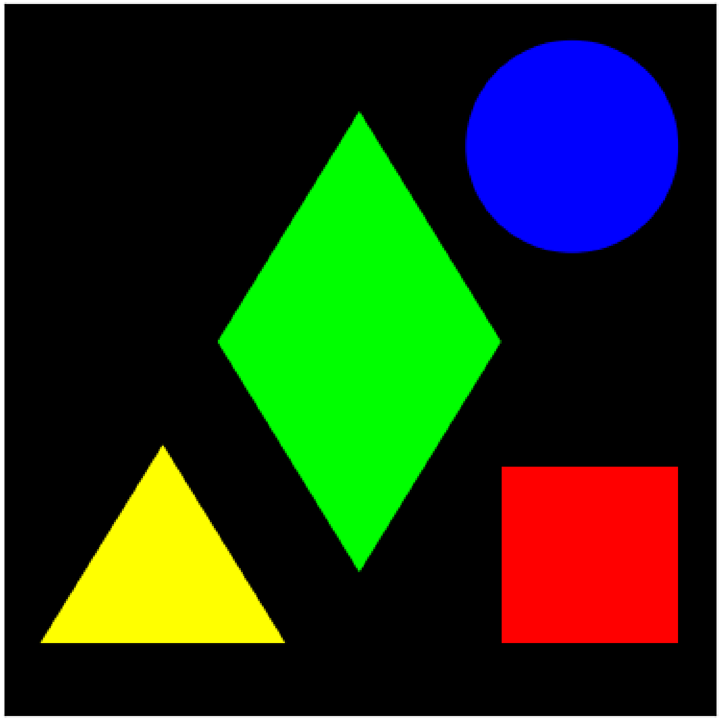
```
Best trial test set accuracy: 0.5268
Overall accuracy of the network  52.76 %
on the 5000 test images
Accuracy per class

airplane     54.92 %
automobile   68.71 %
bird         37.11 %
cat          47.69 %
deer         43.20 %
dog          39.34 %
frog         58.04 %
horse        52.12 %
ship         72.82 %
truck        56.73 %
```

There's a slight decrease in the classification accuracy when noise has been added to images.

8 QUESTION 9 (ADVANCED) - Synthetic generation of dataset

This question is a bit more open ended, will require some creativity and extra work. Your goal is to generate a synthetic dataset of shapes. Below is some code for generating some shapes with matplotlib. Your code should generate random shape using uniform random distributions along the following "dimensions": shape (square, circle, triangle, rhombus), color (red, green, blue, yellow, orange, black), size (continuous but should fit in the image), x-position (continuous but should fit in the image), y_position (continuous but should fit in the image). Once you create a plot you will need to figure out how to convert it to an image. All your images should be 64 by 64 which is bigger than the CIFAR-10 images. Generate a dataset that has 6000 instances of each shape, and 1000 instances of each color within each shape. Show some sample images by appropriately calling/modifying if needed the imshow() function from the deep learning notebook.

[5]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon, Circle, Rectangle

red, blue, yellow, green = '#ff0000', '#0000ff', '#ffff00', '#00ff00'
square = Rectangle((0.7, 0.1), 0.25, 0.25, facecolor=red)
circle = Circle((0.8, 0.8), 0.15, facecolor=blue)
triangle = Polygon(((0.05,0.1), (0.396,0.1), (0.223, 0.38)), fc=yellow)
rhombus = Polygon(((0.5,0.2), (0.7,0.525), (0.5,0.85), (0.3,0.525)),  fc=green)

fig = plt.figure()
ax = fig.add_subplot(111, facecolor='k', aspect='equal')
for shape in (square, circle, triangle, rhombus):
    ax.add_artist(shape)
ax.xaxis.set_visible(False)
ax.yaxis.set_visible(False)

plt.show()
```



9 QUESTION 10 (ADVANCED) - Deep learning for the synthetic shapes

Using the deep learning notebook code as a template build a traditional machine learning classifier using Histogram-of-Oriented Gradients features followed by PCA and using a SVM as a classifier. Train classifiers for the following 3 problems: classify shape (irrespective of color), classify color (irrespective of shape), or classify both color and shape (you can train two SVMs one for each problem). Then repeat the same three configurations using a deep learning neural network. Report on the classification accuracy and confusion matrices for all 6 configurations (shape-SVM, color-SVM, shape+color SVM, shape-DNN, color-DNN, shape+color DNN).

```
[ ]:
```