Student Name: Bowen Li
Email: bli61001@usc.edu

Code Sample MRI Image

Business problem

Our business goal is to use Magnetic Resonance Images(MRI) to realize automatic medical identification with deep learning model(ResNet50).

Introduction

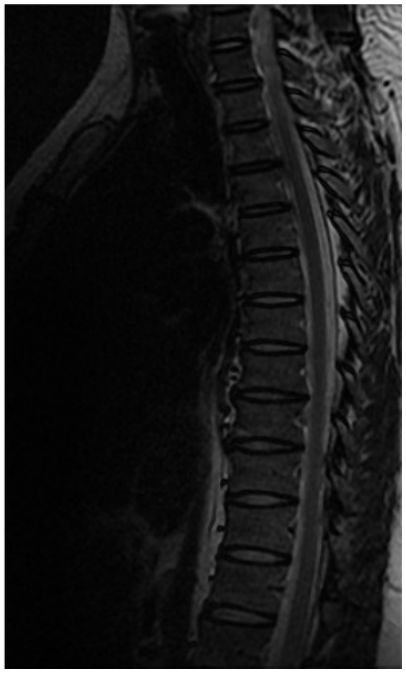
Based on our business problems, we need to realize automatic medical identification with MRI using deep learning models. The images are shown in normal case and patient case images are below.

(a) Normal Case

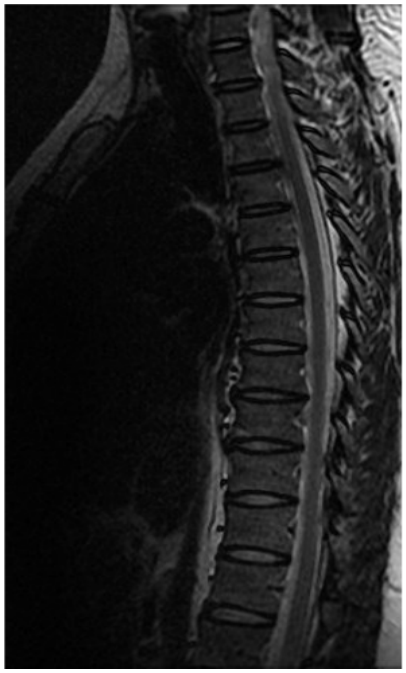
(b) Multiple Myeloma

Histogram Equalization

Normalize image distribution to [0, 255] through gray histogram normalization due to varying gray scale values among images, as demonstrated in the figure below.



(a) Before Hist Normalization



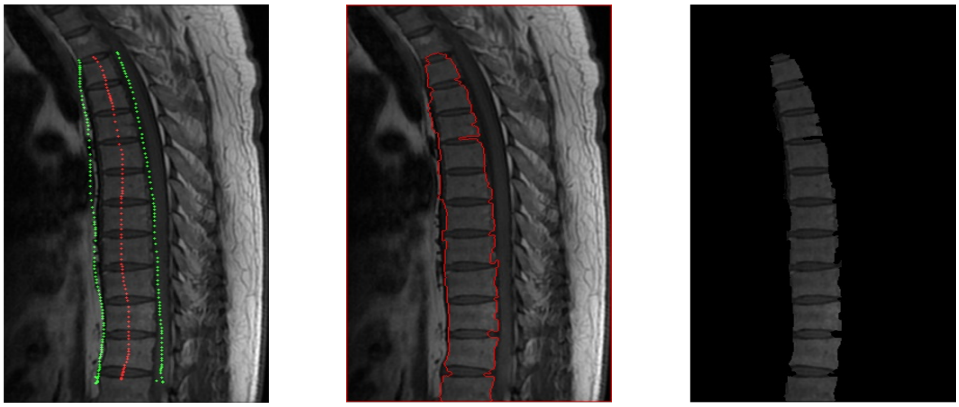
(b) After Hist Normalization

```python
import os
import SimpleITK as sitk
import numpy as np

def adaptive_normal(image_path, outpath):
    """
    Rescale image to [-1, 1] and exclude background voxels in statistical analysis.

    Inputs:

    image_path: The path to the Nii image in .nii.gz format.
    outpath: The .nii.gz format path to save the normalized image.
    Outputs:

    The absolute path to the normalized .nii.gz file.
    """

    min_p = 0.001
    max_p = 0.999  # quantile prefer 98~99

    image = sitk.ReadImage(image_path)
    image_array = sitk.GetArrayFromImage(image)
    imgArray = np.float32(image_array)

    imgPixel = imgArray[imgArray >= 0]
    imgPixel.sort()
    index = int(round(len(imgPixel) - 1) * min_p + 0.5)
    if index < 0:
        index = 0
```

```
    if index > (len(imgPixel) - 1):
        index = len(imgPixel) - 1
    value_min = imgPixel[index]

    index = int(round(len(imgPixel) - 1) * max_p + 0.5)
    if index < 0:
        index = 0
    if index > (len(imgPixel) - 1):
        index = len(imgPixel) - 1
    value_max = imgPixel[index]

    mean = (value_max + value_min) / 2.0
    stddev = (value_max - value_min) / 2.0

    imgArray = (imgArray - mean) / stddev
    imgArray[imgArray < -1] = -1.0
    imgArray[imgArray > 1] = 1.0

    img = sitk.GetImageFromArray(imgArray, isVector=False)
    sitk.WriteImage(img, outpath)
    return os.path.abspath(outpath)
```

