Classifying CIFAR-10 Dataset Using SVM and CNN

Group 8

Group Member: Yefan Li, Jue Li, Yijin Wang, Xiao Ma, Wenxuan Gu, Pengru Lyu, Ziyi Xue, Yanqing Li

1 Introduction

In this project, our task and goal are to implement machine learning methods for image recognition based on the CIFAR-10 dataset. The CIFAR-10 dataset is a widely-used image classification dataset that consists of 60,000 32×32 3 channel color RGB images in 10 classes, with 6,000 images per class. The classes are mutually exclusive, which include airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. Specifically, the dataset is divided into 5 training batches and a test batches, each with 10,000 images, and we use the data in training batches for training our model and use the test batches to get the test accuracy. We used SVM and CNN models for our task. SVM is very efficient in high dimensional space and CNN could be shown to be effective at this task and has high robustness when it comes to image classification tasks.

2 Methodologies

2.1 Support Vector Machine

The first model we used is the Support Vector Machine (SVM). The SVM models could apply different kernels to map data into high dimensions, and find hyperplanes to separate data. This means SVM models could be a useful model when meeting the data with high dimensions. However, there are also some disadvantages of the SVM, such as the high computation complexity. In this part, we will introduce the details of how we use the SVM model.

2.1.1 Preprocessing data

Before building the SVM model, we preprocessed the 2 dimensional image data with color into one dimension. During the transformation process, each object of the data with a shape of 32*32*3 are changed into 1*3072, with each column representing one pixel. Then, we scaled each pixel value into [0,1]. After the data preprocessing, we also checked the balance of the data. We observed each class has the same sample size for both the training and testing dataset (shown in *Figure 2.1*), which indicates the data is balanced.

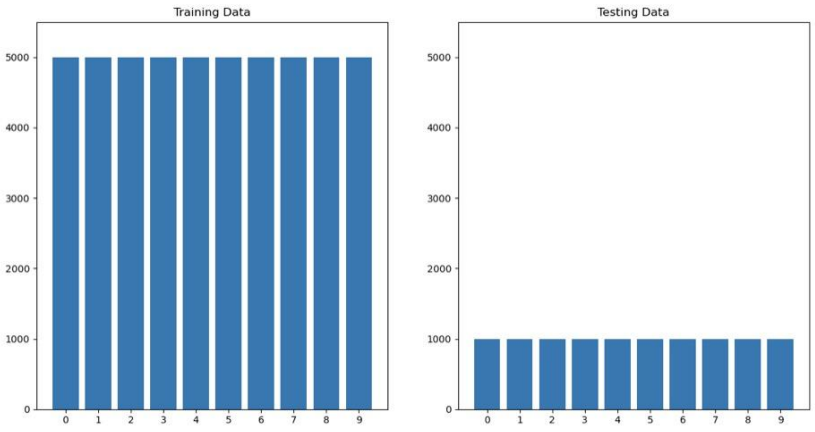

Figure 2.1 The balance of training data and testing data

2.1.2 Experiment Work

The method we used to build the SVM model is the cross validation. In this method, we used the 5 folds and selected the best parameters based on different kernels parameters, such as c and gamma. Before we started using the cross validation, we tested our code with the polynomial kernel and the default c and gamma. However, the problems of high computation complexity arises when attempting to fit only one model, which took as long as 9376.61 seconds (~2.6 hours). Because the core idea of cross validation is to leave one out and repeat training many times, the long runtime would without doubt appear in this method, even if we tried the parallel processing method. So we changed our strategies by just fitting models with trying different kernels and parameters and the histogram of oriented gradient methods.

1) Parameter Analysis

We used four different types of kernel model and used to train the data, those are linear model, polynomial model, radial basis function(RBF) model and sigmoid model. We used the different parameters value for the support vector machine model as a comparison benchmark to evaluate the ability of the model, and c and gamma parameters are mainly used to help improve the accuracy. These is the two method we used to improve accuracy:

1. Changing parameter C values as 0.001,0.01,0.1, 1, 5,10,50,100 and parameter Gamma values as 0.0001, 0.001, 0.01, 1, 3, 5 with four kernel model
2. Changing parameter degree values as 2, 3, 4, 5 with default c and gamma values and only use for polynomial kernel model

By two methods, the best testing accuracy is 52.17% which uses the RBF kernel, with default gamma value and the parameter C value equal to 5.

