kaggle Search Competitions Datasets Kernels Discussion Learn

Jonne

CNN Segmentation + connected components

last run 2 months ago · IPython Notebook HTML · 11,310 views
using data from RSNA Pneumonia Detection Challenge · Public

164
voters

Notebook Code Data (1) Output Comments (39) Log Versions (10) Forks (682) Fork Notebook

Notebook

Approach

- Firstly a convolutional neural network is used to segment the image, using the bounding boxes directly as a mask.
- Secondly connected components is used to separate multiple areas of predicted pneumonia.
- Finally a bounding box is simply drawn around every connected component.

Network

- The network consists of a number of residual blocks with convolutions and downsampling blocks with max pooling.
- At the end of the network a single upsampling layer converts the output to the same shape as the input.

As the input to the network is 256 by 256 (instead of the original 1024 by 1024) and the network downsamples a number of times without any meaningful upsampling (the final upsampling is just to match in 256 by 256 mask) the final prediction is very crude. If the network downsamples 4 times the final bounding boxes can only change with at least 16 pixels.

In [1]:

```
import os
import csv
import random
import pydicom
import numpy as np
import pandas as pd
from skimage import io
from skimage import measure
from skimage.transform import resize

import tensorflow as tf
from tensorflow import keras

from matplotlib import pyplot as plt
import matplotlib.patches as patches
```

```
/opt/conda/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarn
ing: Conversion of the second argument of issubdtype from `float` to `
np.floating` is deprecated. In future, it will be treated as `np.float
64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

Load pneumonia locations

Table contains [filename : pneumonia location] pairs per row.

- If a filename contains multiple pneumonia, the table contains multiple rows with the same filename but different pneumonia locations.
- If a filename contains no pneumonia it contains a single row with an empty pneumonia location.

The code below loads the table and transforms it into a dictionary.

- The dictionary uses the filename as key and a list of pneumonia locations in that filename as value.
- If a filename is not present in the dictionary it means that it contains no pneumonia.

In [2]:

```
# empty dictionary
pneumonia_locations = {}
# load table
with open(os.path.join('../input/stage_1_train_labels.csv'), mode='r')
 as infile:
    # open reader
    reader = csv.reader(infile)
    # skip header
    next(reader, None)
    # loop through rows
    for rows in reader:
        # retrieve information
        filename = rows[0]
        location = rows[1:5]
        pneumonia = rows[5]
        # if row contains pneumonia add label to dictionary
        # which contains a list of pneumonia locations per filename
        if pneumonia == '1':
            # convert string to float to int
            location = [int(float(i)) for i in location]
            # save pneumonia location in dictionary
            if filename in pneumonia_locations:
                pneumonia_locations[filename].append(location)
            else:
                pneumonia_locations[filename] = [location]
```

Load filenames

In [3]:

```
# load and shuffle filenames
folder = '../input/stage_1_train_images'
filenames = os.listdir(folder)
random.shuffle(filenames)
# split into train and validation filenames
n_valid_samples = 2560
```

```
train_filenames = filenames[n_valid_samples:]
valid_filenames = filenames[:n_valid_samples]
print('n train samples', len(train_filenames))
print('n valid samples', len(valid_filenames))
n_train_samples = len(filenames) - n_valid_samples
```

