Mahidol University International College

EGCI463: Pattern Recognition

Report: CNN and VGG16 Comparison

Submitted to

Aj. Mingmanas Sivaraksa

Submitted by

Chawarat Iampramoon 5980028

Run Ngaowijit 5980755

Pirayuj Chaisuk 6080570

Trimester 2/2019

Contents

Explain the data

We select data from the source at kaggle.com. The data we selected is Fruit 360. We choose 3 species of apples to be in our methods which are Braebrun, Crimson Snow and Red Delicious. These 3 species of apple differ in size, color and shape. Braeburn has red color with some yellow and orange on it. Crimson Snow is mainly red with a hint of yellow. Red Delicious is dark red and curvier than the other two. Each type of apple contains around 400-500 pictures for training. Moreover, training pictures are with background cut off and images of fruits are taken from different angles to make sure that the training is effective and the original image size of each image is 100x100 pixel.

Data Preparation

First, we sort those data into three separate folders. Second, we load the path file of the data into a dataframe. Third, check if the program can call the images according to the path that exists in the dataframe and plot it. Fourth, split the train and test images into 80% train images and 20% test images. This gives us 34% of Braeburn apple , 31% of Crimson Snow and 34% of Red Delicious. After splitting the images, it will store into a data frame of file file and labels where label 0 represents Braeburn apple. 1 represents Crimson Snow Apple and 2 represents Red Delicious apple. The x_train and x_test data that can be inserted into the CNN or VGG model must have a shape of (amount of image, image size, image, size, image channel). Def function is written where it passes a resized 64x64 image into a numpy array. The y_train and y_test data should be in binary form so by using the python function to_categorical (run once only), it can transform [0,1,1,2,0] of array size equal to amount of image into array of (amount of image, amount of unique variable) [[1,0,0], [0,1,0], [0,1,0], [0,0,1], [1,0,0]]. After this

transformation of the data it can now be put into CNN/VGG or similarly structured neural network models.

Methods

For the methods, we use two methods to process our data to get the prediction. First is a convolutional neural network that we made with 3 layers. The first 2 layers are convolutional layers and the last one is a dense layer for classification with kernel size of 3 in all layers.

Second is VGG16 , it is one of the simplest neural network architectures. It is also a deep convolutional network that is trained and developed by Visual Geometry Group (VGG) from Oxford. The network has 16 layers. 13 of them are convoluted layers and the last 3 are dense layers for classification. The thing that made VGG16 stand out from prior convolutional neural networks is a smaller but deeper receptive field up to 19 layers deep in one of its variants. The 13 convoluted layers have a filter size of 3x3 instead of 7x7 or 11x11 for prior networks. This method shows that the depth of the network is very important for the result and not only the filter size. If we use the Sigmoid function as a logistic classifier for this method, the accuracy will be very bad because it is for two-class but when we change the classifier

to MaxSoft, the accuracy bumps up. Therefore, logistic classifiers have played a big role in determining the accuracy of the model.

Results

From our CNN, we get very high precision and recall in all three types of apple. With Braeburn, it gets 1.00 in precision and 1.00 for recall which are very high and acceptable. Therefore, we get a high f1 score which is a harmonic mean of precision and recall to conclude the accuracy of the model in the confusion matrix. Next, Crimson snow also has similar numbers with the result from Braeburn. It got 1.00 in precision, 1.00 for recall and ended up with a 1.00 f1 score which is also very promising for the model. Lastly, Red Delicious gets 1.00 for precision and 1.00 for recall and 1.00 for f1 score which are quite high and similar to the result of other kinds of apple. In support, we use a similar number of images in each type; therefore, the numbers are close together. We can also see that the overall accuracy is tremendously good at 1.00. Macro and weighted average also get the score of 1.00 too due to the prediction being correct.

In VGG16, the scores are also quite high but they are not the same as in CNN. We get 0.99 for precision and recall in Braeburn so we eventually get 0.99 in f1 score. For Crimson Snow, we get 0.96 in precision and 0.99 for recall and 0 .97 f1 score which is a little bit lower than CNN but it is still a good result. In prediction of Red Delicious, we get 1.00 precision and 0.97 recall and eventually get 0.98 f1 score. Overall score is very good and almost perfect. However, we had to run with 5 or more epochs to get an acceptable result.

