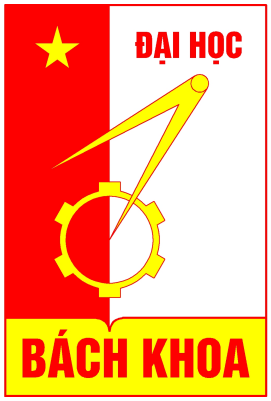
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
_____*******_____



BÁO CÁO
MÔN HỌC
Học máy

Đề tài: So sánh thử nghiệm các phương pháp học máy
cho bài toán phân loại ảnh.

Nhóm sinh viên thực hiện:

Họ và tên	MSSV	Lớp
Nguyễn Tuấn Đạt	20130856	CNTT2.02-K58
Vũ Minh Đức	20130856	CNTT2.02-K58
Nguyễn Ngọc Huyền	20130856	CNTT2.02-K58
Đặng Quang Trung	201341456	CNTT2.02-K58
Phan Anh Tú	20130856	CNTT2.02-K58

Giáo viên hướng dẫn: TS.Thân Quang Khoát

Hà Nội 12–2016

Mục lục

Lời cảm ơn

Danh sách hình vẽ

Phần 1

Mở đầu

Phần 2

Giới thiệu bài toán

2.1 Giới thiệu bài toán

2.2 Bộ dữ liệu sử dụng

Phần 3

Các phương pháp sử dụng và kết quả thực nghiệm

3.1 KNN

3.1.1 Cơ sở lý thuyết

3.1.2 Cài đặt

(ghi chú: cần nêu rõ cấu trúc mã nguồn, chương trình, vai trò của các lớp và các phương thức chính)

3.1.3 Kết quả

3.2 Mạng neural

3.3 Convolutional Neural Networks(CNN)

3.3.1 Giới thiệu

Convolutional Neural Networks tương tư như các mạng nerual network khác. Chúng được tạo thành từ các tế bào noron có thể học trọng số và biases.Mỗi noron thực hiện hiện môt số sản xuất và tùy chọn với đường phi tuyến.Toàn bộ mạng vẫn thể hiện một hàm số khả vi duy nhất: từ các điểm ảnh hình ảnh thô trên một đầu với điểm số lớp học khác. Chúng vẫn có một hàm lỗi(ví dụ như SVM/softmax) ở lớp cuối (với kết nối đầy đủ).

3.3.2 Kiến trúc tổng quan CNN

Mạng CNN có 4 tầng chính: Convolution Layer, ReLu Layer , Pooling Layer, Fully-Connected Layer.

Lớp Convolution Layer (Conv)

- Nhận đầu vào là ma trận các điểm ảnh $(W_1 * H_1 * D_1)$ ($W_1 * H_1$: kích thước của ảnh, D_1 số kênh màu (3 kênh màu nếu là ảnh màu đại diện

cho 3 kênh màu (r,g,b))).

- Các tham số:
  - Số các filters K: Tham số chính của tâng Conv. Mỗi filter sẽ trượt qua ma trận đầu vào và đưa ra ma trận đầu ra. Mỗi filter là một ma trận có kích thước là tham số cần chọn, chiều của ma trận là chiều của ma trận đầu vào (trong bài toán này chiều của filter là 3 (số kênh màu)).
  - Kích thước của mỗi filter F*F.
  - Số ô nhat qua mỗi lần trượt.
  - Zerro padding P: Tham số quyết định ma trận đầu vào và ma trận đầu ra có cùng kích thước không (có trượt hết tất cả các ô của ma trận đầu vào không hay nói cách khác là có trượt vượt ra khỏi ma trận đầu vào không).
- Đầu ra là ma trận kích thước $W_2 * H_2 * D_2$:
  - $W_2 = (W_1 - F + 2P)/S + 1$.
  - $H_2 = (H_1 - F + 2P)/S + 1$.
  - $D_2 = K$.

