data_augmentation+classfication

March 12, 2022

0.1 Data augmentation Brief Intro

1. Selection of data augmentation methods:

First, we want to know if the images cannot include the whole tumor, stromal, or TIL, would the performance become better if more details are detected or decreasing. Second, examine whether the different directions of the same medical images can be seen as new data. Last, we inverted the color of half of the whole dataset. We choose four different data augmentation methods for the experiment, which are Random Crop (Fig 1), Random Rotation (Fig 2), Random Flip (Fig 3), and Random Invert (Fig 4). Figures

2. Design of experiment:

The experiment contains two main parts: choosing efficient data augmentation methods and examining whether the original dataset is overfitting or not through changing the dataset volume.

3. Data manipulation:

The dataset is from the NuCLS, which contains over 220,000 labeled nuclei from breast cancer images.

[ ]:
```
import os, random, keras, keras_preprocessing, cv2
import tensorflow as tf
import matplotlib.pylab as plt
from skimage import io
import numpy as np
from PIL import Image
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from keras.preprocessing.image import ImageDataGenerator
from google.colab import files, drive
drive.mount('/content/drive')
!unzip /content/drive/MyDrive/project/release/classification.zip -d /content
```

[ ]:
```
def prepro_data_aug(filename):
    # create image data augmentation generator
    dataset_aug = [], name_aug = []
    image_directory = './classification/'
    for image_name in filename[::2]:
      if (image_name.split('.')[4] == 'png'):
```

```
        image = io.imread(image_directory + image_name)
        image = Image.fromarray(image, 'RGB')
        dataset_aug.append(np.array(image))
        name_aug.append
    x = np.array(dataset_aug)
    return x
```

[ ]:
```
files = os.listdir('./classification/')
files = [file for file in files if os.path.splitext(file)[1] == '.png']
img = prepro_data_aug(files)
```

four kinds of data augmentation

[ ]:
```
!mkdir /content/drive/MyDrive/project/release/classificationHalf# make a folder
↪for new augmentation images
```

[ ]:
```
# augmentation method
data_augmentation = keras.Sequential(
  [layers.experimental.preprocessing.Resizing(90,90),
  # layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
  # layers.experimental.preprocessing.RandomRotation(0.1),
  # layers.experimental.preprocessing.RandomZoom(0),
  # layers.experimental.preprocessing.RandomCrop(60,60)
])
plt.imshow(data_augmentation(img[0]))
```

[ ]:
```
def random_invert_img(x, p=0.5):
  if  tf.random.uniform([]) < p:
    x = (255-x)
  else:
    x
  return x
def random_invert(factor=0.5):
  return layers.Lambda(lambda x: random_invert_img(x, factor))
random_invert = random_invert()
plt.imshow(random_invert(img[0]))
```

[ ]:
```
index = 0
for file in files[::2]:
  name = ".".join([file.split('.')[0], file.split('.')[1],file.split('.
↪')[2],file.split('.')[3],'classificationHalf',file.split('.')[4]])
  aug = data_augmentation(img[index])
  # aug = random_invert(aug)
  cv2.imwrite('/content/drive/MyDrive/project/release/classificationHalf/
↪'+name,aug.numpy())
  index = index +1
```

```
[ ]: # zip the folder with images for model
     !zip -r "/content/drive/MyDrive/project/release/classificationHalf.zip" '/
      ↪content/drive/MyDrive/project/release/classificationHalf'
```

0.2 Classification Brief Intro

We use the dataset which we have already used the data augmentation and then to examine the performance about classifying tumor, stromal, and TIL cell nuclei.