```
n train samples 23124
n valid samples 2560
```

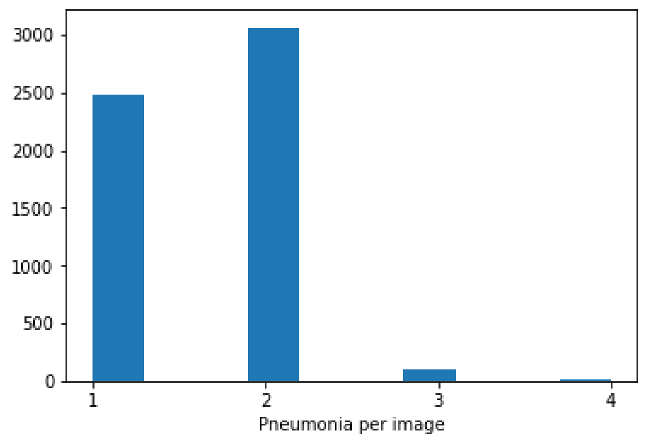
Exploration

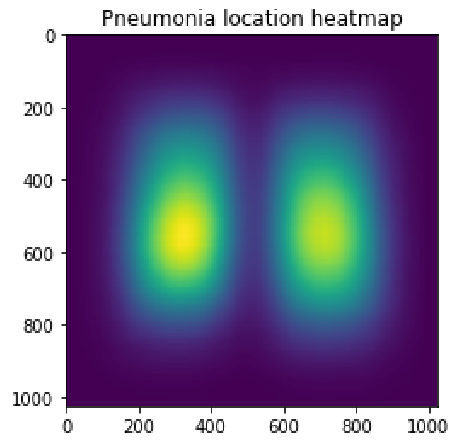
In [4]:

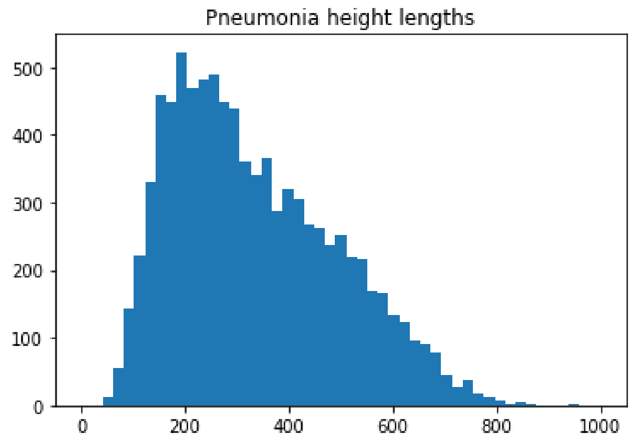
```
print('Total train images:',len(filenames))
print('Images with pneumonia:', len(pneumonia_locations))

ns = [len(value) for value in pneumonia_locations.values()]
plt.figure()
plt.hist(ns)
plt.xlabel('Pneumonia per image')
plt.xticks(range(1, np.max(ns)+1))
plt.show()

heatmap = np.zeros((1024, 1024))
ws = []
hs = []
for values in pneumonia_locations.values():
    for value in values:
        x, y, w, h = value
        heatmap[y:y+h, x:x+w] += 1
        ws.append(w)
        hs.append(h)
plt.figure()
plt.title('Pneumonia location heatmap')
plt.imshow(heatmap)
plt.figure()
plt.title('Pneumonia height lengths')
plt.hist(hs, bins=np.linspace(0,1000,50))
plt.show()
plt.figure()
plt.title('Pneumonia width lengths')
plt.hist(ws, bins=np.linspace(0,1000,50))
plt.show()
print('Minimum pneumonia height:', np.min(hs))
print('Minimum pneumonia width: ', np.min(ws))
```

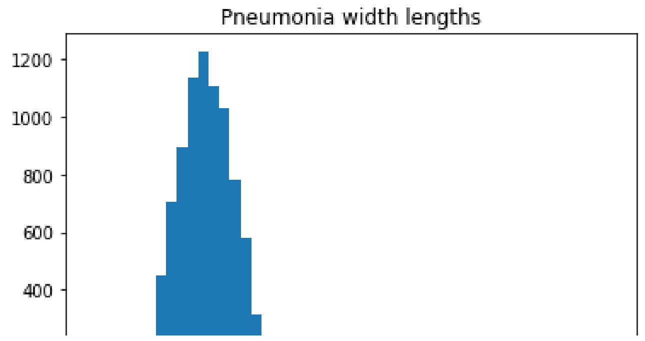
```
Total train images: 25684
Images with pneumonia: 5659
```

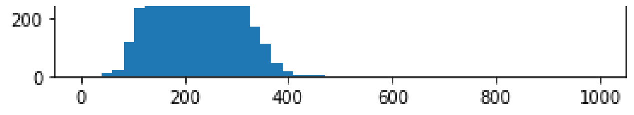
```
images with pneumonia: 6009
```











```
Minimum pneumonia height: 45
Minimum pneumonia width:  40
```

Data generator

The dataset is too large to fit into memory, so we need to create a generator that loads data on the fly.

- The generator takes in some filenames, batch_size and other parameters.
- The generator outputs a random batch of numpy images and numpy masks.

In [5]:

```
class generator(keras.utils.Sequence):

    def __init__(self, folder, filenames, pneumonia_locations=None, ba
tch_size=32, image_size=256, shuffle=True, augment=False, predict=Fals
e):
        self.folder = folder
        self.filenames = filenames
        self.pneumonia_locations = pneumonia_locations
        self.batch_size = batch_size
        self.image_size = image_size
        self.shuffle = shuffle
        self.augment = augment
        self.predict = predict
        self.on_epoch_end()