Semantic segmentation

The watershed algorithm is a method for image region segmentation. In the segmentation process, it takes the similarity between adjacent pixels as an important reference and connects pixels that are similar in space and gray value to form a closed outline. Closeness is an important characteristic of the watershed algorithm. As demonstrated in the figure below.



(a) Pick Pixels Points (b) Pick Region of Interest (c) Extract Region of Interest

```
Interactive Watershed + Optical Flow Spinal Cord Segmentation Method

Operating Procedure:

1. Run the program.
2. Click the left mouse button (or hold alt and slide the mouse) to select foreground
    markers (red).
```

```
   Click the right mouse button (or hold shift and slide the mouse) to select background markers (green).
   Hold ctrl and slide the mouse to delete markers.
3. After markers are selected, press the esc key to view the segmentation result.
  If the result is not satisfactory, press the R key to continue modifying marker positions.
  If the result is satisfactory, press any key to end the program.

import cv2
import os
import numpy as np

#Input
imgs_in = "/Users/brycelee/Desktop/MMT/image%d.jpg"
#Output
imgs_out = "/Users/brycelee/Desktop/MMT/result"

'''
cv2.circle(image, center_coordinates, radius, color, thickness)
image: It is the image on which the circle is to be drawn.
center_coordinates: It is the center coordinates of the circle. The coordinates are represented as a tuple of two values, i.e. (X coordinate value, Y coordinate value).
radius: It is the radius of the circle.
color: It is the color of the border line of the circle to be drawn. For BGR, we pass it through a tuple. For example, (255, 0, 0) is blue.
thickness: It is the thickness of the circle border line in pixels. A thickness of -1 pixel will fill the rectangle shape with the specified color.
Return value: It returns an image.

cv2.setMouseCallback(windowName, onMouse [, userdata])
winname: Name of the window.
onMouse: Mouse response function, callback function.
userdata: The param passed back to userdate in onMouse.
Out of the above three parameters, the second one, the mouse callback function, is the most important. This function is executed when a mouse event occurs.

on_mouse(event, x, y, flags, param)
The on_mouse above can be any function name, and the other parameters are explained as follows:
event is one of the CV_EVENT_* variables, see the table below;
x and y are the coordinates of the mouse in the image coordinate system (not the window coordinate system);
flags is a combination of CV_EVENT_FLAG;
param is the user-defined parameter passed to the setMouseCallback function call.

event Specific description:
EVENT_MOUSEMOVE 0 // Slide
EVENT_LBUTTONDOWN 1 // Left button click
EVENT_RBUTTONDOWN 2 // Right button click
EVENT_MBUTTONDOWN 3 // Middle button click
EVENT_LBUTTONUP 4 // Left button release
EVENT_RBUTTONUP 5 // Right button release
EVENT_MBUTTONUP 6 // Middle button release
EVENT_LBUTTONDBLCLK 7 // Left button double click
```

```
EVENT_RBUTTONDBLCLK 8 // Right button double click
EVENT_MBUTTONDBLCLK 9 // Middle button double click

flags Specific description:
EVENT_FLAG_LBUTTON 1 // Left button drag
EVENT_FLAG_RBUTTON 2 // Right button drag
EVENT_FLAG_MBUTTON 4 // Middle button drag
EVENT_FLAG_CTRLKEY 8 //(815) Ctrl is not released
EVENT_FLAG_SHIFTKEY 16 //(1631) Shift is not released
EVENT_FLAG_ALTKEY 32 //(32~39) Alt is not released
'''

def mouse_click3(event, x, y, flags, para):
    '''
    Mouse response callback function
    event is one of the CV_EVENT_* variables, see the table below;
    x and y are the coordinates of the mouse in the image coordinate system (not the window coordinate system);
    flags is a combination of CV_EVENT_FLAG;
    param is the user-defined parameter passed to the setMouseCallback function call.
    '''
    #if event == cv2.EVENT_LBUTTONDOWN:  # Left mouse button click
    #if flags == cv2.EVENT_FLAG_LBUTTON:  # Mouse press
    if flags == cv2.EVENT_FLAG_ALTKEY or flags == cv2.EVENT_FLAG_LBUTTON:  # Hold alt+mouse click
        para[0]=0
        para[1]=x
        para[2]=y
        print('Select foreground pixel point coordinates: ',x,y)
    #elif event == cv2.EVENT_RBUTTONDOWN:  # Right mouse button click
    #elif flags == cv2.EVENT_FLAG_RBUTTON:  # Hold shift+mouse click
    elif flags == cv2.EVENT_FLAG_SHIFTKEY or flags == cv2.EVENT_FLAG_RBUTTON:  # Hold shift+mouse click
        para[0]=1
        para[1]=x
        para[2]=y
        print('Select background pixel point coordinates: ',x,y)
    elif flags == cv2.EVENT_FLAG_CTRLKEY:
        para[0]=2
        para[1]=x
        para[2]=y
        print('Erase pixel point coordinates: ',x,y)
def watershed_opticalflow_interactive(imgs_in, imgs_out_path):
    '''
    Interactive labeling: watershed + optical flow tracking labeling points
    :param imgs_in:
    :param imgs_out_path:
    :return:
    '''

    start_index = 1
    end_index = 46 # The parameter that needs to be adjusted according to the number of data

