CVPR_Project

December 29, 2020

1 Gourango Modak , 18-37102-1

2 Shuvra Smaran Das , 18-37161-1

3 Md.Salimullah , 18-37161-1

4 Problem Statement

In our daily life, it is easy for us to determine whether or not a natural picture is being rotated. But it may be hard for computers to pridict human level output. So, we are going to pridict rotation angle of a given image [0, 90, 180, 270].

4.1 Dataset

The database contains 67 Indoor categories, and a total of 15620 images. The number of images varies across categories, but there are at least 100 images per category. All images are in jpg format.

From this dataset we randomly pick an image and rotate it into these [0, 90, 180, 270] any random rotation and build our custom dataset.

Dataset Link

[3]:
```
# import the necessary packages
from imutils import paths
import numpy as np
import progressbar
import imutils
import random
import cv2
import os
from keras.applications import VGG16
from keras.applications import imagenet_utils
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import load_img
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
import pickle
import h5py
from keras.models import Sequential
from keras.layers import MaxPool2D, Conv2D, Flatten
```

[4]:

```
#configure setting

# initialize dataset and rotationed dataset location
dataset_path = "H:\CVPR project\Dataset\indoorCVPR_09\Images"
rotated_path = "H:\CVPR project\Rotationed_Images"

# HDF5 dataset store location
hdf5_path = "H:\CVPR project\Hdf5\correcting_rotation_dataset.hdf5"
model_path="H:\CVPR project\Model\orientation_correction_classifier.cpickle"

# initialize vgg16 weight path location
weight_path = r"H:\CVPR␣
 ↪project\Weight\vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5"

# initialize batch size and buffer size
batch_size = 32
buffer_size = 1000
```

5 Dataset Building

[9]:

```
# here we take 10k images from total images and shuffle the images
imagePaths = list(paths.list_images(dataset_path))[:10000]
random.shuffle(imagePaths)

# initialize a dictionary to keep track of the number of each angle
angles = {}
widgets = ["Building Dataset: ", progressbar.Percentage(), " ",progressbar.
 ↪Bar(), " ", progressbar.ETA()]
pbar = progressbar.ProgressBar(maxval=len(imagePaths),widgets=widgets).start()

# loop over the image paths
for (i, imagePath) in enumerate(imagePaths):

    # determine the rotation angle, and load the image
    angle = np.random.choice([0, 90, 180, 270])
    image = cv2.imread(imagePath)

    # if the image is None
    if image is None:
        continue
    # rotate the image based on the selected angle, then construct
    # the path to the base output directory
    image = imutils.rotate_bound(image, angle)
    base = os.path.sep.join([rotated_path, str(angle)])

    # if the base path does not exist already, create it
    if not os.path.exists(base):
        os.makedirs(base)

    # extract the image file extension, then construct the full path
    # to the output file
    ext = imagePath[imagePath.rfind("."):]
    outputPath = [base, "image_{}{}".format(str(angles.get(angle, 0)).zfill(5),␣
 ↪ext)]
    outputPath = os.path.sep.join(outputPath)

    # save the image
    cv2.imwrite(outputPath, image)
    # update the count for the angle
    c = angles.get(angle, 0)
    angles[angle] = c + 1
    pbar.update(i)


# finish the progress bar
pbar.finish()

# loop over the angles and display counts for each of them
for angle in sorted(angles.keys()):
    print("[INFO] angle={}: {:,}".format(angle, angles[angle]))
```

```
Building Dataset: 100% |########################################| Time: 0:06:00
```

```
[INFO] angle=0: 2,499
[INFO] angle=90: 2,480
[INFO] angle=180: 2,506
[INFO] angle=270: 2,493
```