Lớp RELU Layer

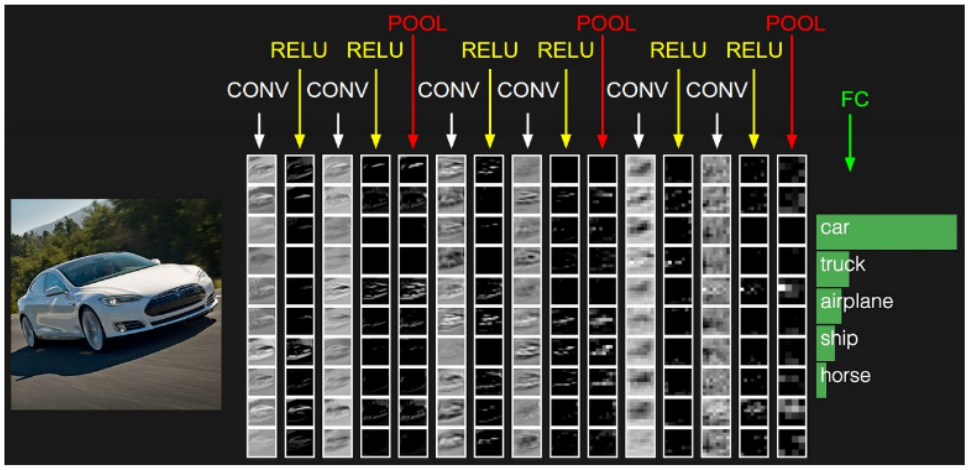
Tầng sẽ áp dụng hàm tác động, thông thường sẽ làm hàm ReLu max(0,x).Hàm sẽ tác động lên từng phần tử của ma trận đầu vào nên ma trận đi qua tầng này sẽ không thay đổi kích thước.

Lớp PoolingLayer

- Thường chèn tầng pooling vào giữa các tầng Conv để giảm kích thước ma trận.
- Nhận đầu vào là ma trận $W_1 * H_1 * D_1$ là kích thước của ma trận đầu ra của tầng Conv.
- Các tham số:
  - Kích thước của ma trận pooling $F * F$.
  - Số ô nhảy qua sau mỗi lần trượt trên ma trận input.
- Đầu ra là mậ trận kích thước $W_2 * H_2 * D_1$.
  - $W_2 = \frac{(W_1 - F)}{S} + 1$.
  - $H_2 = \frac{(H_1 - F)}{S} + 1$.
  - $D_2 = D_1$.

Lớp Fully-connected Layer

- Như mạng neuron thông thường,các neuron trong tầng Fully-connected kết nối với toàn bộ các neuron ở tầng trước nó: Nhận đầu vào là vector áp dụng hàm tác động và đưa ra đầu ra.



Lớp Max norm constraint và Dropout

Để giảm overfit ta có thể áp dụng các kỹ thuật như:

- Thêm max norm constraint cho các trọng số.
  - Khi cập nhật các trọng số thì các trường số phải thỏa mãn điều kiện:
    $\|\overrightarrow{w}\|_2 < c$(c thường được chọn 3 hoặc 4).
- Thêm tầng dropout
  - Đầu ra của các tầng trước tầng drop-out khi đi qua tầng drop-out sẽ được giữ nguyên với xác là p hoặc sẽ được chuyển thành 0 với xác suất 1-p.

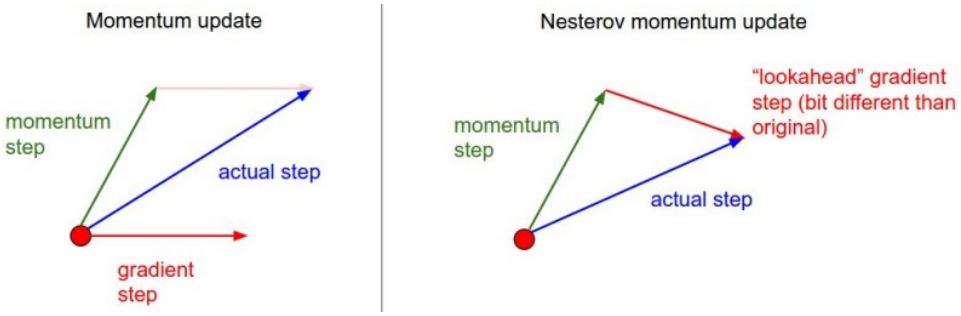
3.3.3 Update tham số

Momentum update

$$\Delta w^{t+1} = \Delta w^t + \mu.\Delta w^t - \eta.\nabla E^t.$$

Nesterov momentum

$w^{(t+1)=w^t+\mu.\Delta w^t}$
$$\Delta w^{(t+1)} = \mu.\Delta w^t - \eta.\nabla E^{t+1}.$$



3.3.4 Mạng CNN Đơn giản

Cấu trúc mạng

Cài đặt chương trình

Sử dụng thư viện keras

- Load data và xử lí dữ liệu:

```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# normalize inputs from 0-255 to 0.0-1.0
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0


# one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
```