```

```
    first_img = cv2.imread(imgs_in % (start_index))
    # first_img_gray = cv2.cvtColor(first_img,cv2.COLOR_BGR2GRAY)
    mask_img = np.zeros_like(first_img)

    # Initialize parameters
    para = [0, -1, -1]

    red_point = []
    green_point = []

    i = start_index

    # Process the first one
    while True:

        img_ = cv2.imread(imgs_in % (i))
        origin_img = np.copy(img_)

        # Convert BGR format to grayscale image, which is single-channel
        gray_ = cv2.cvtColor(img_, cv2.COLOR_BGR2GRAY)
        # mask_img = np.zeros_like(img)
        markers = np.zeros_like(gray_, dtype=np.int32)

        # Name window title
        cv2.namedWindow('img')
        # Display image in window
        cv2.imshow('img', img_)
        # Mouse interaction function
        cv2.setMouseCallback('img', mouse_click3, para)

        # Initialize Key
        key = 0

        # B,G,R
        red_label = (0, 0, 255)
        green_label = (0, 255, 0)
        clear_label = (0, 0, 0)

        # Manually select label points
        while key != 27:  # ASCII code of esc key is 27

            if key == 99:  # If you think the overall labeling is not ideal, you can
                press c to clear all
                mask_img = np.zeros_like(first_img)
                red_point = []
                green_point = []
                para = [0, -1, -1]
                cv2.setMouseCallback('img', mouse_click3, para)
            if para[0] == 0:
                cv2.circle(mask_img, (para[1], para[2]), 1, red_label, -1)
                # Image, center coordinate, radius, boundary color, boundary thickness
                    (-1 pixels fill the rectangle shape with the specified color)
                # Return image
            elif para[0] == 1:
```

```
                cv2.circle(mask_img, (para[1], para[2]), 1, green_label, -1)
            elif para[0] == 2:
                cv2.circle(mask_img, (para[1], para[2]), 5, clear_label, -1)
            point_img_ = cv2.add(mask_img, img_)
            cv2.imshow('img', point_img_)
            key = cv2.waitKey(30)


        cv2.circle(mask_img,(0,0),20,(0,0,0),-1)

        h = mask_img.shape[0]
        w = mask_img.shape[1]

        for row in range(h):
            for col in range(w):
                if mask_img[row][col][2] == 255:
                    markers[row][col]=1
                    red_point.append([[col,row]])
                elif mask_img[row][col][1] == 255:
                    markers[row][col]=2
                    green_point.append([[col,row]])
        # WaterShed Algorithm
        markers = cv2.watershed(img_,markers=markers)


        #
        img_edge = np.copy(img_)
        img_edge[markers==-1] = [0,0,255]    #
        cv2.imshow('res_bonder_%d'%(i),img_edge)

        #
        img_res = np.copy(img_)
        img_res[markers==1] = [255,255,255] #####            #####
        img_res[markers!=1] = [0,0,0]
        #cv2.imshow('res_seg_%d'%(i),img_res)

        #
        seg_res=np.copy(img_)
        seg_res[markers!=1] = [0,0,0]

        #
        #cv2.imshow('origin_%d'%(i),img_bk)
        r_key = cv2.waitKey(0)  #cv2.waitKey()
                                                0
        if r_key == 114:  # r_key=='r'
            cv2.destroyAllWindows() #
            continue
        else:
            cv2.destroyAllWindows()
        #                BGR
        img_gray_res = cv2.cvtColor(img_res,cv2.COLOR_BGR2GRAY)

        img_origin_mask = cv2.add(origin_img,mask_img)