data preprocessing

```
[ ]: !unzip /content/drive/MyDrive/project/release/classificationHalf.zip -d /content
```

```
[ ]: size = tf.constant([224, 224], tf.int32)
     normalizer = tf.keras.applications.resnet50.preprocess_input #normalization for␣
      ↪input images
     optimizer = 'adam' #gradient optimizer
     loss = tf.keras.losses.BinaryCrossentropy() #classification loss
     tf_random_seed = 0
     tf.random.set_seed(tf_random_seed)
     glorot_initializer=tf.keras.initializers.GlorotNormal(seed=tf_random_seed)
     mapping = ['stromal', 'TIL', 'tumor']#label mapping - these classes will be␣
      ↪mapped to 0-stromal, 1-TIL, etc.
```

```
[ ]: path = '/content/content/drive/MyDrive/project/release/classificationHalf/'
     path_test = './classification_test/'
     batch = 64 #batch size (# images / batch)
     epochs = 10 #number of training epochs
     metrics = [tf.keras.metrics.AUC(name='macro_auc', multi_label=True,␣
      ↪num_labels=len(mapping))] #performance metric
     validation_freq = 1 #measure validation performance after every 10 epochs
     mixed_precision = True
     train_cache = False
     validation_cache = False
     py_random_seed = 0
```

```
[ ]: def parse_filename(file):
         label = file.split('.')[0]   #One of 'tumor', 'stromal', 'TIL', or␣
      ↪'unknown'.
         patient = file.split('.')[3][0:12]   #Unique patient identifier within the␣
      ↪dataset.
         lab = file.split('.')[3].split('-')[1]  #Unique identifier of lab that␣
      ↪produced patient sample.
         return label, patient, lab
     def load(filename):
         img = tf.io.read_file(filename)
```

```
    img = tf.image.decode_png(img, channels=3)
    return img
def dataset(path, load, cells, mapping, labs, size, normalizer, batch, cache):
    AUTOTUNE = tf.data.experimental.AUTOTUNE
    ds_cells = [cell for cell in cells if cell['lab'] in labs]
    ds_files = [path + cell['file'] for cell in cells if cell['lab'] in labs]
    ds_labels = [mapping.index(cell['label']) for cell in cells if cell['lab']␣
 ↪in labs]
    ds = tf.data.Dataset.from_tensor_slices((ds_files, ds_labels))
    ds = ds.shuffle(len(ds_labels), reshuffle_each_iteration=True) #shuffle list
    ds = ds.map(lambda x, y: (load(x),  tf.one_hot(y, len(mapping))))
    ds = ds.map(lambda x, y: (tf.image.resize(x, size), y))
    ds = ds.batch(batch)
    ds = ds.map(lambda x, y: (normalizer(x), y))
    ds = ds.prefetch(AUTOTUNE)
    if cache:
        ds = ds.cache()
    return ds
random.seed(py_random_seed)
files = os.listdir('/content/content/drive/MyDrive/project/release/
 ↪classificationHalf')
files = [file for file in files if os.path.splitext(file)[1] == '.png']
fields = [parse_filename(file) for file in files
          if os.path.splitext(file)[1] == '.png']
cells = [{'file': file, 'label': field[0], 'patient': field[1], 'lab': field[2]}
          for (file, field) in zip(files, fields)]
cells = [cell for cell in cells if cell['label'] != 'unknown']
labs = list(set([cell['lab'] for cell in cells]))
labs.sort()
validation_labs = [labs[random.randint(0, len(labs)-1)]]
train_labs = list(set(labs).difference(set(validation_labs)))
train_ds =  dataset(path, load, cells, mapping, train_labs, size,
                    normalizer,  batch, train_cache)
validation_ds = dataset(path, load, cells, mapping, validation_labs, size,
                        normalizer, batch, validation_cache)
```

[ ]:
```
files = os.listdir('./classification_test/')
files = [file for file in files if os.path.splitext(file)[1] == '.png']
fields = [parse_filename(file) for file in files
          if os.path.splitext(file)[1] == '.png']
cells = [{'file': file, 'label': field[0], 'patient': field[1], 'lab': field[2]}
          for (file, field) in zip(files, fields)]
cells = [cell for cell in cells if cell['label'] != 'unknown']
labs = list(set([cell['lab'] for cell in cells]))
labs.sort()
validation_labs = [labs[random.randint(0, len(labs)-1)]]
train_labs = list(set(labs).difference(set(validation_labs)))
```

```
test_ds = dataset(path_test, load, cells, mapping, validation_labs, size,
                        normalizer, batch, validation_cache)
```

