**CS-5542 Project Wiki Report**

Cow face recognition system

Kent Xu, Huanwei Zhang

**The motivation behind the idea:**

At first, our project idea was to make a prediction of movie ratings and box office based on reviews.But after an ICP on image recognition and classification, we decided to change the topic of our project.

Our final project is a cow face recognition system ,This system can identify which number the cow is by using the photos taken.Because this system can actually help the farm to manage their cows in the real world. Compared with the many movie rating predictions that already exist now, I think we have a better application value to make a cow face recognition system.

**Significance:**

For real-world applications, using our system can effectively identify cows by directly taking pictures. Our data is from 66 cows on a real farm, and the system can effectively identify 66 cows. In actual use, for example, when it is necessary to vaccinate these cows, the system can be used to mark the vaccinated one or not. Different from the traditional manual marking method such as tagging on the ears, using this system will be more efficient.

**Objectives:**

Objective1: Import necessary libraries and configure some parameters. upload the picture, convert it to gray, use the classifier(cv2) to find out the range of the face area and the record them.

Objective2：Define and train the model

Objective3：Save the result and visualize the outcome

Objective4: Use the trained model to recognize the cow face

Objective5: SVM VS. CNN

**Features:**

1.The training data of our system is all collected from 66 cows on a real farm, not from a certain data set in the network, so it is very useful.

2.Our system can directly input data by taking photos, and then bring it into the system and run it directly.

3.The output result is a model，use this model for cow face recognition

**Detail of the project:**

The project is to recognize cow face. The problem we tried to solve is a real-world problem that help a farm owner to recognize the cow. The data comes from a farm with 67 cows. In the past, stud earrings are used to identify a cow. But with this project, we develop a new way to identify a cow.

The dataset:

The data are pictures of 67 cows. Here are some examples.

Cow No.1:



Cow No.9:



**train.py:**

This is the core of the whole project. In this module, we use CNN to train a model.

Here is the specific explanation of our code.

First of all, we import some library and configure some parameters.

import os  
import sys  
import cv2  
import numpy as np  
from imutils import paths  
from tensorflow.keras.utils import to\_categorical  
from keras.preprocessing.image import img\_to\_array  
import matplotlib.pyplot as plt  
from tensorflow.keras.optimizers import Adam  
from keras.callbacks import EarlyStopping  
from keras.callbacks import ModelCheckpoint  
from keras.preprocessing.image import ImageDataGenerator  
from keras import regularizers  
from keras import backend as bk  
from keras.models import Sequential  
from keras.layers.core import Dense  
from keras.layers.core import Flatten  
from keras.layers.core import Dropout  
from keras.layers.core import Activation  
from keras.layers.convolutional import Conv2D  
from keras.layers.convolutional import MaxPooling2D  
height = 250  
width = 250  
classes = 67  
epochs = 400  
batch\_size = 50  
save\_path = 'model/0/'

Then, there are some functions.

This function is to find the face area of the picture using a classifier in cv2. First, we convert our color picture to a gray one. Then we use the classifier to find out the range of the face area and the record them.But finally we didn’t use this function. This is only an attempt.

def splitimage(frame, i):  
 face\_class = cv2.CascadeClassifier(  
 r'F:\Python\Python36\Lib\site-packages\cv2\data\haarcascade\_frontalface\_default.xml')  
  
 frame\_gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
 # Use the classifier to identify which area is the face  
 faceRects = face\_class.detectMultiScale(frame\_gray, scaleFactor=1.2, minNeighbors=3, minSize=(32, 32))  
 if len(faceRects) > 0:  
 for faceRect in faceRects:  
 x, y, w, h = faceRect  
 image = frame[y - 10: y + h + 10, x - 10: x + w + 10]  
 cv2.imwrite(str(i) + '.jpg', image)

This is the crucial function to preprocess the graphs. We use the function to read the graphs and make the image size uniform.

def getimage(path):  
 lst\_img = []  
 lst\_label = []  
 i = 0  
 for label in os.listdir(path):  
 # print(label)  
 # images = paths.list\_images(path+label)  
 for fn in os.listdir(path + label):  
 img = path + label + '/' + fn  
 # print(images)  
 # for img in images:  
 # image = cv2.imread(img)  
 image = cv2.imdecode(np.fromfile(img, dtype=np.uint8), -1)  
 i = i + 1  
 if image is None:  
 os.system('rm -rf %s' % img)  
 continue  
 image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)  
 try:  
 image = cv2.resize(image, (height, width), interpolation=cv2.INTER\_AREA)  
 image = img\_to\_array(image)  
 except:  
 print(img)  
 lst\_img.append(image)  
 lst\_label.append(int(label))  
  
 np\_img = np.array(lst\_img, dtype="float") / 255.0  
 np\_label = np.array(lst\_label)  
 np\_label = to\_categorical(np\_label, num\_classes=classes)  
 return np\_img, np\_label

In this function, we define our model.

def buildModel(width, heigth, classes, depth=3):  
 model = Sequential()  
 if bk.image\_data\_format() == "channels\_first":  
 shape = (depth, width, heigth)  
 else:  
 shape = (width, heigth, depth)  
  
 # Convolutional layer  
 model.add(Conv2D(20, (3, 3), padding="same", input\_shape=shape, name='filter1'))  
 model.add(Activation("relu")) # Activate the function layer  
 model.add(MaxPooling2D(strides=(2, 2), name="max1"))  
 model.add(Dropout(0.5))  
  
 model.add(Conv2D(30, (3, 3), padding="same", name='filter2'))  
 model.add(Activation("relu"))  
 model.add(MaxPooling2D(strides=(2, 2), name="max2"))  
  
 model.add(Conv2D(50, (5, 5), padding="same", name='filter3'))  
 model.add(Activation("relu"))  
 model.add(MaxPooling2D(strides=(2, 2), name="max3")) # Pooling layer  
 model.add(Dropout(0.5)) # Dropout layer  
  
 model.add(Flatten()) # Flatten layer  
 model.add(Dense(250)) # Fully connected layer  
 model.add(Dropout(0.5))  
 model.add(Activation("relu"))  
  
 model.add(Dense(classes))  
 model.add(Activation("softmax")) # Classification layer  
  
 return model

In this function, we train our model. First, we configure some parameters. Then, we randomly alter the graphs to avoid over-fitting. After training the model, we save the result, save the model and visualize the outcome.

def train\_model(train\_path, test\_path):  
 model = buildModel(width=width, heigth=height, classes=classes)  
 # Compiling the model requires three parameters, the optimizer, the loss function, and the metrics  
 model.compile(optimizer=Adam(lr=1e-3, decay=0.0), loss="categorical\_crossentropy", metrics=['accuracy'])  
 train\_img, train\_label = getimage(train\_path)  
 test\_img, test\_label, lst\_path = getimage2(test\_path)  
 # print(len(test\_img))  
  
 datagen = ImageDataGenerator(  
 rotation\_range=20, # The angle at which the image rotates randomly when the data is increased (range 0～180)  
 width\_shift\_range=0.2, # The amplitude of the horizontal offset of the picture when the data is promoted  
 height\_shift\_range=0.2, # Same as above, but here is vertical  
 horizontal\_flip=True) # Whether to perform random horizontal flip  
 # If it exists, it can be superimposed  
 if os.path.exists('checkpoint.chk'):  
 model.load\_weights("checkpoint.chk")  
 try:  
 fit = model.fit\_generator(  
 datagen.flow(train\_img, train\_label, batch\_size=batch\_size),  
 validation\_data=(test\_img, test\_label),  
 steps\_per\_epoch=len(train\_img) // epochs,  
 epochs=epochs,  
 verbose=1,  
 )  
 except:  
 fit = model.fit\_generator(  
 datagen.flow(train\_img, train\_label, batch\_size=batch\_size),  
 validation\_data=(test\_img, test\_label),  
 epochs=epochs,  
 verbose=1,  
 )  
 score = model.evaluate(test\_img, test\_label, batch\_size=32)  
 y\_pred = model.predict(test\_img, batch\_size=32, )  
  
 print("Training completed,the model score is %s" % score)  
 print("Training prediction results: %s" % y\_pred)  
 # print(fit.history)  
 lst\_result = []  
 for t in y\_pred:  
 t = list(t)  
 lst\_result.append(t.index(max(t)))  
  
 print(lst\_result)  
  
 file = open('result/cnnresult.txt', 'w')  
 for i in range(len(lst\_result)):  
 file.write(('image%s' % lst\_path[i]) + ('recognized as cow No.%d' % lst\_result[i]) + '\n')  
 file.write('Accuracy：%s' % str(score[1]))  
 file.close()  
  
 file = open('result/check.txt', 'w')  
 cwj = 1  
 for i in range(len(lst\_result)):  
 file.write(('%s:' % lst\_result[i]) + ('%d' % cwj) + '\n')  
 file.close()  
 model.save(save\_path)

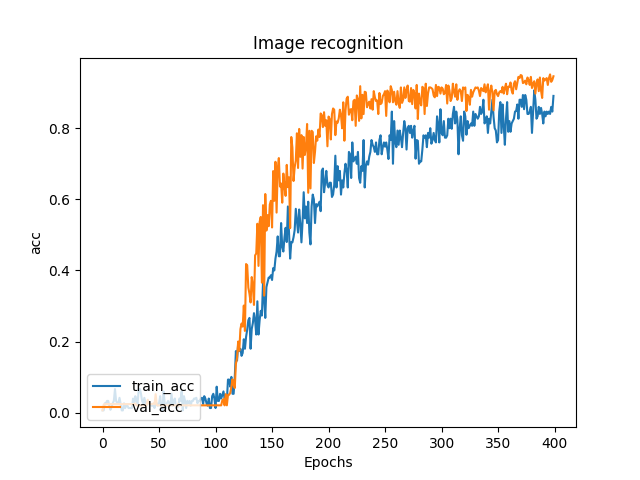
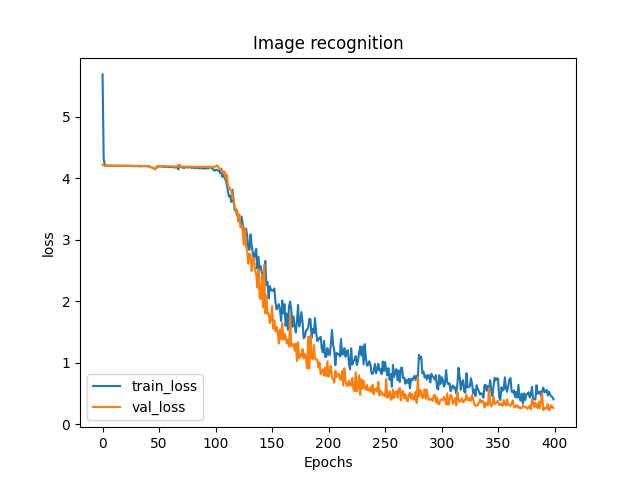
Visualization of the history:

plt.figure()  
try:  
 #plt.plot(fit.history["loss"], label="train\_loss")  
 plt.plot(fit.history["accuracy"], label="train\_acc")  
 #plt.plot(fit.history["val\_loss"], label="val\_loss")  
 plt.plot(fit.history["val\_accuracy"], label="val\_acc")  
except:  
 # plt.plot(fit.history["loss"], label="train\_loss")  
 plt.plot(fit.history["acc"], label="train\_acc")  
 #plt.plot(fit.history["val\_loss"], label="val\_loss")  
 plt.plot(fit.history["val\_acc"], label="val\_acc")  
  