```

```
208         cv2.imwrite(os.path.join(imgs_out_path,"%d_mask.jpg"%i),img_origin_mask)
209         cv2.imwrite(os.path.join(imgs_out_path,"%d_res.jpg"%i),img_gray_res)
210         cv2.imwrite(os.path.join(imgs_out_path,"%d_edge.jpg"%i),img_edge)
211         cv2.imwrite(os.path.join(imgs_out_path, "%d_seg_res.jpg"%i), seg_res)
212         break
213 
214     #
215     # Parameters for lucas kanade optical flow
216     lk_params = dict(winSize=(15, 15),
217                      maxLevel=3,
218                      criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 150, 1))
219     #
220     max_point_dist = 15.0
221 
222     first_frame = np.copy(first_img)
223     old_gray = cv2.cvtColor(first_frame, cv2.COLOR_BGR2GRAY)
224 
225     first_frame_mask=np.zeros_like(first_frame)
226 
227     p0_red_point = np.array(red_point,dtype=np.float32)
228     p0_green_point = np.array(green_point,dtype=np.float32)
229 
230     #           mask
231     mask_track = np.zeros_like(first_img)
232 
233     i = start_index+1
234     while i != end_index:
235 
236         img_cur = cv2.imread(imgs_in%(i))
237         img_cur_copy = np.copy(img_cur)
238         gray_cur = cv2.cvtColor(img_cur, cv2.COLOR_BGR2GRAY)
239 
240         #Lucas -  K a n a d e
241         #
                                                                                1
242         p1_red_point, st_red, err_red = cv2.calcOpticalFlowPyrLK(old_gray, gray_cur,
                p0_red_point, None, **lk_params)
243         p1_green_point, st_green, err_green = cv2.calcOpticalFlowPyrLK(old_gray, gray_cur
                , p0_green_point, None, **lk_params)
244 
245         # Select good points
246         red_good_new = p1_red_point[st_red == 1]
247         red_good_old = p0_red_point[st_red == 1]
248 
249         green_good_new = p1_green_point[st_green == 1]
250         green_good_old = p0_green_point[st_green == 1]
251 
252         #         mask_img
253         #mask_img = np.zeros_like(img)
254 
255         markers = np.zeros_like(gray_cur,dtype=np.int32)
256 
257         red_delete_index = []
```

```
        green_delete_index = []

        # draw the red points tracks
            # enumerate()                                                (
                                                 )
            #                                                  for

            # zip()


        for j, (new, old) in enumerate(zip(red_good_new,red_good_old)):
            a, b = new.ravel()
            c, d = old.ravel()

            #                     newold
            dist = np.sqrt(np.sum(np.square(new.ravel() - old.ravel())))

            #
            if dist > max_point_dist:
                red_delete_index.append(j)
            else:
                #
                #print(type(mask_track),mask_track.shape)
                #print(a,b,c,d)
                mask_track = cv2.line(mask_track, (int(a), int(b)), (int(c), int(d)),
                    (50,20,100), 1, lineType=cv2.LINE_AA)

                #
                mask_img[int(d)][int(c)] = [0,0,0]
                #cv2.circle(mask_img, (c, d),1,(0,0,0),-1)
                #
                #cv2.circle(mask_img, (a, b), 1, (0,0,255), -1)
                if b < mask_img.shape[0] and a < mask_img.shape[1]:
                    mask_img[int(b)][int(a)] = [0,0,255]

                #if b<512 and a<512:
                #   markers_temp[int(b)][int(a)] = 1

        # draw the green points tracks
        for j, (new, old) in enumerate(zip(green_good_new,green_good_old)):
            a, b = new.ravel()
            c, d = old.ravel()

            #                     newold
            dist = np.sqrt(np.sum(np.square(new.ravel() - old.ravel())))

            #
            if dist > max_point_dist:
                green_delete_index.append(j)
            else:
                #
                mask_track = cv2.line(mask_track, (int(a), int(b)), (int(c), int(d)),
                    (50,70,50), 1, lineType=cv2.LINE_AA)

```

```
                #
                mask_img[int(d)][int(c)] = [0,0,0]
                #cv2.circle(mask_img, (c, d),1,(0,0,0),-1)
                #
                #cv2.circle(mask_img, (a, b), 1, (0,0,255), -1)
                if b < mask_img.shape[0] and a < mask_img.shape[1]:
                    mask_img[int(b)][int(a)] = [0,255,0]

                #if b<512 and a<512:
                #   markers_temp[int(b)][int(a)] = 1

        #            +      +
        cv2.namedWindow('img')
        show_img = cv2.add(mask_img,img_cur)
        show_img = cv2.add(show_img,mask_track)
        cv2.imshow('img',show_img)
        #
        cv2.imwrite(os.path.join(imgs_out_path,"%d_track.jpg"%i),show_img)
        cv2.waitKey(0)
        cv2.setMouseCallback('img',mouse_click3,para)
        key = 0

        red_label = (0,0,255)
        green_label = (0,255,0)
        clear_label = (0,0,0)

        #
        while key != 27:  # esc
            if key == 99:   #
                                                c
                mask_img=np.zeros_like(first_img)
                red_point = []
                green_point = []
                para = [0, -1, -1]
                cv2.setMouseCallback('img', mouse_click3, para)

            if para[0] == 0:
                cv2.circle(mask_img,(para[1],para[2]),1,red_label,-1)
            elif para[0] == 1:
                cv2.circle(mask_img,(para[1],para[2]),1,green_label,1)
            elif para[0] == 2:
                cv2.circle(mask_img,(para[1],para[2]),5,clear_label,-1)
            point_img=cv2.add(mask_img,img_cur)
            cv2.imshow('img',point_img)
            key = cv2.waitKey(30)

        cv2.circle(mask_img,(0,0),20,(0,0,0),-1)

        h = mask_img.shape[0]
        w = mask_img.shape[1]

        #          red_point
        red_point.clear()
        #          green_point
```

```
        green_point.clear()

        for row in range(h):
            for col in range(w):
                if mask_img[row][col][2] == 255:
                    markers[row][col]=1
                    red_point.append([[col,row]]) #
                elif mask_img[row][col][1] == 255:
                    markers[row][col]=2
                    green_point.append([[col,row]])

        #
        markers = cv2.watershed(img_cur,markers=markers)


        #
        img_edge = np.copy(img_cur)
        img_edge[markers==-1] = [0,0,255]    #
        cv2.imshow('res_bonder_%d'%(i),img_edge)

        #
        img_res = np.copy(img_cur)
        img_res[markers==1] = [255,255,255] #####            #####
        img_res[markers!=1] = [0,0,0]
        #cv2.imshow('res_seg_%d'%(i),img_res)

        #
        seg_res=np.copy(img_cur)
        seg_res[markers!=1] = [0,0,0]

        #cv2.imshow('origin_%d'%(i),img_bk) #


        #
        p0_red_point = np.array(red_point,dtype=np.float32)
        p0_green_point = np.array(green_point,dtype=np.float32)
        old_gray = gray_cur.copy()

        #           esc            ,          r
        r_key = cv2.waitKey(0)

        if r_key == 114:  # r_key=='r'
            cv2.destroyAllWindows()     #
            continue
        else:
            cv2.destroyAllWindows()
        # if r_key == 114:  # r_key=='r'
        #     cv2.destroyAllWindows()     #
        #     continue
        #
        # r_key = cv2.waitKey(0)
        # while r_key!=32:   #
        #     a=[]
        # cv2.destroyAllWindows()
```

```
414
415         #                 BGR
416         img_gray_res = cv2.cvtColor(img_res,cv2.COLOR_BGR2GRAY)
417
418         img_origin_mask = cv2.add(img_cur_copy,mask_img)
419
420         cv2.imwrite(os.path.join(imgs_out_path,"%d_mask.jpg"%i),img_origin_mask)
421         cv2.imwrite(os.path.join(imgs_out_path,"%d_res.jpg"%i),img_gray_res)
422         cv2.imwrite(os.path.join(imgs_out_path,"%d_edge.jpg"%i),img_edge)
423         cv2.imwrite(os.path.join(imgs_out_path, "%d_seg_res.jpg"%i), seg_res)
424         i = i + 1
425
426
427 if __name__ == '__main__':
428     watershed_opticalflow_inter(imgs_in,imgs_out)
```

Image Augmentation

Image augmentation is a technique that involves making a series of random changes to training images in order to generate similar but different training samples, thereby expanding the size of the training dataset. At the same time, image augmentation can reduce the model's dependence on certain attributes, thereby improving the model's generalization ability.

```
1  import os
2  import cv2
3  import imgaug as ia
4  from imgaug import augmenters as iaa
5
6  # Set the path to the directory containing the input images
7  input_dir = "path/to/input/directory"
8
9  # Set the path to the directory where augmented images will be saved
10 output_dir = "path/to/output/directory"
11
12 # Create the output directory if it doesn't exist
13 if not os.path.exists(output_dir):
14     os.makedirs(output_dir)
15
16 # Define the augmentation pipeline
17 seq = iaa.Sequential([
18     iaa.Fliplr(0.5),  # horizontally flip 50% of the images
19     iaa.Affine(rotate=(-10, 10)),  # rotate images by -10 to +10 degrees
20     iaa.GaussianBlur(sigma=(0, 1.0))  # apply gaussian blur with a sigma between 0 and
           1.0
21 ])
22
23 # Loop through each image in the input directory
24 for filename in os.listdir(input_dir):
25     if filename.endswith(".jpg") or filename.endswith(".png"):
26         # Read the image
27         image = cv2.imread(os.path.join(input_dir, filename))
28
```

```
        # Apply augmentation
        augmented_images = seq.augment_images([image])

        # Save augmented images to the output directory
        for i, augmented_image in enumerate(augmented_images):
            output_filename = os.path.splitext(filename)[0] + "_augmented_" + str(i) + ".jpg"
            output_path = os.path.join(output_dir, output_filename)
            cv2.imwrite(output_path, augmented_image)
```

