DermAssist

Franco Charette

October 14, 2021

[ ]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
import os
import cv2
import math
from pathlib import Path
from sklearn.model_selection import train_test_split, cross_validate,␣
 ↪cross_val_score
from lightgbm import LGBMClassifier
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.dummy import DummyClassifier
from sklearn import datasets
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import img_to_array, load_img
from tensorflow.keras import utils
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.resnet50 import ResNet50
```

We obtained the dataset from kaggle at this link: https://www.kaggle.com/shubhamgoel27/dermnet

We will try to classify the pictures of 23 skin diseases and aim for the highest accuracy, recall, precision and f1Score. To be clear, we will intake pictures and ouput a class prediction. The goal is to predict the right type of skin disease out of 23 types.

This task could be useful for patients in developing countries who don't have access to a qualifed doctor due to their geographic location, economic condition, language barriers, lack of trust, etc. Such patient could get a rough opinion from the image classifier, then read potential course of action, solution or precautions about the disease on the internet.

[ ]:
```
# function to use data folder names to build dataframe
def extractInfo(location):
    arr = []
    for folders in location.glob("*"):
        for file in folders.glob("*jpg"):
            d = {}
            d["disease"] = folders.name.replace("Photos", "")
            # print(file.name)
            d["file"] = file.name.replace(".jpg", "")
            d["image_path"] = str(location) + "/" + folders.name + "/" + file.
 ↪name
            arr.append(d)
    return pd.DataFrame(arr)

trainLocation = Path('../input/dermnet/train')
testLocation = Path('../input/dermnet/test')

df_train = extractInfo(trainLocation)
df_test = extractInfo(testLocation)

print(df_test[:2])
```

[ ]:
```
# Randomly select a small percentage of available dataset due to small laptop␣
 ↪CPU.
# Given 23 classifications and abundance of data, we select a relatively
# large test set.This will provide confidence in the test score result

# NOTE: only the fraction of the dataset can be run due to memory constraints in␣
 ↪kaggle
# Use this split for now
df_train = df_train.sample(frac=0.075)
df_test = df_test.sample(frac=0.29)

# create X_train, X_test
y_train = df_train["disease"]
X_train = df_train.drop(columns=["disease"])

# create X_valid, X_valid
# create a validation set to use especially for confusion matrix
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train,␣
 ↪test_size=0.5, random_state=123)

# create y_train, y_test
y_test = df_test["disease"]
X_test = df_test.drop(columns=["disease"])

print("success")
```

[ ]:
```
print("The number of images contained in the training set is", X_train.shape[0])
print("The number of images contained in the validation set is", X_valid.
 ↪shape[0])
```

```
print("The number of images contained in the test set is", X_test.shape[0])
```

[ ]:
```
print("The distribution of diseases in the traning set is:\n", y_train.
 ↪value_counts())
```

[ ]:
```
print("The distribution of diseases in the validation set is:\n", y_valid.
 ↪value_counts())
```

[ ]:
```
print("The distribution of diseases in the test set is:\n", y_test.
 ↪value_counts())
```

[ ]:
```
# defines some metrics pertaining to images size, made by hand
def dimensionAnalysis(array):
        h = []
        w = []
        answer = []
        for img in array:
            im = cv2.imread(img)
            h.append(im.shape[0])
            w.append(im.shape[1])
        answer.insert(0, np.mean(h))
        answer.insert(1, np.mean(w))
        answer.insert(2, np.min(h))
        answer.insert(3, np.min(w))
        answer.insert(4, np.max(h))
        answer.insert(5, np.max(w))
        return answer


dims = dimensionAnalysis(X_train['image_path'])
print("The average image height is", dims[0])
print("The minimum image height is", dims[2])
print("The maximum image height is", dims[4])
print("The average image width is", dims[1])
print("The minimum image width is", dims[3])
print("The maximum image width is", dims[5])
```

We can see that the images are generally similar in size and that they are large enough for using more than the 256 x 256 pixel during training. Clearly our computation power is the greatest limit here.

A pipeline is not necessary in our use case, but we have much processing to do…

[ ]:
```
# read X_train images from the image directory.
X_train = np.array([img_to_array(
                    load_img(img, target_size=(256,256))
                    ) for img in X_train['image_path'].values.tolist()])
```

```
X_train.shape
```

[ ]:
```
# read X_valid images from the image directory.
X_valid = np.array([img_to_array(
                    load_img(img, target_size=(256,256))
                    ) for img in X_valid['image_path'].values.tolist()])

X_valid.shape
```

[ ]:
```
# read X_test images from the image directory.
X_test = np.array([img_to_array(
                    load_img(img, target_size=(256,256))
                    ) for img in X_test['image_path'].values.tolist()])

X_test.shape
```

[ ]:
```
# scale image data
X_train = X_train.astype('float32')/255.0
X_valid = X_valid.astype('float32')/255.0
X_test = X_test.astype('float32')/255.0
```

[ ]:
```
# Create feature extractor featuring transfer learning with  imagenet
base_inception = InceptionV3(weights='imagenet', include_top=False,␣
 ↪input_shape=(256, 256, 3))
feature_extractor = Model(inputs=base_inception.input,␣
 ↪outputs=GlobalAveragePooling2D()(base_inception.output))

