Machine Learning – HW3

Li-Hen Chen

Department of Computer Science & Engineering, Texas A&M University

Question 1: Convolution Operations

(a) Consider a 1-dimensional signal $\mathrm{F} = [1\ 2\ 1\ 3\ 2\ 3\ 1\ 2\ 3\ 8\ 7\ 8\ 9\ 9\ 7\ 8]$

The 1-dimensional convolution of the signal F with a filter F at point i is defined as:

$$(\mathrm{F} * \mathrm{W})[\mathrm{i}] = \sum_{-\infty}^{\infty} F[j]W[i-j]$$

(a.i) Calculate the convolution of F with filter $\mathbf{W_1}$ = [1 1 1].

Let's operate to each element. (using 0~2 represents i and j in an array and ignoring the edge elements)

$$(\mathrm{F} * \mathrm{W})[0] = F[2]W[-2] + F[1]W[-1] + F[0]W[0] = 1 + 2 + 1 = 4$$

$$(\mathrm{F} * \mathrm{W})[1] = F[2]W[-2] + F[1]W[-1] + F[0]W[0] = 3 + 1 + 2 = 6$$

$$\vdots$$

$$(\mathrm{F} * \mathrm{W})[14] = F[14]W[-2] + F[14]W[-1] + F[14]W[0] = 24$$

When ignoring the edge elements and only compute the convolution for those elements which have all three neighbors, the result is [4, 6, 6, 8, 6, 6, 6, 13, 18, 23, 24, 26, 25, 24].

What the operation W_1 filter performs is to add up the neighbor elements to get the sum of elements in this filter window.

(a.ii) Calculate the convolution of F with filter $\mathbf{W_2}$ = [1 0 −1]

As the above operation.

$$(\mathrm{F} * \mathrm{W})[0] = F[2]W[-2] + F[1]W[-1] + F[0]W[0] = 1 + 0 - 1 = 0$$

$$(\mathrm{F} * \mathrm{W})[1] = F[2]W[-2] + F[1]W[-1] + F[0]W[0] = 3 + 0 - 2 = 1$$

$$\vdots$$

When ignoring the edge elements and only compute the convolution for those elements which have all three neighbors, the result is [0, 1, 1, 0, -1, -1, 2, 6, 4, 0,2, 1, -2, -1].

What the operation W_2 filter performs is to subtract the neighbor elements, which means keeping the high frequency information in the signal. This kind of operation is like the edge extraction in 2D filter for an image.

(b) Consider an image $I \in R^{5\times5}$ and a kernel filter $F \in R^{3\times3}$. The convolution operation between I and F at pixel [m, n] is defined as:

$$(\mathrm{I} * \mathrm{F})[\mathrm{m}, \mathrm{n}] = \sum_{-\infty}^{\infty}\sum_{-\infty}^{\infty} I[i,j]F[m-i, n-j]$$

(b.i) (2 points) Implement a 2-d convolution between the image and the filter.

In the following, consider the following 5 × 5 image:

$$I = \begin{bmatrix} 164 & 188 & 164 & 161 & 195 \\ 178 & 201 & 197 & 150 & 137 \\ 174 & 168 & 181 & 190 & 184 \\ 131 & 179 & 176 & 185 & 198 \\ 92 & 185 & 179 & 133 & 167 \end{bmatrix}$$

The implementation of the 2-d convolution between image and the filter can be found at the appendix, where the function is called convolve2d_scratch. Then I utilize this function to finish (b.ii)'s question.

(b.ii) Using the function from (b.i), compute the 2-d convolution between the above image I and the following filters:

$$F_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$F_2 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$F_3 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Result of F1:

$$\begin{bmatrix} 1615 & 1600 & 1559 \\ 1585 & 1627 & 1598 \\ 1465 & 1576 & 1593 \end{bmatrix}$$

Result of F2:

$$\begin{bmatrix} 13 & -43 & -64 \\ 112 & 29 & -110 \\ 50 & 44 & 133 \end{bmatrix}$$

Result of F3:

$$\begin{bmatrix} 395 & 370 & -59 \\ 95 & 183 & 302 \\ 325 & 184 & 257 \end{bmatrix}$$

- **Discussion**

First of all, if we would like to have a reasonable result, the sum of the elements in the filter should be “0”.