    def __load__(self, filename):
        # load dicom file as numpy array
        img = pydicom.dcmread(os.path.join(self.folder, filename)).pix
el_array
        # create empty mask
        msk = np.zeros(img.shape)
        # get filename without extension
        filename = filename.split('.')[0]
        # if image contains pneumonia
        if filename in self.pneumonia_locations:
            # loop through pneumonia
            for location in self.pneumonia_locations[filename]:
                # add 1's at the location of the pneumonia
                x, y, w, h = location
                msk[y:y+h, x:x+w] = 1
        # resize both image and mask
        img = resize(img, (self.image_size, self.image_size), mode='re
flect')
        msk = resize(msk, (self.image_size, self.image_size), mode='re
flect') > 0.5
        # if augment then horizontal flip half the time
        if self.augment and random.random() > 0.5:
            img = np.fliplr(img)
            msk = np.fliplr(msk)
        # add trailing channel dimension
        img = np.expand_dims(img, -1)
        msk = np.expand_dims(msk, -1)
        return img, msk

    def __loadpredict__(self, filename):
        # load dicom file as numpy array
        img = pydicom.dcmread(os.path.join(self.folder, filename)).pix
el_array
        # resize image
        img = resize(img, (self.image_size, self.image_size), mode='re
flect')
        # add trailing channel dimension
        img = np.expand_dims(img, -1)
        return img

    def __getitem__(self, index):
        # select batch
        filenames = self.filenames[index*self.batch_size:(index+1)*sel
f.batch_size]
        # predict mode: return images and filenames
        if self.predict:
            # load files
            imgs = [self.__loadpredict__(filename) for filename in fil
enames]
            # create numpy batch
            imgs = np.array(imgs)
            return imgs, filenames
        # train mode: return images and masks
        else:
            # load files
            items = [self.__load__(filename) for filename in filenames
]
            # unzip images and masks
            imgs, msks = zip(*items)
            # create numpy batch
            imgs = np.array(imgs)
            msks = np.array(msks)
            return imgs, msks

    def on_epoch_end(self):
        if self.shuffle:
            random.shuffle(self.filenames)
    def __len__(self):
        if self.predict:
            # return everything
            return int(np.ceil(len(self.filenames) / self.batch_size))
        else:
            # return full batches only
            return int(len(self.filenames) / self.batch_size)
```

Network

In [6]:

```
def create_downsample(channels, inputs):
    x = keras.layers.BatchNormalization(momentum=0.9)(inputs)
    x = keras.layers.LeakyReLU(0)(x)
    x = keras.layers.Conv2D(channels, 1, padding='same', use_bias=Fals
e)(x)
    x = keras.layers.MaxPool2D(2)(x)
    return x

def create_resblock(channels, inputs):
    x = keras.layers.BatchNormalization(momentum=0.9)(inputs)
    x = keras.layers.LeakyReLU(0)(x)
    x = keras.layers.Conv2D(channels, 3, padding='same', use_bias=Fals
e)(x)
    x = keras.layers.BatchNormalization(momentum=0.9)(x)
    x = keras.layers.LeakyReLU(0)(x)
    x = keras.layers.Conv2D(channels, 3, padding='same', use_bias=Fals
e)(x)
    return keras.layers.add([x, inputs])

def create_network(input_size, channels, n_blocks=2, depth=4):
    # input
    inputs = keras.Input(shape=(input_size, input_size, 1))
    x = keras.layers.Conv2D(channels, 3, padding='same', use_bias=Fals
e)(inputs)
    # residual blocks
    for d in range(depth):
        channels = channels * 2
        x = create_downsample(channels, x)
        for b in range(n_blocks):
            x = create_resblock(channels, x)
    # output
    x = keras.layers.BatchNormalization(momentum=0.9)(x)
    x = keras.layers.LeakyReLU(0)(x)
    x = keras.layers.Conv2D(1, 1, activation='sigmoid')(x)
    outputs = keras.layers.UpSampling2D(2**depth)(x)
    model = keras.Model(inputs=inputs, outputs=outputs)
    return model
```

Train network

In [7]:

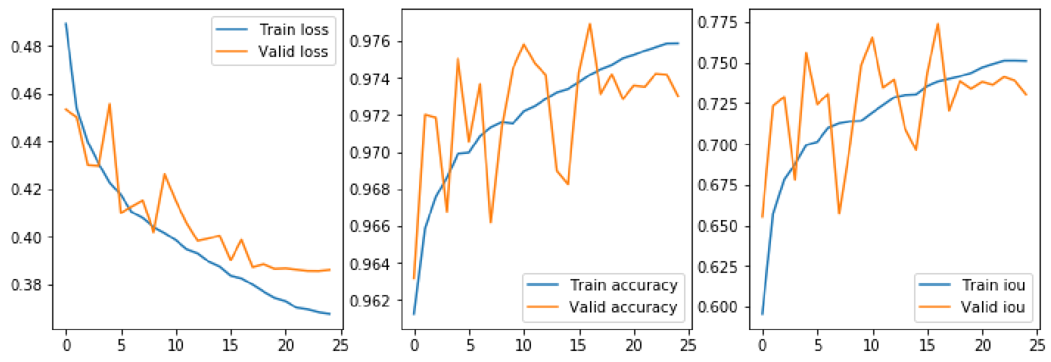
```
# define iou or jaccard loss function
def iou_loss(y_true, y_pred):
    y_true = tf.reshape(y_true, [-1])
    y_pred = tf.reshape(y_pred, [-1])
    intersection = tf.reduce_sum(y_true * y_pred)
    score = (intersection + 1.) / (tf.reduce_sum(y_true) + tf.reduce_s
um(y_pred) - intersection + 1.)
    return 1 - score

# combine bce loss and iou loss
def iou_bce_loss(y_true, y_pred):
    return 0.5 * keras.losses.binary_crossentropy(y_true, y_pred) + 0.
5 * iou_loss(y_true, y_pred)

# mean iou as a metric
def mean_iou(y_true, y_pred):
    y_pred = tf.round(y_pred)
    intersect = tf.reduce_sum(y_true * y_pred, axis=[1, 2, 3])
    union = tf.reduce_sum(y_true, axis=[1, 2, 3]) + tf.reduce_sum(y_pr
ed, axis=[1, 2, 3])
    smooth = tf.ones(tf.shape(intersect))
    return tf.reduce_mean((intersect + smooth) / (union - intersect +
smooth))

# create network and compiler
model = create_network(input_size=256, channels=32, n_blocks=2, depth=
4)
model.compile(optimizer='adam',
              loss=iou_bce_loss,
              metrics=['accuracy', mean_iou])

# cosine learning rate annealing
def cosine_annealing(x):
    lr = 0.001
    epochs = 25
    return lr*(np.cos(np.pi*x/epochs)+1.)/2
learning_rate = tf.keras.callbacks.LearningRateScheduler(cosine_anneal
ing)

# create train and validation generators
folder = '../input/stage_1_train_images'
train_gen = generator(folder, train_filenames, pneumonia_locations, ba
tch_size=32, image_size=256, shuffle=True, augment=True, predict=False
)
valid_gen = generator(folder, valid_filenames, pneumonia_locations, ba
tch_size=32, image_size=256, shuffle=False, predict=False)

history = model.fit_generator(train_gen, validation_data=valid_gen, ca
llbacks=[learning_rate], epochs=25, workers=4, use_multiprocessing=Tru
e)
```

```
Epoch 1/25
722/722 [==============================] - 783s 1s/step - loss: 0.4892
- acc: 0.9613 - mean_iou: 0.5957 - val_loss: 0.4533 - val_acc: 0.9632
- val_mean_iou: 0.6553
Epoch 2/25
722/722 [==============================] - 767s 1s/step - loss: 0.4539
- acc: 0.9659 - mean_iou: 0.6573 - val_loss: 0.4502 - val_acc: 0.9720
- val_mean_iou: 0.7234
Epoch 3/25
722/722 [==============================] - 763s 1s/step - loss: 0.4397
- acc: 0.9676 - mean_iou: 0.6782 - val_loss: 0.4302 - val_acc: 0.9719
- val_mean_iou: 0.7290
Epoch 4/25
722/722 [==============================] - 757s 1s/step - loss: 0.4307
- acc: 0.9686 - mean_iou: 0.6878 - val_loss: 0.4298 - val_acc: 0.9668
- val_mean_iou: 0.6781
Epoch 5/25
722/722 [==============================] - 740s 1s/step - loss: 0.4227
- acc: 0.9699 - mean_iou: 0.6993 - val_loss: 0.4556 - val_acc: 0.9750
- val_mean_iou: 0.7560
Epoch 6/25
722/722 [==============================] - 740s 1s/step - loss: 0.4176
- acc: 0.9700 - mean_iou: 0.7013 - val_loss: 0.4100 - val_acc: 0.9706
- val_mean_iou: 0.7243
Epoch 7/25
722/722 [==============================] - 744s 1s/step - loss: 0.4105
- acc: 0.9709 - mean_iou: 0.7099 - val_loss: 0.4127 - val_acc: 0.9737
- val_mean_iou: 0.7307
Epoch 8/25
722/722 [==============================] - 750s 1s/step - loss: 0.4079
- acc: 0.9713 - mean_iou: 0.7127 - val_loss: 0.4152 - val_acc: 0.9662
- val_mean_iou: 0.6572
Epoch 9/25
722/722 [==============================] - 742s 1s/step - loss: 0.4041
- acc: 0.9716 - mean_iou: 0.7139 - val_loss: 0.4020 - val_acc: 0.9714
- val_mean_iou: 0.6991
Epoch 10/25
722/722 [==============================] - 741s 1s/step - loss: 0.4016
- acc: 0.9715 - mean_iou: 0.7142 - val_loss: 0.4263 - val_acc: 0.9745
- val_mean_iou: 0.7483
Epoch 11/25
722/722 [==============================] - 745s 1s/step - loss: 0.3987
```

```
- acc: 0.9722 - mean_iou: 0.7193 - val_loss: 0.4152 - val_acc: 0.9758
- val_mean_iou: 0.7653
Epoch 12/25
515/722 [===================>.........] - ETA: 3:20 - loss: 0.3954 -
acc: 0.9727 - mean_iou: 0.7254
```

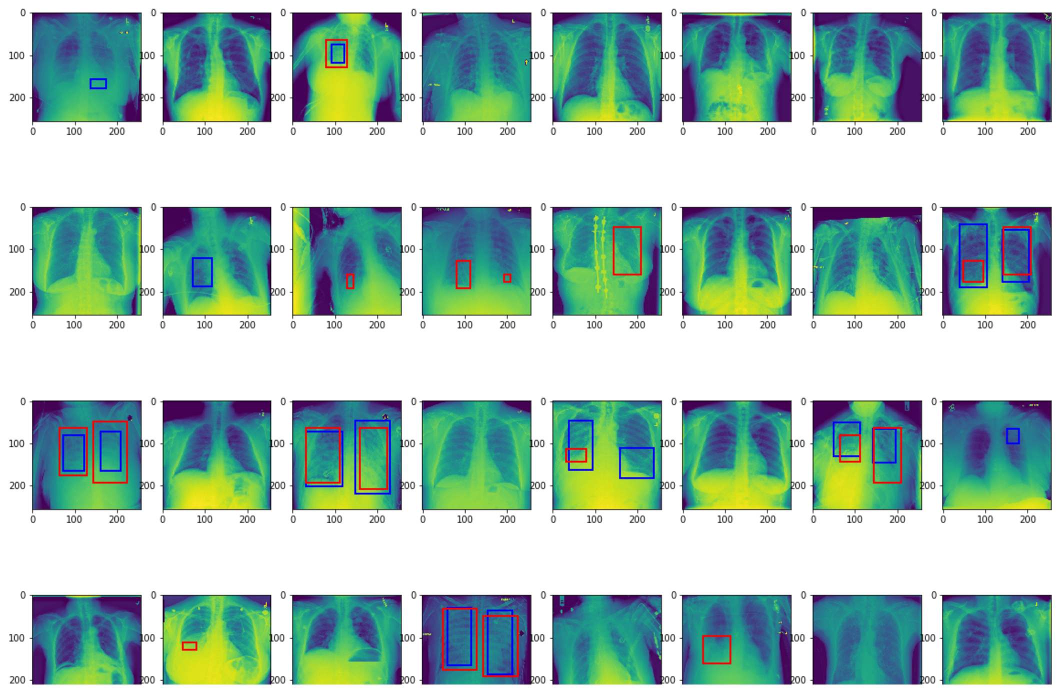
In [8]:

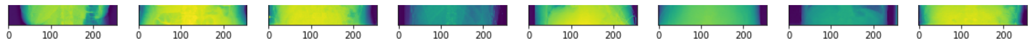
```
plt.figure(figsize=(12,4))
plt.subplot(131)
plt.plot(history.epoch, history.history["loss"], label="Train loss")
plt.plot(history.epoch, history.history["val_loss"], label="Valid los
s")
plt.legend()
plt.subplot(132)
plt.plot(history.epoch, history.history["acc"], label="Train accuracy"
)
plt.plot(history.epoch, history.history["val_acc"], label="Valid accur
acy")
plt.legend()
plt.subplot(133)
plt.plot(history.epoch, history.history["mean_iou"], label="Train iou"
)
plt.plot(history.epoch, history.history["val_mean_iou"], label="Valid
 iou")
plt.legend()
plt.show()
```



In [9]:

```
for imgs, msks in valid_gen:
    # predict batch of images
    preds = model.predict(imgs)
    # create figure
    f, axarr = plt.subplots(4, 8, figsize=(20,15))
    axarr = axarr.ravel()
    axidx = 0
    # loop through batch
    for img, msk, pred in zip(imgs, msks, preds):
        # plot image
        axarr[axidx].imshow(img[:, :, 0])
        # threshold true mask
        comp = msk[:, :, 0] > 0.5
        # apply connected components
        comp = measure.label(comp)
        # apply bounding boxes
        predictionString = ''
        for region in measure.regionprops(comp):
            # retrieve x, y, height and width
            y, x, y2, x2 = region.bbox
            height = y2 - y
            width = x2 - x
            axarr[axidx].add_patch(patches.Rectangle((x,y),width,heigh
t,linewidth=2,edgecolor='b',facecolor='none'))
        # threshold predicted mask
        comp = pred[:, :, 0] > 0.5
        # apply connected components
        comp = measure.label(comp)
        # apply bounding boxes
        predictionString = ''
        for region in measure.regionprops(comp):
            # retrieve x, y, height and width
            y, x, y2, x2 = region.bbox
            height = y2 - y
            width = x2 - x
            axarr[axidx].add_patch(patches.Rectangle((x,y),width,heigh
t,linewidth=2,edgecolor='r',facecolor='none'))
        axidx += 1
    plt.show()
    # only plot one batch
    break
```





Predict test images

In [10]:
```
# load and shuffle filenames
folder = '../input/stage_1_test_images'
test_filenames = os.listdir(folder)
print('n test samples:', len(test_filenames))

# create test generator with predict flag set to True
test_gen = generator(folder, test_filenames, None, batch_size=25, imag
e_size=256, shuffle=False, predict=True)

# create submission dictionary
submission_dict = {}
# loop through testset
for imgs, filenames in test_gen:
    # predict batch of images
    preds = model.predict(imgs)
    # loop through batch
    for pred, filename in zip(preds, filenames):
        # resize predicted mask
```

Did you find this Kernel useful?
Show your appreciation with an upvote

164

Comments (39)   All Comments   Sort by Hotness

Click here to enter a comment...

Chien Hsi Cheng • Posted on Latest Version • a month ago • Options • Reply   2

Thanks for sharing this kernel ! This really helps me a lot. One thing I want to ask is about the horizontal flip for data augmentation. As far as I know, chest radiograph is not symmetric(Ex. heart apex usually points to the left side, rarely the heart apex points to the right side, except the condition such as situ invertus). I'm not sure if this will hurt the performance but might be taken into consideration. Thank you!

Andy Yu • Posted on Latest Version • 2 months ago • Options • Reply   6

Hi , the kernel is great! I have a question, why do you ues leakyReLu as the activation function? Does it perform better than ReLu? Thanks

anokas • Posted on Version 4 • 2 months ago • Options • Reply 21

Nice kernel :)

One thing I want to point out is that `collections.defaultdict` can be useful to simplify your data loading code:

```
pneumonia_locations = {}
...
if filename in pneumonia_locations:
    pneumonia_locations[filename].append(location)
else:
    pneumonia_locations[filename] = [location]
```

You can clean up the conditional by doing:

```
from collections import defaultdict
pneumonia_locations = defaultdict(list)
...
pneumonia_locations[filename].append(location)
```

Which initialises all elements in the dictionary as a list by default (hence you can just append without needing to initialise it)

Ann Antonova • Posted on Version 7 • 2 months ago • Options • Reply 3

Great! But one little question. Why do you get coordinates in this order: `y, x, y2, x2 = region.bbox` ? Not in `x, y, x2, y2` ?

Jonne Kernel Author • Posted on Version 7 • 2 months ago • Options • Reply 4

http://scikit-image.org/docs/dev/api/skimage.measure.html#skimage.measure.regionprops

The bbox property of regionprops contains the bounding box in the format: (min_row, min_col, max_row, max_col).

Sebastian Nor... • Posted on Latest Version • 2 months ago • Options • Reply 1

Is it for the same reason that on the generator class the number "1" is assigned at the location of the pneumonia like this?: msk[y:y+h, x:x+w] = 1 instead of msk[x:x+w, y:y+h] = 1

DimitriF • Posted on Version 4 • 2 months ago • Options • Reply 5

Hi, loved the kernel, this little piece of code allows to visualize the predictions on one batch of the validation set, thought you can use it

```
for imgs, mask in valid_gen:
    # predict batch of images
    preds = model.predict(imgs)
    plt.figure(figsize=(20, 20))
    plt.axis('off')
    nbr=32
    for i in range(nbr):
        plt.subplot(nbr/4, 12, i*3+1)
        plt.imshow(imgs[i,:,:,:].reshape(256,256))
        plt.subplot(nbr/4, 12, i*3+2)
        plt.imshow(mask[i,:,:,:].reshape(256,256))
        plt.subplot(nbr/4, 12, i*3+3)
        plt.imshow(preds[i,:,:,:].reshape(256,256))
    break
```

raddar • Posted on Version 3 • 2 months ago • Options • Reply 5

nodule is such a bad term to use in the competition context, as nodules usually refer to small cancerous areas up to 3cm. it is opacity we are predicting here, which is whole different thing - areas affected by bacterias or virus.

Jonne Kernel Author • Posted on Version 4 • 2 months ago • Options • Reply 1

Thanks, edited.

Siddhartha Sinha • Posted on Version 4 • 2 months ago • Options • Reply 1

Very nice kernel. Would it be possible to use patient age and sex as additional model features?

Giulia Savorgnan • Posted on Version 4 • 2 months ago • Options • Reply 1

Thanks a lot for sharing, I found a pseudo-bug here: `def __load__(self, filename): # load dicom file as numpy array img = pydicom.dcmread(os.path.join(self.folder, filename)).pixel_array # create empty mask msk = np.zeros(img.shape) # get filename without extension filename =`

`filename.split('.')[0] # if image contains pneumonia if filename in *pneumonia_locations*: # loop through pneumonia for location in *pneumonia_locations*[filename]:` you are using the global variable `pneumonia_locations` rather than the local `self.pneumonia_locations`

Andy Harless • Posted on Version 4 • 2 months ago • Options • Reply 2

I forked version 5, which had a verified score of 0.100, ran it with no changes, and got a score of 0.114, which put me at #8/309. I guess the current LB is mostly a random assignment.

Joli • Posted on Latest Version • 2 months ago • Options • Reply 0

I forked version 10 and had a score of 0.118 (ran with no changes), 0.002 higher than the verified score.

Moshel • Posted on Latest Version • 2 months ago • Options • Reply -1

That's a really educational kernel, thank you! Histogram equilization adds about 2% without doing anything

Glacier_Li • Posted on Latest Version • a month ago • Options • Reply 0

Thank you! Do you make histogram equalization for all training set and validation set, or select part training set and validation set? How about test set?

TRAN QUANG TH... • Posted on Version 4 • 2 months ago • Options • Reply 0

I learned a lot from your kernel. Thank you!

TSHLNIHAO • Posted on Version 4 • 2 months ago • Options • Reply 0

good ,have some algrithms ,such as ssd ,r-fcn,faster r-cnn etc

Tom Bu • Posted on Latest Version • 2 months ago • Options • Reply 0

it already implements resnet blocks

Andy Harless • Posted on Version 7 • 2 months ago • Options • Reply 0

I'm getting some weird results and starting to wonder about the mean_iou calculation. I don't know Tensorflow well, so I'm semi-guessing at the meaning, but it looks like you are calculating (I+N)/(U+N) whereas you may have meant to calculate (I+1)/(U+1). It looks like it sums a whole tensor full of ones in both the numerator and the denominator, which would explain why IoU appears to be high on the first epoch and then goes down afterward.

Jonne Kernel Author • Posted on Version 7 • 2 months ago • Options • Reply 0

Both the intersect as the union have a shape of (batch_size,) after the reduce_sum, with an intersect and union value per item in the batch. The smooth also has a shape of (batch_size,) to calculate (I+1)/(U+1) per item.

The mean_iou function is not used for training, just for tracking. It is not perfect as it gives a score of 1. per item if both y_true and y_pred are 0., whereas in the competition those items are ignored for calculating the score.

My guess is that in the beginning the network learns to predict no lung opacities anywhere, resulting in a lot of items (all items with no lung opacities) with a score of 1., which explains the high mean.

Andy Harless • Posted on Latest Version • 2 months ago • Options • Reply 0

That makes sense, but it doesn't seem consistent with the results I'm getting. If I run it for just one epoch, it predicts opacities nowhere. (That is, I have to use a threshold less than 0.5 to identify areas it thinks might be opacities.) In itself, this seems reasonable: when it hasn't learned much yet, it will be conservative in assigning probabilities. But it doesn't seem consistent with the high mean IoU metric usually reported for the first epoch.

Tom Bu • Posted on Latest Version • 2 months ago • Options • Reply 0

I was trying to go for a greater depth, but whenever I run it, I always end up with Resource Exhausted Error. Do you know any way to get around it?

Andy Harless • Posted on Latest Version • 2 months ago • Options • Reply 1

@tommy260 You can run with a smaller batch size (and maybe increase the momentum parameter to compensate).

Marsh • Posted on Latest Version • 2 months ago • Options • Reply 0

Thank you for explaining clearly and for the commented code. I've learned a lot from this kernel.

