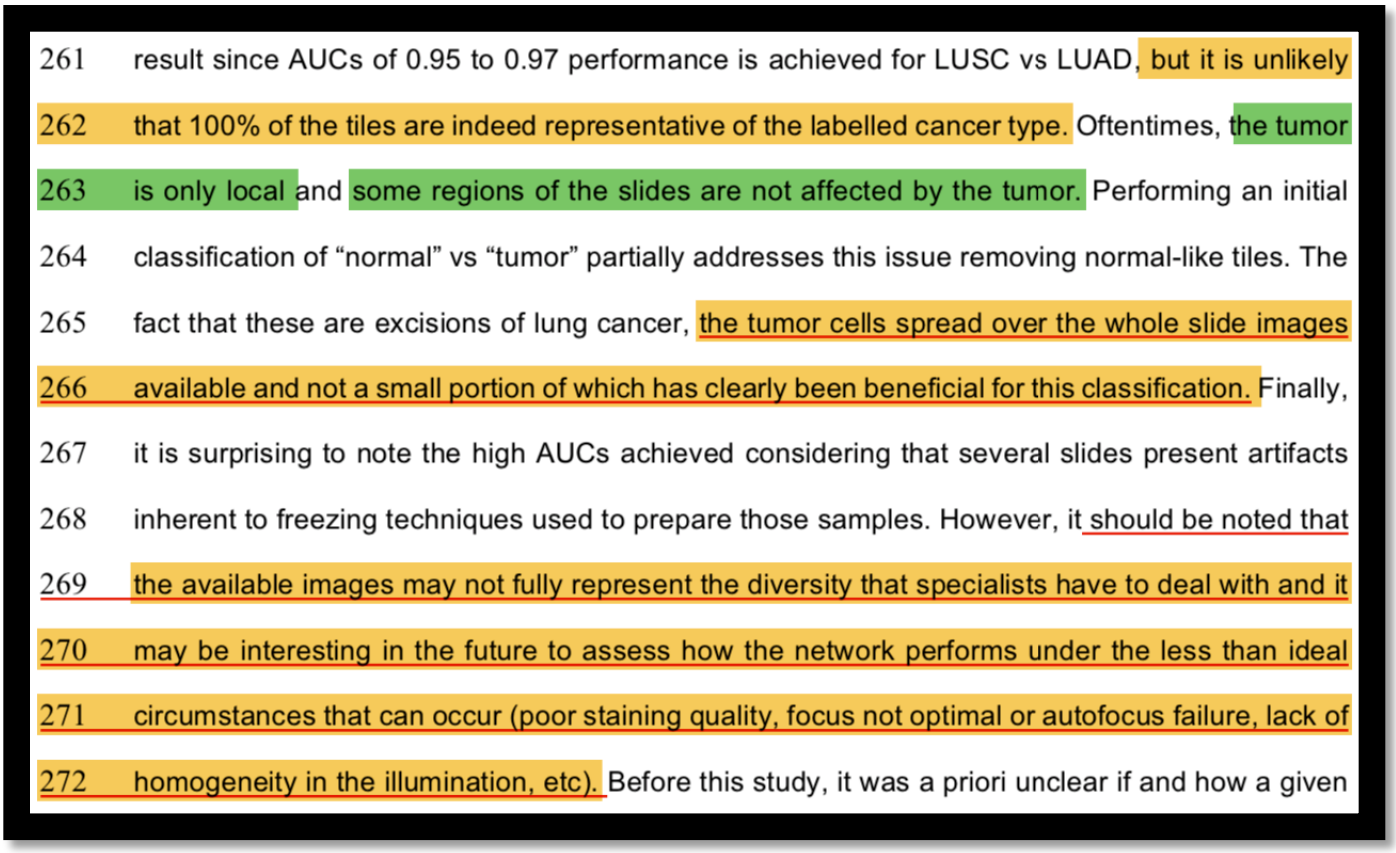
Predicting gene mutational status from whole slide images.
Feb28th/2020

- The result from rerunning the model by changing the preprocessing parameters:
  - Not much change for micro and macro AUC/ROC values.
  - Still have NA values for each Class files.
  - Reasons:
    - Computational:
      ROC/AUC calculation script in:
      </nfs/home/xwang/DeepPATH/DeepPATH_code/03_postprocessing/0h_ROC_MultiOutput_BootStrap.py>

      ⇨ Is not working / not generalize well for our datasets, this may due to fact that differences in image generations. Somehow this script does not have generalizability for our image dataset from Providence.