6 Helper Classes

[58]:

```
# we used this class to store all images into hdf5 file format.
class HDF5DatasetWriter:

    def __init__(self, dims, outputPath, dataKey="images", bufSize=1000):
```

```
    # check to see if the output path exists, and if so, raise
    # an exception
        if os.path.exists(outputPath):
            raise ValueError("The supplied ‘outputPath‘ already "
             "exists and cannot be overwritten. Manually delete "
            "the file before continuing.", outputPath)

        # open the HDF5 database for writing and create two datasets:
        # one to store the images/features and another to store the
        # class labels
        self.db = h5py.File(outputPath, "w")
        self.data = self.db.create_dataset(dataKey, dims,
        dtype="float")
        self.labels = self.db.create_dataset("labels", (dims[0],),
        dtype="int")

        # store the buffer size, then initialize the buffer itself
        # along with the index into the datasets
        self.bufSize = bufSize
        self.buffer = {"data": [], "labels": []}
        self.idx = 0

    def add(self, rows, labels):
        # add the rows and labels to the buffer
        self.buffer["data"].extend(rows)
        self.buffer["labels"].extend(labels)
        # check to see if the buffer needs to be flushed to disk
        if len(self.buffer["data"]) >= self.bufSize:
            self.flush()

    def flush(self):
        # write the buffers to disk then reset the buffer
        i = self.idx + len(self.buffer["data"])
        self.data[self.idx:i] = self.buffer["data"]
        self.labels[self.idx:i] = self.buffer["labels"]
        self.idx = i
        self.buffer = {"data": [], "labels": []}

    def storeClassLabels(self, classLabels):
        # create a dataset to store the actual class label names,
        # then store the class labels
        dt = h5py.special_dtype(vlen=str)
        labelSet = self.db.create_dataset("label_names",(len(classLabels),),␣
 ↪dtype=dt)
        labelSet[:] = classLabels

    def close(self):
        # check to see if there are any other entries in the buffer
        # that need to be flushed to disk
        if len(self.buffer["data"]) > 0:
            self.flush()

        # close the dataset
        self.db.close()
```

7 Model Architecture

7.1 VGG 16

pic/vgg16.png

[5]:

```
class VGGNet_16:

    @staticmethod
    def build(weights_path=None):

        model = Sequential()

        model.
 ↪add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,3),padding="same",␣
 ↪activation="relu"))
        model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",␣
 ↪activation="relu"))
        model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

        model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same",␣
 ↪activation="relu"))
        model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same",␣
 ↪activation="relu"))
        model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

        model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",␣
 ↪activation="relu"))
        model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",␣
 ↪activation="relu"))
        model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",␣
 ↪activation="relu"))
        model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
        model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",␣
↪activation="relu"))
        model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",␣
↪activation="relu"))
        model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",␣
↪activation="relu"))
        model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

        model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",␣
↪activation="relu"))
        model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",␣
↪activation="relu"))
        model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",␣
↪activation="relu"))
        model.add(MaxPool2D(pool_size=(2,2),strides=(2,2),name='vgg16'))

        model.add(Flatten(name='flatten'))

        if weights_path:
            model.load_weights(weights_path)

        return model
```

[6]:

```
model=VGGNet_16.build(weight_path)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 224, 224, 64)      1792
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 224, 224, 64)      36928
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 112, 112, 64)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 128)     73856
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 112, 112, 128)     147584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 56, 56, 256)       295168
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 56, 56, 256)       590080
_________________________________________________________________
```

```
conv2d_6 (Conv2D)            (None, 56, 56, 256)       590080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 256)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 512)       1180160
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 28, 28, 512)       2359808
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 28, 28, 512)       2359808
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 512)       0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
vgg16 (MaxPooling2D)         (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

8 Extracting Features

[59]:

```python
# initlize the batch size
bs = batch_size

# array slicing during training time
print("[INFO] loading images...")
imagePaths = list(paths.list_images(rotated_path))
random.shuffle(imagePaths)

labels = [p.split(os.path.sep)[-2] for p in imagePaths]
le = LabelEncoder()
labels = le.fit_transform(labels)

# load the VGG16 network
# print("[INFO] loading network...")
# model = VGG16(weights="imagenet", include_top=False)

# initialize the HDF5 dataset writer, then store the class label
# names in the dataset
dataset = HDF5DatasetWriter((len(imagePaths), 512 * 7 * 7), hdf5_path,␣
 ↪dataKey="features", bufSize=buffer_size)

dataset.storeClassLabels(le.classes_)

# initialize the progress bar
widgets = ["Extracting Features: ", progressbar.Percentage(), " ", progressbar.
 ↪Bar(), " ", progressbar.ETA()]
pbar = progressbar.ProgressBar(maxval=len(imagePaths),widgets=widgets).start()

# loop over the images in patches
for i in np.arange(0, len(imagePaths), bs):

    batchPaths = imagePaths[i:i + bs]
    batchLabels = labels[i:i + bs]
    batchImages = []

    # loop over the images and labels in the current batch
    for (j, imagePath) in enumerate(batchPaths):

        # we resized all images to 224x224 pixels
        image = load_img(imagePath, target_size=(224, 224))
        image = img_to_array(image)

        image = np.expand_dims(image, axis=0)
        image = imagenet_utils.preprocess_input(image)

        # add the image to the batch
        batchImages.append(image)

    batchImages = np.vstack(batchImages)
    features = model.predict(batchImages, batch_size=bs)