Model Built

```
from keras.callbacks import EarlyStopping
from keras.layers import Dense, Conv2D,  MaxPool2D, Flatten, GlobalAveragePooling2D, BatchNormalization, Layer, Add
from keras.models import Sequential
from keras.models import Model
import tensorflow as tf

#        ResNet18


class ResnetBlock(Model):
    """
    A standard resnet block.
    """

    def __init__(self, channels: int, down_sample=False):
        """
        channels: same as number of convolution kernels
        """
        super().__init__()

        self.__channels = channels
        self.__down_sample = down_sample
        self.__strides = [2, 1] if down_sample else [1, 1]

        KERNEL_SIZE = (3, 3)
        # use He initialization, instead of Xavier (a.k.a 'glorot_uniform' in Keras), as suggested in [2]
        INIT_SCHEME = "he_normal"

        self.conv_1 = Conv2D(self.__channels, strides=self.__strides[0],
                             kernel_size=KERNEL_SIZE, padding="same", kernel_initializer=INIT_SCHEME)
        self.bn_1 = BatchNormalization()
        self.conv_2 = Conv2D(self.__channels, strides=self.__strides[1],
                             kernel_size=KERNEL_SIZE, padding="same", kernel_initializer=INIT_SCHEME)
        self.bn_2 = BatchNormalization()
        self.merge = Add()

```

```
        if self.__down_sample:
            # perform down sampling using stride of 2, according to [1].
            self.res_conv = Conv2D(
                self.__channels, strides=2, kernel_size=(1, 1), kernel_initializer=
                    INIT_SCHEME, padding="same")
            self.res_bn = BatchNormalization()

    def call(self, inputs):
        res = inputs

        x = self.conv_1(inputs)
        x = self.bn_1(x)
        x = tf.nn.relu(x)
        x = self.conv_2(x)
        x = self.bn_2(x)

        if self.__down_sample:
            res = self.res_conv(res)
            res = self.res_bn(res)

        # if not perform down sample, then add a shortcut directly
        x = self.merge([x, res])
        out = tf.nn.relu(x)
        return out


class ResNet18(Model):

    def __init__(self, num_classes, **kwargs):
        """
            num_classes: number of classes in specific classification task.
        """
        super().__init__(**kwargs)
        self.conv_1 = Conv2D(64, (7, 7), strides=2,
                             padding="same", kernel_initializer="he_normal")
        self.init_bn = BatchNormalization()
        self.pool_2 = MaxPool2D(pool_size=(2, 2), strides=2, padding="same")
        self.res_1_1 = ResnetBlock(64)
        self.res_1_2 = ResnetBlock(64)
        self.res_2_1 = ResnetBlock(128, down_sample=True)
        self.res_2_2 = ResnetBlock(128)
        self.res_3_1 = ResnetBlock(256, down_sample=True)
        self.res_3_2 = ResnetBlock(256)
        self.res_4_1 = ResnetBlock(512, down_sample=True)
        self.res_4_2 = ResnetBlock(512)
        self.avg_pool = GlobalAveragePooling2D()
        self.flat = Flatten()
        self.fc = Dense(num_classes, activation="softmax")

    def call(self, inputs):
        out = self.conv_1(inputs)
        out = self.init_bn(out)
        out = tf.nn.relu(out)
        out = self.pool_2(out)
```

```
        for res_block in [self.res_1_1, self.res_1_2, self.res_2_1, self.res_2_2, self.
            res_3_1, self.res_3_2, self.res_4_1, self.res_4_2]:
            out = res_block(out)
        out = self.avg_pool(out)
        out = self.flat(out)
        out = self.fc(out)
        return out


# Summary of this model

Model: "res_net18_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_80 (Conv2D)          multiple                  9472

 batch_normalization_80 (Bat  multiple                 256
 chNormalization)

 max_pooling2d_4 (MaxPooling  multiple                 0
 2D)

 resnet_block_32 (ResnetBloc  multiple                 74368
 k)

 resnet_block_33 (ResnetBloc  multiple                 74368
 k)

 resnet_block_34 (ResnetBloc  multiple                 231296
 k)

 resnet_block_35 (ResnetBloc  multiple                 296192
 k)

 resnet_block_36 (ResnetBloc  multiple                 921344
 k)

 resnet_block_37 (ResnetBloc  multiple                 1182208
 k)

 resnet_block_38 (ResnetBloc  multiple                 3677696
 k)

 resnet_block_39 (ResnetBloc  multiple                 4723712
 k)

 global_average_pooling2d_4   multiple                 0
 (GlobalAveragePooling2D)

 flatten_4 (Flatten)         multiple                  0

 dense_4 (Dense)             multiple                  1026

```

```
==========================================================================
Total params: 11,191,938
Trainable params: 11,182,338
Non-trainable params: 9,600
--------------------------------------------------------------------------
```