    - Biological:
      ⇨ The model trained on TCGA datasets and tested on their own testing set for different biopsy images. They may figure out the same preprocess step for images they trained on. For example, their images may obtain from similar lung cancer stage or microenvironment.
      ⇨ Biologically, the staining process on these images may be different and the cancer stages for these images may be different in clinical definition that causing bad classifying results.



result since AUCs of 0.95 to 0.97 performance is achieved for LUSC vs LUAD, but it is unlikely that 100% of the tiles are indeed representative of the labelled cancer type. Oftentimes, the tumor is only local and some regions of the slides are not affected by the tumor. Performing an initial classification of "normal" vs "tumor" partially addresses this issue removing normal-like tiles. The fact that these are excisions of lung cancer, the tumor cells spread over the whole slide images available and not a small portion of which has clearly been beneficial for this classification. Finally, it is surprising to note the high AUCs achieved considering that several slides present artifacts inherent to freezing techniques used to prepare those samples. However, it should be noted that the available images may not fully represent the diversity that specialists have to deal with and it may be interesting in the future to assess how the network performs under the less than ideal circumstances that can occur (poor staining quality, focus not optimal or autofocus failure, lack of homogeneity in the illumination, etc). Before this study, it was a priori unclear if and how a given

- **Next Step:** Gene mutation prediction from image.
    - Only using LUAD tiles for gene mutation prediction to see what's the results.

From https://github.com/ncoudray/DeepPATH/tree/master/DeepPATH_code/example_TCGA_lung:

To process mutations of LUAD images, there are different ways to do it.

First, to extract probability of LUAD tiles on all LUAD tiles, we'll run them through the above classifier: Sort the tiles, assigning them all to "test".

"""
mkdir r2_LUAD_segmentation
cd r2_LUAD_segmentation
python 00_preprocessing/0d_SortTiles.py --SourceFolder='../512px_Tiled/' --Magnification=20.0 --MagDiffAllowed=0 --SortingOption=3 --PatientID=12 --nSplit 0 --JsonFile='../downloaded_data/metadata.cart.2017-03-02T00_36_30.276824.json' --PercentTest=100 --PercentValid=0
"""

Since Normal and LUSC do not interest us, delete their content (the content only - not the folder - the number of folders in that directory is used to identify the total number of possible classes):

"""
rm -rf TCGA-lUSC/*
rm -rf Solid_Tissue_Normal/*
"""

convert to TFRecord:

"""
mkdir r2_TFRecord_test

python 00_preprocessing/TFRecord_2or3_Classes/build_TF_test.py --directory='r2_LUAD_segmentation/' --output_directory='r2_TFRecord_test' --num_threads=1 --one_FT_per_Tile=False --ImageSet_basename='test'
"""

Segment the LUAD tiles using the checkpoint giving the best validation/test AUC.

"""
export CHECKPOINT_PATH='r1_results'
export OUTPUT_DIR='r2_test'
export DATA_DIR='r2_TFRecord_test'
export LABEL_FILE='labelref_r1.txt'

Best checkpoints
declare -i count=69000
declare -i NbClasses=3

```
# create temporary directory for checkpoints
mkdir  -p $OUTPUT_DIR/tmp_checkpoints
export CUR_CHECKPOINT=$OUTPUT_DIR/tmp_checkpoints

export TEST_OUTPUT=$OUTPUT_DIR/test_$count'k'
mkdir -p $TEST_OUTPUT

ln -s $CHECKPOINT_PATH/*-$count.* $CUR_CHECKPOINT/.
touch $CUR_CHECKPOINT/checkpoint
echo 'model_checkpoint_path: "'$CUR_CHECKPOINT'/model.ckpt-'$count'"' > $CUR_CHECKPOINT/checkpoint
echo 'all_model_checkpoint_paths: "'$CUR_CHECKPOINT'/model.ckpt-'$count'"' >>
$CUR_CHECKPOINT/checkpoint

# Test
python 02_testing/xClasses/nc_imagenet_eval.py --checkpoint_dir=$CUR_CHECKPOINT --
eval_dir=$OUTPUT_DIR --data_dir=$DATA_DIR  --batch_size 300  --run_once --ImageSet_basename='test_' --
ClassNumber $NbClasses --mode='0_softmax'  --TVmode='test'
                                # wait

mv $OUTPUT_DIR/out* $TEST_OUTPUT/.
"""
```

Retrieve the mutation information from the GDC website. In this particular example, we use mutect2 "masked somatic mutations". We **label samples/slides as mutated with respect to every gene if** it had a non-silent mutation. We used maftools to parse the Mutect2 variants from TCGA which by default uses Variant Classifications with High/Moderate variant consequences. These include: "Frame_Shift_Del", "Frame_Shift_Ins", "Splice_Site", "Translation_Start_Site", "Nonsense_Mutation", "Nonstop_Mutation", "In_Frame_Del", "In_Frame_Ins", "Missense_Mutation". We then picked the top 10 "known cancer genes" (https://cancer.sanger.ac.uk/census) with respect to the number of (non-silent) mutation across our dataset, excluding genes like TNN which are known to be frequently mutated in general (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4267152/). **We can then generate a label file where the first column is the slide ID, and the second the mutation name (if a slide has several mutations, then it will have several lines - example in file attached labels_r3.txt), and a reference file with the list of possible mutations (see labelref_r3.txt).**

sort the LUAD tiles identified as LUAD intro a train, valid a test set for mutation analysis.

```
"""
mkdir r3_LUAD_sorted
cd r3_LUAD_sorted

python ../00_preprocessing/0d_SortTiles.py --SourceFolder='../512px_Tiled_NewPortal/'  --Magnification=20  -
-MagDiffAllowed=0 --SortingOption=10  --PatientID=-1 --PercentTest=15 --PercentValid=15 --nSplit 0 --
outFilenameStats='../r2_test/test_69000k/out_filename_Stats.txt'
"""
```

Convert to TFRecord:
"""

```
# valid
python 00_preprocessing/TFRecord_multi_Classes/build_TF_test_multiClass.py --
directory='r3_LUAD_sorted/512px_Tiled_NewPortal/'  --output_directory='r3_TFRecord_valid' --
num_threads=1 --one_FT_per_Tile=False --ImageSet_basename='valid' --labels_names='labelref_r3.txt' --
labels='labels_r3.txt' --PatientID=14

# test
python  00_preprocessing/TFRecord_multi_Classes/build_TF_test_multiClass.py --
directory='r3_LUAD_sorted/512px_Tiled_NewPortal'  --output_directory='r3_TFRecord_test' --num_threads=1
--one_FT_per_Tile=False --ImageSet_basename='test' --labels_names='labelref_r3.txt' --labels='labels_r3.txt' --
PatientID=14

# train
python 00_preprocessing/TFRecord_multi_Classes/build_image_data_multiClass.py --
directory='r3_LUAD_sorted/512px_Tiled_NewPortal' --output_directory='r3_TFRecord_train' --
train_shards=1024 --validation_shards=128 --num_threads=16  --labels_names='labelref_r3.txt' --
labels='labels_r3.txt' --PatientID=14
"""
```

train the model with 10-class sigmoid classifier:

"""

```
bazel-bin/inception/imagenet_train --num_gpus=4 --batch_size=400 --train_dir="r3_results_train" --
data_dir="r3_TFRecord_train" --ClassNumber=10 --mode='1_sigmoid' --NbrOfImages=326613 --
save_step_for_chekcpoint=815  --max_steps=81501
```

"""

once the checkpoints start being saved, we can start runing the valid and test sets:

"""

```
export CHECKPOINT_PATH='full_ath_to/r3_results_train/'
export OUTPUT_DIR='full_path_to/r3_valid'
export DATA_DIR='r3_TFRecord_valid'
export LABEL_FILE='labelref_r3.txt'

# create temporary directory for checkpoints
mkdir  -p $OUTPUT_DIR/tmp_checkpoints
export CUR_CHECKPOINT=$OUTPUT_DIR/tmp_checkpoints

# check if next checkpoint available
declare -i count=815
declare -i step=815
declare -i NbClasses=10

while true; do
    echo $count
    if [ -f $CHECKPOINT_PATH/model.ckpt-$count.meta ]; then
        echo $CHECKPOINT_PATH/model.ckpt-$count.meta " exists"
        # check if there's already a computation for this checkpoinmt
```

```
        export TEST_OUTPUT=$OUTPUT_DIR/test_$count'k'
        if [ ! -d $TEST_OUTPUT ]; then
            mkdir -p $TEST_OUTPUT


            ln -s $CHECKPOINT_PATH/*-$count.* $CUR_CHECKPOINT/.
            touch $CUR_CHECKPOINT/checkpoint
            echo 'model_checkpoint_path: "'$CUR_CHECKPOINT'/model.ckpt-'$count'"' >
$CUR_CHECKPOINT/checkpoint
            echo 'all_model_checkpoint_paths: "'$CUR_CHECKPOINT'/model.ckpt-'$count'"' >>
$CUR_CHECKPOINT/checkpoint

            # Test
            python
/gpfs/scratch/coudrn01/NN_test/code/DeepPATH/DeepPATH_code/02_testing/xClasses/nc_imagenet_eval.p
y --checkpoint_dir=$CUR_CHECKPOINT --eval_dir=$OUTPUT_DIR --data_dir=$DATA_DIR  --batch_size 200  --
run_once --ImageSet_basename='valid_' --ClassNumber $NbClasses --mode='1_sigmoid'  --TVmode='test'
            # wait

            mv $OUTPUT_DIR/out* $TEST_OUTPUT/.

            # ROC
            export OUTFILENAME=$TEST_OUTPUT/out_filename_Stats.txt
            python
/gpfs/scratch/coudrn01/NN_test/code/DeepPATH/DeepPATH_code/03_postprocessing/0h_ROC_MultiOutput
_BootStrap.py --file_stats=$OUTFILENAME  --output_dir=$TEST_OUTPUT --labels_names=$LABEL_FILE

        else
            echo 'checkpoint '$TEST_OUTPUT' skipped'
        fi

    else
        echo $CHECKPOINT_PATH/model.ckpt-$count.meta " does not exist"
        break
    fi

    # next checkpoint
    count=`expr "$count" + "$step"`
done

# summarize all AUC per slide (average probability) for class 1:
ls -tr $OUTPUT_DIR/test_*/out2_roc_data_AvPb_c1a*  | sed -e 's/k\/out2_roc_data_AvPb_c1a/ /' | sed -e
's/test_/ /' | sed -e 's/_/ /g' | sed -e 's/.txt//'   > $OUTPUT_DIR/valid_out2_AvPb_AUCs_1.txt


# summarize all AUC per slide (average probability) for macro average:
ls -tr $OUTPUT_DIR/test_*/out2_roc_data_AvPb_macro*  | sed -e 's/k\/out2_roc_data_AvPb_macro_/ /' |
sed -e 's/test_/ /' | sed -e 's/_/ /g' | sed -e 's/.txt//'   > $OUTPUT_DIR/valid_out2_AvPb_AUCs_macro.txt
```

```
# summarize all AUC per slide (average probability) for micro average:
ls -tr $OUTPUT_DIR/test_*/out2_roc_data_AvPb_micro*  | sed -e 's/k\/out2_roc_data_AvPb_micro_/ /' | sed
-e 's/test_/ /' | sed -e 's/_/ /g' | sed -e 's/.txt//'   > $OUTPUT_DIR/valid_out2_AvPb_AUCs_micro.txt

ls -tr $OUTPUT_DIR/test_*/out2_roc_data_AvPb_c2*  | sed -e 's/k\/out2_roc_data_AvPb_c2/ /' | sed -e
's/test_/ /' | sed -e 's/_/ /g' | sed -e 's/.txt//'   > $OUTPUT_DIR/valid_out2_AvPb_AUCs_2.txt

ls -tr $OUTPUT_DIR/test_*/out2_roc_data_AvPb_c3*  | sed -e 's/k\/out2_roc_data_AvPb_c3/ /' | sed -e
's/test_/ /' | sed -e 's/_/ /g' | sed -e 's/.txt//'   > $OUTPUT_DIR/valid_out2_AvPb_AUCs_3.txt

ls -tr $OUTPUT_DIR/test_*/out2_roc_data_AvPb_c4*  | sed -e 's/k\/out2_roc_data_AvPb_c4/ /' | sed -e
's/test_/ /' | sed -e 's/_/ /g' | sed -e 's/.txt//'   > $OUTPUT_DIR/valid_out2_AvPb_AUCs_4.txt

ls -tr $OUTPUT_DIR/test_*/out2_roc_data_AvPb_c5*  | sed -e 's/k\/out2_roc_data_AvPb_c5/ /' | sed -e
's/test_/ /' | sed -e 's/_/ /g' | sed -e 's/.txt//'   > $OUTPUT_DIR/valid_out2_AvPb_AUCs_5.txt

ls -tr $OUTPUT_DIR/test_*/out2_roc_data_AvPb_c6*  | sed -e 's/k\/out2_roc_data_AvPb_c6/ /' | sed -e
's/test_/ /' | sed -e 's/_/ /g' | sed -e 's/.txt//'   > $OUTPUT_DIR/valid_out2_AvPb_AUCs_6.txt

ls -tr $OUTPUT_DIR/test_*/out2_roc_data_AvPb_c7*  | sed -e 's/k\/out2_roc_data_AvPb_c7/ /' | sed -e
's/test_/ /' | sed -e 's/_/ /g' | sed -e 's/.txt//'   > $OUTPUT_DIR/valid_out2_AvPb_AUCs_7.txt

ls -tr $OUTPUT_DIR/test_*/out2_roc_data_AvPb_c8*  | sed -e 's/k\/out2_roc_data_AvPb_c8/ /' | sed -e
's/test_/ /' | sed -e 's/_/ /g' | sed -e 's/.txt//'   > $OUTPUT_DIR/valid_out2_AvPb_AUCs_8.txt

ls -tr $OUTPUT_DIR/test_*/out2_roc_data_AvPb_c9*  | sed -e 's/k\/out2_roc_data_AvPb_c9/ /' | sed -e
's/test_/ /' | sed -e 's/_/ /g' | sed -e 's/.txt//'   > $OUTPUT_DIR/valid_out2_AvPb_AUCs_9.txt

ls -tr $OUTPUT_DIR/test_*/out2_roc_data_AvPb_c10*  | sed -e 's/k\/out2_roc_data_AvPb_c10/ /' | sed -e
's/test_/ /' | sed -e 's/_/ /g' | sed -e 's/.txt//'   > $OUTPUT_DIR/valid_out2_AvPb_AUCs_10.txt
"""
```