If “I” is an image, we can tell from the above results that only the filter 2 has the reasonable result (0~255). For filter 1, if it multiplies 1/9, it is a common mean filter for an image. The idea of mean filtering is simply to replace each pixel value in an image with the mean (`average') value of its neighbors, including itself. This has the effect of eliminating pixel values which are unrepresentative of their surroundings.

For filter 2, it is a filter that keeps the high frequency of the image of horizontal direction. It is designed to respond maximally to edges running horizontally relative to the pixel grid. The kernels can be applied to the input image, to produce measurements of the gradient component in horizontal orientation, or to gain edge information.

For filter 3, it is kind of Laplace filter, which substitutes the element "9" into "8". Laplacian filters are derivative filters used to find areas of rapid change (edges) in images.

Question 2: Image processing for human faces

(a) Implement Principal Component Analysis (PCA) on the input images. Assume that the input vector of PCA contains all rows of an image stacked one on top of the other.

There are five steps for us to implement PCA algorithm.

1. Mean normalize input features

2. Compute covariance matrix $\mathrm{S} = \mathrm{cov(X)} = \left(\frac{1}{\mathrm{n}}\right) X^T X$

3. Diagonalize S and find the eigenvector matrix P

4. Take the first M eigenvectors or principal components and form reduced matrix U.

5. Project data into reduced space.

Code Implementation:

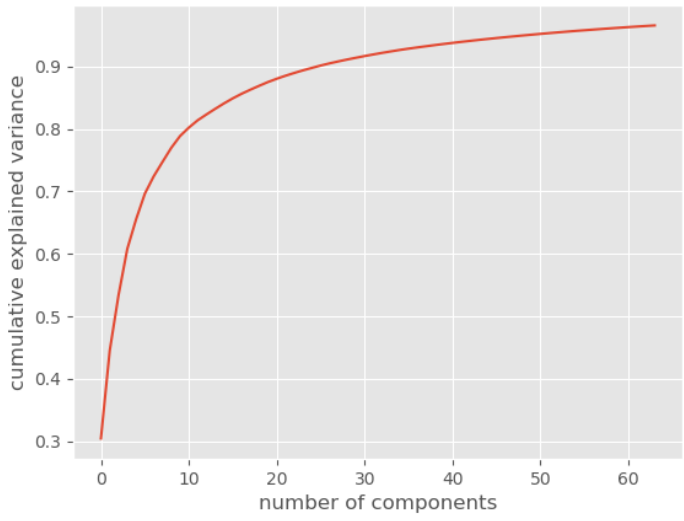
```
def PCA_scratch(A):
    M = np.mean(A.T,axis=1)
    Center = A-M
    V = np.cov(Center.T)
    values,vectors = np.linalg.eig(V)
```

```
P = vectors.T.dot(Center.T)
return P
```

(b) (0.5 points) Plot a curve displaying the first k eigenvalues $\lambda_1,...,\lambda_K$, i.e. the energy of the first K principal components. How many components do we need to capture 50% of the energy?

components	1	2	3	4	5	6	7	8	9	10
energy	30.47%	44.51%	53.39%	60.82%	65.54%	69.66%	72.42%	74.72%	76.98%	78.87%

First, Let's talk about PCA. Principal Component Analysis performs a linear mapping of the data to a lower-dimensional space in such a way that the variance of the data in the low-dimensional representation is maximized. PCA it's a dimensionality reduction algorithm. In this question, I perform PCA with 64 eigenvalues and plot it with cumulative experienced variance. The above table shows the first 10 components with its energy. The following image curve quantifies how much of the total, 64-dimensional variance is contained within the first N components. We need 3 components to capture 50% of energy. Besides, we see that with the digits the first 10 components contain approximately 78% of the variance, while you need around 50 components to describe close to 95% of the variance.



