**COMP3314 Image Classification Report**

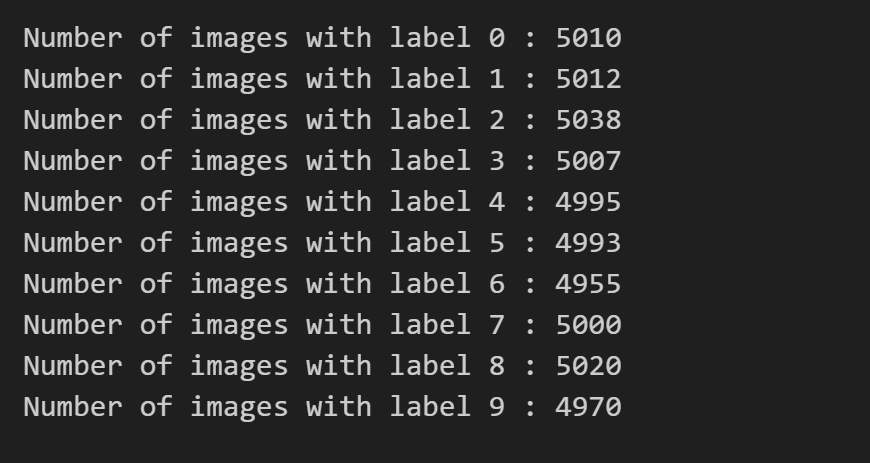
Final Accuracy (public board): 79.15%

Zhong Lin, Shi Boao, Li Junzhe

# **1 Data Analysis**

**1.1 Dataset Statistics**

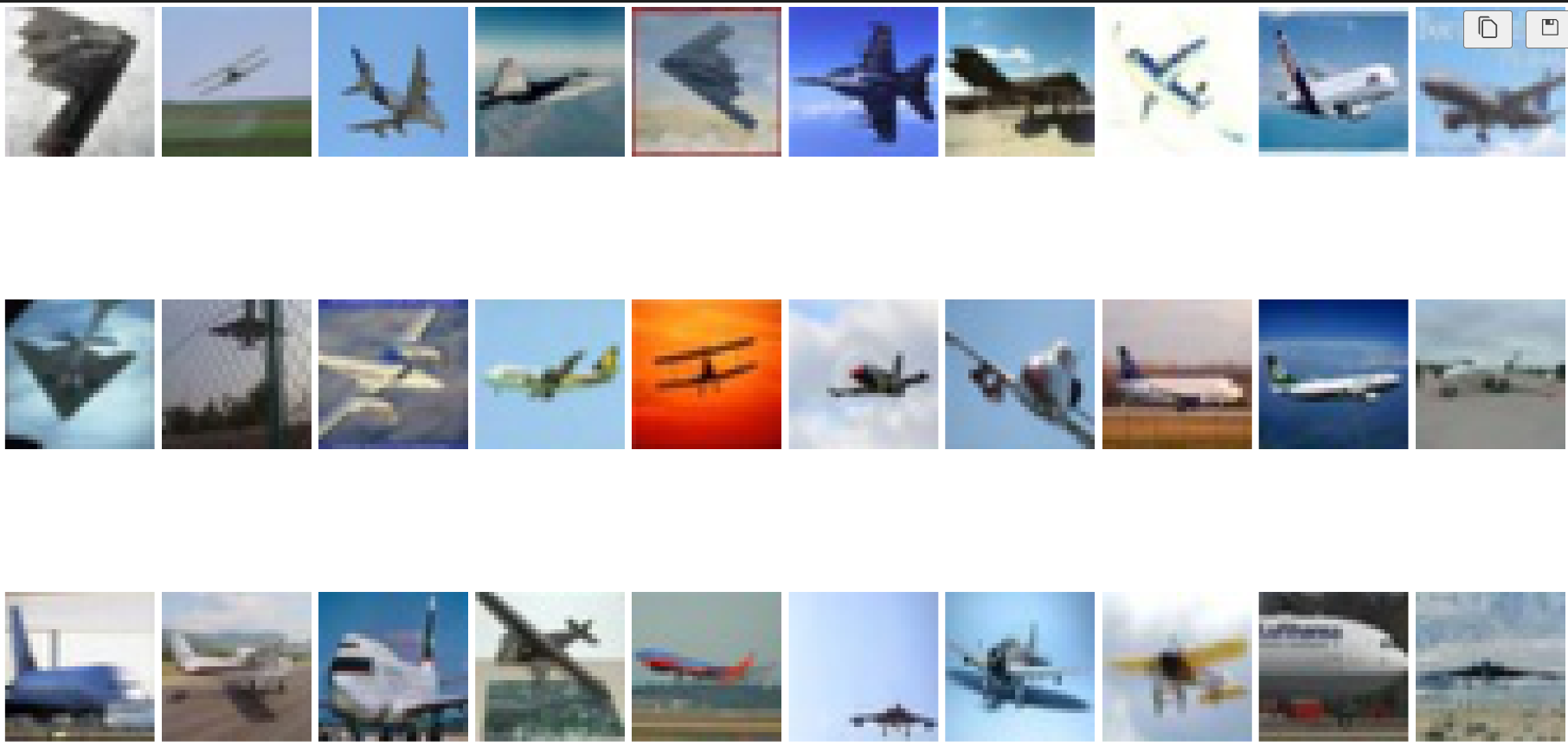
There are 10 classes of images in total. The number of images for each class are as follows:



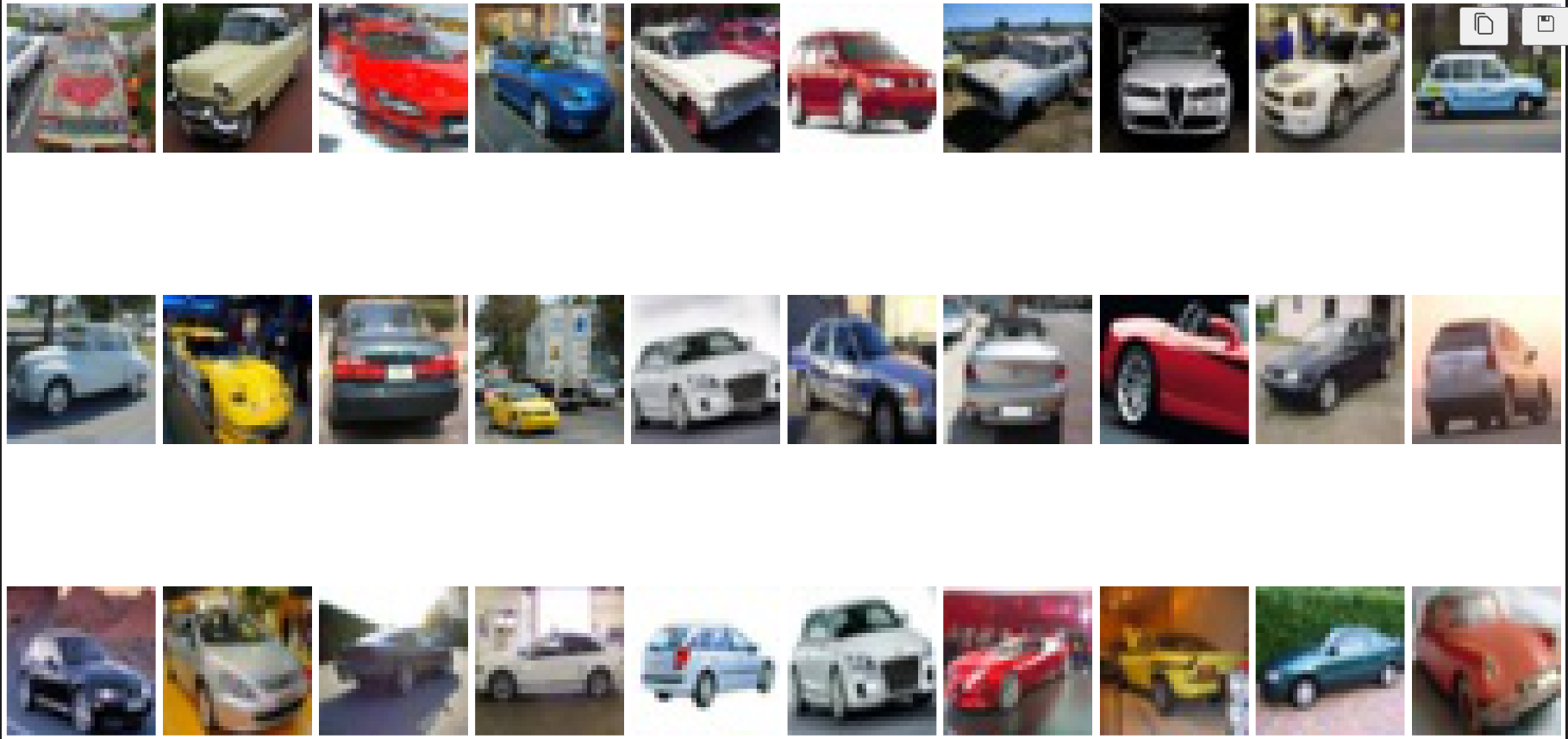
**1.2 Data Visualization**

For each class of images, the examples are as follows:

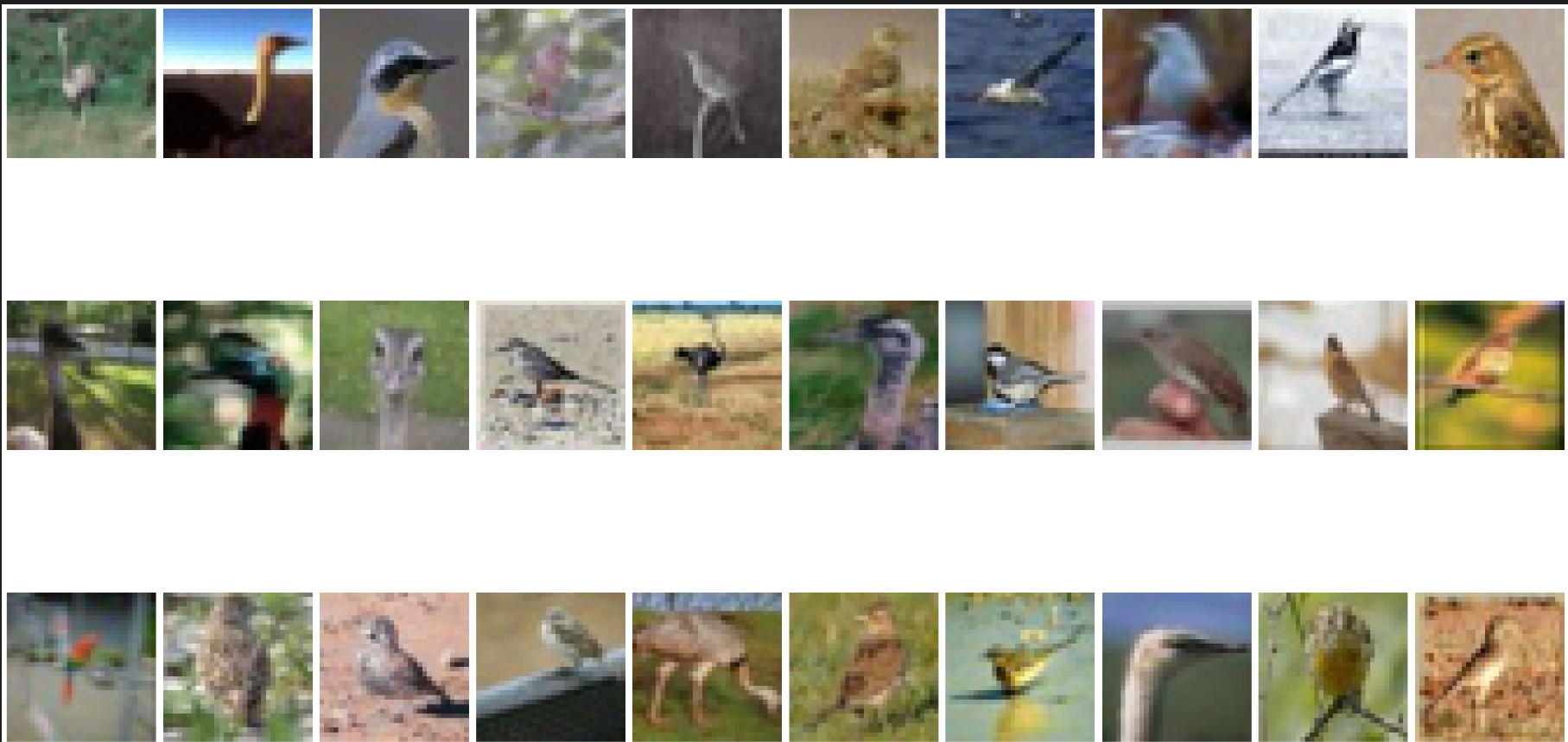
**class 0: airplanes**

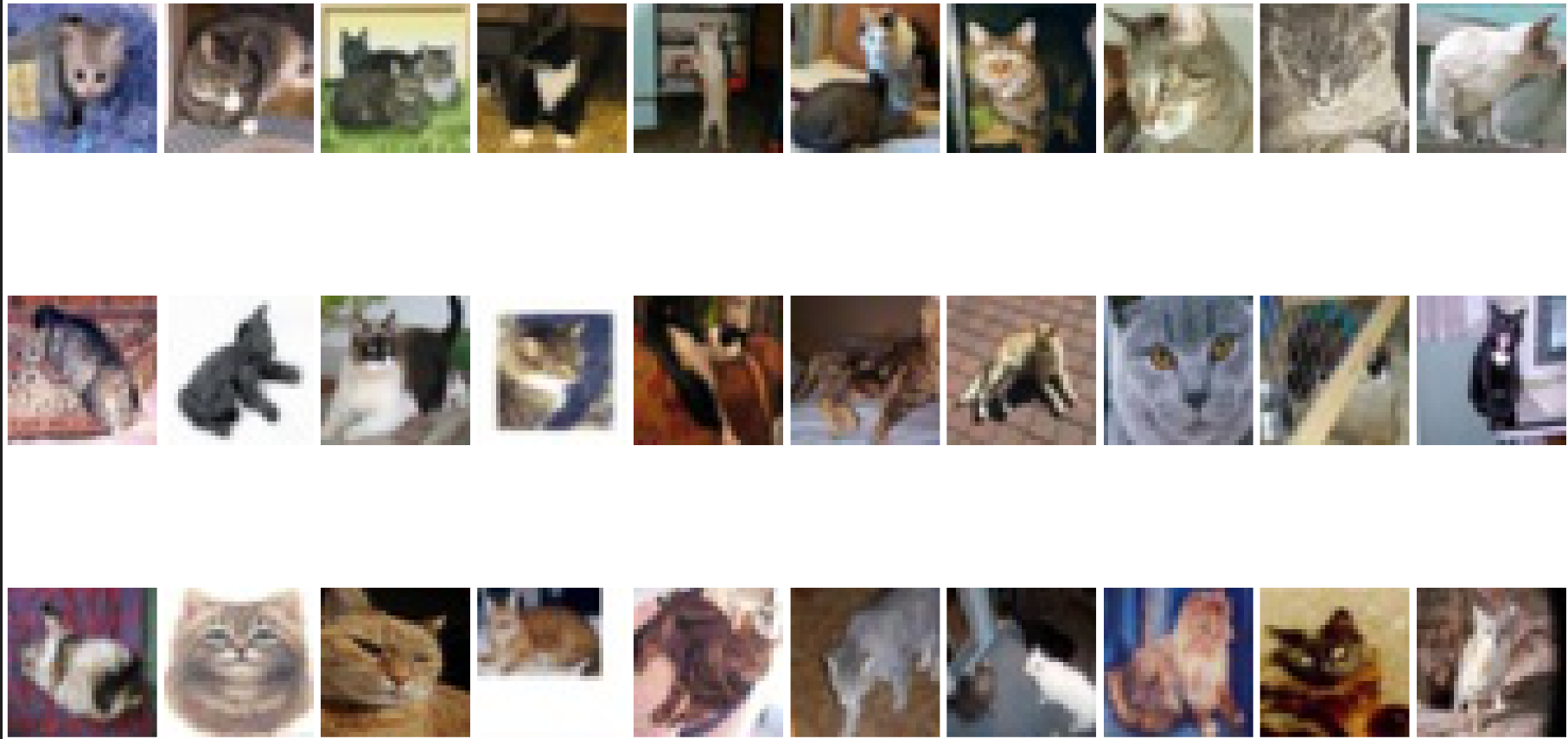


**class 1: cars**

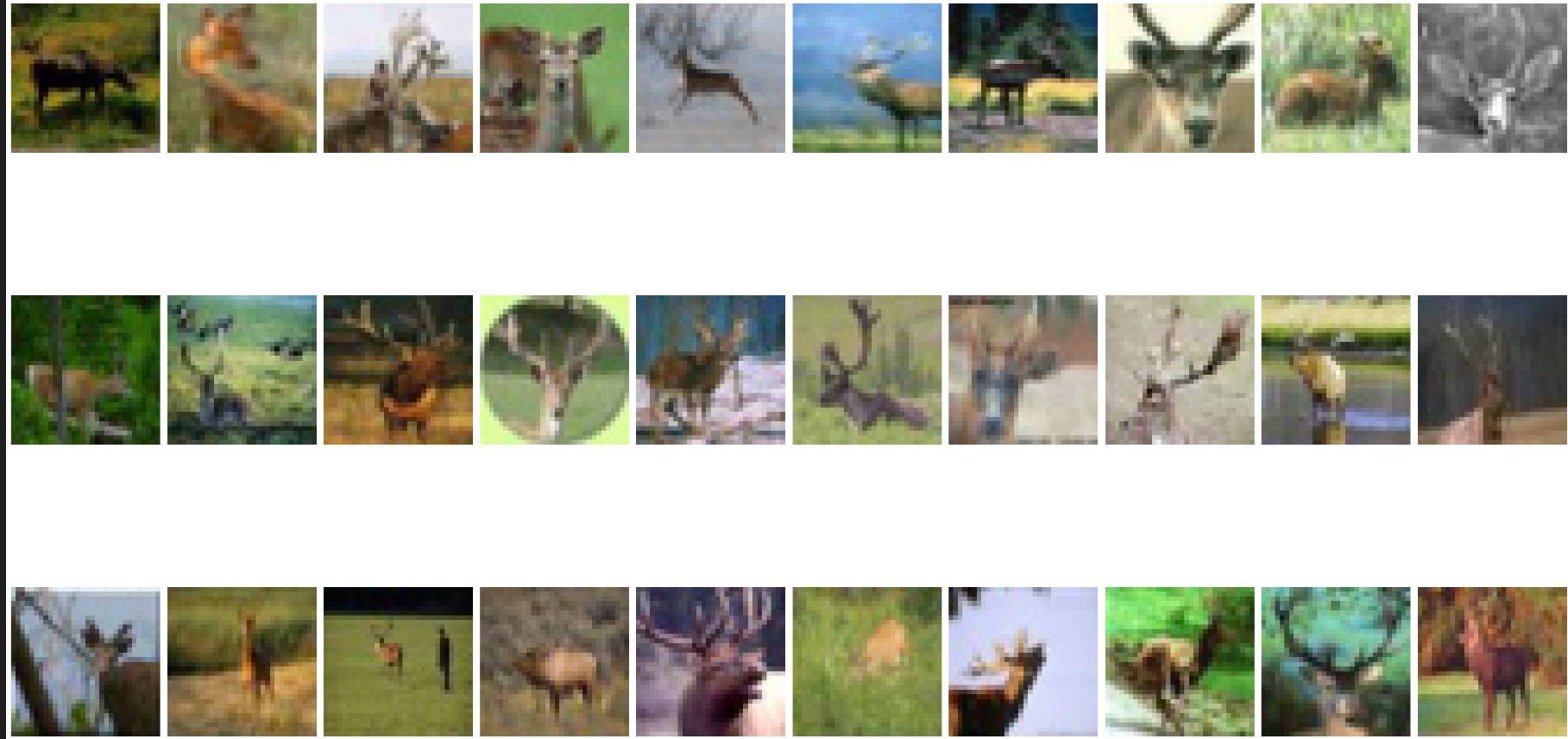


**class 2: birds**

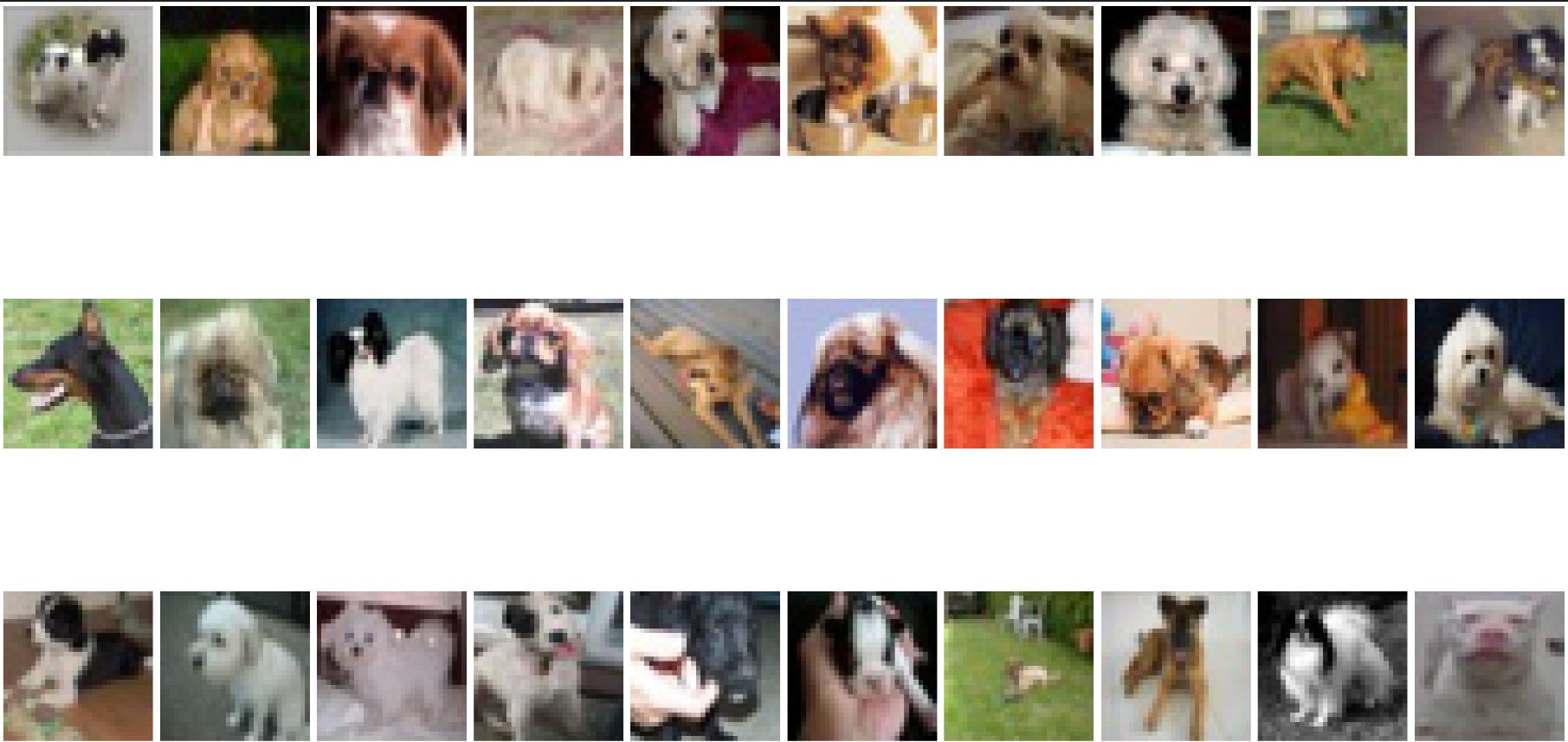


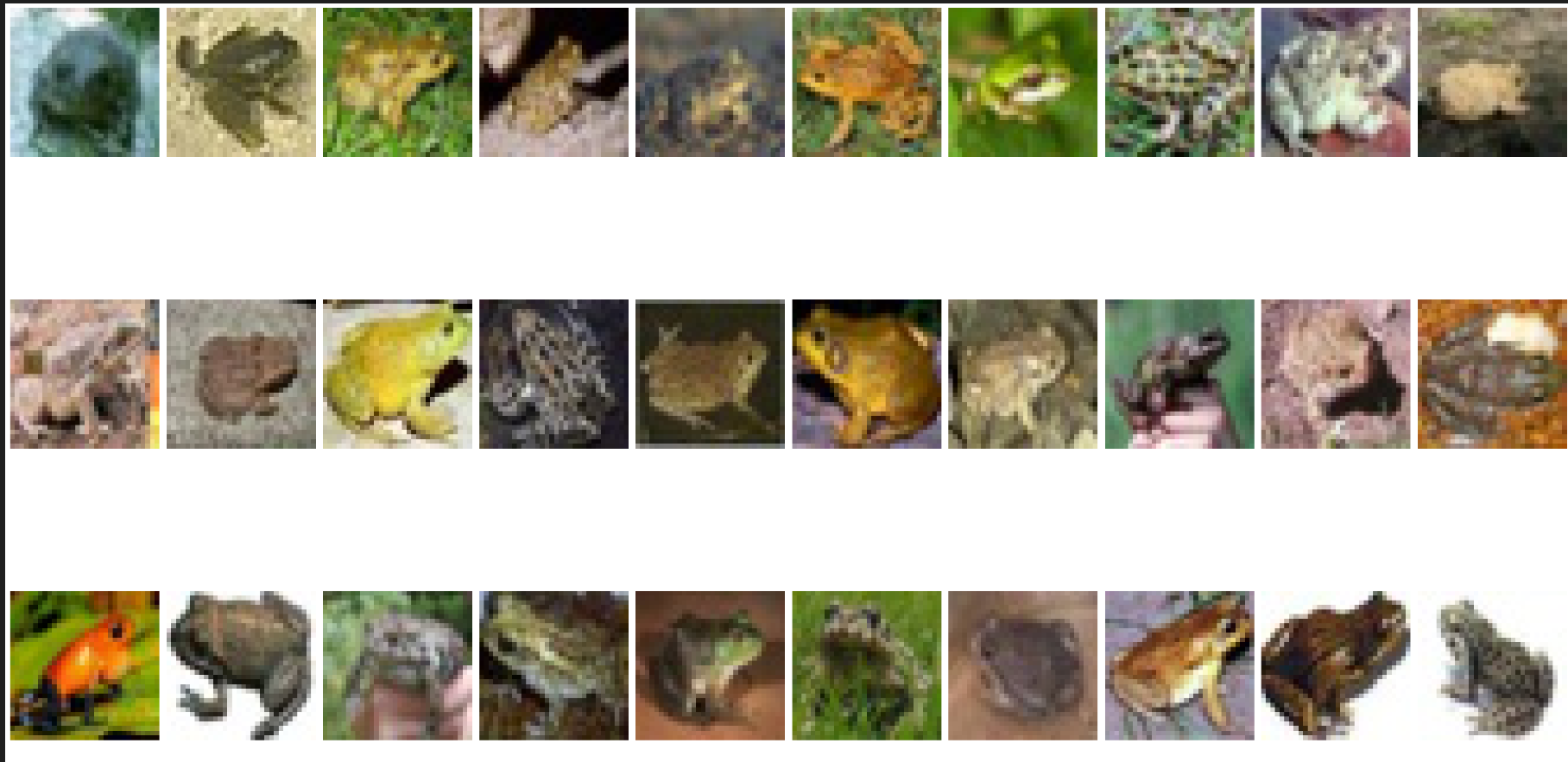
**class 3: cats**

**class 4: dears**

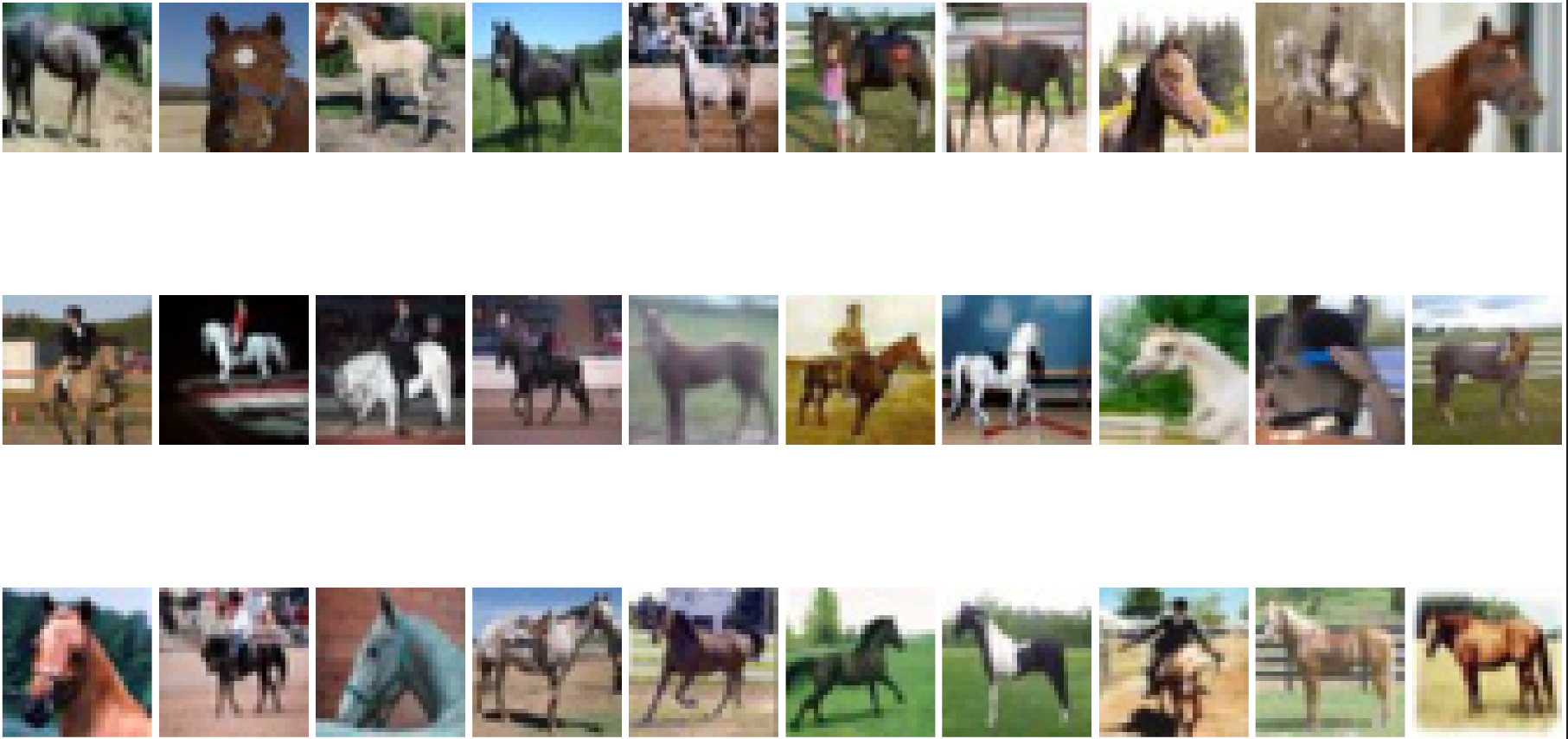


**class 5: dogs**

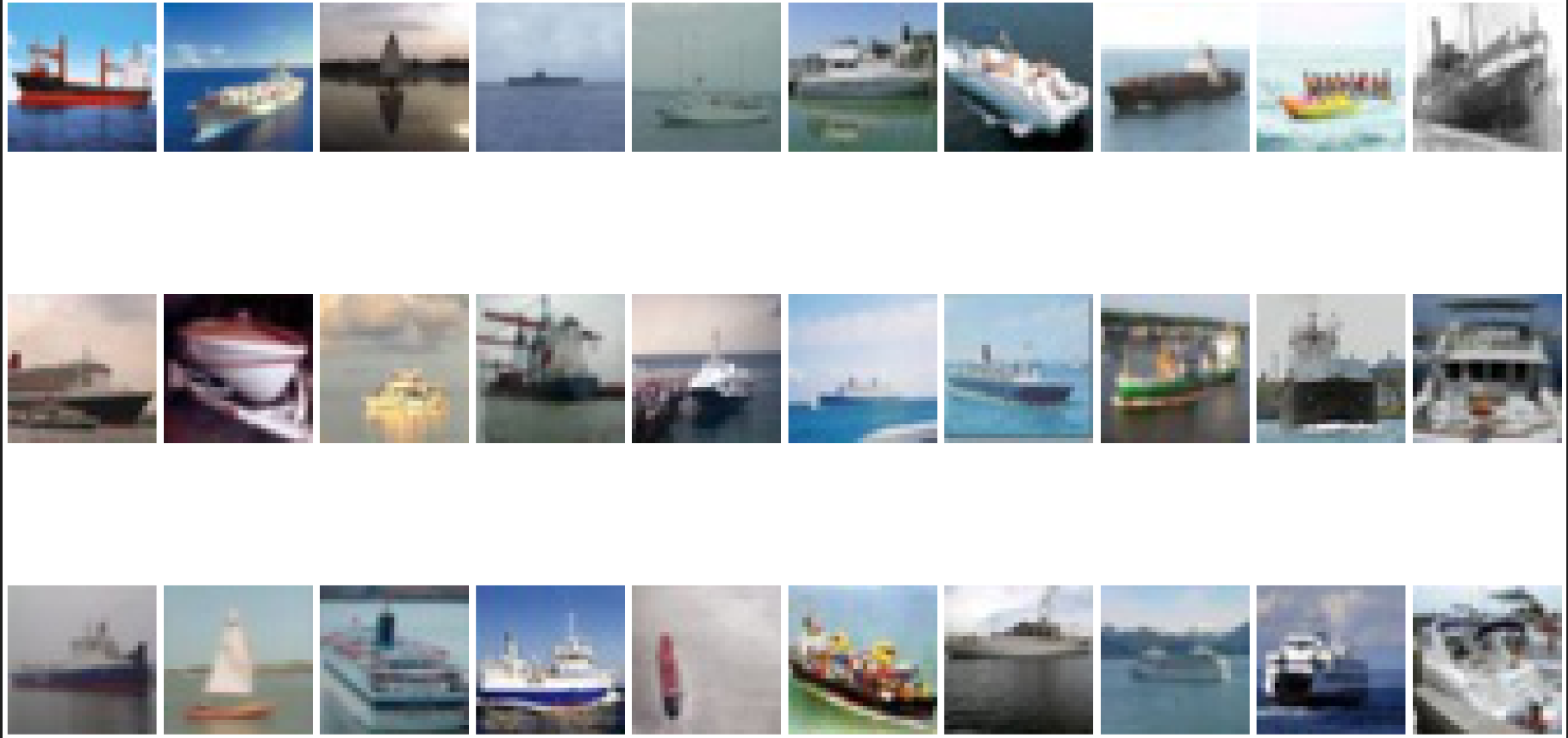


**class 6: frogs**

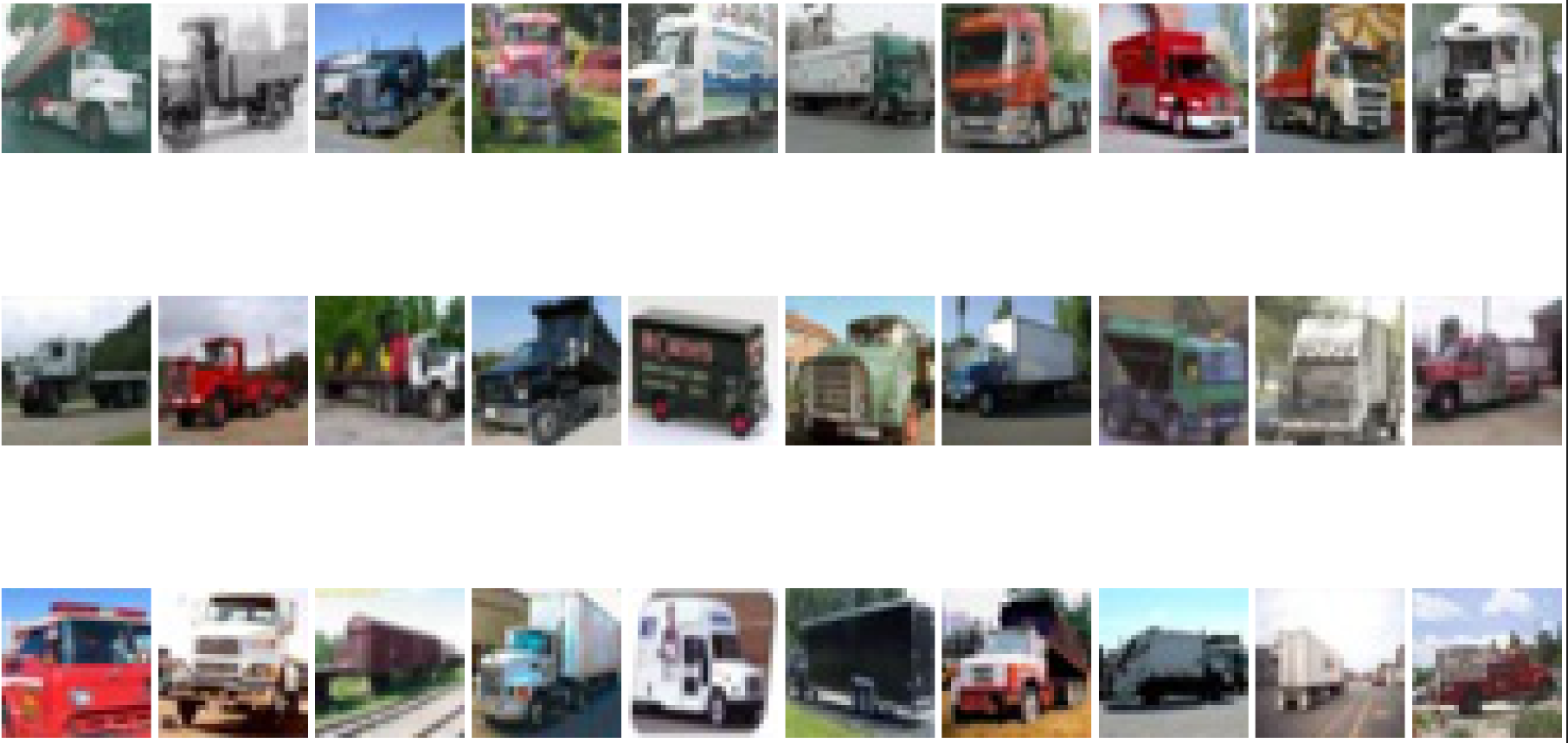
**class 7: horses**



**class 8: ships**



**class 9: trucks**



**2 Methods and Experiments**

**2.1 Data Augmentation**

**2.1.1 Flip**

We use flipping horizontally to augment our dataset. It boosts our model performance in a large extent.

In our sota model:

- without Flip: 0.722

- with Flip: 0.7539

**2.1.2 Rotation**

It seems that doing rotation will only benefit the local results. The prediction that was uploaded on Kaggle to predict new images has lower accuracy. It may be because repeatedly doing tricks on the same set of pictures may cause the overfit.