Discussion

The reason for using a visually similar dataset (complex data) to test whether CNN or VGG worked. The reasons we use CNN and VGG16 is because it holds similar convolutional/deep neural network architecture. The similar architecture should result in both models having similar accuracy, f1-score, loss-rate plot, recall and precision. However, the result from two methods show CNN is better to use than VGG16 for this model. CNN scores for the three types of apples are more than VGG16 in all fields (accuracy, recall, precision, f1-score). One of the problems occurring during multiple runs of the code is the imprecision score of VGG16. The model seems to flip up and down a lot while CNN remains constant throughout most of the runtime. VGG16 outcome is random. The VGG16 original code has a dense layer using sigmoid which results in accuracy dropping in every epoch however after changing from sigmoid to softmax the accuracy score improves. This is due to the fact that sigmoid is used for 2 neurons while softmax can be used for three classes. However even after changing the neurons the VGG16 still performs the same. The dataset doesn't

exceed 2000 images resulting in the reason that simple CNN works better than complex VGG16. The way the code is written also affects the accuracy as VGG16 also contains pretrain weight ‘imagenet’ while CNN is built from scratch.

Appendix

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import os
import time
import copy
from tqdm import tqdm_notebook as tqdm
#Braeburn 0
#Crimson 1
#Red Delicious 2
files = os.listdir('Apple TrainDataset')  #Put this PY file in same folder
with this
print(len(files))
files[0:10]


# In[2]:


base_path = 'Apple TrainDataset'
folder = os.listdir(base_path)
len(folder)


# In[3]:


total_images = 0
for n in range(1):
    for c in [0, 1, 2]:
        class_path = base_path + "/" + str(c) + "/"
        print(class_path)
        subfiles = os.listdir(class_path)
        total_images += len(subfiles)
print(total_images)
```

```
# In[4]:


import pandas as pd
import numpy as np
from os import listdir
k=0
data = pd.DataFrame(index=np.arange(0, total_images), columns=["path",
"target"])
for c in [0,1,2]:
    class_path = base_path + "/" + str(c) + "/"
    subfiles = listdir(class_path)
    for m in range(len(subfiles)):
        image_path = subfiles[m]
        data.iloc[k]["path"] = class_path + image_path
        data.iloc[k]["target"] = c
        k += 1
data


# In[5]:


brae = np.random.choice(data[data.target==0].index.values, size=50,
replace=False)
crim = np.random.choice(data[data.target==1].index.values, size=50,
replace=False)
red = np.random.choice(data[data.target==2].index.values, size=50,
replace=False)


# In[6]:


from skimage.io import imread
import matplotlib.pyplot as plt
fig, ax = plt.subplots(5,10,figsize=(20,10))
```

```
for n in range(5):
    for m in range(10):
        idx = red[m + 10*n]
        image = imread(data.loc[idx, "path"])
        ax[n,m].imshow(image)
        ax[n,m].grid(False)


# In[7]:


from sklearn.model_selection import train_test_split, StratifiedKFold
def class_count(input):
    grp = input.groupby(['target'])['path'].nunique()
    return {key: grp[key] for key in list(grp.keys())}

def class_prop(input):
    cc = class_count(input)
    return{val[0]: (round(val[1]/input.shape[0],4)*100) for val in cc.items()}

def namestr(obj, namespace):
    return [name for name in namespace if namespace[name] is obj]

x_train_path,x_test_path =
train_test_split(data,test_size=0.2,stratify=data['target'],random_state=42)
tt = [x_train_path,x_test_path]
for i in range(len(tt)):
    print(namestr(tt[i], globals()),'image
count:',class_count(tt[i]),class_prop(tt[i]))


# In[8]:


print(x_train_path)


# In[9]:
```

```
import cv2
def importimg(df):
    IMG_SIZE = 64
    all_images = []
    label = []
    for i in range(len(df)):
        image_path = df.path.values[i]
        labels = df.target.values[i]
        img = cv2.imread(image_path)
        img = cv2.resize(img,(IMG_SIZE,IMG_SIZE))
        all_images.append(img)
        label.append(labels)
    train = np.array(all_images)
    tlabel = np.array(label)
    return train,tlabel
x_train,y_train = importimg(x_train_path)
x_test,y_test = importimg(x_test_path)
print(x_train.shape,x_train.shape)


# In[10]:


from keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
print(y_train.shape,y_test.shape)