2) Improvement – Using HOG (Histogram of Oriented Gradient) Method

The SVM model spends a lot of time mapping data with different kernels, because of the high dimensions. And this also caused the problems of overfitting, which led to the low testing accuracy. So in order to reduce the problem of high dimensions, we applied the histogram of oriented gradient method (HOG). This method allowed us to transform the color image to grayscale, and got the gradient histogram by calculating the gradient values of pixels, which can better identify the patterns and features in the image. With this process, we reduced the dimensions of original data from 3072 to 324 columns, by setting the size of cells equal to 8. With the HOG methods, we reduced the runtime effectively. This method only needs around 30 minutes for fitting a whole model. Then we continued our steps of model fitting with different kernels and parameter turning with the transformed training and testing data.

2.2 Convolutional Neural Network (CNN)

We choose CNN because it is often used for image recognition tasks, where the filters can learn to recognize edges, textures, and patterns in the input images. They have been shown to be very effective at these tasks, achieving state-of-the-art performance on many image classification benchmarks. For preprocessing of data, we use the same strategy as SVM that scales each pixel into [0,1].

2.2.1 Experiment Work

1) Basic Model

The original basic model structure we proposed was 32 - 32 - 64 - 64 (Two convolutional layers with 32 filters of size 3x3, two convolutional layers with 64 filters of size 3x3, we have ReLU as our activation function by comparison with the results of Sigmoid function). At the end, we added two dense layers to shrink the network. For the training process of the CNN model, batch size of 64 and epoch of 20 are applied.

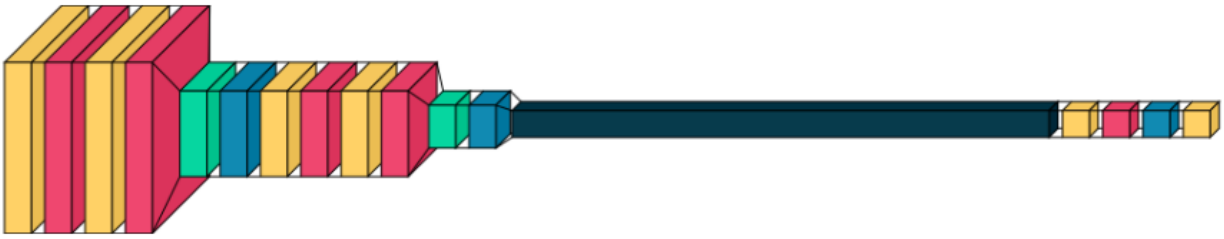

Figure 2.2.2.1 Structure of the Basic Model

2) Final Model

The original basic model structure we proposed was 32 - 32 - 64 - 64 - 128 - 128(Two convolutional layers with 32 filters of size 3x3, two convolutional layers with 64 filters of size 3x3, we have ReLU as our activation function). And, we added dropout layers along with every convolutional layer to remain a certain amount of units active in each layer. Besides, we also applied the learning rate schedule technique(Exponential Decay) along with our final model. For the training process of the CNN model, batch size of 64 and epoch of 70 are applied.

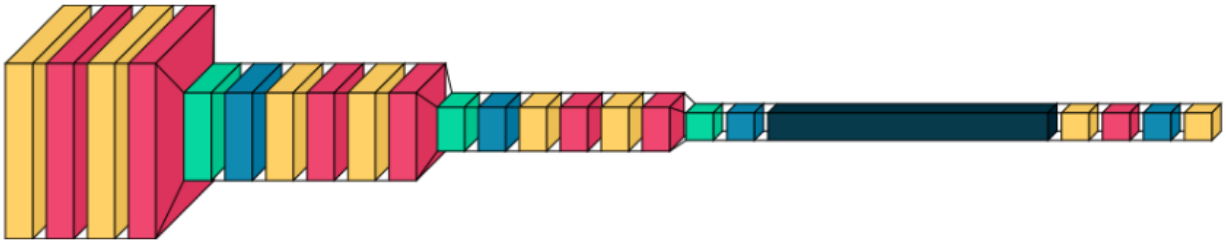

Figure 2.2.2.2 Structure of the Final Model

3 Result of Simulation and Comparison

3.1Results of supporting Vector Machine (SVM):

By using the HOG method as stated previously, we have successfully trained the whole 50,000 dataset and tested with 10,000 data much faster and more accurately than before. We observed 51.42% testing accuracy for the linear SVM model with default c and gamma (confusion matrix shown as Figure 3.1). And we observed 63.79% testing accuracy for the nonlinear SVM model with parameters C = 5, default gamma, and kernel = 'rbf' (confusion matrix shown as Figure 3.2). As mentioned before, we used the number of pixels in the cell's width equal to 8 as the parameter.