**2.1.3 Color**

Contrast, Saturation, Gray Scale. No time to try it.

**2.1.4 Noise**

Gaussian noise. Random mask. No time to try it.

**2.2 Feature**

**2.2.1 Global Feature**

**2.2.1.1 HuMoment**

Hu Moments, also known as Hu Invariant Moments or Hu's Seven Moments, are a set of seven statistical moments that are used as shape descriptors in image processing and pattern recognition. They were introduced by M. K. Hu in 1962 as a method to characterize and distinguish between different shapes regardless of their rotation, scale, and translation.

**2.2.1.2 Color-based features**

We research the train dataset. We find that some classes tend to have some major colors. Like the “airplane” and “ship”, they have blue as their background color. For “dog”, the face of dogs typically is white. For “frog”, the body of them is green. So we have the idea to use the main color of an image as our feature.

We have tried 2 methods:

1. We use KMean to find three of the most significant colors in an image and use their color (r, g, b) as our feature. However, experiment results show that it is not working.
2. We extract the mean value of each RGB value as our feature. It can improve the accuracy by about 1%~2%.

**2.2.1.3 Haralick Texture**

Texture refers to the visual patterns and structures present in an image that are related to the spatial arrangement of pixel intensities. Haralick texture features capture different aspects of texture, such as smoothness, coarseness, regularity, and complexity.

This feature is useful, which can improve our almost-final model by 0.3%

**2.2.2 Local Feature**

**2.2.2.1 Histogram of Orientated Gradient (HOG)**

Histograms of Oriented Gradients can extract local features. The HOG feature descriptor represents the local appearance and shape information of an image by capturing the distribution of gradient orientations. It is based on the observation that object shapes and structures can be characterized by the distribution of intensity gradients or edge directions in an image.

After experiment, only using the HOG feature can get the best accuracy compared with SIFT, Raw image and other features. The possible reasons are:

1. HOG provides feature size of thousands of dimensions. Large dimension is beneficial for SVM classification
2. HOG can extract features using histogram. To some extent, it is more robust to noise like shifting of image or unnormal pixel value.

fine-tuning hyper-parameters of HOG:

model = SVC(kernel='rbf', max\_iter=100, gamma=0.09, C=10) +(rgb)HOG\_feature(block\_size = (16, 16), block\_stride = (8, 8), cell\_size = (8, 8) num\_bins = 9) + TopColor = 0.695

model = SVC(kernel='rbf', max\_iter=100, gamma=0.09, C=10)+smooth\_raw(16\*16)+(grey)HOG\_feature(block\_size = (16, 16), block\_stride = (8, 8), cell\_size = (8, 8) num\_bins = 9) + TopColor = 0.6718

model = SVC(kernel='rbf', max\_iter=100, gamma=0.09, C=10)+(grey)HOG\_feature(block\_size = (16, 16), block\_stride = (8, 8), cell\_size = (8, 8) num\_bins = 9) + TopColor = 0.6672

model = SVC(kernel='rbf', max\_iter=100, gamma=0.09, C=10)+smooth\_raw(32\*32)+(grey)HOG\_feature(block\_size = (16, 16), block\_stride = (8, 8), cell\_size = (8, 8) num\_bins = 9) + TopColor = 0.5062