- Thêm các tầng

```
# Create the model
model = Sequential()
model.add(Convolution2D(32,3,3,input_shape=(3,32,32),...
  border_mode='same',activation='relu',...
      W_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Convolution2D(32,3,3,activation='relu',...
  border_mode='same',...
      W_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(512,activation='relu',...
    W_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

- Các tầng Conv có các tham số:
  * Số filter: 32
  * Kích thước mỗi filter: $3 * 3 * 3$.
  * input_shape: Nhận đầu vào là ma trận ảnh $32 * 32 * 3$.
  * border_mode: kích thước ma trận đầu vào và ma trận đầu ra là như nhau (32*32).
  * W_constraint: max norm constraint với c = 3.
- Sau mỗi tầng Conv là tầng ReLu
- Tầng Drop-out thứ nhất với xác suất chuyển các tham số về 0 là 0.2
- Tầng Pooling với fiter có kích thước 2*2
- Tầng Flattern để chuyển ma trận thành vector
- Tầng Dense (Fully-connected):
  * Kích thước đầu ra: 512 (vector 512 chiều)
  * Hàm tác động: relu
  * W_constraint: max norm constraint với c = 3
- Tầng Drop-out thứ hai với xác suất chuyển các tham số về 0 là 0.2
- Tầng Dense (Fully-connected) cuối cùng:
  * Kích thước đầu ra: Số nhãn lớp (scroce cho mỗi nhàn lớp): 10 nhãn lớp
  * Hàm tác động : softmax (vector x có n chiều)

$$\sigma(x_i) = \frac{e^{x_i}}{\Sigma_{j=1}^{n} e^{x_k}}$$

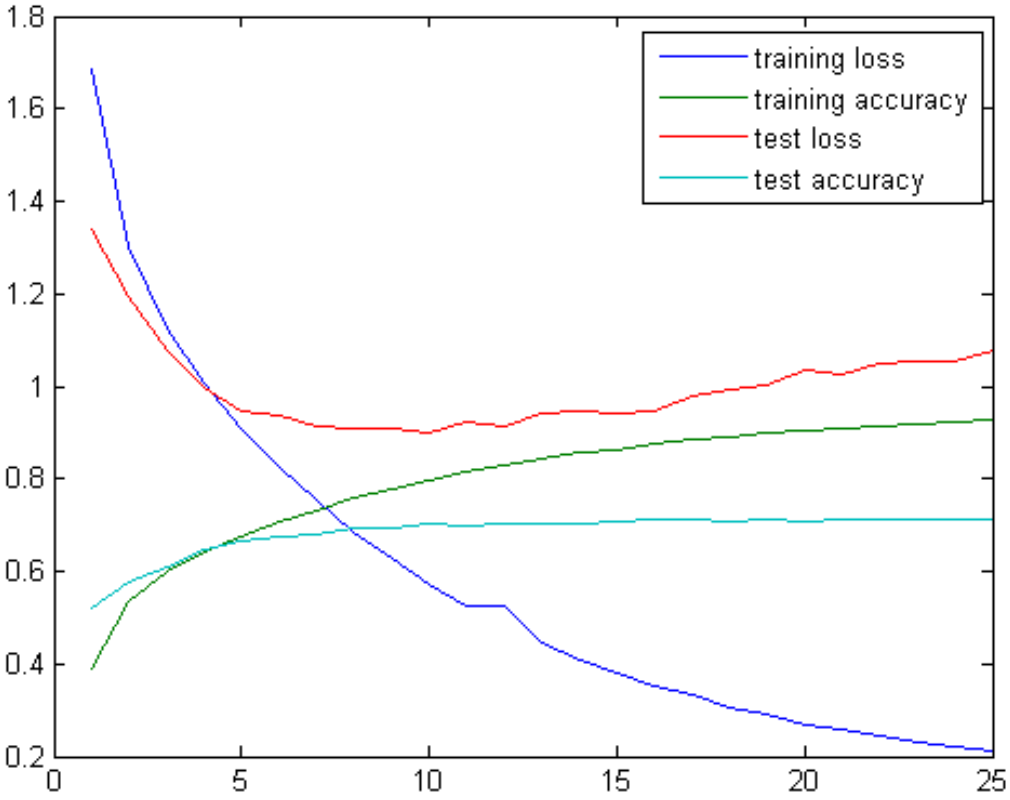
- Traning:

```
#have 5 bacth
#read batch 1
fo = open("data_batch_1", 'rb')
dict = cPickle.load(fo)
da=np.array(dict.get('data'))
lb=np.array(dict.get('labels'))
fo.close()
```

- Số epoch thực hiện 25
- Learning rate: 0.01
- Sử dụng minibatch gradient decent (số ảnh trong một batch là 32) với chiến lược tối ưu là momentum để update các tham số (tham số mu trong momentum là 0.9) không sử dụng nesterov momentum
- Hàm đánh giá lỗi: cross-entropy

$L = -\frac{1}{N}\Sigma_{i=1}^{N}[y_i \log y_i + (1 - \tilde{y_i})]$ (c là nhãn thực tế của ảnh i,$\tilde{y_i}$ là nhãn do mô hình dự đoán)

Kết quả



3.3.5 Mạng CNN Lớn

Cấu trúc mạng

INPUT → CONV → RELU → DropOut → [CONV → RELU → POOL]*2 → DropOut → CONV → RELU → POOL → FLatten[DropOut → Fully-connect(RELU)]*2 → DropOut → Ouput(num_class,activation='softmax')

Bảng 3.1: **Bảng mô tả mạng CNN**

Layer (type)	Output Shape	Param#	Connected to
convolution2d_1 (Convolution2D)	(None, 32, 32, 32)	896	convolution2d_input_1[0][0]
dropout_1 (Dropout)	(None, 32, 32, 32)	0	convolution2d_1[0][0]
convolution2d_2 (Convolution2D)	(None, 32, 32, 32)	9248	dropout_1[0][0]
maxpooling2d_1 (MaxPooling2D)	(None, 32, 16, 16)	0	convolution2d_2[0][0]
convolution2d_3 (Convolution2D)	(None, 64, 16, 16)	18496	maxpooling2d_1[0][0]
maxpooling2d_2 (MaxPooling2D)	(None, 64, 8, 8)	0	convolution2d_3[0][0]
convolution2d_4 (Convolution2D)	(None, 128, 8, 8)	73856	maxpooling2d_2[0][0]
dropout_2 (Dropout)	(None, 128, 8, 8)	0	convolution2d_4[0][0]