# In[11]:


from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten

model = Sequential()
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(64,64,3)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(3, activation='softmax'))
```

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


# In[27]:


N = 5
H = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=N)


# In[28]:


plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy on Apple Species CNN")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.show()


# In[29]:


a = model.predict(x_test[:])
print(a.shape)
print(y_test.shape)
print(a[:5])
print(y_test[:5])


# In[30]:


def test1D(df):
```

```
    pred = []
    for i in range(len(df)):
        if df[i][0] == 1:
            prediction = 'Braeburn'
        elif df[i][1] == 1:
            prediction = 'Crimson Snow'
        elif df[i][2] == 1:
            prediction = 'Red Delicious'
        pred.append(prediction)
    oned = np.array(pred)
    return oned


# In[31]:


mat_pre = test1D(a)
mat_test = test1D(y_test)
print(mat_pre[:10])
print(mat_test[:10])


# In[32]:


from sklearn.metrics import confusion_matrix
cnf_matrix = confusion_matrix(mat_pre,mat_test)
print(cnf_matrix)


# In[33]:


from sklearn.metrics import classification_report
print(classification_report(mat_pre, mat_test,target_names=['Braeburn',
'Crimsom Snow', 'Red Delicious']))


# In[19]:
```

```
import numpy as np

def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):
    """
    given a sklearn confusion matrix (cm), make a nice plot

    Arguments
    ---------
    cm:           confusion matrix from sklearn.metrics.confusion_matrix

    target_names: given classification classes such as [0, 1, 2]
                  the class names, for example: ['high', 'medium', 'low']

    title:        the text to display at the top of the matrix

    cmap:         the gradient of the values displayed from
matplotlib.pyplot.cm
                  see
http://matplotlib.org/examples/color/colormaps_reference.html
                  plt.get_cmap('jet') or plt.cm.Blues

    normalize:    If False, plot the raw numbers
                  If True, plot the proportions

    Usage
    -----
    plot_confusion_matrix(cm           = cm,                  # confusion
matrix created by
                                                              #
sklearn.metrics.confusion_matrix
                          normalize    = True,                # show
proportions
                          target_names = y_labels_vals,       # list of names
of the classes
                          title        = best_estimator_name) # title of graph
```

```
    Citiation
    ---------

http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_ma
trix.html

    """
    import matplotlib.pyplot as plt
    import numpy as np
    import itertools

    accuracy = np.trace(cm) / np.sum(cm).astype('float')
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
```

```
                    horizontalalignment="center",
                    color="white" if cm[i, j] > thresh else "black")


    plt.tight_layout()
    plt.ylabel('True label')
    #plt.xlabel('Predicted label\naccuracy={:0.4f};
misclass={:0.4f}'.format(accuracy, misclass))
    plt.xlabel('Predicted label')
    plt.show()


# In[20]:


plot_confusion_matrix(cnf_matrix,['Braeburn', 'Crimsom Snow', 'Red
Delicious'],plt.cm.Blues,normalize=False)


# In[21]:


from keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(64, 64, 3))

from keras import models
from keras import layers

vggmodel = models.Sequential()
vggmodel.add(conv_base)
vggmodel.add(layers.Flatten())
vggmodel.add(layers.Dense(256, activation='relu'))
vggmodel.add(layers.Dense(3, activation='softmax'))
vggmodel.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

```
# In[ ]:


N = 6
I = vggmodel.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=N)


# In[23]:


plt.figure()
plt.plot(np.arange(0, N), I.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), I.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), I.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), I.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy on Apple Species VGG")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.show()


# In[24]:


b = vggmodel.predict(x_test[:])
print(b.shape)
print(y_test.shape)
print(b[:5])
print(y_test[:5])


# In[25]:


def test1D(df):
    pred = []
    for i in range(len(df)):
        if df[i][0] == 1:
            prediction = 'Braeburn'
```

```
        elif df[i][1] == 1:
            prediction = 'Crimson Snow'
        elif df[i][2] == 1:
            prediction = 'Red Delicious'
        pred.append(prediction)
    oned = np.array(pred)
    return oned


# In[26]:


mat_preb = test1D(b)
mat_testb = test1D(y_test)
print(mat_preb[:10])
print(mat_testb[:10])


# In[ ]:


from sklearn.metrics import confusion_matrix
cnf_matrixb = confusion_matrix(mat_preb,mat_testb)
print(cnf_matrixb)


# In[ ]:


from sklearn.metrics import classification_report
print(classification_report(mat_preb, mat_testb,target_names=['Braeburn',
'Crimsom Snow', 'Red Delicious']))


# In[ ]:


plot_confusion_matrix(cnf_matrixb,['Braeburn', 'Crimsom Snow', 'Red
Delicious'],plt.cm.Blues,normalize=False)
```

```
# In[ ]:
```