ADL Final Project Presentation

Qi Feng (qf2140)

Goal

- Given a gigapixel pathology image (slide), the goal is to classify if the image contains tumor and localize the tumors for a pathologist's review.

Preprocessing

- Data loading
- Patch extraction
- Train-test-split
- Class Imbalance

Data Loading

- Sanity check
  - Number of slides == Number of tumor masks
    - Missing mask for ['/content/drive/My Drive/adl/slides/tumor_038.tif']
    - Remedy: drop slide/mask with no match.
    - Total slides: 21
  - Slide dimensions == Tumor mask dimensions
    - Minimum dimensions: 8 (level 0-7)

Patch Extraction

- Parameters used for sliding-window
  - `LEVELS = [3, 4]`
  - `PATCH_SIZE = 300`
  - `STRIDE = 128`
- Train-test-split
  - Randomly select two slides for testing: `['tumor_075', 'tumor_084']`
  - Draw **overlapping** patches for the **training** set (`stride = 128`).
  - Draw **non-overlapping** patches for the **test** set (`stride = 300`).

Method

- For all levels, iterate the center (x, y) of the patches
  - Coordinates are multiplied by scales
- Pad at higher levels when (x, y) is out of border (i.e., the maximum dimensions of the lowest level)
- Skip the non-tissue areas
  - Threshold = 40%
  - Record the indices to take labels/skip patches in the next level accordingly

```
def _extract_patch(slide_img, max_x, max_y, check, dropped,
                   scale=1, patch_size=PATCH_SIZE, stride=STRIDE):
  """
  Helper function to extract patch at a level.
  """
  max_x_cur = max_x * scale  # Max dim for current level
  max_y_cur = max_y * scale
  # stride *= scale
  half_patch = patch_size // 2
  patches = []

  idx = 0  # used to track dropped patch index
  # Iterate the center of the patch
  for _, y in enumerate(np.arange(half_patch, max_y - half_patch, stride)):
    y *= scale
    for _, x in enumerate(np.arange(half_patch, max_x - half_patch, stride)):
      x *= scale

      if not check:
        if idx in dropped:
          idx += 1
          continue
        else:
          idx += 1

      # Padding will only be executed at higher levels
      # Pad when out of border
      y_top = int(max(0, y - half_patch))
      y_bottom = int(min(max_y_cur, y + half_patch))
      x_left = int(max(0, x - half_patch))
      x_right = int(min(max_x_cur, x + half_patch))

      patch = slide_img[y_top : y_bottom, x_left : x_right, :]

      if patch.shape[0] != patch_size:
        if y_top == 0:
          patch = pad_along_axis(patch, half_patch - y, 0)
        if y_bottom == max_y_cur:
          patch = pad_along_axis(patch, half_patch - (max_y_cur - y), 0)
      if patch.shape[1] != patch_size:
        if x_left == 0:
          patch = pad_along_axis(patch, half_patch - x, 1)
        if x_right == max_x_cur:
          patch = pad_along_axis(patch, half_patch - (max_x_cur - x), 1)

      if check: # if at min level, check and skip non-tissue areas
        tissue_pixels = find_tissue_pixels(patch)
        percent_tissue = len(tissue_pixels) / float(patch.shape[0] * patch.shape[1]) * 100
        if percent_tissue < 40:
          dropped.append(idx)
          idx += 1
          continue
        idx += 1

      assert patch.shape == (PATCH_SIZE, PATCH_SIZE, 3)
      patches.append(patch)

  return patches, dropped
```

Labelling

- Draw patches correspondingly on the **mask image**
- For each input patch, we predict the label of the center 128×128 region.
- We label a patch as **tumor ('1')** if at least one pixel in the center region is annotated as tumor
  - Otherwise, **Non-tumor ('0')**

```python
def crop_center(img, crop_x=128, crop_y=128):
    """
    Crop the center area for an image.
    """
    y, x, _ = img.shape
    start_x = x // 2 - (crop_x // 2)
    start_y = y // 2 - (crop_y // 2)
    return img[start_y:start_y + crop_y, start_x:start_x + crop_x, :]
```

```python
def _extract_patch_labels(mask_img, max_x, max_y, dropped,
                          scale=1, patch_size=PATCH_SIZE, stride=STRIDE):
    """
    Helper function to extract patch labels at a level.
    """
    max_x_cur = max_x * scale  # Max dim for current level
    max_y_cur = max_y * scale
    # stride *= scale
    half_patch = patch_size // 2

    labels = []

    idx = 0  # used to track dropped patch index
    for _, y in enumerate(np.arange(half_patch, max_y - half_patch, stride)):
        y *= scale
        for _, x in enumerate(np.arange(half_patch, max_x - half_patch, stride)):
            x *= scale
            if idx in dropped:
                idx += 1
                continue
            else:
                idx += 1

            # Padding will only be executed at higher levels
            # Pad when out of border
            y_top = int(max(0, y - half_patch))
            y_bottom = int(min(max_y, y + half_patch))
            x_left = int(max(0, x - half_patch))
            x_right = int(min(max_x, x + half_patch))

            patch = mask_img[y_top : y_bottom, x_left : x_right, :]

            if patch.shape[0] != patch_size:
                if y_top == 0:
                    patch = pad_along_axis(patch, half_patch - y, 0)
                if y_bottom == max_y_cur:
                    patch = pad_along_axis(patch, half_patch - (max_y_cur - y), 0)
            if patch.shape[1] != patch_size:
                if x_left == 0:
                    patch = pad_along_axis(patch, half_patch - x, 1)
                if x_right == max_x_cur:
                    patch = pad_along_axis(patch, half_patch - (max_x_cur - x), 1)

            # Crop the center 128 region to label
            patch = crop_center(patch)

            lab = 0 if np.all(patch == 0) else 1
            labels.append(lab)

    return labels
```

Loop Over Levels At All Slides

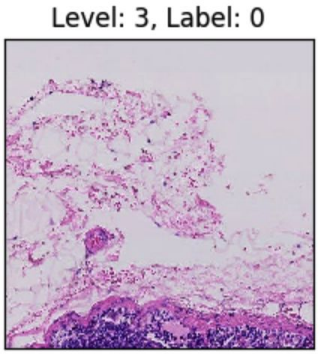
For all slides/tumor masks

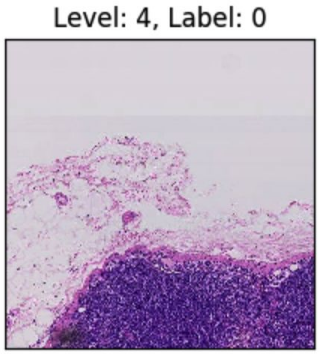
21 Slides
21 Tumor masks

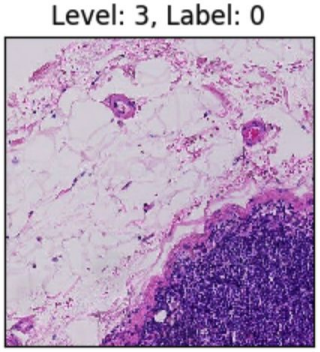
For all levels

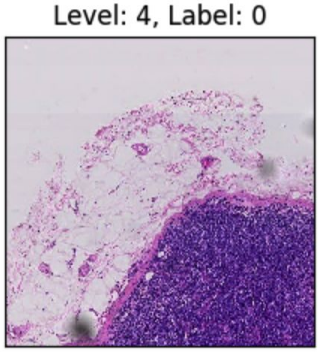
Level 3
Extract patches (filter out non-tissue, return indices)
→ extract labels (center 128 region)

Level 4
Extract (context) patches (matching level 3, pad if out of border)
→ extract labels (center 128 region)

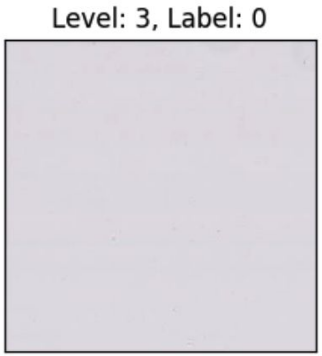


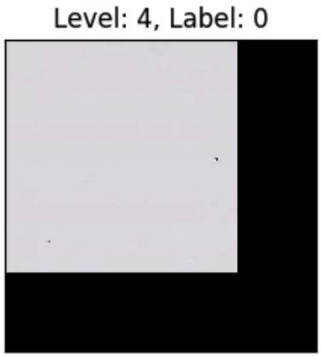


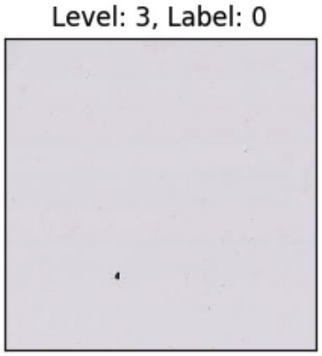


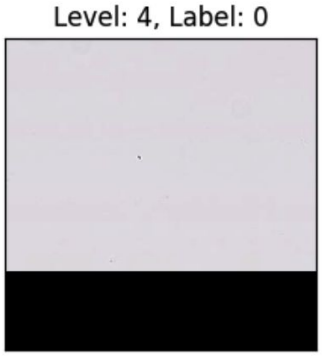


Neighboring training patches (overlapped)

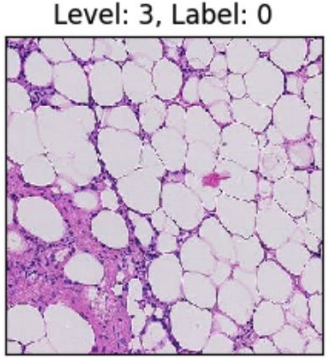


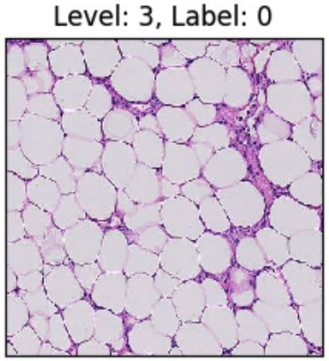






Neighboring training patches, padded (overlapped)

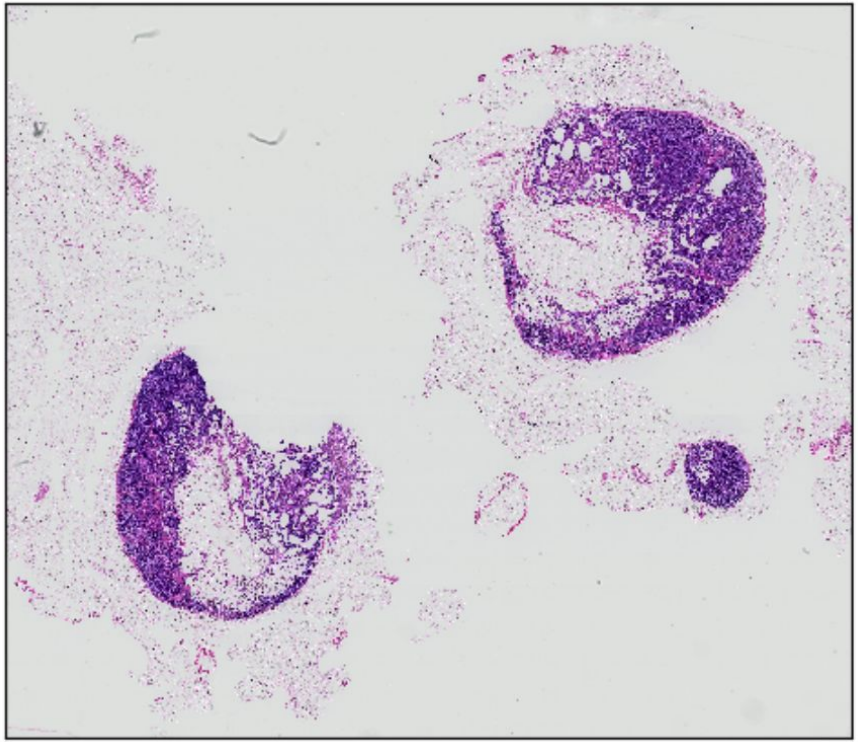




Neighboring test patches, (non-overlapped)