# transform training and test datasets
X_train = feature_extractor.predict(X_train)
X_valid = feature_extractor.predict(X_valid)
X_test = feature_extractor.predict(X_test)
```

[ ]:
```
# cross validate initial results
def cross_validate_std(*args, **kwargs):
    """Like cross_validate, except also gives the standard deviation of the␣
 ↪score"""
    res = pd.DataFrame(cross_validate(*args, **kwargs))
    res_mean = res.mean()

    res_mean["std_test_score"] = res["test_score"].std()
    if "train_score" in res:
        res_mean["std_train_score"] = res["train_score"].std()
    return res_mean
```

[ ]:
```
# running dummy classifyer as benchmark
dc = DummyClassifier(strategy='prior')
dc.fit(X_train, y_train)
```

```
score = dc.score(X_train, y_train)
print("Dummy classifier scores", score)
```

```
[ ]: # Creating a logistic regression object
lr = LogisticRegression(multi_class="multinomial", solver="lbfgs", max_iter=1000)
```

```
[ ]: # Scoring with cross validation standard deviation
lrScore = cross_validate_std(lr, X_train, y_train, cv=10,␣
 ↪return_train_score=True)
print("Logistic regression scores are:\n", lrScore)
```

We obtain very poor accuracy score, likely because we are training the models with only ~450 picture split in 23 classes. This is the best we can do for free on kaggle

Given the very low score, the standard deviation is very high. This higlights the fact that our training set has only 5-10 pictures per disease while we demand cross validation folds of 10. That is very few picures to train and score on!

```
[ ]: classList = ["Nail Fungus and other Nail Disease",
"Tinea Ringworm Candidiasis and other Fungal Infections",
"Eczema",
"Psoriasis pictures Lichen Planus and related diseases",
"Actinic Keratosis Basal Cell Carcinoma and other Malignant Lesions",
"Warts Molluscum and other Viral Infections",
"Seborrheic Keratoses and other Benign Tumors",
"Acne and Rosacea",
"Light Diseases and Disorders of Pigmentation",
"Bullous Disease",
"Melanoma Skin Cancer Nevi and Moles","Exanthems and Drug Eruptions",
"Vasculitis",
"Scabies Lyme Disease and other Infestations and Bites",
"Atopic Dermatitis",
"Vascular Tumors",
"Lupus and other Connective Tissue diseases",
"Cellulitis Impetigo and other Bacterial Infections",
"Systemic Disease",
"Hair Loss  Alopecia and other Hair Diseases",
"Herpes HPV and other STDs",
"Poison Ivy  and other Contact Dermatitis",
"Urticaria Hives"]

# Calculate logistic regression prediction for confusion matrix
lr2 = LogisticRegression(multi_class="multinomial", solver="lbfgs",␣
 ↪max_iter=1000)
lr2.fit(X_train, y_train)
y_pred = lr2.predict(X_valid)
lr2_matrix = confusion_matrix(y_valid, y_pred,labels=classList)
```

```
print(lr2_matrix)
```

The multiclass confusion matrix does a really good job showing where the model does better and worse. As expected, the model does a poor job in the lower right, on classes featuring 5-10 pictures for training.

```
[ ]: # inspired by https://www.youtube.com/watch?v=HBi-P5j0Kec
     # inspired by https://stats.stackexchange.com/questions/51296/
     #␣
      ↪how-do-you-calculate-precision-and-recall-for-multiclass-classification-using-co
     def averageRecall(m):
         pred = np.diag(m)
         total = np.sum(m, axis = 0)
         arr = []
         i=0
         for tp in pred:
             prec = tp / total[i]
             if (math.isnan(prec)):
                 prec = 0
             arr = np.append(arr,prec)
             i+=1
         return np.mean(arr)

     def averagePrecision(m):
         pred = np.diag(m)
         total = np.sum(m, axis = 1)
         arr = []
         i=0
         for tp in pred:
             recall = tp / total[i]
             if (math.isnan(recall)):
                 recall = 0
             arr = np.append(arr,recall)
             i+=1
         return np.mean(arr)

     def f1Score(precision, recall):
         return ((2*precision*recall) / (precision + recall))

     lrRecall = averageRecall(lr2_matrix)
     print("The lr2 recall score is", lrRecall)

     lrPrecision = averagePrecision(lr2_matrix)
     print("The lr2 precision score is", lrPrecision)

     lrF1Score = f1Score(lrPrecision,lrRecall)
     print("The lr2 F1 score is", lrF1Score)
```

The recall, precision and f1 score shows the reality of training the model with too few pictures due to our memory constraint

```
[ ]: # Evaluate model on the test set
     final_lr_score = lr2.score(X_test, y_test)
     print("The final logistic regression score is",final_lr_score )
```

As mentioned before, our laptop CPU restricts us to train both models with a small number of pictures(~500) divided by 23 classes. The accuracy is predictably low.

Given that scoring is less demanding than than training, we were able to test the final model with a larger number of pictures(~1000) to provide more accuracy. However, it is still a relatively low amount given there are 23 classes.

To conclude, we ackowledge the poor accuracy and the relatively poor level of confidence due to our weak CPU. Next, we would like to learn how to train a model by exposing it to successive batches of images or by using some online GPU!!!