Course Project

Xiaowei Zhao (xz2767)
Mingfeng Li (ml4209)

Problem:

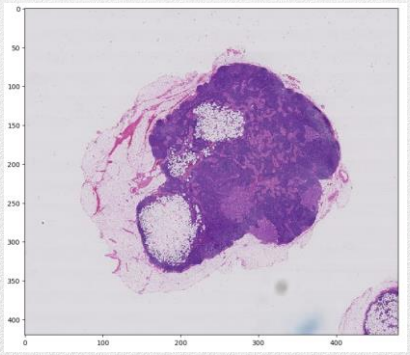
Detecting Cancer Metastases on Gigapixel Pathology Images

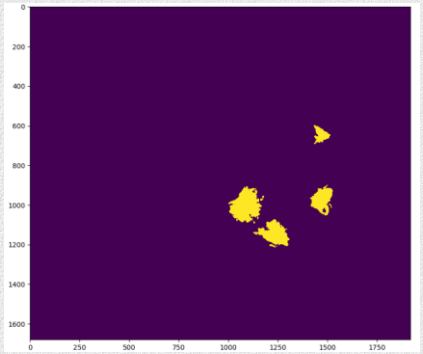
01 Get 299x299 patches of the slides and masks and make another dataset by zooming in these patches

02 Split the data into training and test set, then upsample the unbalanced dataset

03 Define the model by merging two sequential models

04 Train the model and make the predictions, then create heat maps





Starter Functions

```
def read_slide(slide, x, y, level, width, height, as_float=False):
    im = slide.read_region((x,y), level, (width, height))
    im = im.convert('RGB') # drop the alpha channel
    if as_float:
        im = np.asarray(im, dtype=np.float32)
    else:
        im = np.asarray(im)
    assert im.shape == (height, width, 3)
    return im

def find_tissue_pixels(image, intensity=0.8):
    im_gray = rgb2gray(image)
    assert im_gray.shape == (image.shape[0], image.shape[1])
    indices = np.where(im_gray <= intensity)
    return zip(indices[0], indices[1])

def apply_mask(im, mask, color=(255,0,0)):
    masked = np.copy(im)
    for x,y in mask: masked[x][y] = color
    return masked
```

Function read_slide is to read a region from the slide

Functions find_tissue_pixels and apply_mask are used to improve efficiency by ignoring non-tissue areas of the slide

Creating patches

Sliding a window over the slides and masks and creating 299x299 patches of them. The level of origin image is 3.

For each patch, check the percentage tissue in that patch, if percentage tissue > 20%, let the patch be zero, then plot the graph to check if the process of image is right.

```
[19] patches = []
     tumor_patches = []
     pixel = 299

     def create_patches(slides, masks, level, pixel):
         for k, slide in enumerate(slides[:1]):
             for j in range(0, slide.level_dimensions[level][1], pixel):
                 for i in range(0, slide.level_dimensions[level][0], pixel):
                     if i+pixel < slide.level_dimensions[level][0] and j+pixel < slide.level_dimensions[level][1]:
                         patch = read_slide(slide, i*8, j*8, level = level, width = pixel, height = pixel)
                         mask_patch = read_slide(masks[k], i*8, j*8, level = level, width = pixel, height = pixel)
                         tumor_patch = mask_patch[:, :, 0]
                         tissue_pixels = find_tissue_pixels(patch)
                         tissue_pixels = list(tissue_pixels)
                         percent_tissue = len(tissue_pixels) / float(patch.shape[0] * patch.shape[0]) * 100
                         if np.count_nonzero(np.array(tumor_patch)) > 0:
                             patches.append(patch)
                             tumor_patches.append(tumor_patch)
                         elif percent_tissue > 20:
                             patches.append(patch)
                             tumor_patches.append(tumor_patch)
                         else:
                             zero = np.zeros((pixel, pixel))
                             patches.append(zero)
                             tumor_patches.append(zero)

[20] create_patches(slides, masks, 3, pixel)
```

Zoom in all patches to make another dataset for training

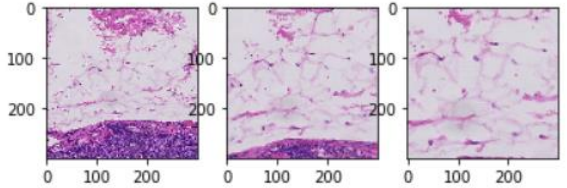
The function is to zoom in the patches. Center zoom in of the given image and returning an enlarged view of the image without changing dimensions.

```
[27] import cv2
     def cv2_clipped_zoom(img, zoom_factor):
             height, width = img.shape[:2]
             new_height, new_width = int(height * zoom_factor), int(width * zoom_factor)
             y1, x1 = max(0, new_height - height) // 2, max(0, new_width - width) // 2
             y2, x2 = y1 + height, x1 + width
             bbox = np.array([y1,x1,y2,x2])
             bbox = (bbox / zoom_factor).astype(np.int)
             y1, x1, y2, x2 = bbox
             cropped_img = img[y1:y2, x1:x2]
             resize_height, resize_width = min(new_height, height), min(new_width, width)
             pad_height1, pad_width1 = (height - resize_height) // 2, (width - resize_width) //2
             pad_height2, pad_width2 = (height - resize_height) - pad_height1, (width - resize_width) - pad_width1
             pad_spec = [(pad_height1, pad_height2), (pad_width1, pad_width2)] + [(0,0)] * (img.ndim - 2)
             result = cv2.resize(cropped_img, (resize_width, resize_height))
             result = np.pad(result, pad_spec, mode='constant')
             assert result.shape[0] == height and result.shape[1] == width
             return result
```

Show some zoom-in examples

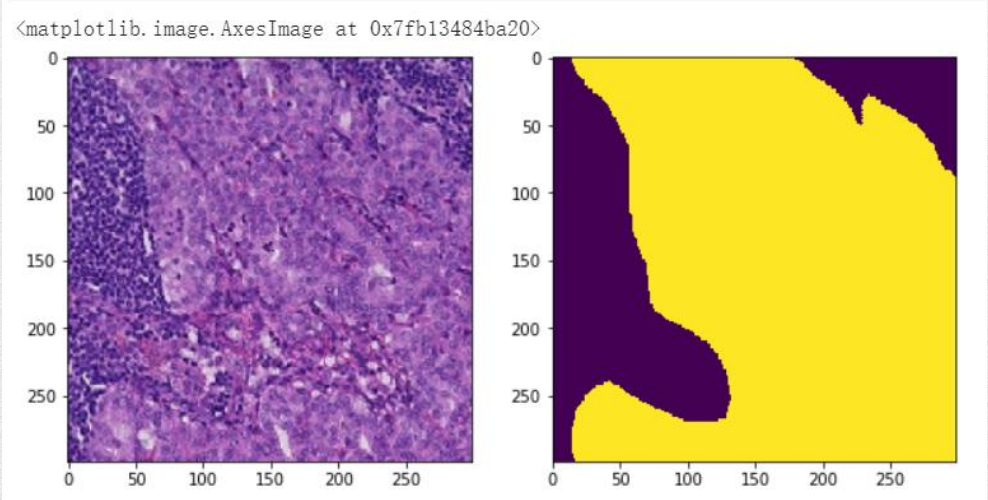
```
img = patches[1]
zm1 = cv2_clipped_zoom(img, 1.5)
zm2 = cv2_clipped_zoom(img, 2)

fig, ax = plt.subplots(1, 3)
ax[0].imshow(img)
ax[1].imshow(zm1)
ax[2].imshow(zm2)
```

```
<matplotlib.image.AxesImage at 0x7fb134bbcc50>
```



Assigning labels to each zoomed patch.

1 if there is at least one non-zero entry in the corresponding zoomed mask patch, 0 if not.

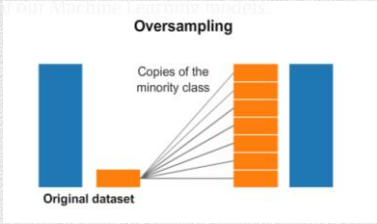


Preprocessing - Oversampling

```
class0_index = np.where(y_train_zoomed == 0)[0]
class1_index = np.where(y_train_zoomed == 1)[0]
class1_index_upsampled = np.random.choice(class1_index, size=len(class0_index), replace=True)

print("The number of class 0 is: {}".format(len(class0_index)))

X_train_upsampled = []
X_train_zoomed_upsampled = []

for i in class1_index_upsampled:
  X_train_upsampled.append(X_train[i])
  X_train_zoomed_upsampled.append(X_train_zoomed[i])

X_train_temp = []
X_train_zoomed_temp = []
for i in class0_index:
  X_train_temp.append(X_train[i])
  X_train_zoomed_temp.append(X_train_zoomed[i])

for i in X_train_upsampled:
  X_train_temp.append(i)

for i in X_train_zoomed_upsampled:
  X_train_zoomed_temp.append(i)

y_train_upsampled = list(y_train_zoomed[class1_index_upsampled])
y_train_temp = list(y_train_zoomed[class0_index])
for i in y_train_upsampled:
  y_train_temp.append(i)

X_train_balanced = np.array(X_train_temp)
y_train_balanced = np.array(y_train_temp)

X_train_zoomed_balanced = np.array(X_train_zoomed_temp)
y_train_zoomed_balanced = y_train_balanced
```

Problem:
Data has high level of class imbalance. Target class (tumor patch, class 1) is only 13% of whole dataset.

Solution:
Random select copies of patches containing tumors to reach the same number as the normal cells' patches.



Defining Network

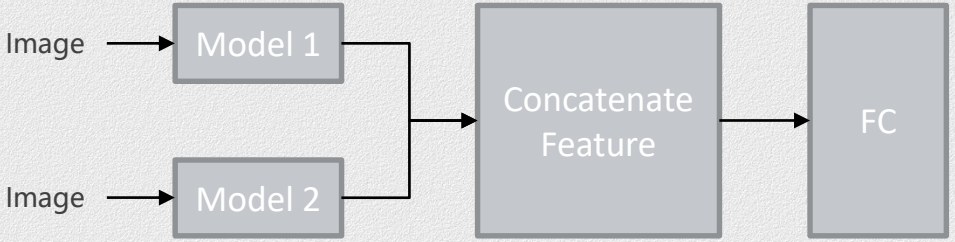
We concatenate two networks together to create a multi scale model, attempting to capture both details and contexts. Two scales used are 8X (level 3 dimension) and 12X.

```
model_base = applications.inception_v3.InceptionV3(weights=None,include_top = False,input_shape=(299,299,3))

model_out = Flatten(name='m_layer_1')(model_base.output)
model_out = Dense(128, activation='relu',name='m_layer_2')(model_out)
model = Model(model_base.input, model_out)

model_zoomed_base = applications.vgg16.VGG16(weights=None,include_top = False,input_shape=(299,299,3))
#model_zoomed_base.get_layer(name='mixed0').name='mixed0_1'
model_zoomed_out = Flatten(name='z_layer_1')(model_zoomed_base.output)
model_zoomed_out = Dense(128, activation='relu',name='z_layer_2')(model_zoomed_out)
model_zoomed = Model(model_zoomed_base.input, model_zoomed_out)

concatenated = concatenate([model_out, model_zoomed_out])
out = Dense(1, activation='sigmoid', name = 'output_layer')(concatenated)

merged_model = Model([model_base.input, model_zoomed_base.input], out)
```

```
sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
adam = optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=False)
rmsprop = optimizers.RMSprop(lr = 0.0001)
merged_model.compile(loss = 'binary_crossentropy',optimizer=sgd,metrics=['accuracy',f1])
```

Model Details:

Accuracy and F1 score are used as metrics to evaluate our model fitting on validation sets. The optimizer used is SGD with learning rate 0.1, decay at 1e-6 and momentum at 0.9.



Training Model

~9K training patches in total.
Deep Learning is trained from scratch with 10 epochs and shuffling.
The accuracy reaches 96% and f1 score reaches 0.85.

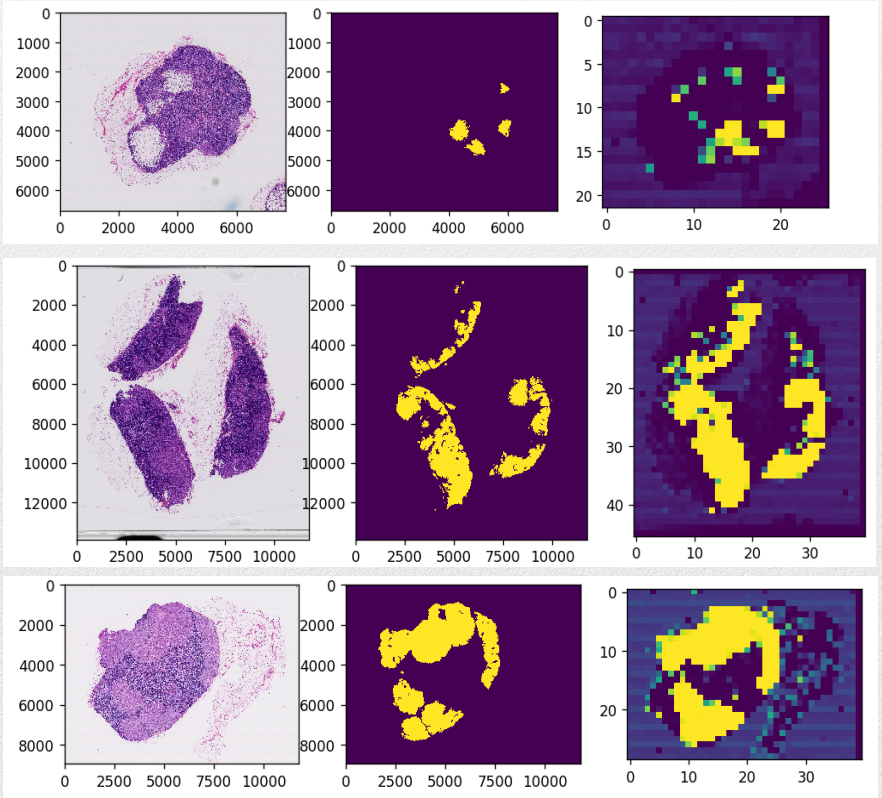
```
merged_model.fit([X_train_balanced, X_train_zoomed_balanced], y_train_balanced, epochs=10 , validation_data=([X_test, X_test_zoomed], y_test_zoomed), shuffle = True)
```

```
Train on 8742 samples, validate on 1270 samples
Epoch 1/10
8742/8742 [==============================] - 173s 20ms/sample - loss: 0.1565 - accuracy: 0.9445 - f1: 0.9398 - val_loss: 0.4670 - val_accuracy: 0.7669 - val_f1: 0.4992
Epoch 2/10
8742/8742 [==============================] - 173s 20ms/sample - loss: 0.1439 - accuracy: 0.9459 - f1: 0.9438 - val_loss: 0.4147 - val_accuracy: 0.8646 - val_f1: 0.6369
Epoch 3/10
8742/8742 [==============================] - 173s 20ms/sample - loss: 0.2537 - accuracy: 0.8989 - f1: 0.8903 - val_loss: 0.1642 - val_accuracy: 0.9425 - val_f1: 0.7777
Epoch 4/10
8742/8742 [==============================] - 173s 20ms/sample - loss: 0.1660 - accuracy: 0.9302 - f1: 0.9260 - val_loss: 0.2011 - val_accuracy: 0.9323 - val_f1: 0.7672
Epoch 5/10
8742/8742 [==============================] - 173s 20ms/sample - loss: 0.1547 - accuracy: 0.9299 - f1: 0.9235 - val_loss: 0.3221 - val_accuracy: 0.9268 - val_f1: 0.6243
Epoch 6/10
8742/8742 [==============================] - 173s 20ms/sample - loss: 0.2039 - accuracy: 0.9016 - f1: 0.8942 - val_loss: 0.1410 - val_accuracy: 0.9504 - val_f1: 0.8244
Epoch 7/10
8742/8742 [==============================] - 173s 20ms/sample - loss: 0.1411 - accuracy: 0.9404 - f1: 0.9384 - val_loss: 0.3494 - val_accuracy: 0.7984 - val_f1: 0.5415
Epoch 8/10
8742/8742 [==============================] - 173s 20ms/sample - loss: 0.1202 - accuracy: 0.9467 - f1: 0.9460 - val_loss: 0.1447 - val_accuracy: 0.9614 - val_f1: 0.8450
Epoch 9/10
8742/8742 [==============================] - 173s 20ms/sample - loss: 0.1014 - accuracy: 0.9505 - f1: 0.9485 - val_loss: 0.2566 - val_accuracy: 0.8843 - val_f1: 0.6644
Epoch 10/10
8742/8742 [==============================] - 173s 20ms/sample - loss: 0.1023 - accuracy: 0.9534 - f1: 0.9521 - val_loss: 0.2381 - val_accuracy: 0.9425 - val_f1: 0.7665
<tensorflow.python.keras.callbacks.History at 0x7f603210b3c8>
```

Evaluating Model

We generate tumor probability heatmap to evaluate the model's predictions.

The heatmaps are shown along the original slide and mask images. (left to right: original slide, mask image, predicted heatmap).



Conclusions

We developed a deep learning framework for detecting cancerous cells in gigapixel images.

- **Architecture:**
  - Used Transfer Learning, based on InceptionV3 pretrained model.
- **Training:**
  - Upsampled to create balanced training sets.
- **Evaluation:**
  - generated tumor probability heatmaps that mimic the actual tumor masks.

Thank you

Video uploaded on the following link:

https://youtu.be/E159hjztAs4