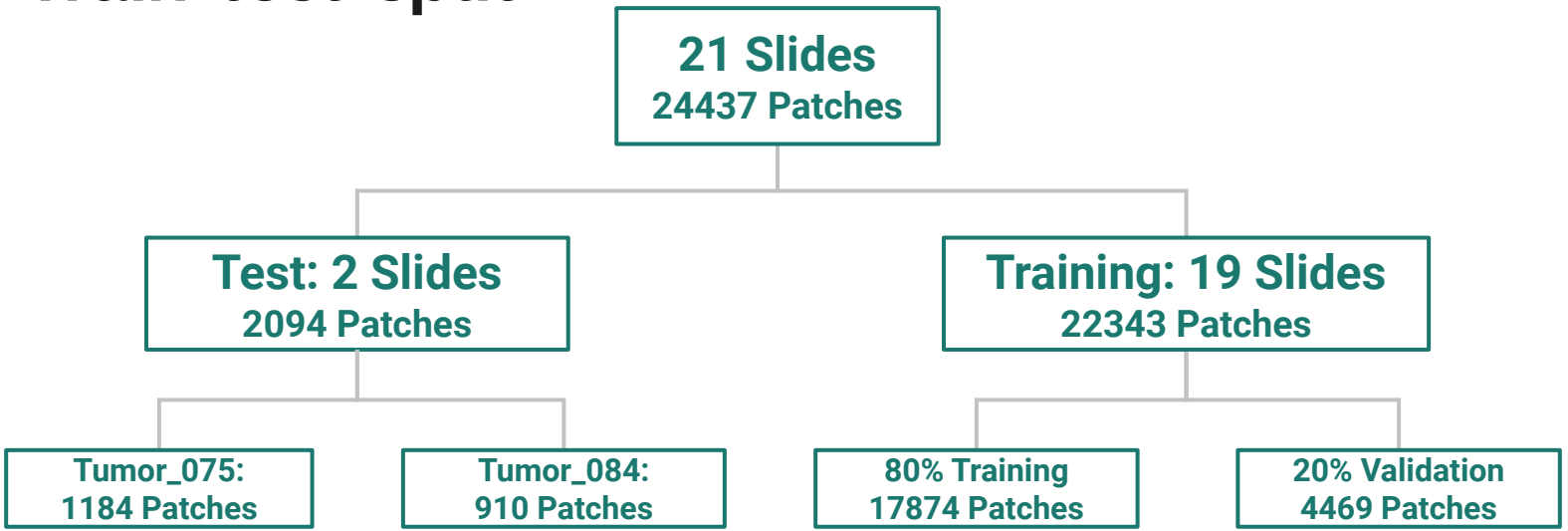
```python
def get_n_patch(slide_path, tumor_mask_path, level):
  slide = open_slide(slide_path)
  tumor_mask = open_slide(tumor_mask_path)

  # number of patches per col
  n_patch_h = slide.level_dimensions[level][0] // PATCH_SIZE
  # number of patches per row
  n_patch_v = slide.level_dimensions[level][1] // PATCH_SIZE

  return (n_patch_h, n_patch_v)
```

```python
import matplotlib.image as mpimg

def concat_patches(patch_path, n_patch_h, n_patch_v, level):
  # Convert path into np array
  img = [mpimg.imread(i) for i in patch_path[level]]

  # 1. Stack patches horizontally
  partition = np.array_split(img, n_patch_v)
  rows = [np.hstack(p) for p in partition]
  # 2. Stack rows of patches vertically
  res = np.vstack(rows)

  return res
```



Sanity checks for patch extraction, tumor_075.tif

Train-test-split



Class Imbalance

- Tumor patches are rare
- Solution: class weights


```
from sklearn.utils import class_weight

class_weights = class_weight.compute_class_weight('balanced',
                                                  [0, 1],
                                                  all_y_train[3])
class_weights = dict(enumerate(class_weights))
class_weights
```

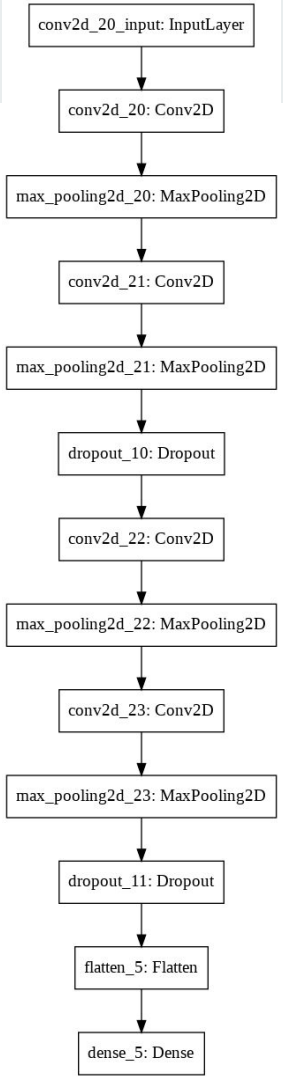
```
{0: 0.6024266936299292, 1: 2.9407699901283317}
```

Modeling

- Single scale, simple CNN
- Single scale, one Inception tower
- Multi-scales, two CNNs

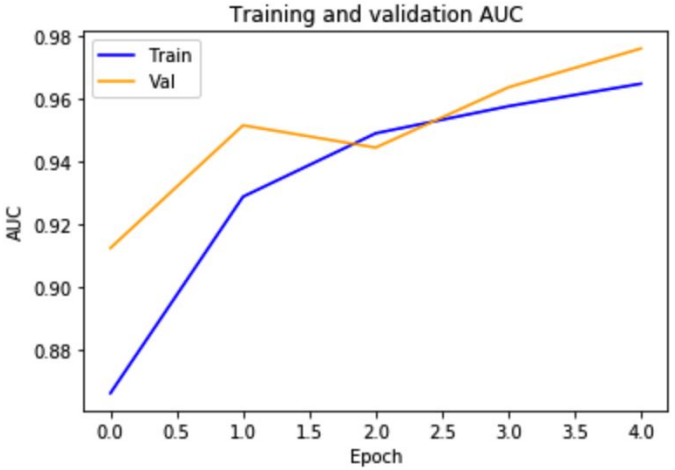
Single Scale, Simple CNN

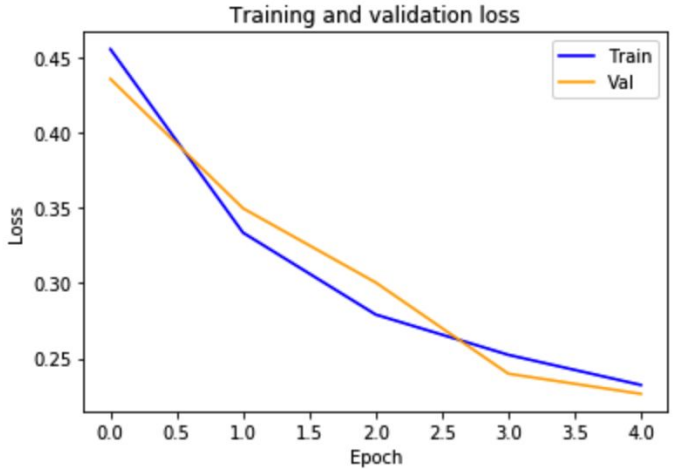
```
def make_model():
  model = Sequential()
  model.add(Conv2D(32, (3, 3), activation='relu',
                   input_shape=(299, 299, 3)))
  model.add(MaxPooling2D())

  model.add(Conv2D(32, (3, 3), activation='relu'))
  model.add(MaxPooling2D())

  model.add(Dropout(0.5))

  model.add(Conv2D(64, (3, 3), activation='relu'))
  model.add(MaxPooling2D())

  model.add(Conv2D(64, (3, 3), activation='relu'))
  model.add(MaxPooling2D())

  model.add(Dropout(0.5))

  model.add(Flatten())

  # Binary classifier
  model.add(Dense(1, activation='sigmoid'))

  return model
```