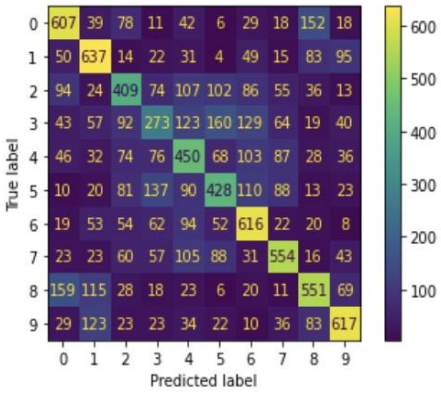

Figure 3.1.1 (51.42% accuracy)

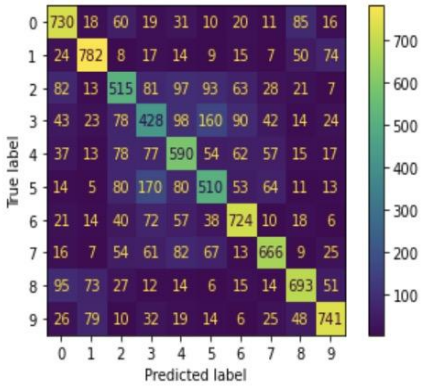

Figure 3.1.2 (63.79% accuracy)

These two confusion matrices explicitly show the differences of each SVM model setting on the testing accuracy level. Figure 3.1.1 and Figure 3.1.2 both showed that there are lots of class labels that are not successfully predicted, which is consistent with their accuracy score. The confusion matrices could visually show the accuracy score each model has.

In a nutshell, apparently, the nonlinear model works better than the linear one in the accuracy score, even without any parameter tuning. Moreover, the SVM model with our final (C = 5, gamma = 'scale', kernel = 'rbf') parameter setting performs best as we tested in the parameter tuning process, and with the help of the HOG feature extraction method, we had successfully improved our accuracy, decreased the running time and achieved the highest accuracy score we got so far - 63.79%.

3.2 Results of CNN:

As the final model shown above, six convolutional layers(32 - 32 - 64 - 64 - 128 - 128) are added. Following the convolutional layers, the dropout lays are added with the retention probability equal to 0.3, 0.5, 0.5, and 0.5. This helps achieve faster convergence and avoid the overfitting problem.

By splitting the whole CIFAR-10 dataset into the training dataset and testing dataset, we achieved the accuracy about 87.16% in the testing dataset.

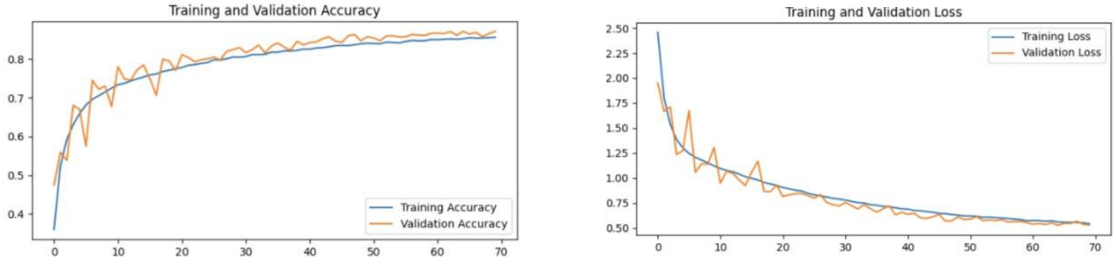

Figure 3.3 Accuracy and Loss of Final Model

With further research, we proposed a way to combine SVM and CNN together which we called Convolutional SVM. This is the same strategy as HOG-SVM. We use CNN as feature extraction that we cut the model at the dense layer and process it as feature extraction. This way is effective that 1-epoch training CNN as feature extraction can make SVM reach 0.58 accuracy.

3.3 Comparison (SVM vs CNN):

Obviously, we can see that CNN performs better on the image classification problem. While SVM is a powerful machine learning algorithm, it is better suited for simpler datasets with fewer dimensions. On the other hand, CNN is a deep learning algorithm that is suitable for image recognition tasks, making it an ideal approach for the CIFAR dataset. Our results demonstrate that CNN is a superior method for approaching the CIFAR dataset, and it is likely to be the preferred method for other similar image recognition tasks.