(c) Plot the top 10 eigenfaces, i.e. the eigenvectors u_k, k = 1, . . . , 10 obtained by PCA.

Eigenfaces are the principal components of a distribution of faces, or equivalently, the eigenvectors of the covariance matrix of the set of face images, where an image with N pixels is considered a point in N-dimensional space. The following figure is the result of top 10 eigenfaces obtained by PCA

(d) Select a couple of images from the data. Use the first k eigenfaces as a basis to reconstruct the images. Visualize the reconstructed images using 1, 10, 20, 30, 40, 50 components. How many components do we need to achieve a visually good result?

I use 3 faces to reconstruct the images using 1, 10, 20, 30, 40, 50, 60 components. We can tell from the following table that with 30 components the result is visually good. From components 30 to 60, I consider that the result only improves a little which corresponds to the question (b) because the result only improves from 92% to 96 %. Although it can still be recognized, compared to 1 to 30, it is relatively slight.

1	10	20	30	40	50	60	Origin

(e) Perform face recognition: Split the input data into training and testing making sure that every person is included in each set. Use any of the classification methods that we have learnt so far to recognize a person's identity. Use as input features the transformed feature space that resulted from PCA. Experiment with different number of PCA components through a 5-fold cross-validation. Report the recognition accuracy on the test set.

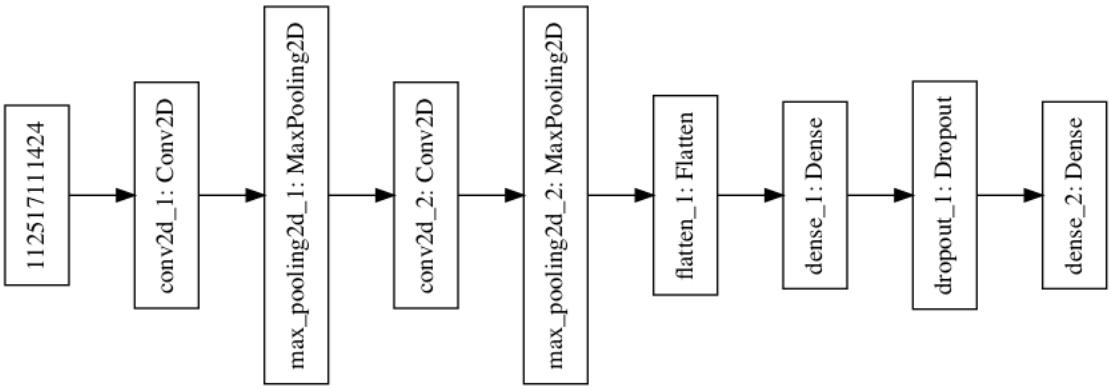
I choose 3 algorithms that we have learnt so far including multiclass Logistic regression, Random Forest and SVM and perform 5-fold cross-validation on different number of principal components (1,2,3,10,20,30,40,50). The results are shown in the following table. Indeed, the more numbers of principal components, the result is better, because it contains more information. However, in practical, it's a trade of between accuracy and computation cost.

# of PC	Logistic	SVM (linear) C=5	Random Forest
1	0.2706 (+/- 0.15)	0.2662 (+/- 0.07)	0.3331 (+/- 0.13)
2	0.2682 (+/- 0.03)	0.3270 (+/- 0.06)	0.5085 (+/- 0.12)
3	0.4883 (+/- 0.10)	0.5353 (+/- 0.12)	0.6036 (+/- 0.12)
10	0.8144 (+/- 0.18)	0.7861 (+/- 0.13)	0.7580 (+/- 0.12)
20	0.8413 (+/- 0.15)	0.8332 (+/- 0.13)	0.8436 (+/- 0.19)
30	0.8661 (+/- 0.11)	0.8451 (+/- 0.13)	0.8746 (+/- 0.16)
40	0.8907 (+/- 0.09)	0.8643 (+/- 0.11)	0.8538 (+/- 0.13)
50	0.8907 (+/- 0.09)	0.8579 (+/- 0.08)	0.8745 (+/- 0.12)