convolution2d_5 (Convolution2D)	(None, 128, 8, 8)	147584	dropout_2[0][0]
maxpooling2d_3 (MaxPooling2D)	(None, 128, 4, 4)	0	convolution2d_5[0][0]
flatten_1 (Flatten)	(None, 2048)	0	maxpooling2d_3[0][0]
dropout_3 (Dropout)	(None, 2048)	0	flatten_1[0][0]
dense_1 (Dense)	(None, 1024)	2098176	dropout_3[0][0]
dropout_4 (Dropout)	(None, 1024)	0	dense_1[0][0]
dense_2 (Dense)	(None, 512)	524800	dropout_4[0][0]
dropout_5 (Dropout)	(None, 512)	0	dense_2[0][0]
dense_3 (Dense)	(None, 10)	5130	dropout_5[0][0]

Tổng params: 2878186

Cài đặt chương trình

Sử dụng thư viện keras,cài đặt tương tự với mạng CNN đơn giản với các tham số:

- epochs = 25
- chiến lược tối ưu Stochastic gradient descent
  - Tỉ lệ học learn rate: lrate = 0.01
  - decay = lrate/epochs

Kết quả

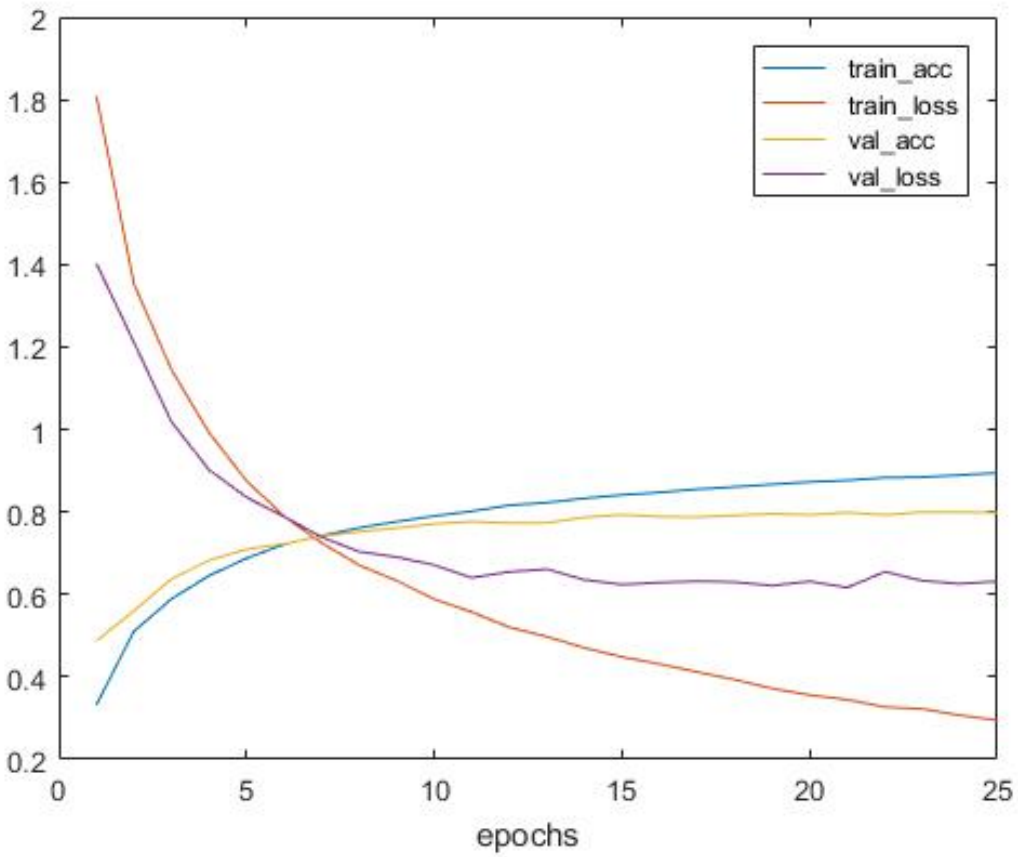
Epoch 1/25
50000/50000 [===] - 1047s - loss: 1.8077 - acc: 0.3323 - val_loss: 1.4027 - val_acc: 0.4874
Epoch 2/25
50000/50000 [===] - 1034s - loss: 1.3529 - acc: 0.5107 - val_loss: 1.2112 - val_acc: 0.5600
Epoch 3/25
50000/50000 [===] - 1035s - loss: 1.1446 - acc: 0.5892 - val_loss: 1.0184 - val_acc: 0.6379
Epoch 4/25
50000/50000 [===] - 1034s - loss: 0.9922 - acc: 0.6457 - val_loss: 0.9022 - val_acc: 0.6826
Epoch 5/25
50000/50000 [===] - 1036s - loss: 0.8755 - acc: 0.6877 - val_loss: 0.8355 - val_acc: 0.7097
Epoch 6/25
50000/50000 [===] - 1036s - loss: 0.7894 - acc: 0.7218 - val_loss: 0.7886 - val_acc: 0.7226