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.09, C=10)+smooth\_raw(16\*16)+(rgb)HOG\_feature(block\_size = (16, 16), block\_stride = (8, 8), cell\_size = (8, 8) num\_bins = 9) + TopColor = 0.7092

model = SVC(kernel='rbf', max\_iter=5000, gamma=0.09, C=10)+smooth\_raw(16\*16)+(rgb)HOG\_feature(block\_size = (16, 16), block\_stride = (8, 8), cell\_size = (8, 8) num\_bins = 9) + TopColor = 0.7092

**model = SVC(kernel='rbf', max\_iter=3000, gamma=0.09, C=10)+smooth\_raw(16\*16)+(rgb+grey)HOG\_feature(block\_size = (16, 16), block\_stride = (8, 8), cell\_size = (8, 8) num\_bins = 9) + TopColor = 0.7118**

**2.2.2.2 SIFT (Scale-Invariant Feature Transform)**

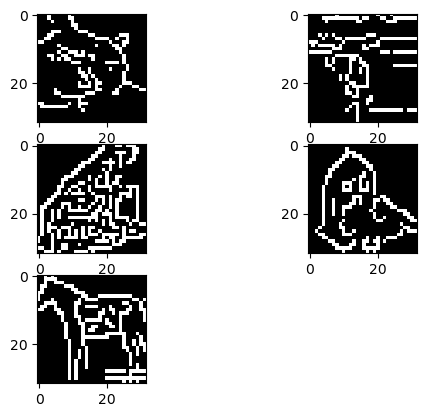
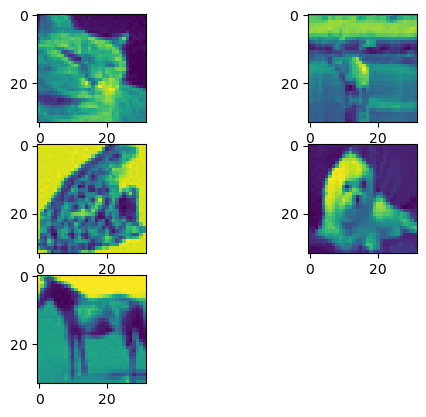
The idea of SIFT is similar to HOG. But it can extract keypoints and their descriptor. At first, we tried to use the “Bags of SIFT” idea. We try to use KMean to abstract the feature and reduce the final feature size, which can also reduce the influence of noise. However the results are not satisfactory. We still only use simple SIFT feature without “Bags”.

**2.2.2.3 Local Binary Patterns (LBP)**

LBP characterizes the local texture patterns by comparing the intensity values of a center pixel with its surrounding neighbors. The basic idea behind LBP is to encode the local structure of an image by quantizing the relationships between neighboring pixels into binary patterns.

**2.2.2.4 Edge Orientation Histogram (EOH)**

Similar to the mechanism of HOG, by dividing the picture to a certain number of blocks of cells, it first detects the edge of each cell using the Sobel Operator. Then the histograms based on the edge orientation are calculated. In the end the value is normalized to be the features.



Bin represents for dividing the picture into bin\*bin number of blocks.

After the experiment, we find bin=8 gets the best result. It is the second-best feature we extracted so far.

model = SVC(kernel='rbf') + (bin=4) = 0.5302

model = SVC(kernel='rbf') + (bin=6) = 0.5684

model = SVC(kernel='rbf') + (bin=7) = 0.5580

model = SVC(kernel='rbf') + (bin=8) = 0.5782

model = SVC(kernel='rbf') + (bin=9) = 0.5422

model = SVC(kernel='rbf') + (bin=10) = 0.5676

model = SVC(kernel='rbf') + (bin=16) = 0.5596

**2.2.3 Feature Reduction**

**2.2.3.1 Principle Components Analysis**

After experiment, we find PCA is a good way to remove redundant or irrelevant feature. Our original feature has a feature size of 5952 and got 0.712 accuracy. However, after putting PCA, the feature size is reduced to 2000, 1000, 500, 200 and the performance even goes up, which justifies that so many features are redundant. Using PCA can allow us to contract features and allow more memory space for other feature extraction methods.

Experiment result:

HOG from package openCV + raw\_feature => noPCA = 0.712 (feature size of 5952)

HOG from package openCV + raw\_feature => PCA(2000) = 0.7146 (feature size of 2000)

HOG from package openCV + raw\_feature => PCA(1000) = 0.716 (feature size of 1000)

HOG from package openCV + raw\_feature => PCA(500) = 0.717 (feature size of 500)

HOG from package openCV + raw\_feature => PCA(200) = 0.7222 (feature size of 200)

HOG from package openCV + raw\_feature => PCA(150) = 0.7202 (feature size of 150)

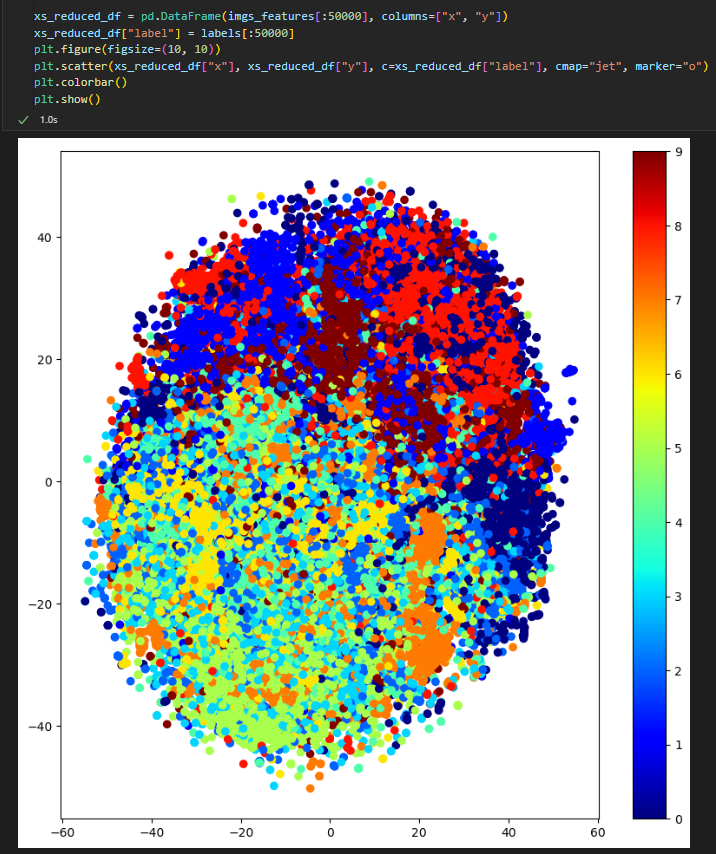
HOG from package openCV + raw\_feature => PCA(100) = 0.714 (feature size of 100)

**2.2.3.2 T-SNE Scale-Invariant Feature Transform**

t-SNE (t-distributed Stochastic Neighbor Embedding) is an unsupervised non-linear dimensionality reduction technique for data exploration and visualizing high-dimensional data.

In Assignment 2, t-SNE performs better than PCA to reduce the dimension from 794 to 2. However, in our experiment, the result turns out to be not so good. Using it we only got an accuracy of 41%.

Maybe it is because we did not get the preferred parameter. However, running t-SNE is much more time costly than PCA (about 10 times longer for the runtime). So, we finally decided not to replace PCA by t-SNE.

The following is the visualization of t-SNE in our experiment.

reference: <https://distill.pub/2016/misread-tsne/>

**2.4 Classifiers Analysis**

**2.4.1 Linear SVM**

Linear Support Vector Machine (SVM) is a Support Vector Machine algorithm that uses a linear decision boundary to classify data points into different classes. In this task, it gives a pretty good performance. However, compared to the results given by kernel SVM, which is shown below, Linear SVM is proven to be primarily used for binary classification tasks, but not an image classification task like this. Because the image classification task is highly nonlinear. LinearSVC can be only used for binary classification. In this task, LinearSVC tends to be affected by slight noise. kernel SVM can project the feature into a high dimensional kernel, which can make the new vector more linearly-separable.

**Experiment results:**

LinearSVC(multi\_class='ovr', max\_iter=3000, C=1) + HOG\_feature + TopColor\_feature = 0.5488

SVC(kernel='linear', max\_iter=3000) + HOG\_feature + TopColor\_feature = 0.5034

**2.4.2 Kernel SVM**

We use the 'rbf' kernel, which is [Radial basis function kernel](https://en.wikipedia.org/wiki/Radial_basis_function_kernel). It projects the original features into an infinite-dimensional space. It is generally good for all kinds of classification tasks.