(f) Face recognition with CNNs: Use a CNN to perform face recognition in the augmented Yale Face Dataset. Use the same split for the train and test set as in

Question 2e. Experiment with different CNN parameters, e.g. # layers, filter size, stride size, activation function, dropout, pool size, etc.

Let's use a CNN to perform face recognition in the augmented Yale Face Dataset. First, we use data augmentation to augment the dataset in order to pursue a better performance. The detail of data augmentation will be discussed in the (g) part. We first focus on the CNN and its result on face recognition. The following figure is a simple CNN model for training. The filter size is basically 3x3. The activation function is "Relu" which is effective for image classification, final we have 1 fully connected layer with 1000 parameters and feed the results into activation function "Softmax" to classify whose face is. The architecture of the CNN is listed below. In order to prevent overfitting, I experiment the model dropout (0.5) which is normally used in image classification and without dropout. The results are the same but the loss value is relatively small when using dropout, which can be seen in the figure and the table. But I only have the epoch to be 10 because my computer does not have GPU. I have run the model on CPU and it's terribly slow.



Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 239, 316, 32)	832
max_pooling2d_1 (MaxPooling2	(None, 119, 158, 32)	0
conv2d_2 (Conv2D)	(None, 115, 154, 64)	51264

max_pooling2d_2 (MaxPooling2 (None, 57, 77, 64) 0

---

flatten_1 (Flatten) (None, 280896) 0

---

dense_1 (Dense) (None, 1000) 280897000

---

dropout_1 (Dropout) (None, 1000) 0

---

dense_2 (Dense) (None, 15) 15015

==========

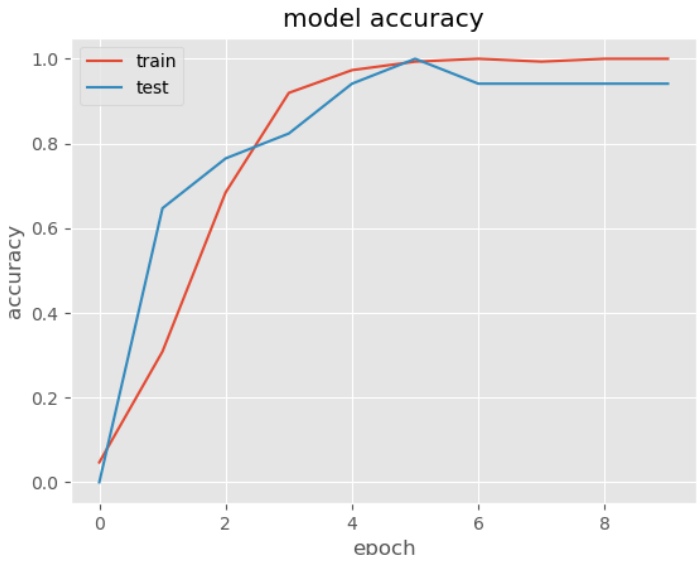
Total params: 280,964,111

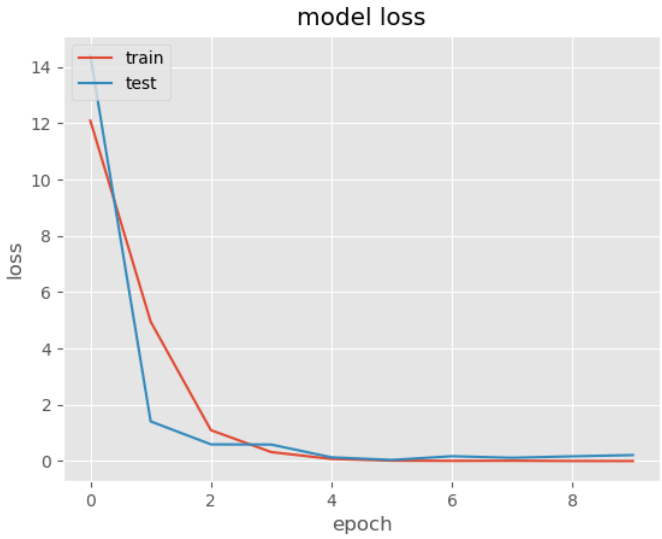
Trainable params: 280,964,111

Non-trainable params: 0

- Result

	Training Accuracy	**Validation Loss**	**Test Accuracy**