Firstly, they are both suitable for classification problems. But, the important reason why CNN achieves a higher accuracy is that CNN can effectively extract the features from the image tasks. Based on the principle of SVM, it maps the input data to some high-dimensional space and tries to find the differences. This will cause low efficiency while processing the image task.
If we focus on the dataset CIFAR-10 in our project, CNN is very suitable for processing quite large and complex datasets like CIFAR-10 because it requires a large dataset to train the whole network. On the other hand,be trained on a smaller dataset and it performs better on low-dimensional problems.

4 Overcame difficulty

SVM:

During the modeling, we have encountered some problems and difficulties such as computational complexity, and long runtime. Afterwards, we discussed and searched these issues online. We found some possible solutions for each problem, as stated below, which we have used a few but they could all be used in the future work.

For computational complexity, we could use the parallel processing method - a type of computation in which many calculations or processes are carried out simultaneously.; For the long run time, we used the HOG feature extraction method to both improve the testing accuracy and reduce the long runtime. And finally, we also use a computer with better CPU or GPU to run the dataset faster, which will be way much easier for us to do future parameter tuning or optimizing as well.

CNN:

During constructing the model of CNN, we generally had two major difficulties. The first one is how to improve the accuracy and the second one is how to prevent overfitting. In order to overcome these two major difficulties, we had three attempts:

1. We added two more convolutional layers with the size of 128 and 128. This attempt improves the accuracy. Furthermore, we tried to add more convolutional layers with the size of 64 filters to reduce the meaningless feature pattern that we extracted before. But, the results showed that adding extra layers with size of 64 did not improve the accuracy obviously. So, we kept the final model with 6 convolutional layers(32 - 32 - 64 - 64 - 128 - 128) to reduce computational cost. Also, the accuracy of our model is still increasing slowly at epoch 70. We suggest a way that uses the while loop to let the model find the

maximum epochs needed by itself. But this way requires large computing units. If we have a more powerful computing tool, we think our model can reach accuracy around 0.90 at epoch 100.

2. We tried transfer learning to improve the accuracy. GoogleNet and ImageNet are applied but the accuracy is barely improved. By adjusting the parameters, we could see the improvements in accuracy but it is still not obvious. This is one point that we can improve more in our future work.

3. We tried learning rate schedule and regularization. We added the experiential decay with decay rate equal to 0.95 to adjust the learning rate during training. Besides, the dropout layers are added in the final model to drop certain percent of units and remain others active. By doing this, we can avoid the potential overfitting problem raised by adding the layers and achieve faster convergence.

