Machine Learning Project Report

Multi-classification on different objects based on LSUN

Team menbers:

Tianyin Huang	N13743096	th2227
Haonan Wang	N10480187	hw2204
Ruiyang Chen	N18051385	rc3877
Kaiyu Yin	N19172766	ky1161

1. Brief overview

We have a large-scale image dataset called LSUN, which contains several different kinds of labeled images. Our goal is to classify different objects based on LSUN dataset using different methods, test the accuracy of these model, analysis the results and comparing three methods. In this project, we use GPU(Tesla p100) on GCP because of large dataset. And three objects we recognize are cat, dog and wolf.

2. Formulation

2.1 SVM approach:

Hyperplane:

$$Z = b + w_1 x_{i1} + \cdots + w_d x_{id}$$

C and λ:

(a) λ: margin line distance

$$m = \frac{\gamma}{\|w\|}$$

(b) C: use for regularization to minimize loss

$$J(w,b) = C\sum_{i=1}^{N} \max(0, 1 - y_i(w^T x_i + b)) + \frac{1}{2}\|w\|^2$$

RBF kernel:

$$K(x, x') = \exp[-\lambda \|x - x'\|^2]$$

$\frac{1}{\lambda}$ indicates width of kernel

2.2 PCA approach:

Standard Scaler: $Xtr = \frac{X - \mu}{s}$

μ means the mean of the training samples

s means the standard deviation of the training samples

PCA :

$$z = \sum_{j=1}^{p} \alpha_j v_j$$

Coefficient expansion of data sample:

$$x_i = \bar{x} + \sum_{j=1}^{p} \alpha_{ij} v_j$$

Approximation with d coefficients:

$$\hat{x}_i = \bar{x} + \sum_{j=1}^{d} \alpha_{ij} v_j$$

n_components:

$$n\_componets = \frac{\sum_{j=1}^{d} \lambda_j}{\sum_{j=1}^{p} \lambda_j}$$

d means components be chosen as PCA

λ_j means corresponding eigenvalues

Svd_solver:

$$X = U\Sigma V^T$$

coefficients are

$$Z = XV = U\Sigma$$

2.3 CNN approach:

Hidden activation function:

Sigmoid: $g_{act}(z) = \frac{1}{1+e^{-z}}$

RELU: $g_{act}(z) = \max(0, z)$

Output activation function:

binary classification:

$\hat{y} = sign(z_0)$ hard decision

$P(y=1 \mid x) = \frac{1}{1+e^{-z_o}}$ soft decision

Multi-classification:

$\hat{y} = \arg\max z_{o,k}$ hard decision

$P(y=k \mid x) = S_k(z_o), S_k(z_o) = \frac{e^{z_{o,k}}}{\sum_l e^{z_{o,l}}}$ soft decision

Regression:

$$\hat{y} = z_o$$

Model compile:

Loss function categorical_crossentropy:

$$C = -\frac{1}{n}\sum_x [y \ln a + (1-y)\ln(1-a)]$$

y means the expected output

a means the actual output

Optimizer Adam: $\hat{m}_\omega = \frac{m_\omega^{t+1}}{1-\beta_1^{t+1}}$ $\hat{v}_\omega = \frac{v_\omega^{t+1}}{1-\beta_2^{t+1}}$ $\omega^{t+1} = \omega^t - \eta \frac{\hat{m}_\omega}{\sqrt{\hat{v}_\omega} + \varepsilon}$

Convolution: $z[i_1, i_2, m] = \sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} \sum_{n=0}^{N_{in}-1} W[k_1, k, n, m] X[i_1 + k_1, i_2 + k_2, n] + b[m]$

3. Approach

The project has two part. The first part is classsifying with SVM and PCA. The second part is classsifying with cnn..

3.1 processing data

Before classification, we need to processing data.

We import libraries at first. Also, the code shows that we are using GPU.

```
1. import numpy as np
2. import matplotlib.pyplot as plt
3. from tensorflow.keras.preprocessing.image import ImageDataGenerator
4. from tensorflow.python.client import device_lib
5. print(device_lib.list_local_devices())
```

OUTPUT:

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 17800032341924909121
, name: "/device:XLA_GPU:0"
device_type: "XLA_GPU"
memory_limit: 17179869184
locality {
}
incarnation: 5196357083987651747
physical_device_desc: "device: XLA_GPU device"
, name: "/device:XLA_CPU:0"
device_type: "XLA_CPU"
memory_limit: 17179869184
locality {
}
incarnation: 8920218747135598995
physical_device_desc: "device: XLA_CPU device"
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 69074944
locality {
  bus_id: 1
  links {
  }
}
incarnation: 9825840636830917610
physical_device_desc: "device: 0, name: Tesla P100-PCIE-16GB, pci bus id: 0000:00:0
4.0, compute capability: 6.0"
]
```

Then, we set the image size 150*150 to make sure a high quality input. And we create a generater to load data. We use ImageDataCenter() in keras which can fetch images on the fly from a directory of images. Because of the limited training data, ImageDataCenter() helps to augment data by creating random deformations of the image to expand the total dataset size. From the result, we can see the total number of images and classes.

```
1. nrow = 150
2. ncol = 150
3.
4. train_data_dir = './train'
5. train_datagen = ImageDataGenerator(rescale=1./255,
6.                                     shear_range=0.2,
```

```
                                    zoom_range=0.2,
                                    horizontal_flip=True)
train_generator = train_datagen.flow_from_directory(
                        train_data_dir,
                        target_size=(nrow,ncol),
                        batch_size=1000,
                        class_mode='sparse')

test_data_dir = './test'
test_datagen = ImageDataGenerator(rescale=1./255,
                                    shear_range=0.2,
                                    zoom_range=0.2,
                                    horizontal_flip=True)
test_generator = test_datagen.flow_from_directory(
                        test_data_dir,
                        target_size=(nrow,ncol),
                        batch_size=300,
                        class_mode='sparse')

# Display the image
def disp_image(im):
    if (len(im.shape) == 2):
        # Gray scale image
        plt.imshow(im, cmap='gray')
    else:
        # Color image.
        im1 = (im-np.min(im))/(np.max(im)-np.min(im))*255
        im1 = im1.astype(np.uint8)
        plt.imshow(im1)

    # Remove axis ticks
    plt.xticks([])
    plt.yticks([])
```

OUTPUT:

```
Found 3000 images belonging to 3 classes.
Found 900 images belonging to 3 classes.
```

Before the training, we display some labeled images. We can see that in the training data, cat has label 0, dog has label 1, and wolf has label 2.

```
Xtr, ytr = train_generator.next()
Xts, yts = test_generator.next()
Xtr_gray = np.dot(Xtr,[0.299,0.587,0.114]).reshape((1000,nrow*ncol))
Xts_gray = np.dot(Xts,[0.299,0.587,0.114]).reshape((300,nrow*ncol))
plt.figure(figsize=(16,16))
for i in range(10):
    plt.subplot(1,10,i+1)
    disp_image(Xtr[i])
    plt.title(int(ytr[i]))
```

OUTPUT:

3.2 SVM approach

Now, we use SVM to classify the data. And we choose the best C and gamma from the valve we set before. We can see that the best accuracy is only 45%. Therefoe, SVM isn't a good method.

```
from sklearn import svm

Xtr_s = Xtr.reshape((1000,nrow*ncol*3))
Xts_s = Xts.reshape((300,nrow*ncol*3))

C_test = [0.1,1,10]
gam_test = [0.001,0.01,0.1]

nC = len(C_test)
ngam = len(gam_test)
acc = np.zeros((nC,ngam))

# Measure and print the accuracy for each C and gamma value.  Store the results in acc
for i in range(nC):
    for j in range(ngam):
        svc = svm.SVC(kernel='rbf',C=C_test[i],gamma=gam_test[j])
        svc.fit(Xtr_s,ytr)
        yhat_ts = svc.predict(Xts_s)
        acc[i][j] = np.mean(yhat_ts == yts)

# Print the maximum accuracy and the corresponding best C and gamma
C_best = C_test[np.where(acc==np.max(acc))[0][0]]
gam_best = gam_test[np.where(acc==np.max(acc))[1][0]]
print('The maximum accuracy is',np.max(acc))
print('The corresponding best C and gamma are',C_best,'and',gam_best)
```

OUTPUT:

```
The maximum accuracy is 0.45
The corresponding best C and gamma are 10 and 0.001
```

3.3 PCA approach

Then we use PCA as a pre- processing step before logistic regression. And the accuracy is 37%, which is low effective.

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression

```

```
scaler = StandardScaler()
scaler.fit(Xtr_gray)
Xtr_scl = scaler.transform(Xtr_gray)

pca = PCA(n_components=0.95,svd_solver='full')
pca.fit(Xtr_scl)
Ztr = pca.transform(Xtr_scl)

logreg = LogisticRegression(multi_class='auto',solver='newton-cg')
logreg.fit(Ztr,ytr)

Xts_scl = scaler.transform(Xts_gray)
Zts = pca.transform(Xts_scl)
yhat = logreg.predict(Zts)

acc = np.mean(yhat==yts)
acc
```

OUTPUT:

$$0.37333333333333335$$

3.4 CNN approach

Then we use CNN method to classify the data. We import libraries first.

```
from tensorflow.keras import applications
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import optimizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout, Flatten, Dense
from sklearn.preprocessing import LabelEncoder
import numpy as np
import matplotlib.pyplot as plt
```

Then we create CNN model. We add four layers reference the LAB from previous lab. Flatten() layer reshapes the outputs to a single channel. Fully-connected layer with 256 output units and relu activation. A Dropout(0.5) layer to prevent over-fitting. A final fully-connected layer. Since this is multi-classification, there should be three output and we use softmax activation.

```
import tensorflow.keras.backend as K
K.clear_session()
nrow = 150
ncol = 150
model = Sequential()
for layer in base_model.layers:
    model.add(layer)
input_shape=(nrow,ncol,3)
base_model = applications.VGG16(weights='imagenet',input_shape=input_shape,include_top=False)
model = Sequential()
```

```
for layer in base_model.layers:
    model.add(layer)
for layer in model.layers:
    layer.trainable=False
model.add(Flatten())
model.add(Dense(256,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3,activation='softmax'))
```

And we can see the detail by model.summary():

```
model.summary()
```

OUTPUT:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 256)               2097408
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 771
=================================================================
Total params: 16,812,867
Trainable params: 2,098,179
Non-trainable params: 14,714,688
```

Now, we load data and use ImageDataCenter() to augment the data just like the step in part one. And we display several labeled images.

```
train_data_dir = './train'
batch_size = 32
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True)
train_generator = train_datagen.flow_from_directory(
                          train_data_dir,
                          target_size=(nrow,ncol),
                          batch_size=batch_size,
                          class_mode='categorical')
test_data_dir = './test'
test_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True)
test_generator = test_datagen.flow_from_directory(
                          test_data_dir,
                          target_size=(nrow,ncol),
                          batch_size=batch_size,
                          class_mode='categorical')
def disp_image(im):
    if (len(im.shape) == 2):
        # Gray scale image
        plt.imshow(im, cmap='gray')
    else:
        # Color image.
        im1 = (im-np.min(im))/(np.max(im)-np.min(im))*255
        im1 = im1.astype(np.uint8)
        plt.imshow(im1)

    # Remove axis ticks
    plt.xticks([])
    plt.yticks([])
Xtr,ytr=train_generator.next()
n=8
plt.figure(figsize=(20,20))
for i in range(n):
    plt.subplot(1,n,i+1)
    disp_image(Xtr[i])
    #plt.title(int(ytr[i]))
    plt.title(np.where(ytr[i]==1)[0][0])
```

OUTPUT:

```
Found 3000 images belonging to 3 classes.
Found 900 images belonging to 3 classes.
```

Because of multi-classification, we use categorical_crossentropy as loss function. After compile, we fit the data for 5 epochs

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
steps_per_epoch =  train_generator.n // batch_size
validation_steps = test_generator.n // batch_size
nepochs = 5  # Number of epochs
# Call the fit_generator function
hist = model.fit_generator(
    train_generator,
    steps_per_epoch=steps_per_epoch,
    epochs=nepochs,
    validation_data=test_generator,
    validation_steps=validation_steps)
```