with dropout	1.00	0.2135	0.9412
without dropout	1.00	0.4285	0.9412


model accuracy
train
test
accuracy
epoch
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8


model loss
train
test
loss
epoch
14
12
10
8
6
4
2
0
0
2
4
6
8

(g) Data augmentation: Data augmentation is a way to increase the size of our dataset and reduce overfitting, especially when we use complicated models with many parameters to learn. Using any available toolbox or your own code, implement some of these techniques and augment the original Yale Face Dataset.

There are plenty of techniques to use in image data augmentation. For example:

1. Flip

You can flip images horizontally and vertically. Some frameworks do not provide function for vertical flips. But, a vertical flip is equivalent to rotating an image by 180 degrees and then performing a horizontal flip.

2. Rotation

One key thing to note about this operation is that image dimensions may not be preserved after rotation. If your image is a square, rotating it at right angles will preserve the image size. If it's a rectangle, rotating it by 180 degrees would preserve the size. Rotating the image by finer angles will also change the final image size.

3. Scale

The image can be scaled outward or inward. While scaling outward, the final image size will be larger than the original image size. Most image frameworks cut out a section from the new image, with size equal to the original image. We'll deal with scaling inward in the next section, as it reduces the image size, forcing us to make assumptions about what lies beyond the boundary.

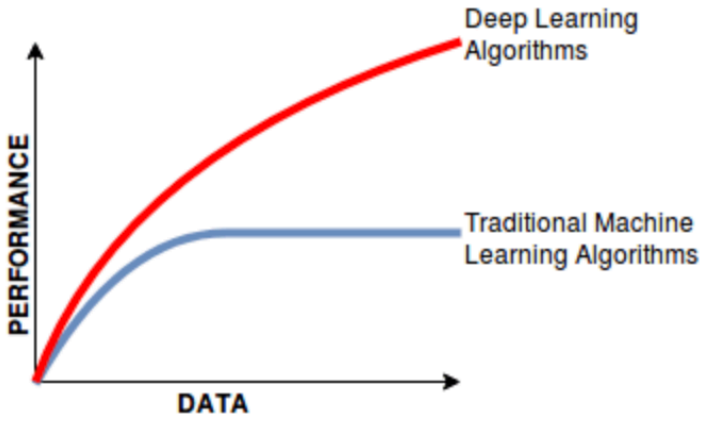
4. Crop

Unlike scaling, we just randomly sample a section from the original image. We then resize this section to the original image size. This method is popularly known as random cropping.

In order to maintain the features of faces, I only use flip and scale as the data augmentation, with simple CNN and limited data, we can find out that the result is already great.