    # reshape the features so that each image is represented by
    # a flattened feature vector of the ‘MaxPooling2D‘ outputs
    features = features.reshape((features.shape[0], 512 * 7 * 7))

    # add the features and labels to our HDF5 dataset
    dataset.add(features, batchLabels)
    pbar.update(i)

# close the dataset
dataset.close()
pbar.finish()
```

```
Extracting Features:   0% |                                    | ETA:  --:--:--
```

```
[INFO] loading images...
```

```
Extracting Features: 100% |######################################| Time: 0:35:53
```

9 Training an Orientation Correction Classifier

Here, we use Logistic Regression model for classification and tune the hyper parameter C using GridSearch.

[13]:

```
# initialize hdf5 dataset path
db=hdf5_path

# open the HDF5 database for reading
db = h5py.File(db, "r")
i = int(db["labels"].shape[0] * 0.75) # take 75% for training and rest for testing

print("[INFO] tuning hyperparameters...")
params = {"C": [0.01, 0.1, 1.0, 10.0, 100.0, 1000.0, 10000.0]}

model = GridSearchCV(LogisticRegression(max_iter=300), params, cv=3, n_jobs=1)
model.fit(db["features"][:i], db["labels"][:i])

print("[INFO] best hyperparameters: {}".format(model.best_params_))

# evaluate the model
print("[INFO] evaluating...")
preds = model.predict(db["features"][i:])
print(classification_report(db["labels"][i:], preds,target_names=db["label_names"]))


# serialize the model to disk
print("[INFO] saving model...")
f = open(model_path, "wb")
f.write(pickle.dumps(model.best_estimator_))
f.close()

# close the database
db.close()
```

```
[INFO] tuning hyperparameters...
[INFO] best hyperparameters: {'C': 1.0}
[INFO] evaluating...
              precision    recall  f1-score   support

           0       0.95      0.92      0.94       615
         180       0.92      0.93      0.92       627
         270       0.89      0.89      0.89       622
          90       0.90      0.91      0.91       631

    accuracy                           0.91      2495
   macro avg       0.91      0.91      0.91      2495
weighted avg       0.91      0.91      0.91      2495

[INFO] saving model...
```

10 Pridicting some random images

[10]:

```
# initialize all paths
db = hdf5_path
model = model_path
dataset = rotated_path

# load the label names from the HDF5 dataset
db = h5py.File(db)
labelNames = [int(angle) for angle in db["label_names"][:]]
db.close()

# grab the paths to the testing images and randomly sample them
print("[INFO] sampling images...")

imagePaths = list(paths.list_images(dataset))
imagePaths = np.random.choice(imagePaths, size=(10,), replace=False)

# load the VGG16 network
print("[INFO] loading network...")
vgg = VGGNet_16.build(weight_path)
# vgg = VGG16(weights="imagenet", include_top=False)


# load the orientation model
print("[INFO] loading model...")
model = pickle.loads(open(model, "rb").read())

# loop over the image paths
for imagePath in imagePaths:

    orig = cv2.imread(imagePath)

    image = load_img(imagePath, target_size=(224, 224))
    image = img_to_array(image)

    image = np.expand_dims(image, axis=0)
    image = imagenet_utils.preprocess_input(image)

    # pass the image through the network to obtain the feature vector
    features = vgg.predict(image)
    features = features.reshape((features.shape[0], 512 * 7 * 7))
    # now that we have the CNN features, pass these through our
    # classifier to obtain the orientation predictions
    angle = model.predict(features)
    angle = labelNames[angle[0]]
    # now that we have the predicted orientation of the image we can
    # correct for it
    rotated = imutils.rotate_bound(orig, 360 - angle)
    orig = cv2.resize(orig, (300,300))
    rotated = cv2.resize(rotated, (300,300))
    # display the original and corrected images
    cv2.imshow("Original", orig)
    cv2.imshow("Corrected", rotated)
    cv2.waitKey(0)

cv2.destroyAllWindows()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:7:
H5pyDeprecationWarning: The default file mode will change to 'r' (read-only) in
h5py 3.0. To suppress this warning, pass the mode you need to h5py.File(), or
set the global default h5.get_config().default_file_mode, or set the environment
variable H5PY_DEFAULT_READONLY=1. Available modes are: 'r', 'r+', 'w', 'w-'/'x',
'a'. See the docs for details.
  import sys
```

```
[INFO] sampling images...
[INFO] loading network...
[INFO] loading model...
```

[ ]: