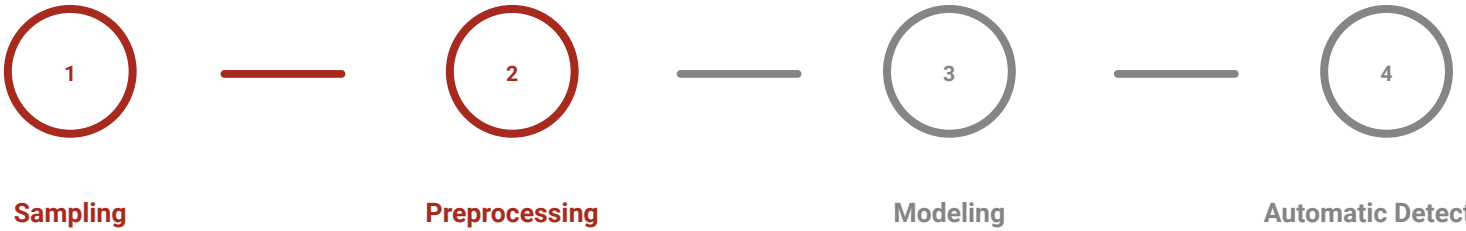
Applied Deep Learning - Final Project
Camelyon Challenge

Seung-jae Bang (sb2927) & Heinrich Peters (hp2500)

https://github.com/lazysjb/camelyon16

Workflow

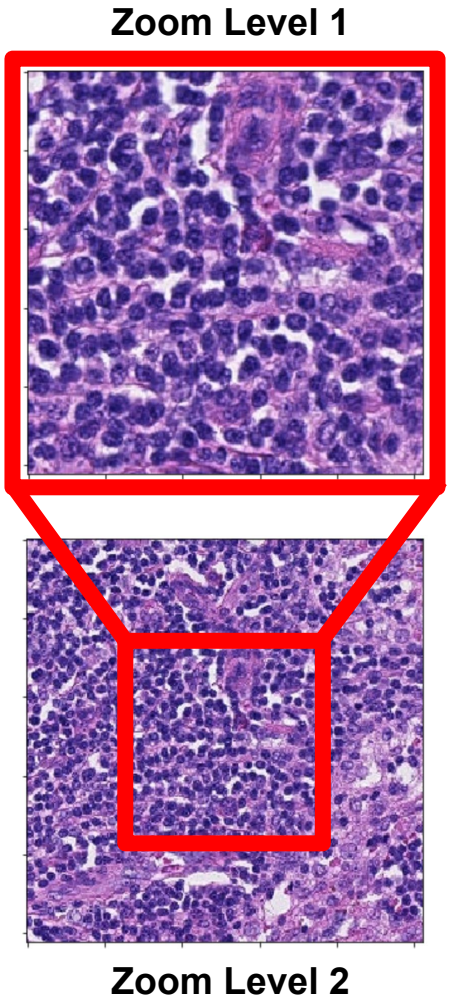

1
2
3
4
Sampling
Preprocessing
Modeling
Automatic Detection

Sampling & Preprocessing

- Train / val / test split at slide-level
  - Used 21 slides total → split to 13 / 4 / 4
  - Stratified sampling based on tumor area size (at zoom level 5) to mitigate sampling bias
- Slice images into patches
  - Patch size: 256 x 256
  - Zoom levels 1 & 2
  - Grayscale > 0.4
  - ROI (Region of Interest)
  - 51283 negative vs 7344 positive samples
- Random undersampling & oversampling
- Data augmentation: vertical & horizontal flip

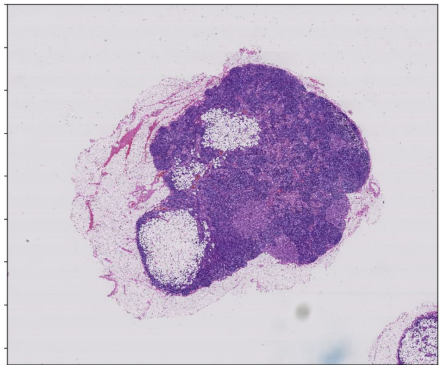
Image Partition

- Use multiple zoom levels as inputs
  - Focus: Patch on zoom level 1
  - Context: Patch on zoom level 2
  - We will use overlapping patches so that the level 1 patch is centered in the level 2 patch
  - This way we can link information from both zoom levels
- This requires the functional API and a multiple input model

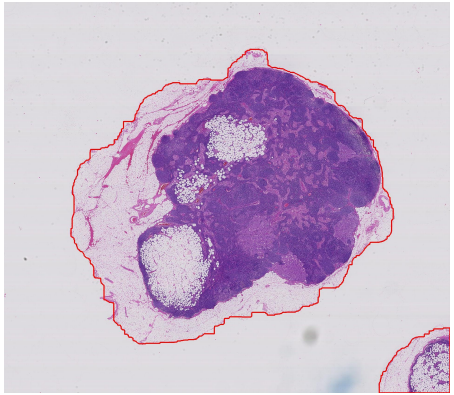


Region of Interest

```
hsv = cv2.cvtColor(slide_img, cv2.COLOR_BGR2HSV)
lower_red = np.array([30, 30, 30])
upper_red = np.array([200, 200, 200])
mask = cv2.inRange(hsv, lower_red, upper_red)
```



```
close_kernel = np.ones((50, 50), dtype=np.uint8)
image_close = Image.fromarray(cv2.morphologyEx(np.array(mask),
                cv2.MORPH_CLOSE, close_kernel))

open_kernel = np.ones((30, 30), dtype=np.uint8)
image_open =
Image.fromarray(cv2.morphologyEx(np.array(image_close),
                cv2.MORPH_OPEN, open_kernel))
```



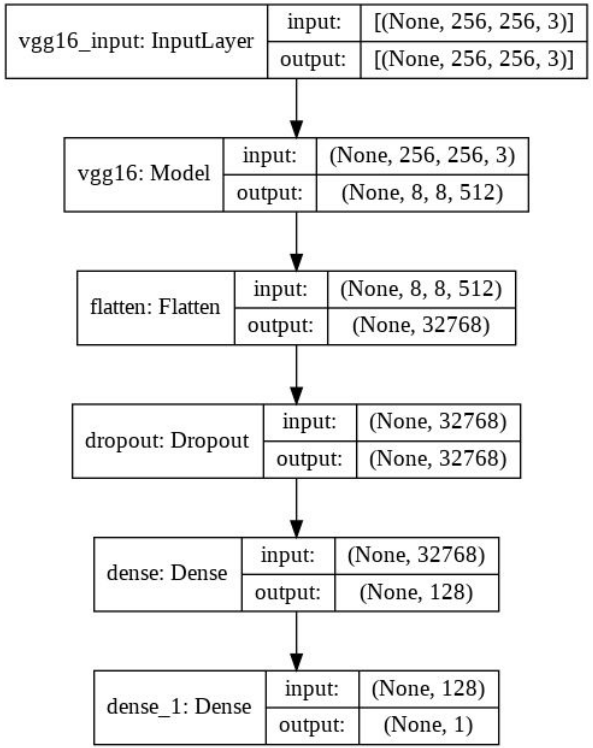
Reference: https://github.com/arjunvekariyagithub/camelyon16-grand-challenge

Modeling

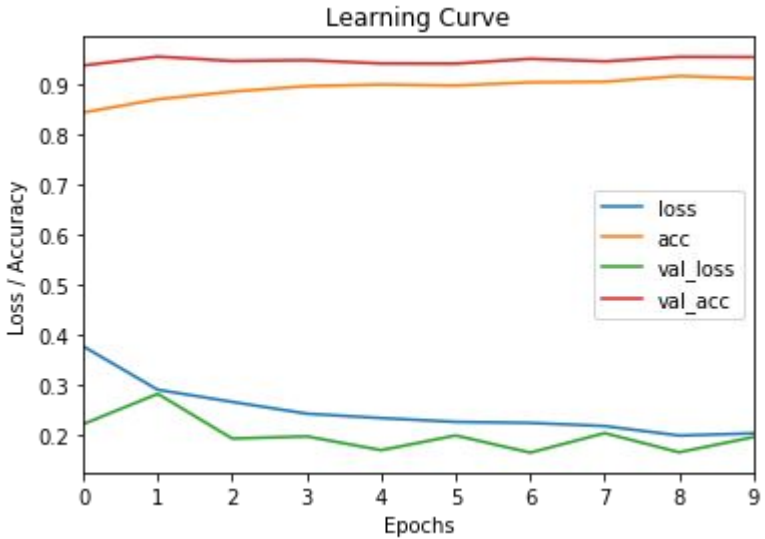
- Naive baseline (predict negative class by default)
- Small custom CNNs (e.g. 6 conv layers)
- Large custom CNNs (up to 50 conv layers)
  - Added batch normalization
  - Added residual layers
- Transfer Learning (e.g. VGG16 with imagenet weights)
- Double input model with custom CNN architecture
- Double input model with pre-trained VGGs

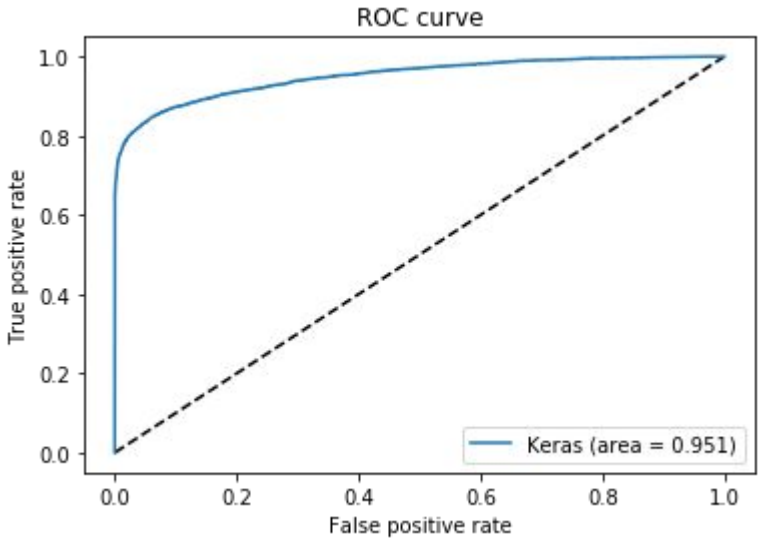
Transfer Learning with VGG16 - Code & Architecture

```
# Single input VGG16 transfer learn
vgg_base = VGG16(weights='imagenet',
                 include_top=False,
                 input_shape=(256, 256, 3)

model = models.Sequential()

model.add(vgg_base)
model.add(Flatten())
model.add(Dropout(rate=0.4))
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

vgg_base.trainable = False

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['acc'])
```



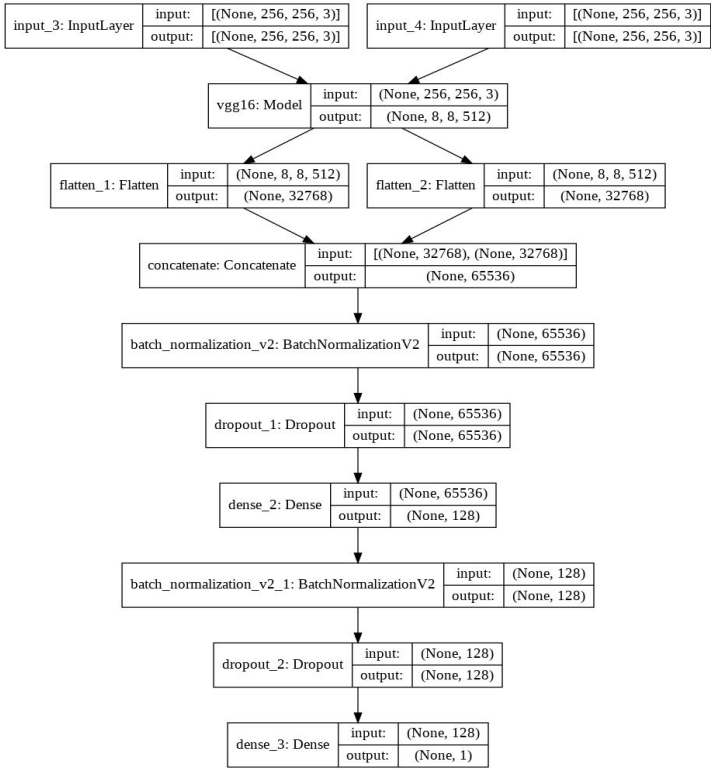
Transfer Learning with VGG16 - Results



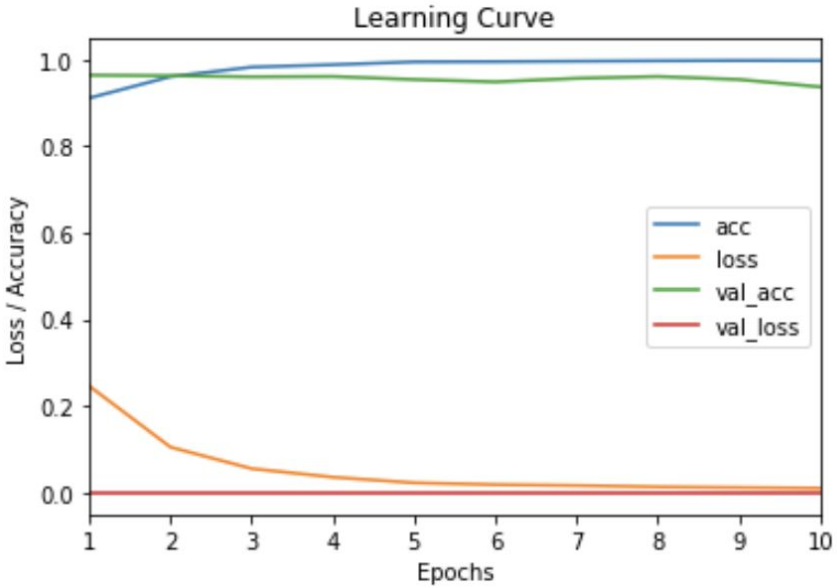


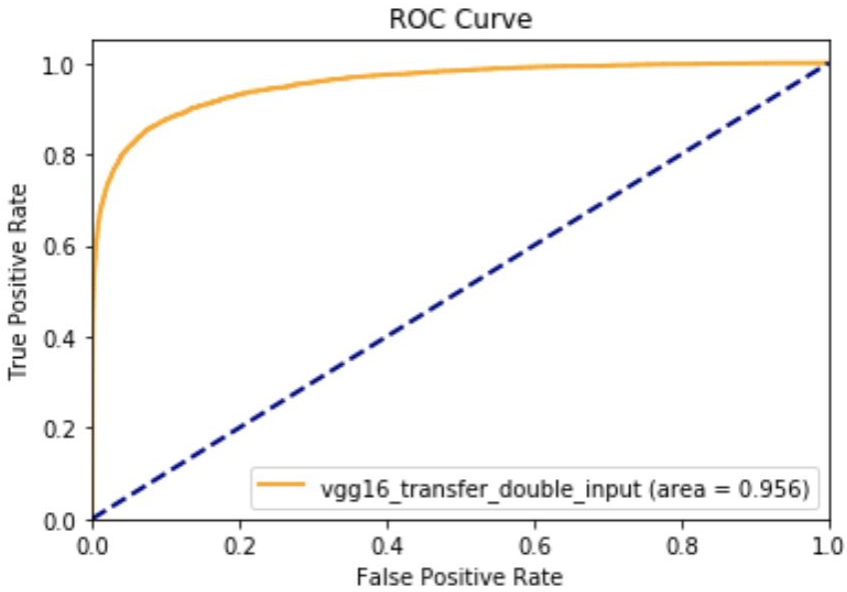
Double Input Transfer Learning - Code & Architecture

```
# Double input VGG16 transfer learn
vgg_base = VGG16(weights='imagenet',
                 include_top=False,
                 input_shape=input_shape)

# define two sets of inputs
zoom_1 = Input(shape=input_shape)
zoom_2 = Input(shape=input_shape)

# process zoom level 1 patch
conv_1 = vgg_base(zoom_1)
flatten_1 = Flatten()(conv_1)

# process zoom level 2 patch
conv_2 = vgg_base(zoom_2)
flatten_2 = Flatten()(conv_2)

# combine output of convolutional layers
combined = concatenate([flatten_1, flatten_2])
combined = BatchNormalization()(combined)
combined = Dropout(rate=0.4)(combined)

# fully connected layer after combined outputs
z = Dense(128, activation="relu")(combined)
z = BatchNormalization()(z)
z = Dropout(rate=0.4)(z)
z = Dense(1, activation="sigmoid")(z)

vgg_base.trainable = False

model = Model(inputs=[zoom_1, zoom_2], outputs=z)
```



Double Input Transfer Learning - Results

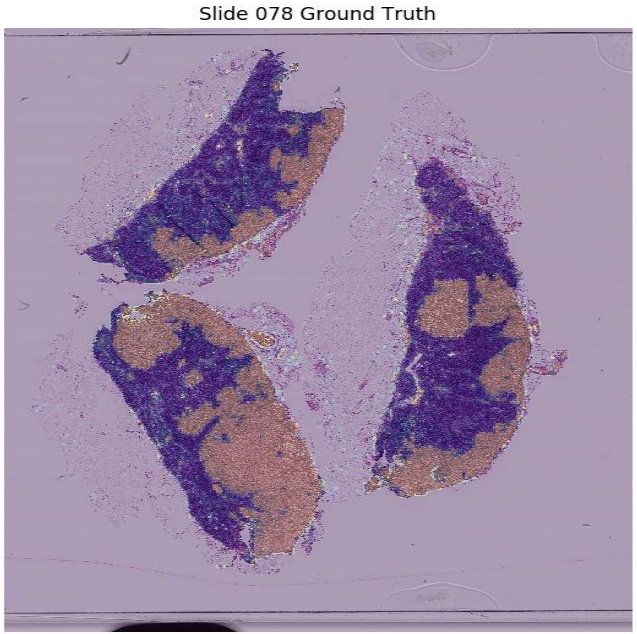




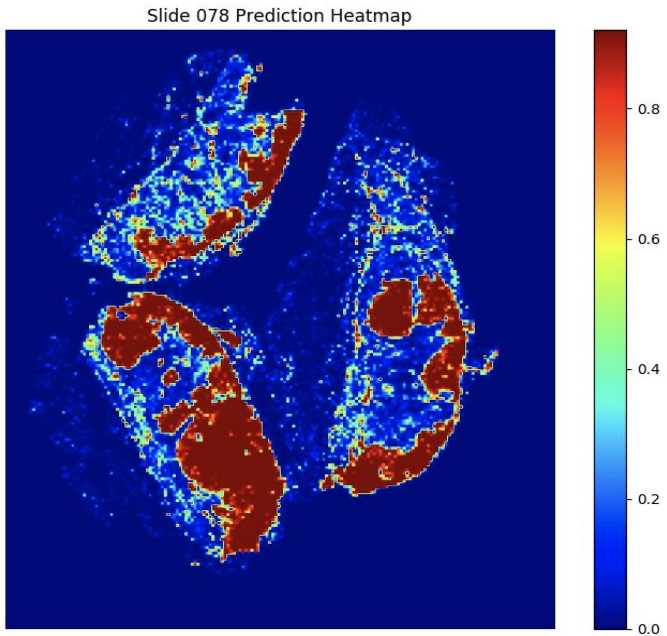
Summary of the Results (Test-Set)

	AUC	Accuracy	LogLoss	Precision	Recall
Single Input VGG16 Transfer Learn	0.951	0.907	0.253	0.798	0.863
Double Input VGG16 Transfer Learn	0.956	0.908	0.26	0.81	0.848
Naive model always predicting negative class	0.5	0.739			

Automatic Detection of Tumor Tissue



Original slide with tumor tissue



Automatic detection of tumor tissue (on test set)

Conclusions

- Final model performance significantly better than naive baseline
- Small CNNs were not very successful
- Transfer learning with imagenet weights helped
- Further improvements with multi-input-model
- Visualization is surprisingly accurate!