- Discussion

In this dataset, we can see the difference between deep learning and traditional machine learning algorithm. The deep learning has better performance but it does need more computation time. The following figure shows the comparison between deep learning and conventional machine learning. Indeed, in this case we have relatively small amount of data but deep learning already has better performance. If we have more input data, I believe Convolution Neural Network can achieve way better performance.



code

```
import numpy as np
from PIL import Image
import os
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from matplotlib import pyplot as plt
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

plt.style.use('ggplot')

def read_faces(filepath='yalefaces'):
    X_origin = []
    X_flattlen = []
    y = []
    for (dirpath, dirnames, filenames) in os.walk(filepath):
        for filename in filenames:
            if filename[0]=='s':
                y.append(filename.split('.')[0])
                img = np.array(Image.open(dirpath+'/'+filename))
                X_origin.append(img)
                X_flattlen.append(img.flatten())
    # normalize
    return np.array(X_origin)/255.0, np.array(X_flattlen)/255.0 ,np.array(y)

def encode(y):
    le = preprocessing.LabelEncoder()
    le.fit(y)
    return le
```

```
def PCA_scratch(A):
    M = np.mean(A.T,axis=1)
    Center = A-M
    V = np.cov(Center.T)
    values,vectors = np.linalg.eig(V)
    P = vectors.T.dot(Center.T)
    return P


def PCA_all(X_flatten, num_eig=64):
    pca_face = PCA(num_eig)
    X_proj = pca_face.fit_transform(X_flatten)
    # print(np.cumsum(pca_face.explained_variance_ratio_))
    # plt.plot(np.cumsum(pca_face.explained_variance_ratio_))
    # plt.xlabel('number of components')
    # plt.ylabel('cumulative explained variance')
    # plt.show()
    return X_proj, pca_face


def classify(X_train_proj, X_test_proj,y_train, y_test):
    logisticRegr = LogisticRegression(
       C=1e5, solver='lbfgs', multi_class='multinomial')
    clf_svm = SVC(kernel='linear', C=5)
    clf_rf =  RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini',
                                     max_features='auto', max_leaf_nodes=None,
                                     min_impurity_decrease=0.0,
min_impurity_split=None,
                                     min_samples_leaf=1,
                                     min_weight_fraction_leaf=0.0,
n_estimators=100,
                                     oob_score=False, random_state=0,
verbose=0, warm_start=False)
```

```
    logistc_score = cross_val_score(logisticRegr, X_train_proj, y_train,
cv=5)
    svm_score = cross_val_score(clf_svm, X_train_proj, y_train, cv=5)
    rf_score = cross_val_score(clf_rf, X_train_proj, y_train, cv=5)
    print("Logistic Accuracy: %0.4f (+/- %0.2f)" %
          (logistc_score.mean(), logistc_score.std() * 2))
    print("SVM Accuracy: %0.4f (+/- %0.2f)" %
          (svm_score.mean(), svm_score.std() * 2))
    print("Random Forest Accuracy: %0.4f (+/- %0.2f)" %
          (rf_score.mean(), rf_score.std() * 2))
```

```
import numpy as np
from PIL import Image
import os
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from matplotlib import pyplot as plt
from sklearn.decomposition import PCA
from keras.utils import to_categorical

plt.style.use('ggplot')


from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
import matplotlib.pyplot as plt
from keras.utils import plot_model
from keras.layers.normalization import BatchNormalization
```

```
from Questionb import *


def basicCNNbuild(batch_size=32, epoch=30, input_shape=(64, 64, 3),
num_classes=15):
    model = Sequential()


    model.add(Conv2D(32, kernel_size=(5, 5), strides=(1, 1),
                     activation='relu',
                     input_shape=input_shape))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    model.add(Conv2D(64, (5, 5), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(1000, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])

    return model


def train_model(model, train_generator, validation_generator,
batch_size=10):
    history = model.fit_generator(
        train_generator,
        steps_per_epoch=3000//batch_size,
        epochs=10,
        validation_data=validation_generator,
        validation_steps=40 // batch_size)
    return history
```

```
def plot_loss(history):
    # summarize history for accuracy
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()
    # summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()


def show_model(model):
    plot_model(model, to_file='images/cnn_model.png')

def augment(X):
    datagen = ImageDataGenerator(
        featurewise_center=True,
        featurewise_std_normalization=True,
        horizontal_flip=True)
    datagen.fit(X)
    return datagen
```