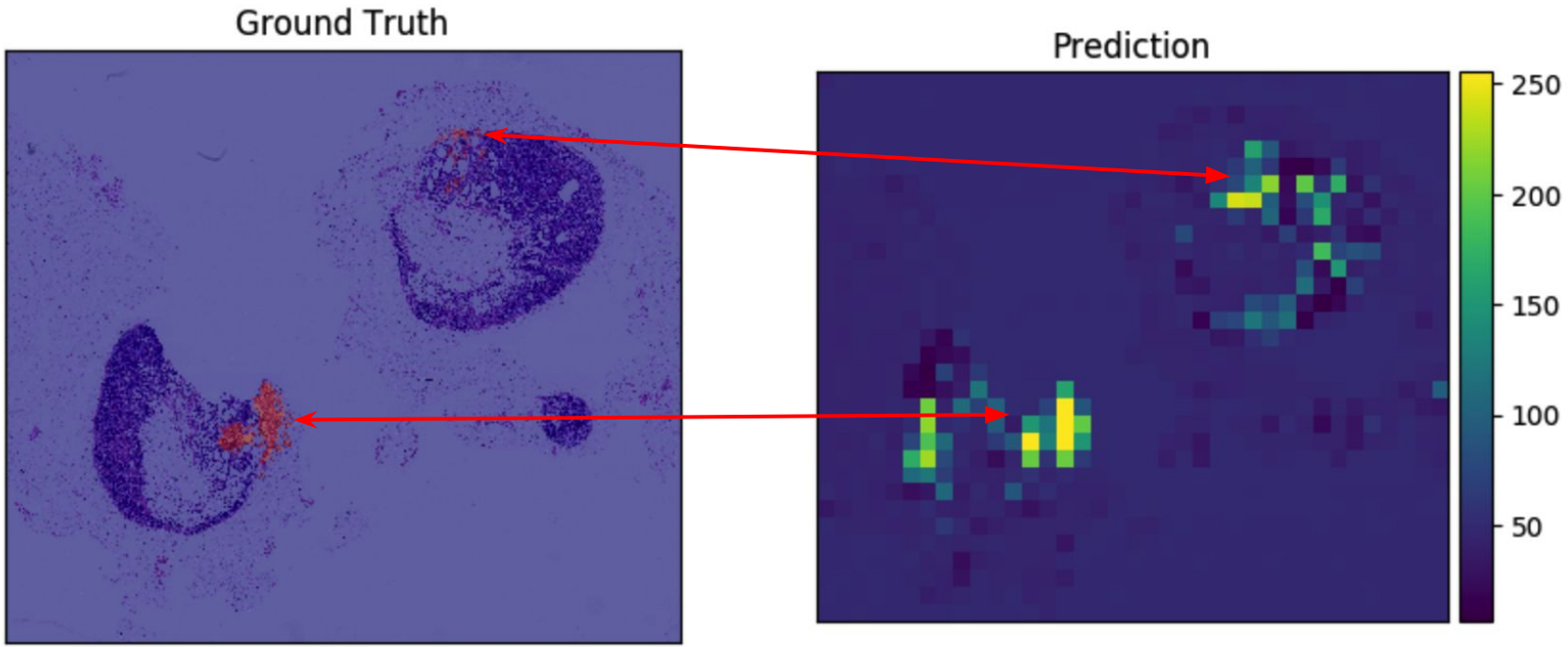


ROC-AUC & Loss

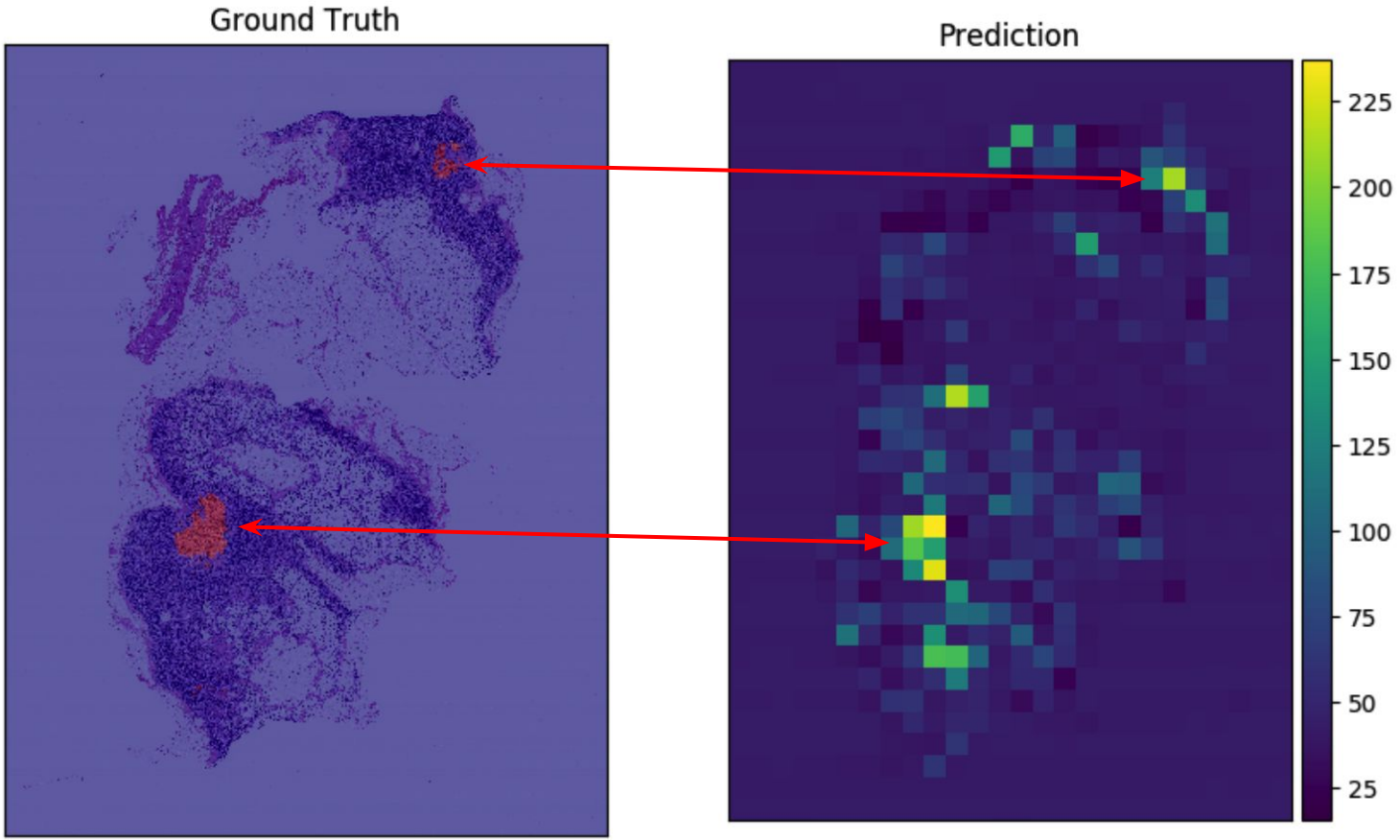
- ROC-AUC was used for imbalanced dataset
- Validation accuracy is higher/loss is lower sometimes
  - Regularization is applied during training but not validation
  - Training loss is measured during each epoch (thus is averaged over the epoch) while validation loss is measured after each epoch (thus is computed only once)
  - The validation set may be easier and there may also be leaks.
- Validation is decreasing at the last epoch
  - Not overfitting







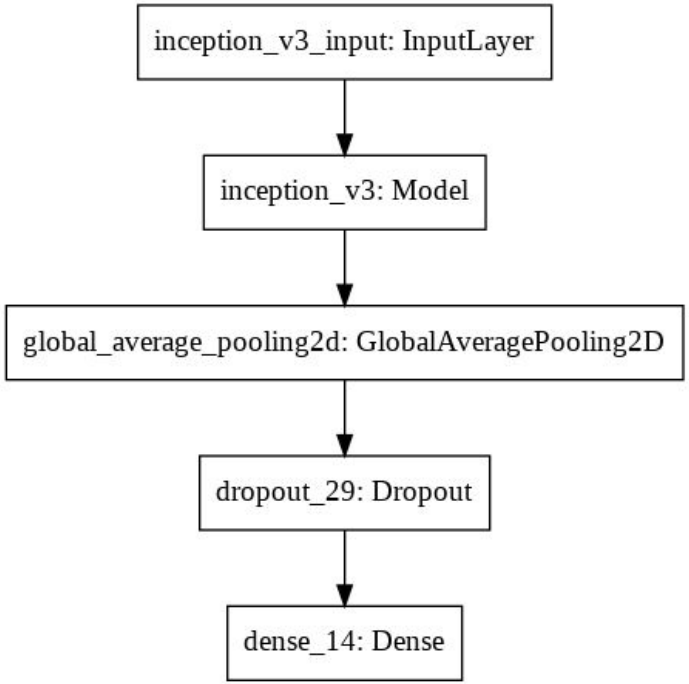
Prediction: tumor_075.tif



Prediction: tumor_084.tif

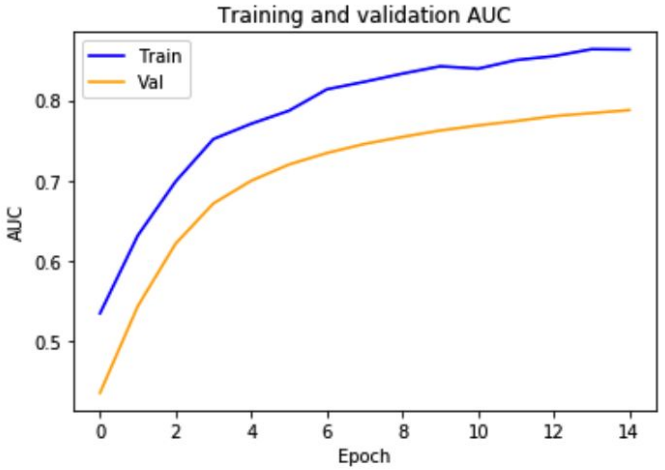
One-scale, InceptionV3

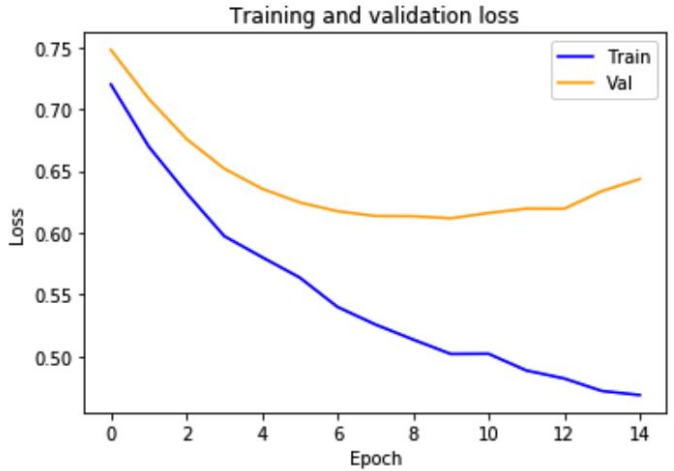
```
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.layers import GlobalAveragePooling2D

def make_model_inception():
  model = Sequential()
  base_model = InceptionV3(include_top=False,
                           weights='imagenet',
                           input_shape=(299, 299, 3))

  # Do not update the pretrained weights during training
  base_model.trainable = False

  model.add(base_model)
  model.add(GlobalAveragePooling2D())

  # model.add(Dense(32, activation='relu'))
  model.add(Dropout(0.5))

  # Binary classifier
  model.add(Dense(1, activation='sigmoid'))

  return model
```

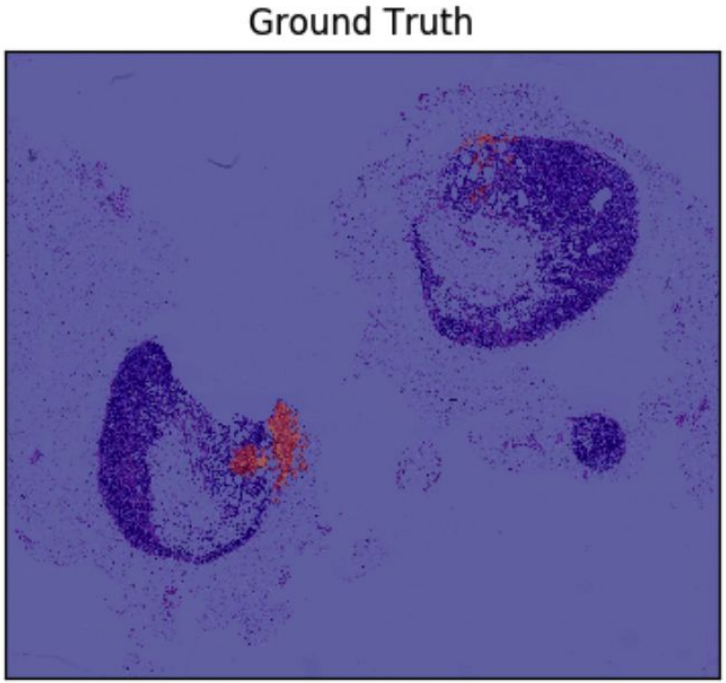


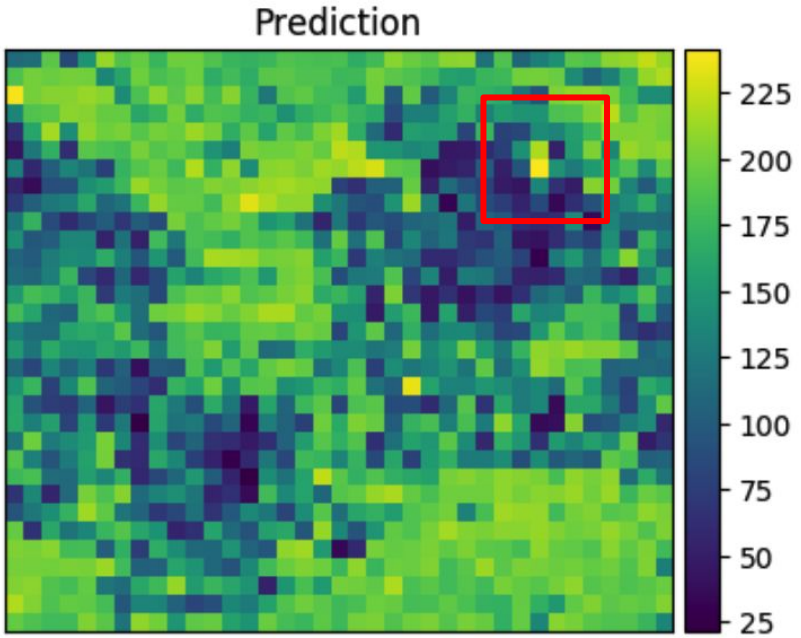
ROC-AUC & Loss

- Epoch = 15
- Validation accuracy is always lower
- Validation increasing at epoch = 7
    - Suggesting early stop

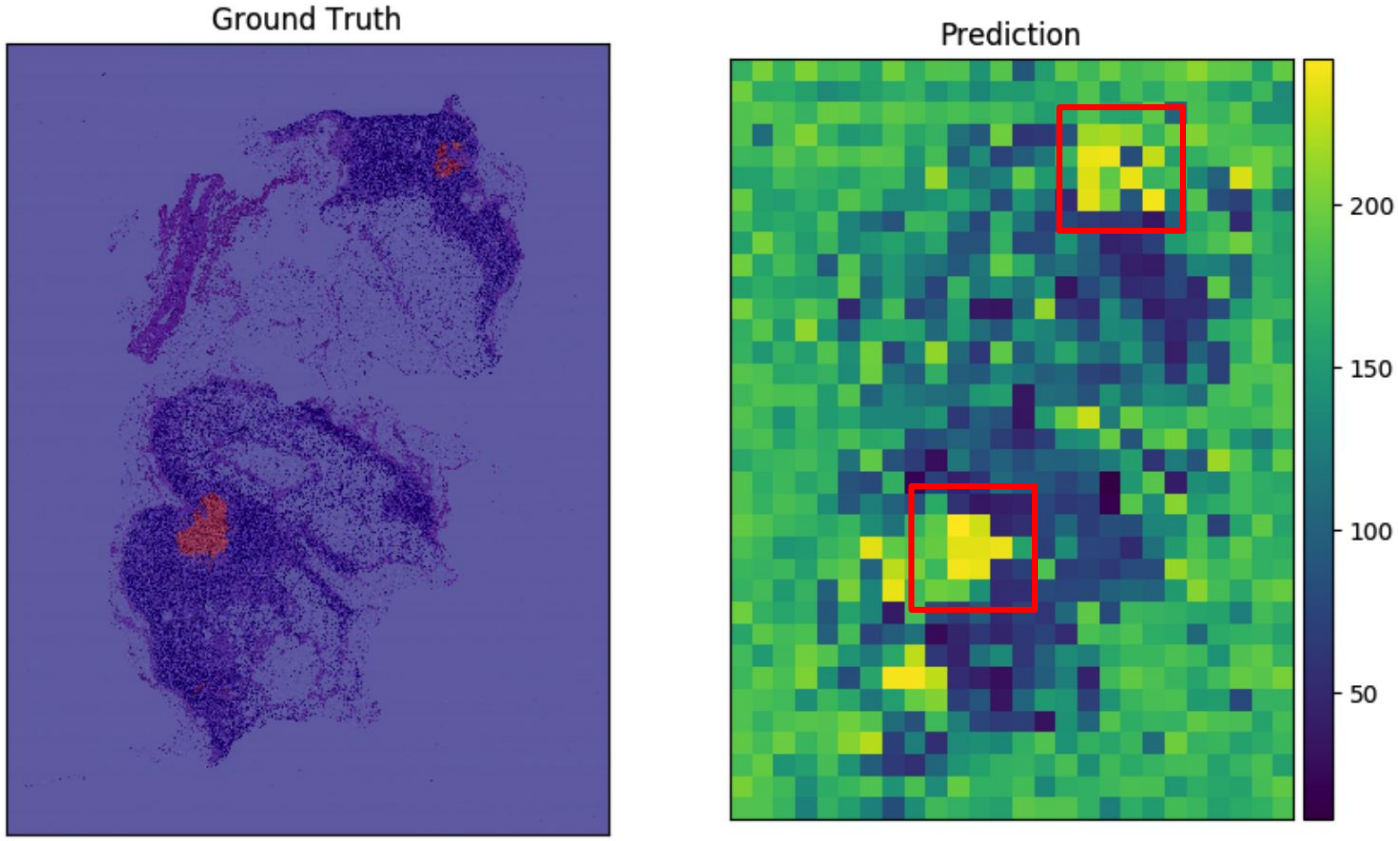








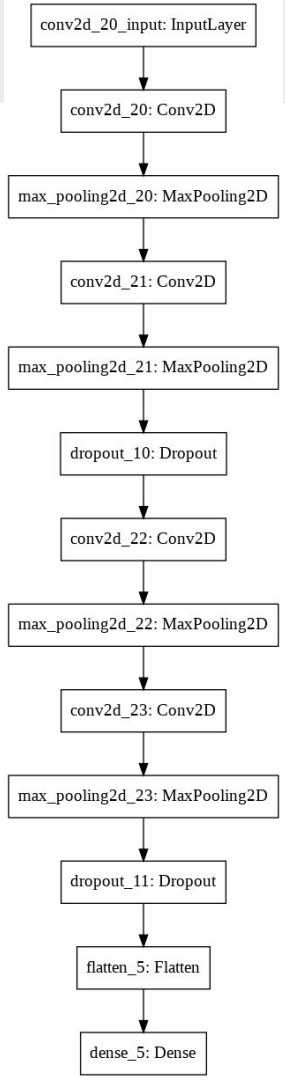
Prediction: tumor_075.tif



Prediction: tumor_084.tif

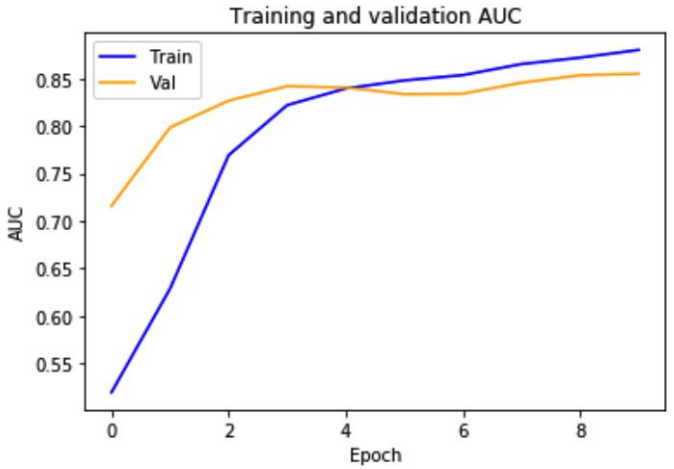
Multi-scale CNNs

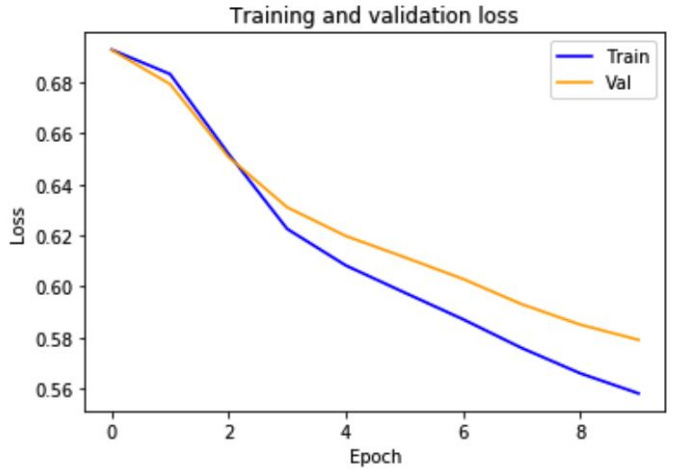
```python
from tensorflow.keras.layers import Input, concatenate

def make_model_multi():
  input1 = Input(shape=(299, 299, 3))
  input2 = Input(shape=(299, 299, 3))

  # The first branch operates on the first input: level 3
  x = Conv2D(32, (3, 3), activation='relu')(input1)
  x = MaxPooling2D()(x)
  x = Conv2D(32, (3, 3), activation='relu')(x)
  x = MaxPooling2D()(x)
  x = Dropout(0.5)(x)

  x = Conv2D(64, (3, 3), activation='relu')(x)
  x = MaxPooling2D()(x)
  x = Conv2D(64, (3, 3), activation='relu')(x)
  x = MaxPooling2D()(x)
  x = Dropout(0.5)(x)
  x = Flatten()(x)

  x = Model(inputs=input1, outputs=x)

  # The second branch opreates on the second input: level 4
  y = Conv2D(32, (3, 3), activation='relu')(input2)
  y = MaxPooling2D()(y)
  y = Conv2D(32, (3, 3), activation='relu')(y)
  y = MaxPooling2D()(y)
  y = Dropout(0.5)(y)

  y = Conv2D(64, (3, 3), activation='relu')(y)
  y = MaxPooling2D()(y)
  y = Conv2D(64, (3, 3), activation='relu')(y)
  y = MaxPooling2D()(y)
  y = Dropout(0.5)(y)
  y = Flatten()(y)

  y = Model(inputs=input2, outputs=y)

  # Combine the output of the two branches
  combined = concatenate([x.output, y.output])

  # Apply a FC layer and then a binary classifier
  z = Dense(2, activation='relu')(combined)

  z = Dense(1, activation='sigmoid')(z)

  model = Model(inputs=[x.input, y.input], outputs=z)

  return model
```

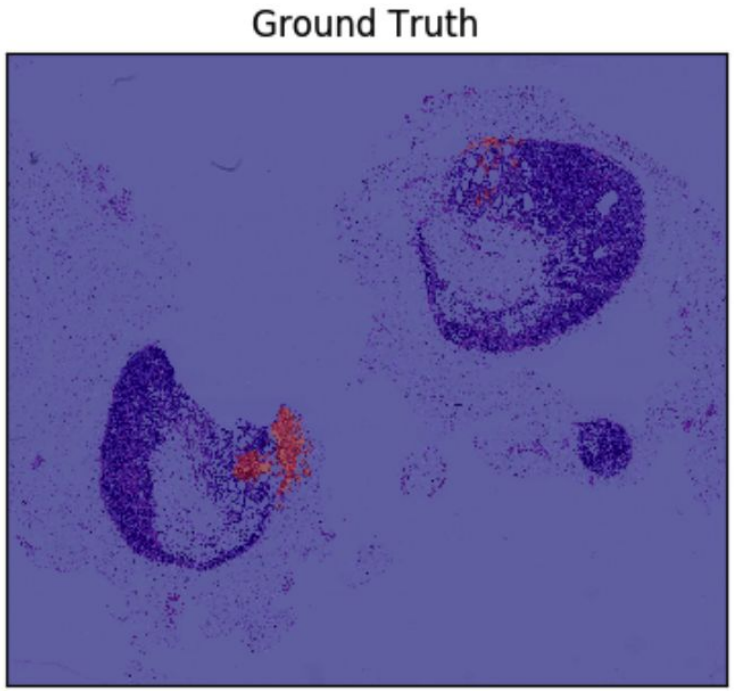


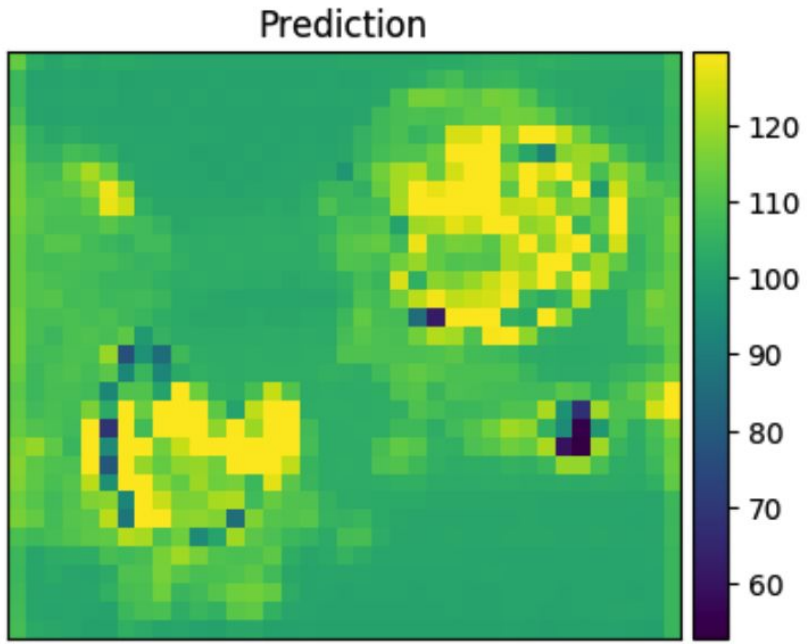
ROC-AUC & Loss

- Epoch = 10
- Validation accuracy is higher/loss is lower at the start, but lower/loss is higher later
- Validation is decreasing at the last epoch
    - Not overfitting

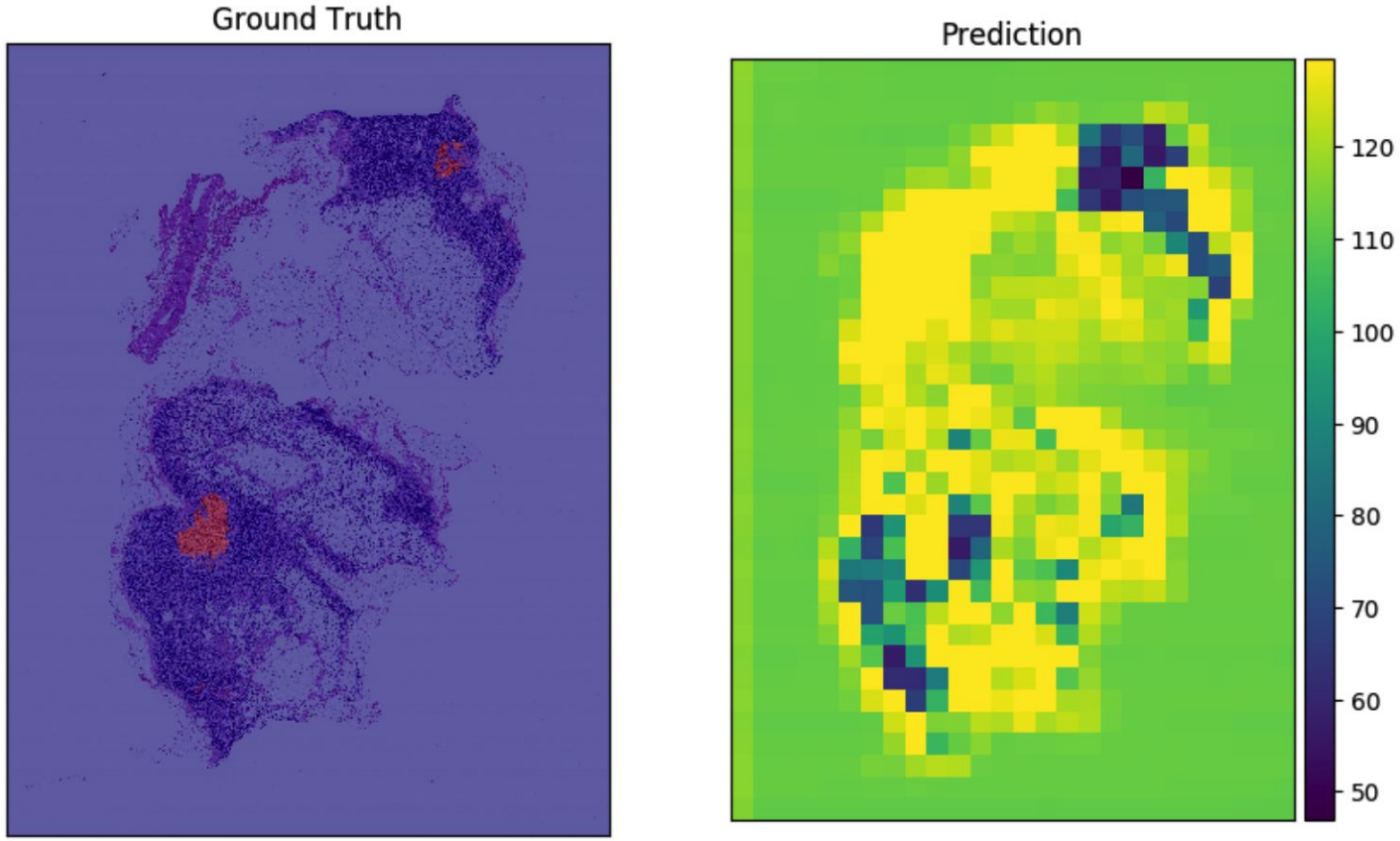








Prediction: tumor_075.tif



Prediction: tumor_084.tif

Conclusions

- The simple CNN has the best performance
  - InceptionV3 is the worst
  - Overfitting (small training set)
- Context patches are not so helpful
- Limitations
  - InceptionV3 fine tuning
  - Data augmentation
  - Larger scales, e.g., level 2