Tim H • Posted on Latest Version • 2 months ago • Options • Reply 0

Thank you for publishing this! I've learned a lot from it.

Nazim Girach • Posted on Latest Version • a month ago • Options • Reply 0

Hey Jonne, would like to know your thoughts on segmentation vs detection topic.

You seem to have a really good score and this kernel of yours is the reason many people are doing segmentation, so thanks for sharing.

I have tried a lot of small increments but in vain. Would you suggest switching over to detection or trying different architectures for segmentation?

Jonne Kernel Author • Posted on Latest Version • a month ago • Options • Reply 3

With this approach I was only able to get 0.16 within the limits of the Kaggle kernel. I definitely recommend switching to bounding box detection. With a simplified (no anchors and no need for object classification) region proposal network I was able to get to 0.185 with a Kaggle kernel.

Hengjian Jia • Posted on Latest Version • a month ago • Options • Reply 0

Hi, this is a nice kernel but the IOU values seem much much higher than the leaderboard's IOU values?

Jonne Kernel Author • Posted on Latest Version • a month ago • Options • Reply 0

Correct. First of all mean_iou is not supposed to reflect the leaderboard metric (it only uses a threshold of 0.5 and doesn't ignore items with zero true and predicted boxes). Secondly even when implementing the correct metric there is a disconnect between the mean average precision on the validation set and the score on the leaderboard (see: https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/discussion/66323).

Andy Harless • Posted on Latest Version • a month ago • Options • Reply 0

@jonnedtc I appreciate your two points in response to @hengjianjia, as well as your reply to me earlier in the thread, but there is still something very weird about these IoU values. Either they are wrong, or they mean something completely different from what I thought they meant. I implemented (following @vbookshelf's suggestion in another kernel to use tf.py_func) a function to calculate mean IoU, and it gets results that are what I expect (see the `raw_iou` function in this kernel, for example). I think it should do the same thing as your function, but yours gives weird results, for reasons I don't understand.

Jonne Kernel Author • Posted on Latest Version • a month ago • Options • Reply 0

The mean_iou function is just to track the learning, so I didn't put too much thought into it. I took a look into the raw_iou function and I think the difference stems from images without any true lung opacities and without any predicted lung opacities: the raw_iou calculates: 0/(0+0-0+1e-7) = 0 while mean_iou calculates (0+1)/(0+0-0+1) = 1 due to the smooth. Note that the competition metric simply ignores these images.

Carl S. • Posted on Latest Version • a month ago • Options • Reply 0

Thanks for the great kernel! For people forking the kernel and building off it, one parameter to adjust might be the threshold for mask predictions. Instead of 0.5, 0.3 and 0.4 seemed to allow more opportunities for post-processing analysis of the predicted bounding boxes.

TPloveYXT520 • Posted on Latest Version • a month ago • Options • Reply 0

Hi~ Jonne~ Do you use the segmentation model to get LB 0.190+ score?

abhishek singh • Posted on Latest Version • a month ago • Options • Reply 0

one little question. Why do you get coordinates in this order: y, x, y2, x2 = region.bbox? what is the purpose of doing it ?

PC Jimmmy • Posted on Latest Version • 18 days ago • Options • Reply 0

Love your work! thanks for sharing

I have been using what I think is version 2 of your generator code. Played around with lots of the other pieces but have kept that part intact.

I needed to do a clean Windows installation so I have nice shiny Win 10 version 1803 with the latest Anaconda and all libraries up to date.

The script now dies with a 'stopiteration' error in model.fit_generator cell. I am using a balanced data set of 15000 images and the stop appears to show up when I get to the end of data set.

Did you run across this error over your 10 versions? I am assuming that one of the libraries needs to be downgraded but have no clues where to start. The exact same script still runs on my second PC which did not have to die and do a clean restart.

So I have started by plugging in your latest generator code and have an hour or two until the error is due to hit :)

I understand this error occurs for an improperly built generator but you built it well and its been doing fine for many weeks. Any clues?

Master • Posted on Latest Version • 15 days ago • Options • Reply 0

Great kernel! I noticed that going deeper (using pretrained resnet models) doesn't work. In fact, it lowers the score significantly. Did anyone else experience the same thing?

Vignesh • Posted on Latest Version • 11 days ago • Options • Reply 0

Great kernel !!!Very informative and easy understanding visualization part. I have learnt a lot in your this kernel

2 months ago

This Comment was deleted.

a month ago

This Comment was deleted.

Our Team Terms Privacy Contact/Support