A similar code can be used for the test check by modifying the corresponding options and inputs.

labelref_r3.txt

```
EGFR
FAT1
FAT4
KEAP1
KRAS
LRP1B
NF1
SETBP1
STK11
TP53
```

run1b_10way_MutationClassifier

labels_r3.txt

TCGA-95-7039-0	TP53
TCGA-95-7039-0	LRP1B
TCGA-95-7039-0	KRAS
TCGA-95-7039-0	EGFR
TCGA-95-7039-0	FAT1
TCGA-95-7567-0	TP53
TCGA-95-7567-0	LRP1B
TCGA-95-7567-0	KRAS
TCGA-95-7567-0	FAT4
TCGA-05-4427-0	TP53
TCGA-05-4427-0	LRP1B
TCGA-05-4427-0	KRAS
TCGA-05-4427-0	SETBP1
TCGA-64-5778-0	TP53
TCGA-64-5778-0	LRP1B
TCGA-64-5778-0	KRAS
TCGA-55-8507-0	TP53
TCGA-55-8507-0	LRP1B
TCGA-55-8507-0	FAT4
TCGA-55-8507-0	STK11
TCGA-MN-A4N4-0	TP53
TCGA-MN-A4N4-0	LRP1B
TCGA-05-4382-0	TP53
TCGA-05-4382-0	LRP1B
TCGA-05-4382-0	FAT4
TCGA-05-4382-0	EGFR
TCGA-05-4382-0	SETBP1
TCGA-05-4398-0	TP53
TCGA-05-4398-0	LRP1B
TCGA-05-4398-0	FAT1
TCGA-05-4398-0	SETBP1
TCGA-49-AAR4-0	TP53
TCGA-49-AAR4-0	LRP1B
TCGA-49-AAR4-0	FAT1

- Figure out if we need at least one gene mutation label for each tile name.
    - Try first with tile labels only from tiles had KRAS mutation.
    - Then, if not working properly, use other gene name to represent tiles without KRAS mutation.

➔ Need to pull out the KRAS mutation Patient ID ---- also Image ID ---- Tile ID list / dict / table. In order to generate this label file.