**Experiment results:**

SVC(kernel='rbf', max\_iter=3000) + HOG\_feature + TopColor\_feature = 0.6626

**Experiment results for tuning gamma and C:**

**Gamma**

model = SVC(kernel='rbf', max\_iter=3000, gamma=1) + HOG\_feature + TopColor = 0.6004

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.3) + HOG\_feature + TopColor = 0.6109

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.1) + HOG\_feature + TopColor = 0.6764

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.08) + HOG\_feature + TopColor = 0.6755

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.01) + HOG\_feature + TopColor = 0.596

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.001) + HOG\_feature + TopColor= 0.5302

model = SVC(kernel='rbf', max\_iter=3000, gamma=scale) + HOG\_feature + TopColor = 0.6638

**C**

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.1, C=0.1) + HOG\_feature = 0.5693

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.1, C=1) + HOG\_feature = 0.6647

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.1, C=10) + HOG\_feature = 0.6845

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.09, C=10) + HOG\_feature = 0.6853

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.09, C=10) + HOG\_feature + TopColor = 0.6966

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.09, C=10) + HOG\_feature + ColorEntropy = 0.6778

**2.4.3 KNN**

**2.4.3.1 Mechanism**

KNN is short for k-nearest neighbors, which is a quite straightforward algorithm. Firstly, the number of k and a distance metric will be chosen. Then, k-nearest neighbors of the sample will be found. Finally, the class label will be assigned by the majority vote.

The reason for adopting this classifier is that KNN is a memory-based approach. So the runtime will be extremely low. Particularly, when we struggle to run an SVM model for 10 minutes, KNN only takes a few seconds to run. This largely saves us time when deciding the quality of various features at the beginning of our project. However, the memory problem is serious while dealing with samples with high dimensions.

**2.4.3.2 Parameter**

Impact of p

p is the distance metric to choose while computing the nearest neighbours. p=1 represents for Manhattan distance and p=2 represents for Euclidean distance. In our experiment, the Manhattan distance (p=1) fits better.

**Impact of p to KNN:**

model = KNeighborsClassifier(n\_neighbors=10, p=1, metric='minkowski') + HOG\_feature = 0.6311

model = KNeighborsClassifier(n\_neighbors=10, p=2, metric='minkowski') + HOG\_feature = 0.5637

Impact of n\_neighbors

The number of neighbours is also significant. If it is too small, the model may be underfitting, if it is too large, it may be overfitting. In our experiment, n\_neighbours = 10 achieved the best balance.

**Impact of n to KNN:**

model = KNeighborsClassifier(n\_neighbors=5, p=2, metric='minkowski') + HOG\_feature = 0.5629

model = KNeighborsClassifier(n\_neighbors=10, p=2, metric='minkowski') + HOG\_feature = 0.5637

model = KNeighborsClassifier(n\_neighbors=15, p=2, metric='minkowski') + HOG\_feature = 0.5546

model = KNeighborsClassifier(n\_neighbors=20, p=2, metric='minkowski') + HOG\_feature = 0.5484

**2.4.4 Logistic regression**

The reason why SVM is better than logistic regression may lie in that logistical regression is vulnerable under occurrence of outliers, which is quite often in this task.

**2.4.5 Random forest**

Random Forest combines multiple decision trees to make predictions. In this task, its performance is not quite satisfying. This may be because there are noise or irrelevant patterns in the training data, and decision trees are prone to overfitting, especially when the tree becomes too deep and complex, leading to poor generalization and low accuracy on unseen data.

**Experiment results:**

RandomForestClassifier(entropy,100,42)+ HOG\_feature=0.4918

**3 Final Model**

Our methods to find the best model follows the following procedure:

1. Select a suitable classifier

2. Find the best feature extractors

3. try some resembling learning and data argumentation

4. fine-tuning hyperparameters

**3.1 Data Augmentation**

We only use flip to double our dataset size. We only flipped our training images horizontally.

print("begin data augmentation")

size = len(train\_imgs)

for i in range(size):

img = train\_imgs[i]

# flip

img = cv2.flip(img, 1)

train\_imgs.append(img)

train\_labels.append(train\_labels[i])

print("data augmentation done")

**3.2 Feature Extraction**

After hundreds of experiment, we selected the best 12 features as our final image embeddings. Some have major impacts on performance of our model. Some have less impact but by working together, they can improve our model significantly.

The 12 features are;

1. cvHOG\_feature; HOG feature

2. raw\_feature; flatten our raw image into one-dimensional 32\*32\*3 vector

3. Color\_feature; take mean value of each color channel

4. EOH\_feature; use Canny to extract the local feature

5. HuMoments\_feature; use moment to calculate the global moment feature

6. SIFT\_feature; find keypoints and descriptors of images as feature

7. LBP\_feature; local binary pattern

8. cooccurrence\_feature;

9. LPQ\_feature;

10. ColorHistogram\_feature;

11. Tamura\_feature;

12. Daubechies\_Wavelets\_feature;

Daubechies\_Wavelets\_feature = Daubechies\_Wavelets\_extractor.featureExtract(imgs)

Tamura\_feature = Tamura\_extractor.featureExtract(imgs)

ColorHistogram\_feature = ColorHistogram\_extractor.featureExtract(imgs)

LPQ\_feature = LPQ\_extractor.featureExtract(imgs)

cooccurrence\_feature = cooccurrence\_extractor.featureExtract(imgs)

cvHOG\_feature = cvHOG\_extractor.featureExtract(imgs)

Color\_feature = Color\_extractor.featureExtract(imgs)

raw\_feature = Raw\_extractor.featureExtract(imgs)

EOH\_feature = EOH\_extractor.featureExtract(imgs)

HuMoments\_feature = HuMoments\_extractor.featureExtract(imgs)

SIFT\_feature = SIFT\_extractor.featureExtract(imgs)

LBP\_feature = LBP\_extractor.featureExtract(imgs)

imgs\_features = np.concatenate(

(cvHOG\_feature, raw\_feature, Color\_feature, EOH\_feature, HuMoments\_feature, SIFT\_feature, LBP\_feature,

cooccurrence\_feature, LPQ\_feature, ColorHistogram\_feature, Tamura\_feature, Daubechies\_Wavelets\_feature,),

axis=1)

**3.3 PCA**

We use pca to reduce feature dimension for SVM training. With PCA, we can reduce our original feature vector from 7889 dimensions to less than 400 dimensions but maintain the model performance. PCA also accelerates our SVM training speed. After fine-tuning, we find n\_component=0.87 have the best performance.

**print("begin pca")**

**pca = PCA(0.87, copy=False)**

**pca.fit(imgs\_features)**

**imgs\_features = pca.fit\_transform(imgs\_features)**

**print("pca done")**

**3.4 Classifier**

Lastly, we feed our embedding vectors into our SVM model for training. For Classifier, we choose kernel SVM. The kernel is 'rbf', which is a good kernel for most classification jobs. We further fine-tuned hyperparameters like C. We find C=5 > C=10 > C=20 (sorted by prediction accuracy).

print("begin training")

model = SVC(kernel='rbf', C=1, random\_state=42)

model.fit(train\_features, train\_labels)

print("training done")

**4 Appendix**

Here are some additional experiment results for reference:

Experiment Results:

Logistic Regression (C=1) + raw\_feature = 0.4017

Logistic Regression (C=0.1) + raw\_feature = 0.3985

Logistic Regression (C=0.01) + raw\_feature = 0.4004

Logistic Regression (C=0.001) + raw\_feature = 0.3998

**This series of experiment compares *num\_bins***

Logistic Regression (C=1) +

HOG\_feature (win\_size = (32, 32), block\_size = (32, 32), block\_stride = (8, 8), cell\_size = (16, 16), num\_bins = 6) = 0.3979

Logistic Regression (C=1) +

HOG\_feature (win\_size = (32, 32), block\_size = (32, 32), block\_stride = (8, 8), cell\_size = (16, 16), num\_bins = 9) = 0.4227

Logistic Regression (C=1) +

HOG\_feature (win\_size = (32, 32), block\_size = (32, 32), block\_stride = (8, 8), cell\_size = (16, 16), num\_bins = 12) = 0.4245

**This series of experiment compares *cell\_size***

Logistic Regression (C=1) +

HOG\_feature (win\_size = (32, 32), block\_size = (32, 32), block\_stride = (8, 8), cell\_size = (16, 16), num\_bins = 12) = 0.4245

Logistic Regression (C=1) +

HOG\_feature (win\_size = (32, 32), block\_size = (32, 32), block\_stride = (8, 8), cell\_size = (8, 8), num\_bins = 12) = 0.52

Logistic Regression (C=1) +

HOG\_feature (win\_size = (32, 32), block\_size = (32, 32), block\_stride = (8, 8), cell\_size = (4, 4), num\_bins = 12) = 0.5413

**This series of experiment compares *block\_size***

Logistic Regression (C=1) +

HOG\_feature (win\_size = (32, 32), block\_size = (32, 32), block\_stride = (8, 8), cell\_size = (4, 4), num\_bins = 12) = 0.5413