```
Epoch 7/25
50000/50000 [===] - 1037s - loss: 0.7251 - acc: 0.7422 - val_loss: 0.7389 - val_acc: 0.7412
Epoch 8/25
50000/50000 [===] - 1036s - loss: 0.6710 - acc: 0.7617 - val_loss: 0.7038 - val_acc: 0.7522
Epoch 9/25
50000/50000 [===] - 1036s - loss: 0.6332 - acc: 0.7766 - val_loss: 0.6907 - val_acc: 0.7613
Epoch 10/25
50000/50000 [===] - 1037s - loss: 0.5886 - acc: 0.7902 - val_loss: 0.6712 - val_acc: 0.7712
Epoch 11/25
50000/50000 [===] - 1036s - loss: 0.5572 - acc: 0.8018 - val_loss: 0.6407 - val_acc: 0.7768
Epoch 12/25
50000/50000 [===] - 1036s - loss: 0.5205 - acc: 0.8163 - val_loss: 0.6548 - val_acc: 0.7731
Epoch 13/25
50000/50000 [===] - 1036s - loss: 0.4976 - acc: 0.8226 - val_loss: 0.6613 - val_acc: 0.7737
Epoch 14/25
50000/50000 [===] - 1036s - loss: 0.4705 - acc: 0.8332 - val_loss: 0.6355 - val_acc: 0.7869
Epoch 15/25
50000/50000 [===] - 1036s - loss: 0.4486 - acc: 0.8412 - val_loss: 0.6241 - val_acc: 0.7933
Epoch 16/25
50000/50000 [===] - 1041s - loss: 0.4310 - acc: 0.8475 - val_loss: 0.6288 - val_acc: 0.7886
Epoch 17/25
50000/50000 [===] - 1037s - loss: 0.4122 - acc: 0.8552 - val_loss: 0.6321 - val_acc: 0.7877
Epoch 18/25
50000/50000 [===] - 1037s - loss: 0.3932 - acc: 0.8608 - val_loss: 0.6300 - val_acc: 0.7916
Epoch 19/25
50000/50000 [===] - 1037s - loss: 0.3721 - acc: 0.8669 - val_loss: 0.6214 - val_acc: 0.7956
Epoch 20/25
50000/50000 [===] - 1038s - loss: 0.3561 - acc: 0.8729 - val_loss: 0.6317 - val_acc: 0.7929
Epoch 21/25
50000/50000 [===] - 1037s - loss: 0.3448 - acc: 0.8765 - val_loss: 0.6168 - val_acc: 0.7987
Epoch 22/25
50000/50000 [===] - 1037s - loss: 0.3271 - acc: 0.8833 - val_loss: 0.6551 - val_acc: 0.7927
Epoch 23/25
50000/50000 [===] - 1037s - loss: 0.3225 - acc: 0.8844 - val_loss: 0.6334 - val_acc: 0.7999
Epoch 24/25
50000/50000 [===] - 1037s - loss: 0.3071 - acc: 0.8895 - val_loss: 0.6259 - val_acc: 0.7993
Epoch 25/25
50000/50000 [===] - 1037s - loss: 0.2948 - acc: 0.8946 - val_loss: 0.6318 - val_acc: 0.7991
```

Trong đó:

- loss: giá trị lỗi tập train
- acc: giá trị chính xác
- val_loss: giá trị lỗi theo tập test
- val_acc: giá trị lỗi theo tâp test

Đồ thị biểu diễn



Thực nghiệm

- Train với bộ dữ liệu là 50000 mẫu và 10000 mẫu test.
- Sử dụng phương pháp đánh giá hand-out.
  - Dữ liệu được chia làm 2 phần tách biệt không giao nhau.
  - Bộ dữ liệu với các ví dụ nhiễu lỗi ít
- Kết quả thu đươc:
  - Chạy 8h với bộ xử lý Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz - 3.0GHz
  - Chạy theo chiến lược min-batch(32).
  - Độ chính xác khi đánh giá với tập test là: 79.91%.

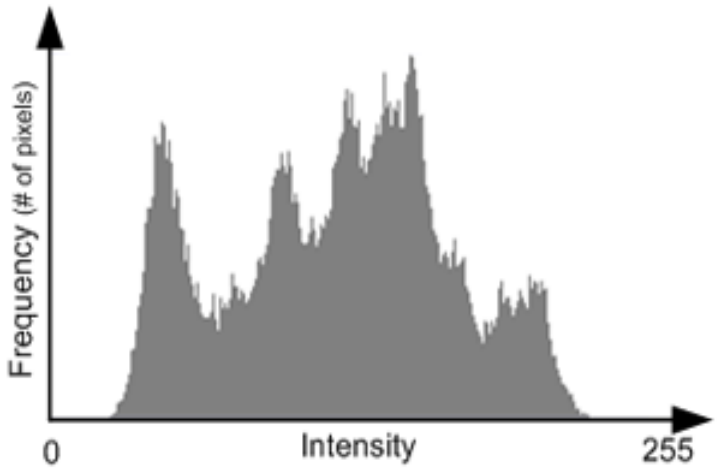
3.4 SVM

3.4.1 Cơ sở lý thuyết SVM

Khái niệm-đặc điểm

SVM là một phương pháp phân lớp tuyến tính (linear classifier), với mục đích xác định một siêu phẳng (hyperplane) để phân tách hai lớp của dữ liệu.
SVM có một nền tảng lý thuyết chặt chẽ
SVM là một phương pháp tốt (phù hợp) đối với những bài toán phân lớp có không gian rất nhiều chiều (các đối tượng cần phân lớp được biểu diễn bởi một tập rất lớn các thuộc tính)

Biểu đồ histogram: Histogram là một dạng biểu đồ biểu diễn số lượng điểm ảnh tương ứng với mức độ sáng tối của bức ảnh.



Hình 3.1: Histogram

3.4.2 Cơ sở lý thuyết

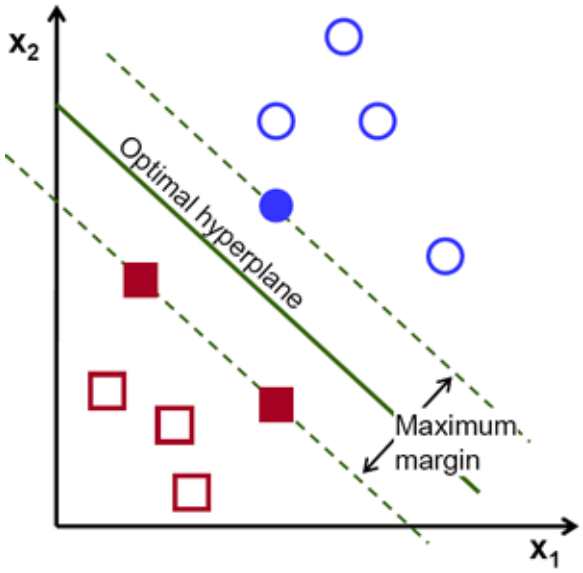
Biểu diễn tập huấn luyện

$$x_1, y_1, (x_2, y_2), ...(x_r, y_r)$$

- x_i là một vectơ đầu vào $x_i \subseteq R^n$
- $y_i \in 1, -1$ là một nhãn lớp Hàm phân tách tuyến tính

$$f(x) =< w.x > +b$$

w là vectơ trọng số các thuộc tính, b là một giá trị số thực



Hình 3.2: Siêu phẳng phân tách

Khoảng cách từ một điểm đến siêu phẳng

$$\frac{|<w.x_i>+b|}{||w||}$$

$$margin = d_+ + d_- = \frac{2}{||w||}$$

Tương đương bài toán cực tiểu hóa $\frac{<w.w>}{2}$ với điều kiện

$$\begin{cases} <w.x_i>+b \geq 1 if y_i = 1 \\ <w.x_i>+b \leq -1 if y_i = -1 \end{cases}$$

Áp dụng phương pháp Lagrange

$$L = 1/2||\overline{w}||^2 - \sum \alpha_i[y_i(\overline{w}.\overline{x_i}+b)-1]$$

Điều kiện

$$\frac{\partial}{\partial \overline{w}} = \overline{w} - \sum \alpha_i y_i x_i = 0 \Longrightarrow \overline{w} = \sum \alpha_i y_i \overline{x_i}$$

$$\frac{\partial}{\partial b} = -\sum \alpha_i y_i \Longrightarrow \sum x_i y_i = 0$$

Thế vào ta được biểu thức :

$$L_D(\alpha) = \sum_{i=1}^{r} \alpha_i - 1/2 \sum_{i,j=1}^{r} \alpha_i \alpha_j y_i y_j <x_i.x_j>$$

Điều kiện:

$$\begin{cases} \sum_{i=1}^{r} \alpha_i y_i = 0 \\ \alpha_i \geq 0 \forall i = 1..r \end{cases}$$

Dung phương pháp lặp giải cuối cùng thu được

$$w^* = \sum_{x_i \in SV} \alpha_i y_i x_i$$

$$b^* = y_k - < w*, x_k >$$

Để phân lớp cho giá trị mới ta tìm dấu của siêu phẳng

$$f(x) = \sum_{x_i \in SV} \alpha_i y_i < x_i.x > + b^*$$

Nới lỏng điểu kiện :

$$L_D(\alpha) = \sum_{i=1}^{r} \alpha_i - 1/2 \sum_{i,j=1}^{r} \alpha_i \alpha_j y_i y_j < x_i.x_j >$$

$$\begin{cases} \sum_{i=1}^{r} \alpha_i y_i = 0 \\ 0 \leq \alpha_i \leq C \forall i = 1..r \end{cases}$$

Phân lớp không tuyến tính một vài hàm nhân: Đa thức $K(x,z) = (< x.z > + \Theta)^d$ Gaussian RBF $K(x,z) = e^{\frac{||x-z||^2}{2*\sigma}}$ trong đó $\sigma > 0$ Phân lớp nhiều nhãn 2 chiến lược :

- one-versus-rest
- one-versus-one

3.4.3 Cài đặt áp dụng với bài toán phân loại ảnh

Công cụ

Ngôn ngữ lập trình python, thư viện học máy scikit-learn, opencv Đọc dữ liệu :