plt.title("Image recognition")  
plt.xlabel("Epochs")  
plt.ylabel("acc")  
plt.legend(loc="lower left")  
plt.savefig("accreco")  
plt.show()  
  
plt.figure()  
try:  
 plt.plot(fit.history["loss"], label="train\_loss")  
 #plt.plot(fit.history["accuracy"], label="train\_acc")  
 plt.plot(fit.history["val\_loss"], label="val\_loss")  
 #plt.plot(fit.history["val\_accuracy"], label="val\_acc")  
except:  
 plt.plot(fit.history["loss"], label="train\_loss")  
 #plt.plot(fit.history["acc"], label="train\_acc")  
 plt.plot(fit.history["val\_loss"], label="val\_loss")  
 #plt.plot(fit.history["val\_acc"], label="val\_acc")  
  
plt.title("Image recognition")  
plt.xlabel("Epochs")  
plt.ylabel("loss")  
plt.legend(loc="lower left")  
plt.savefig("lossreco")  
plt.show()

the main function:

if \_\_name\_\_ == '\_\_main\_\_':  
 train = "./data/train/"  
 test = "./data/test/"  
 train\_model(train, test)

Visualization of the history:



**Recogniton.py**

In this module, we use the trained model to recognize the cow face. We call the laptop built-in camera to take a picture.Then we print the result of prediction. In this module,we tried to detect the face area. But it seems unnecessary to do so. In this module we can solve the real-world problem, the farmer can recognize his cow anytime he wants.

import cv2  
from keras.models import load\_model  
import numpy as np  
from keras.preprocessing.image import img\_to\_array  
#Call the laptop built-in camera, the parameter is 0, if there are other cameras, you can adjust the parameter to 1,2  
cap = cv2.VideoCapture(0)  
  
face\_detector = cv2.CascadeClassifier(r'F:\Python\Python36\Lib\site-packages\cv2\data\haarcascade\_frontalface\_default.xml')  
#Mark an id for the face to be entered  
#sampleNum is used to count the number of samples  
count = 0  
height=250  
width=250  
def image\_array(image):  
 image = cv2.resize(image, (height, width))  
 image = image.astype("float") / 255.0  
 image = img\_to\_array(image)  
 image = np.expand\_dims(image, axis=0)  
 return image  
   
model = load\_model('./model/classify.model')  
while True:  
 #read the picture from the camera  
 success,img = cap.read()  
 image = image\_array(img)  
 result = model.predict(image)[0]  
 print(result)  
 print("recognized as cow No.%d"%list(result).index(max(list(result))))  
  
 if success is True:  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 else:  
 break  
  
 faces = face\_detector.detectMultiScale(gray, 1.3, 5)  
 for (x, y, w, h) in faces:  
 cv2.rectangle(img, (x, y), (x+w, y+w), (255, 0, 0))  
 count += 1  
 cv2.imshow('image',img)  
 #Keep the picture continuity. The waitkey method can bind the buttons to ensure that the picture is retracted, and exit the camera by pressing the q key  
 k = cv2.waitKey(10)  
 if k & 0xFF == ord('q'):  
 break  
  
cap.release()  
cv2.destroyAllWindows()

The outcome of the module:



**trainsvm.py:**

We tried another way to recognize the cow face. We use SVM to do a prediction. But we fount it very inaccurate to do classification. Anyway, we can make a contrast between CNN and SVM.

#Training model  
def train\_model(train\_path, test\_path):  
 model = svm.SVC(gamma='auto')  
 train\_img, train\_label = getimage(train\_path)  
 test\_img, test\_label, lst\_path = getimage2(test\_path)  
   
 pca = PCA().fit(train\_img)  
   
 train\_img = pca.transform(train\_img)  
 test\_img = pca.transform(test\_img)  
   
 model.fit(train\_img, train\_label)

y\_pred = model.predict(test\_img)  
 print(y\_pred)  
 score = model.score(test\_img, test\_label)#[1,2,3]

**qt.py:**

A simple Graphical User Interface of the project to show the outcome.