Logistic Regression (C=1) +

HOG\_feature (win\_size = (32, 32), block\_size = (24, 24), block\_stride = (8, 8), cell\_size = (4, 4), num\_bins = 12) = 0.5589

Logistic Regression (C=1) +

HOG\_feature (win\_size = (32, 32), block\_size = (16, 16), block\_stride = (8, 8), cell\_size = (4, 4), num\_bins = 12) = 0.5624

**This series of experiment compares *block\_stride***

Logistic Regression (C=1) +

HOG\_feature (win\_size = (32, 32), block\_size = (16, 16), block\_stride = (8, 8), cell\_size = (4, 4), num\_bins = 12) = 0.5624

Logistic Regression (C=1) +

HOG\_feature (win\_size = (32, 32), block\_size = (16, 16), block\_stride = (4, 4), cell\_size = (4, 4), num\_bins = 12) = 0.5639

SVC(kernel='rbf', max\_iter=3000) + HOG\_feature + TopColor\_feature = 0.6626

Logistic Regression (C=1) +

HOG\_feature (win\_size = (32, 32), block\_size = (16, 16), block\_stride = (8, 8), cell\_size = (4, 4), num\_bins = 12) + TopColor\_feature= 0.5722

**This series of experiment compares *LinearSVM and kernel SVM***

SVC(kernel='rbf', max\_iter=3000) + HOG\_feature + TopColor\_feature = 0.6626

LinearSVC(multi\_class='ovr', max\_iter=3000, C=1) + HOG\_feature + TopColor\_feature = 0.5488

SVC(kernel='linear', max\_iter=3000) + HOG\_feature + TopColor\_feature = 0.5034

**Gamma**

model = SVC(kernel='rbf', max\_iter=3000, gamma=1) + HOG\_feature + TopColor = 0.6004

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.3) + HOG\_feature + TopColor = 0.6109

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.1) + HOG\_feature + TopColor = 0.6764

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.08) + HOG\_feature + TopColor = 0.6755

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.01) + HOG\_feature + TopColor = 0.596

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.001) + HOG\_feature + TopColor= 0.5302

model = SVC(kernel='rbf', max\_iter=3000, gamma=scale) + HOG\_feature + TopColor = 0.6638

**C**

**0.1 Accuracy: 0.5693**

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.1, C=0.1) + HOG\_feature = 0.5693

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.1, C=1) + HOG\_feature = 0.6647

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.1, C=10) + HOG\_feature = 0.6845

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.09, C=10) + HOG\_feature = 0.6853

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.09, C=10) + HOG\_feature + TopColor = 0.6966

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.09, C=10) + HOG\_feature + ColorEntropy = 0.6778

**This series of experiment compares the impact of n to KNN**

model = KNeighborsClassifier(n\_neighbors=5, p=2, metric='minkowski') + HOG\_feature = 0.5629

model = KNeighborsClassifier(n\_neighbors=10, p=2, metric='minkowski') + HOG\_feature = 0.5637

model = KNeighborsClassifier(n\_neighbors=15, p=2, metric='minkowski') + HOG\_feature = 0.5546

model = KNeighborsClassifier(n\_neighbors=20, p=2, metric='minkowski') + HOG\_feature = 0.5484

**This series of experiment compares the impact of p to KNN**

model = KNeighborsClassifier(n\_neighbors=10, p=2, metric='minkowski') + HOG\_feature = 0.5637

model = KNeighborsClassifier(n\_neighbors=10, p=1, metric='minkowski') + HOG\_feature = 0.6311

model = KNeighborsClassifier(n\_neighbors=10, p=1, metric='minkowski', weight=’distance’) + HOG\_feature = 0.633

**This series of experiment compares different features on KNN**

model = KNeighborsClassifier(n\_neighbors=10, p=2, metric='minkowski', weights='distance') + raw\_feature = 0.34

model = KNeighborsClassifier(n\_neighbors=10, p=2, metric='minkowski', weights='distance') + Haralick\_feature = 0.1685

model = KNeighborsClassifier(n\_neighbors=10, p=2, metric='minkowski', weights='distance') + HOG\_feature = 0.5636

model = KNeighborsClassifier(n\_neighbors=10, p=2, metric='minkowski', weights='distance') + EntropyColor\_feature = 0.162

model = KNeighborsClassifier(n\_neighbors=10, p=2, metric='minkowski', weights='distance') + LBP\_feature = 0.2318

**This series of experiment compares the impact of depth to Decision Tree**

model = DecisionTreeClassifier(max\_depth=5) + HOG\_feature = 0.2396

model = DecisionTreeClassifier(max\_depth=15) + HOG\_feature = 0.264

**Random forest**

RandomForestClassifier(entropy,100,42)+ HOG\_feature=0.4918

**Whether TopColor is useful?**

model = SVC(kernel='rbf', max\_iter=3000, gamma=1, C=1) + HOG\_feature= 0.6009

model = SVC(kernel='rbf', max\_iter=3000, gamma=1, C=1) + HOG\_feature + TopColor = 0.6004

model = KNeighborsClassifier(n\_neighbors=10, p=1, metric='minkowski') + HOG\_feature = 0.6311

model = KNeighborsClassifier(n\_neighbors=10, p=1, metric='minkowski') + HOG\_feature + TopColor = 0.6313 确实没用

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.09, C=10) + HOG\_feature + TopColor = 0.6966 好像又有点用！！！

**PCA**

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.09, C=10)+pca\_image(dim=30)+(rgb+grey)HOG\_feature(block\_size = (16, 16), block\_stride = (8, 8), cell\_size = (8, 8) num\_bins = 9) + TopColor = 0.5204

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.09, C=10)+pca\_image(dim=200)+(rgb+grey)HOG\_feature(block\_size = (16, 16), block\_stride = (8, 8), cell\_size = (8, 8) num\_bins = 9) + TopColor = 0.6562

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.09, C=10)+pca\_image(dim=500)+(rgb+grey)HOG\_feature(block\_size = (16, 16), block\_stride = (8, 8), cell\_size = (8, 8) num\_bins = 9) + TopColor = 0.6558

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.09, C=10)+pca\_image(dim=500)+(rgb+grey)HOG\_feature(block\_size = (16, 16), block\_stride = (8, 8), cell\_size = (8, 8) num\_bins = 9) + TopColor = 0.656

**SIFT**

We use SIFT feature extractor….

only SIFT

​​​model = SVC(kernel='rbf', max\_iter=20000, C=0.01) = 0.2616

​​​model = SVC(kernel='rbf', max\_iter=20000, C=0.1) = 0.3109

​model = SVC(kernel='rbf', max\_iter=20000, C=100) = 0.3076

model = SVC(kernel='rbf', max\_iter=20000, C=10) = 0.3244

model = SVC(kernel='rbf', max\_iter=20000) = 0.3526

model = SVC(kernel='rbf', max\_iter=10000) = 0.3526

model = SVC(kernel='rbf', max\_iter=10000, gamma=0.09, C=1) = 0.2266

model = SVC(kernel='rbf', max\_iter=3000, gamma=0.09, C=1) = 0.1016

model = SVC(kernel='rbf’) = 0.3454

**Edge Histogram**

model = SVC(kernel='rbf') + (bin=4) = 0.5302

model = SVC(kernel='rbf') + (bin=6) = 0.5684

model = SVC(kernel='rbf') + (bin=7) = 0.5580

model = SVC(kernel='rbf') + (bin=8) = 0.5782

model = SVC(kernel='rbf') + (bin=9) = 0.5422

model = SVC(kernel='rbf') + (bin=10) = 0.5676

model = SVC(kernel='rbf') + (bin=16) = 0.5596

**TopColor**

model = SVC(kernel='rbf') = 0.3074

**HOG**

model = SVC(kernel='rbf') = 0.6706

**Color**

model = SVC(kernel='rbf') = 0.3502

**Combination**

Increase, Decrease, Equal

TopColor + SIFT = 0.3454

HOG + EOH(bin=4) = 0.6768

HOG + EOH(bin=8) = 0.6776

HOG + Color = 0.6874

HOG + TopColor = 0.6564

Color + EOH(bin=8) = 0.53

Color + EOH(bin=4) = 0.5366

HOG + EOH(bin=4) + Color = 0.6878

**HOG + EOH(bin=4) + Color = 0.6878**

HOG + Color + EOH + HuM = 0.7564

HOG + EOH + HuM = 0.7588

HOG + EOH = 0.7578

HOG + EOH(bin=4) = 0.7592

HOG + HuM = 0.757

HOG = 0.749

HOG + EOH(bin=4) + HuM = 0.755

HOG and SIFT analysis

HOG from package openCV: 0.6596

HOG from package sklearn: 0.6434

HOG from package openCV + Color\_feature = 0.665