Appendix

 # SVM Python code

```
#process data
# X value
(X_train, Y_train), (X_test, Y_test) = cifar10.load_data()
X_train_new = X_train.reshape(-1, 3072)  # 1 dimensional array
X_test_new = X_test.reshape(-1, 3072)

col_names = ['pixel_%s' % i for i in range(0, 32*32*3)]  # column name with
pixel values
X_train_new = pd.DataFrame(X_train_new, columns=col_names)
X_test_new = pd.DataFrame(X_test_new, columns=col_names)

# Y value
Y_train_new = Y_train.reshape(-1, 1)  # 1 dimensional array
Y_test_new = Y_test.reshape(-1, 1)

Y_train_new = pd.DataFrame(Y_train_new, columns=["label"])
Y_test_new = pd.DataFrame(Y_test_new, columns=["label"])

# scale data
X_train_new = X_train_new/255
X_test_new = X_test_new/255

# balance of data
fig, ax = plt.subplots(1, 2, figsize=(16, 8))
train_size = Y_train_new["label"].value_counts()
test_size = Y_test_new["label"].value_counts()
ax[0].bar(train_size.index, train_size.values)
ax[0].set_title("Training Data")
ax[0].set_xticks(range(0, 10))
ax[0].set_ylim([0,5500])
ax[1].bar(test_size.index, test_size.values)
ax[1].set_title("Testing Data")
ax[1].set_xticks(range(0, 10))
ax[1].set_ylim([0,5500])
plt.show()

# Method 1: Cross Validation
# folds = KFold(n_splits=5, shuffle=True, random_state=542)
# gamma = [0.1, 0.01, 0.001]
# C = [5, 10]
# params = [{'gamma': gamma, 'C': C}]
# model_cv = GridSearchCV(estimator= linear_model,param_grid=params,
#                         scoring='accuracy',cv=folds,verbose=1,
#                         return_train_score=True,n_jobs=-2)

# Method 2: HOG
def hog_extraction(data, size):
    num = data.shape[0]
    data1_hog_feature = []
    for i in range(num):
        x = data[i]
        # Create a image object
        r = Image.fromarray(x[0])
```

```
        g = Image.fromarray(x[1])
        b = Image.fromarray(x[2])
        img = Image.merge("RGB", (r, g, b))

        # Change the image to gray scale
        gray = img.convert('L')
        gray_array = np.array(gray)

        # Extract the hog feature
        hog_feature = ft.hog(gray_array, pixels_per_cell=(size, size))
        data1_hog_feature.append(hog_feature)

    data_hog_feature = np.reshape(np.concatenate(data1_hog_feature), [num, -1])
    return data_hog_feature

# Try size equal to 8
size = 8

# Change the data shape for the hog_extraction function
X_train_tmp = np.reshape(X_train, [len(X_train), 3, 32, 32])
X_train_hog = hog_extraction(X_train_tmp, size)
Y_train_hog = Y_train_new
X_test_tmp = np.reshape(X_test, [len(X_test), 3, 32, 32])
X_test_hog = hog_extraction(X_test_tmp, size)
Y_test_hog = Y_test_new

# Fit the model
# Function of SVM testing accuracy
def model_accuracy(c, g, k, train_X, train_Y, test_X, test_Y):
    model = SVC(C=c, gamma=g, probability=False, kernel=k)
    model.fit(train_X, train_Y["label"])

    Y_predict = model.predict(test_X)
    accuracy = metrics.accuracy_score(y_true=test_Y["label"], y_pred=Y_predict)
    return accuracy

# parameters selection
parameter_c = [0.001, 0.01, 0.1, 1, 5, 10, 50, 100]
gamma = [0.0001, 0.001, 0.01, 1, 3, 5]
kernel = ['linear', 'poly', 'rbf', 'sigmoid']

# fit models
for k in range(0, len(kernel)):
    for c in range(0, len(parameter_c)):
        for g in range(0, len(gamma)):
            accuracy = model_accuracy(parameter_c[c], gamma[g], kernel[k],
X_train_new,Y_train_new, X_test_new, Y_test_new)
            print("parameter C value: " + str(parameter_c[c])+ " parameter gamma
value: " + str(gamma[g]) + " kernel model: " + str(kernel[k]) + " Accuracy = "
+ str(accuracy))
            accuracy_hog = model_accuracy(parameter_c[c], gamma[g], kernel[k],
X_train_hog,Y_train_hog, X_test_hog, Y_test_hog)
            print("parameter C value: " + str(parameter_c[c])+ " parameter gamma
value: " + str(gamma[g]) + " kernel model: " + str(kernel[k]) +" Accuracy = " +
str(accuracy) + " with hog method")
```

CNN Base Model Structure:

```
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', padding='same',
input_shape=(32, 32, 3)),
    layers.BatchNormalization(),
    layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.3),

    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.5),

    layers.Flatten(),
    layers.Dense(64, activation='sigmoid'),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])
epochs = 20
```

CNN Final Model Structure:

```
l2_reg = regularizers.l2(0.001)

model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', padding='same',
kernel_regularizer=l2_reg, input_shape=(32, 32, 3)),
    layers.BatchNormalization(),
    layers.Conv2D(32, (3, 3), activation='relu', padding='same',
kernel_regularizer=l2_reg),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.3),

    layers.Conv2D(64, (3, 3), activation='relu', padding='same',
kernel_regularizer=l2_reg),
    layers.BatchNormalization(),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same',
kernel_regularizer=l2_reg),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.5),

    layers.Conv2D(128, (3, 3), activation='relu', padding='same',
kernel_regularizer=l2_reg),
    layers.BatchNormalization(),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same',
kernel_regularizer=l2_reg),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.5),
```

```
    layers.Flatten(),
    layers.Dense(128, activation='relu', kernel_regularizer=l2_reg),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])
epochs = 70
```

Decay learning rate:

```
initial_learning_rate = 0.001

batch_size = 64

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=train_images.shape[0] // batch_size,
    decay_rate=0.95,
    staircase=True)
```

Model fitting:

```
model.fit(datagen.flow(train_images, train_labels, batch_size=batch_size),
                    steps_per_epoch=train_images.shape[0] // batch_size,
                    epochs=epochs,
                    validation_data=(test_images, test_labels))
```