```
#have 5 bacth
#read batch 1
fo = open("data_batch_1", 'rb')
dict = cPickle.load(fo)
da=np.array(dict.get('data'))
lb=np.array(dict.get('labels'))
fo.close()
```

Huấn luyện mô hình kernel là hàm nhân sử dụng (rbf,linear,poly) C là tham số nới lỏng mô hình, gamma tham số ảnh của hàm nhân(ảnh hưởng đến giá trị). Với hàm poly ta còn có tham số degree(mũ) và coef0(b) phương thức fit của SVC dùng để train mô hình

```
clf = SVC(C=5, kernel='rbf',gamma='auto')
from datetime import datetime
print("start",str(datetime.now()))
clf.fit(trainX, trainY)
print("end train",str(datetime.now()))
```

Đoán tập test: phương thức predict dùng để đoán nhãn cho tập test

```
res=clf.predict(testX)
print("end predict ",str(datetime.now()))
sol=(res==testY)
print(res[:10])
print(testY[:10])
print(sol[:10])
print(sol.sum())
```

Sử dụng histogram để xây dựng đầu vào cho SVM :

Histogram với rbg color: sử dụng cv2 để tạo ra ma trận histogram , cv nhận đầu vào là ảnh nên ta phải chuyển ma trận từ dữ liệu thành ma trận ảnh rbg kích thước 32 32 3.
Các tham số của hàm calcHist như sau

- Đầu vào ảnh
- Số đại diện cho màu cần tạo histogram [0 1 2] tức là cả 3 màu r b g
- bin tương ứng từng màu với dữ liệu ảnh theo (1) khuyên nên chọn 16
- Vùng màu chọn

```
lhistr=[]
for i in range(50000):
    #print i
    histr = cv2.calcHist(da[i].reshape((3,1024)).T.reshape((32,32,3))
    lhistr.append(histr.ravel())

trainX=np.array(lhistr)
```

Histogram với gray color: Sử dụng hàm chuyển đổi màu của opencv cvtColor() việc này đồng nghĩa sẽ giảm số chiều của dữ liệu xuống bằng bins ở đây là 16.

```
lhistr=[]
for i in range(50000):
    #print i
    histr = cv2.calcHist(cv2.cvtColor(da[i].reshape((3,1024)).T.resha
    lhistr.append(histr.ravel())

trainX=np.array(lhistr)
```

Phương pháp đánh giá sử dụng: hold on

3.4.4 Kết quả thực nghiệm

Áp dụng toàn bộ với chiền lược one vs all

- SVM rbf C=5 autosklearn=(gamma=1/3072(xích ma=3072)) độ chính xác $\approx 25\%$

- SVM poly C=5 autosklearn=(gamma=1/3072 n=3) độ chính xác $\approx 38,2\%$
- SVM linear C=5 độ chính xác $\approx 32\%$
- SVM poly histogram bin=16 C=5 autosklearn=(gamma=1/3072 n=3) độ chính xác $\approx 37,6\%$
- SVM poly gray histogram bin=16 C=5 autoskcearn=(gamma=1/3072 n=3) độ chính xác $\approx 29,6\%$

Phần 4

Kết luận

4.1 So sánh các phương pháp

4.2 Khó khăn gặp phải

Phương pháp SVM

- Bộ dữ liệu lớn train rất lâu poly (5 tiếng) rbf(10 tiếng) rất khó để xử lý chọn các tham số
- SVM rất nhiều tham số cần phải thử nghiệm rất nhiều

Phương Pháp CNN

- Phần cứng chưa đủ mạnh trên dữ liệu lâu (6h - 8h).

4.3 Kinh nghiệm rút ra được

Phần 5

Tài liệu tham khảo

[+] Slide Học Máy TS. Thân Quang Khoát
[+] Opencourse in Mit-SVM
[+] http://www.svm-tutorial.com/2016/09/unconstrained-minimization/
[+] https://ongxuanhong.wordpress.com/2015/09/19/support-vector-machine
[+] http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.htm
[+] SVMs for Histogram-Based Image Classification
Olivier Chapelle, Patrick Haffner and Vladimir Vapnik
[+] http://docs.opencv.org/3.1.0/d1/db7/tutorial_py_histogram_begins.htm
[+] http://cs231n.github.io/convolutional-networks/
[+] https://keras.io/getting-started/sequential-model-guide/