learning structure

[ ]:
```
#train
if mixed_precision:
    tf.keras.mixed_precision.set_global_policy('mixed_float16')
with tf.device('/device:GPU:0'): #define model
    base = tf.keras.applications.ResNet50(
        include_top=False, weights='imagenet', input_tensor=None,
        input_shape=(size[0], size[1], 3), pooling='avg')
    inputs = tf.keras.Input(shape=(size[0], size[1], 3))#define base network␣
 ↪input and output
    x = base(inputs)
    x = tf.keras.layers.Dense(len(mapping),␣
 ↪kernel_initializer=glorot_initializer)(x)#add dense network
    outputs = tf.keras.layers.Activation('softmax', dtype='float32')(x)
    model = tf.keras.Model(inputs, outputs)
model.compile(optimizer=optimizer, loss=loss, metrics=metrics) #compile the␣
 ↪model with the adam optimizer and cross-entropy loss
model.fit(train_ds, epochs=epochs, validation_data=validation_ds,
          validation_freq=validation_freq, verbose=1)  #fit and validate every␣
 ↪10 epochs
```

final result(just demonstrate one of results)

[ ]:
```
result = model.evaluate(test_ds)
```

```
3/3 [==============================] - 1s 225ms/step - loss: 1.6799 - macro_auc:
0.8087
```

1 Result:

(All the results is the outcome of the model tested by the test dataset)

From performance comparison with single data augmentation at original size dataset (Fig5) Fig6), we can see there is one model, random invert, outperforming the original models, which means all medical images in the same color can result in the model learning some unrelated patterns. When a random invert is applied, the unrelated patterns due to similar color in all images can be decreased. Also, the inverted images can break the consistency in the dataset. Next, the model of the random crop is worse than anticipated, showing that the images after cropping are too small. If they are not totally located to the tumor, the interesting patterns of cancer cannot be learned by the models effectively. The other two models have very similar output to the original model, which means those medical images are not directional, so those two models can be seen as almost the same as the original one. Based on the first experiment step, we have to test two combinations of the data augmentation methods, random invert plus random rotation and random invert plus random

flip, to know which is the best combination of those four methods. In the end, the results show that the random invert plus random flip model has the most remarkable accuracy at the original size of a dataset, which may result from that rotating the cells in the images don't have any actual benefit since the image are not directional. Subsequently, the comparison of half dataset volume (Fig7) Fig8) has the same ranking as the original dataset for the data augmentation method. There is no noticeable change in performance for the single process applied. However, when combining two data augmentation methods used in the models, both of the accuracy of two models, random rotation plus random invert and random flip plus random invert, has dramatically improved. Last, we compare the three different dataset volumes in random invert and random flip models. The outcome (Fig9) shows that the dataset volume is saturation, which means increasing more data to the original dataset volume cannot improve the performance.

2 Discussion:

For discussion, we want to give more ideas about the reasons for various accuracy in different models due to the data augmentation training dataset. First, the geometry modification for medical images doesn't present excellent performance alone, leading to an increasing loss in the model. Take the rotation method, for example; this method doesn't enhance the boundary, the exciting region, or give new information, but lets some parts of the image out of the edge, filled by nearby pixels. Compared to the flip method, which also has the problems that don't provide valuable labels, this method wouldn't lose data. Therefore, those methods have a very similar outcome with the original dataset, but the loss of the flip process is lower than the loss of the rotation method. Another geometry-changing way, random crop, has more apparent problems when examining its loss and accuracy. First, when we zoom in on the images, there are plenty of discarded areas, significantly increasing the loss. However, if the region we zoom in on is the region of interest of the medical images, we consider the accuracy is still acceptable. But, because we zoom in randomly in the whole picture, there is likely no tumor in the region. So, the dataset contains plenty of useless data, and every image has a higher loss due to the abandoned area. Last, the only method outperforms the original dataset significantly in different dataset volumes. First, it enhances the boundary and the region of interest through different colors. Second, it increases diversity in the dataset, preventing the model from becoming overfitting to the dataset. Based on the above discussion for examining the appropriate size of dataset volume, the result of accuracy versus dataset volume (Fig 9) for three models shows that I should decrease the dataset volume more in the experiment. The highest accuracy model, random invert plus random flip, reaches almost 90% accuracy in this project which is trained by half of the original dataset volume but not original or double dataset volume. To sum up, the geometry-changing data augmentation methods don't work as well as the color-changing process, and the training dataset volume needs to be decreased.

3 Future work:

In this project, we only choose four data-augmented methods. And we are looking for more different methods to be implemented. For example, noise and grayscale filters. Moreover, we could attempt to add more layer into network or another deep learning method like GAN.