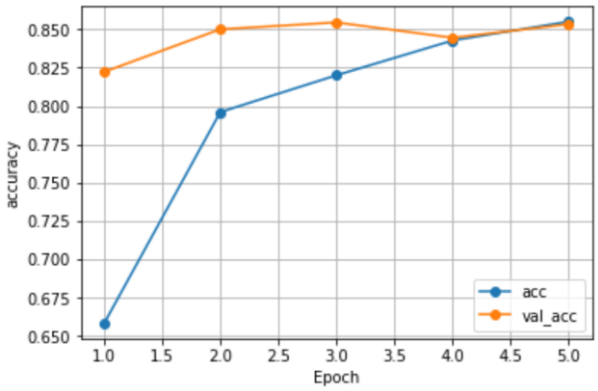
OUTPUT:

```
WARNING:tensorflow:From /usr/local/lib/python3.5/dist-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/5
29/29 [==============================] - 9s 319ms/step - loss: 0.4731 - acc: 0.8222
94/94 [==============================] - 44s 470ms/step - loss: 0.9298 - acc: 0.6583 - val_loss: 0.4731 - val_acc: 0.8222
Epoch 2/5
29/29 [==============================] - 8s 281ms/step - loss: 0.4077 - acc: 0.8500
94/94 [==============================] - 28s 298ms/step - loss: 0.5126 - acc: 0.7960 - val_loss: 0.4077 - val_acc: 0.8500
Epoch 3/5
29/29 [==============================] - 8s 279ms/step - loss: 0.3985 - acc: 0.8544
94/94 [==============================] - 28s 294ms/step - loss: 0.4573 - acc: 0.8200 - val_loss: 0.3985 - val_acc: 0.8544
Epoch 4/5
29/29 [==============================] - 8s 281ms/step - loss: 0.3741 - acc: 0.8444
94/94 [==============================] - 28s 296ms/step - loss: 0.4159 - acc: 0.8427 - val_loss: 0.3741 - val_acc: 0.8444
Epoch 5/5
29/29 [==============================] - 8s 280ms/step - loss: 0.3701 - acc: 0.8533
94/94 [==============================] - 28s 294ms/step - loss: 0.3830 - acc: 0.8550 - val_loss: 0.3701 - val_acc: 0.8533
```

Print training accuracy and validation accuracy, we can see that accuracy is close to 86%.

```
acc=hist.history['acc']
val_acc=hist.history['val_acc']
n=len(acc)
plt.plot(np.arange(1,n+1),acc,'o-')
plt.plot(np.arange(1,n+1),val_acc,'o-')
plt.legend(['acc','val_acc'])
plt.grid()
plt.xlabel('Epoch')
plt.ylabel('accuracy')
```

OUTPUT:



To find the reason why the accuracy is 86%, we also write code to plot some of error predicts.

```
n_error=4
X_err=[]
y_err=[]
yhat_err=[]
err=[]
yts2=[]
n=0
while(n<n_error):
    Xts,yts=test_generator.next()
    for j in range(yts.shape[0]):
        yts2.append(np.where(yts[j]==1)[0][0])
    z=model.predict(Xts)
    yhat=np.argmax(z,axis=1)
    err=np.where(yhat!=yts2)[0]
    for i in err:
            n+=1
            X_err.append(Xts[i])
            y_err.append(yts2[i])
            yhat_err.append(yhat[i])
plt.figure(figsize=(20,20))
for i in range(n_error):
    plt.subplot(1,n_error,i+1)
    disp_image(X_err[i])
    title='true={0:d} est={1:d}'.format(int(y_err[i]),int(yhat_err[i]))
    plt.title(title)
```

OUTPUT:

4. Analysis

From results, wee see that the accuracy of classification with SVM and PCA are much lower than the accuracy of CNN. It doesn't mean SVM and PCA are not good. They are just not suitable for this multiclassification. We will discuss according to their basic principle.

SVM doesn't have deep learning network, which is used to analysis different kinds of features. And in SVM, RGB value of pixels is its feature. This method can work well with recognition handwriting words. Because handwriting words are actually black and white pixels, and the number of pixels is small. But it can't work to recognize pictures like animals. Because they are more complicated and have lots of features and colors. So SVM has a low accuracy.

PCA will do data dimension reduction to all pixels. And it can't extract features like CNN. Moreover, both SVM and PCA ravel the images, we lost the positional relation between different pixels, like edge detection, which plays a important role in recognizing animals. Instead, CNN will keep all these features because it has convolutional layers. So CNN will has the best accuracy.

In this project, classification with CNN has accuracy of 86%. But in previous lab in class, we usually have accuracy more than 90%. We try to find the reason. One is about the dataset. When plot the error images, we find that some error image are blurred and we can't see exactly what it is with eyes, let alone the model. Additionally, we only use four layer to train the data. The result will be better if more well-designed layers are used.