Wednesday Update

Managing Different Image sizes

Current Preprocessing: Removing images smaller than 2KB

- Need medical perspective on this threshold

Approach #1: Resize before CNN (perplexity)

- Pro: Very compute **efficient**
- Con: Can make artifacts (upscaling)/hide patterns (downscaling), **pre-CNN distortion**

Approach #2: Adaptive Pooling (perplexity)

- Pro: moderate compute, **compression late in CNN**
- Con: Compression proportional to image size, not cell size

Approach #3: FCN U- Net (perplexity)

- Pro: Compression of a single cell is **consistent** no matter image size
- Con: High compute, difficult and slow training

Better size handling but more compute

Preprocessing

```
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

```
def preprocess(image_path, label):
    img = tf.io.read_file(image_path)
    img = tf.image.decode_png(img, channels=3) # Renders image into a 8-bit integer tensor and converts to RGB (3-channel)
    img = tf.image.resize(img, [256, 256]) # Resize to 256 x 256 array
    img = tf.image.central_crop(img, central_fraction=224/256) # Crop out center 224 x 224 pixels to finish resizing
    img = tf.cast(img, tf.float32) / 255.0 # Cast each tensor to float type and normalize
    img = tf.image.per_image_standardization(img) # Standardize each image to have mean 0 and variance 1
    return img, label
```

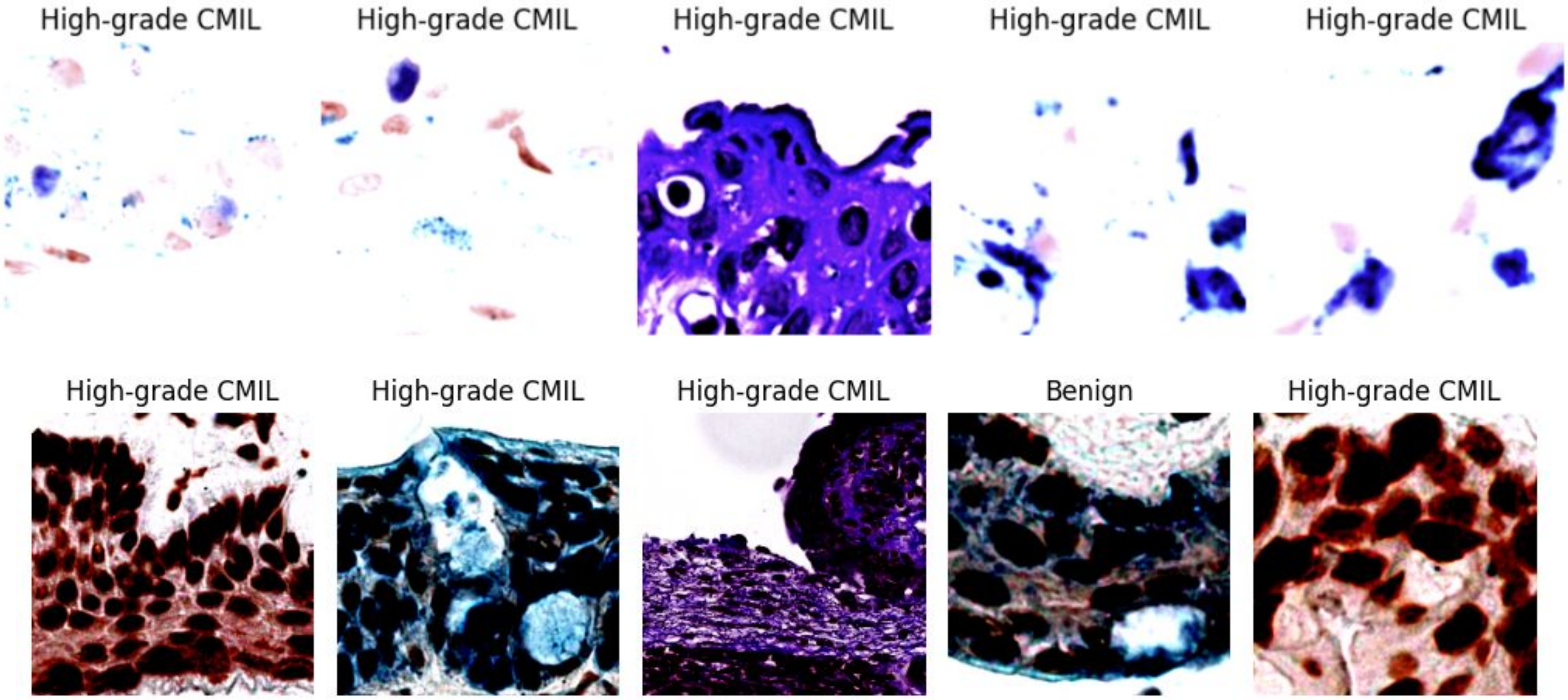
Pytorch

- Pytorch normalizes across the entire data set, based on mean and variance values set for each channel

Tensorflow

- TF normalizes image by image, in this case so that the overall image mean pixel value is 0 and variance is 1

Preprocessing - Visual Differences



Preprocessing - Resizing

https://pytorch.org/vision/main/auto_examples/transforms/plot_transforms_illustrations.html#sphx-glr-auto-examples-transforms-plot-transforms-illustrations-py

Resize

The `Resize` transform (see also `resize()`) resizes an image.

```
resized_imgs = [v2.Resize(size=size)(orig_img) for size in (30, 50, 100, orig_img.size)]
plot([orig_img] + resized_imgs)
```

+ G
+ P
+ A
+ R

- the transform will scale the shorter side of the image to that size while preserving the aspect ratio
- The longer side is resized proportionally so the entire image keeps the same aspect ratio, just smaller or larger overall
- If original images are high resolution, downsampling them so that the shorter side is 256 should still retain enough detail after interpolation
- If original images have very low resolution, then resizing (i.e., upscaling them) to 256 could indeed cause a pixelated effect.

Preprocessing - Padding

https://pytorch.org/vision/main/auto_examples/transforms/plot_transforms_illustrations.html#sphx-glr-auto-examples-transforms-plot-transforms-illustrations-py

Pad

The `Pad` transform (see also `pad()`) pads all image borders with some pixel values.

```
padded_imgs = [v2.Pad(padding=padding)(orig_img) for padding in (3, 10, 30, 50)]
plot([orig_img] + padded_imgs)
```

- Padding
    - Instead of a CenterCrop, pad the image to **preserve the original resolution** and **avoids clipping details at the edges**
    - Define a custom transform that pads an image to a square shape without cropping any of the original content, resizing into uniform 224x224 after
    - Plan to introduce padding before resizing, maintain the spatial structure of the original image

Training Transformation Ideas

- **Stain Normalization**
  - Reduce color variability
  - Help focus on morphology
  - May not be effective on smaller datasets collected from similar labs/sources
- **Randomized Horizontal Flipping**
  - Horizontal because vertical will affect epithelium structure, which is important in learning
- **Randomized Rotations**
  - Increase variance
  - May distort morphological features
- **Randomized Cropping**
  - Again, introduces variance
  - May eliminate learning epithelium-wide structures, which is important

Buda, M., Maki, A., & Mazurowski, M. A. (2018).
“A Systematic Study of the Class Imbalance Problem in Convolutional Neural Networks.”
Neural Networks, 106, 249–259.

Class Imbalance

Explains why models are having a hard time predicting the benign class, overfitting

```
# Create PNGDataset instances for train, validation, and test
train_dataset = PNGDataset(train_patches, labels, transform=transform)
val_dataset = PNGDataset(val_patches, labels, transform=transform)
test_dataset = PNGDataset(test_patches, labels, transform=transform)
```

```
[ ] sum(train_dataset.labels)/len(train_dataset.labels)
```

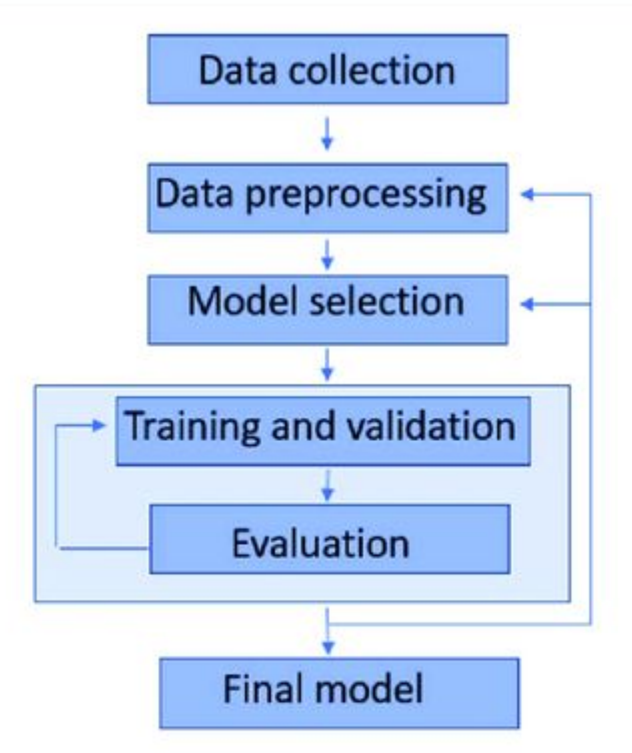
```
0.7442116868798236
```

- Training on 74 % high-grade class images
- Last quarter’s model attempted randomoversampler, demonstrated overfitting

- Other techniques to try
  - Weighted Loss Functions and Focal Loss
    - Assigns penalty for misclassifying minority class examples
  - Synthetic Sample Generation:
    - synthetic oversampling methods like SMOTE, create new, diverse minority class examples.

```
def oversample_dataset(dataset):

    # Get all labels from the dataset
    labels = [label for _, label in dataset]

    # Initialize RandomOverSampler
    oversampler = RandomOverSampler(random_state=42)

    # Resample the indices of the dataset
    resampled_indices, _ = oversampler.fit_resample(np.arange(len(dataset)).reshape(-1, 1), labels)

    # Create a new dataset with the resampled indices
    resampled_dataset = [dataset[i[0]] for i in resampled_indices]

    return resampled_dataset

# Apply oversampling to the training dataset
train_dataset_resampled = oversample_dataset(train_dataset)

# Create DataLoaders using the PNGDataset instances
# benefits of using DataLoaders: num_workers for parallel processing (when one is training other is getting pre
# collate_fn for padding patches to the same dimensions when there are slight variations despite resizing
```

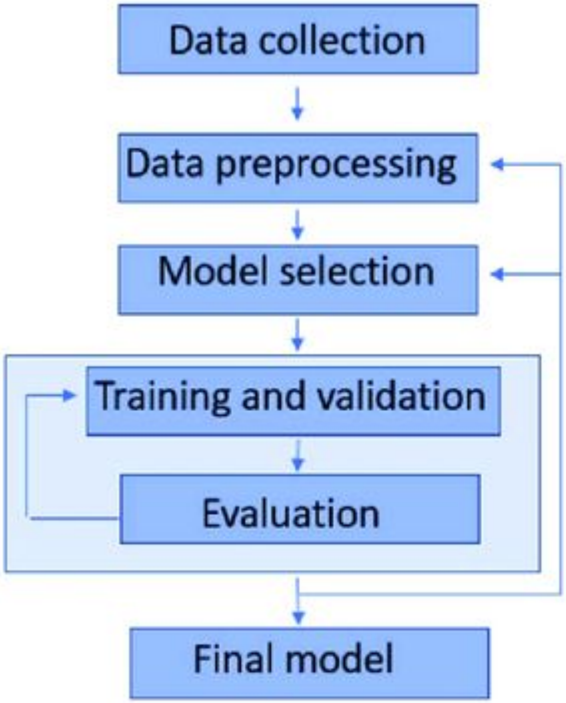
ML Workflow: Data Collection




```python
# #Define filtering function
def filter(input_folder, output_base_folder):···
# Define a function to group patches by case number
def group_patches(patch_dir):···
# Define a custom dataset class for loading PNG images
class PNGDataset(Dataset):···
```

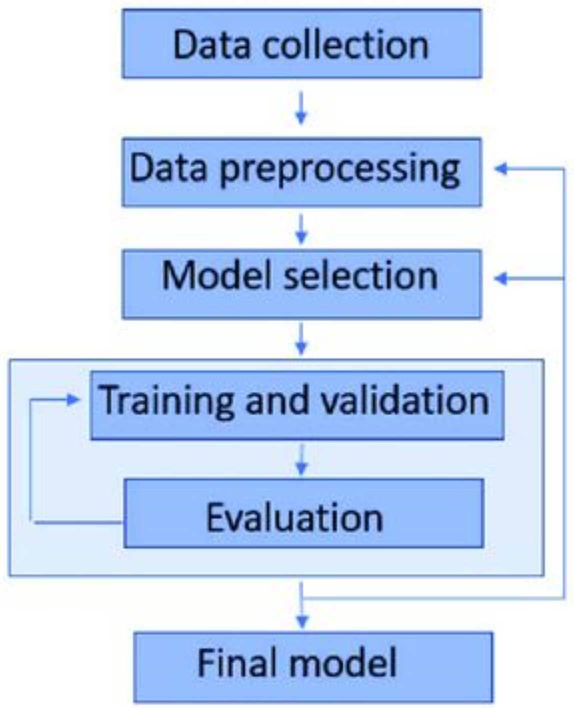
Ng, Frederick & Jiang, Runqing & Chow, James. (2020). Predicting radiation treatment planning evaluation parameter using artificial intelligence and machine learning. IOP SciNotes. 1. 014003. 10.1088/2633-1357/ab805d.

ML Workflow: Preprocessing



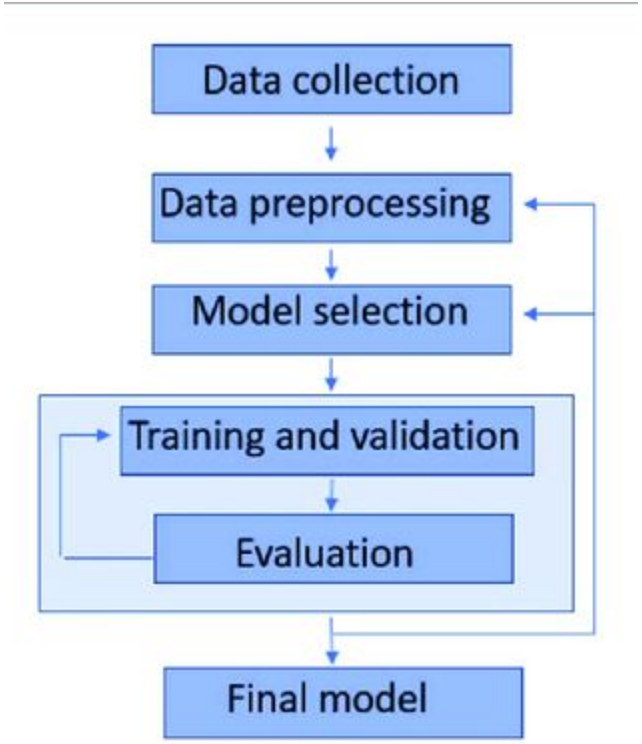

```python
# Resize all patch images to 256x256
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# Possible data augmentation for training data
train_transform = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

ML Workflow: Tuning and Validation




```python
# Define function to validate the model
def validation(model, criterion, val_loader):...
# Define a function to save the model checkpoint
def save_checkpoint(model, arch, checkpoint_dir, epoch):...
# Define Training Function
def train_model(model, optimizer, criterion, train_loader, val_loader, arch, checkpoint_dir, epochs=5, start_epoch=0):
# Loss function and gradient descent
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=0.001)

# Freeze all convolutional layers (optional)
for param in model.parameters():
    param.requires_grad = False

# Unfreeze just the final FC layer (optional)
for param in model.fc.parameters():
    param.requires_grad = True

# Replace the final fully connected layer
num_classes = 2
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, num_classes)
```

ML Workflow: Evaluation



Model	Accuracy	Notes
DenseNet	0.76	Last Quarter (Margaret) ● 7 epochs ● Trained everything
AlexNet	0.66	Last Quarter (Nathan) ● Best of many iterations ● Not recommended
ResNet50	0.68	Last Quarter (Sharon) ● Bad at predicting Benign patches
VGGNet	0.67	This Quarter (Jeffrey) ● Slightly better at predicting Benign
ResNet50	0.75	This Quarter (Jeffrey) ● Added training transformation
EfficientNetB2	0.72	This Quarter (Harvey) ● Added training transformation

ML Workflow: Next Steps

- Try new patches
- Implement Rohan's preprocessing steps
- Try oversampling
- Different pooling methods
- Additional data augmentation

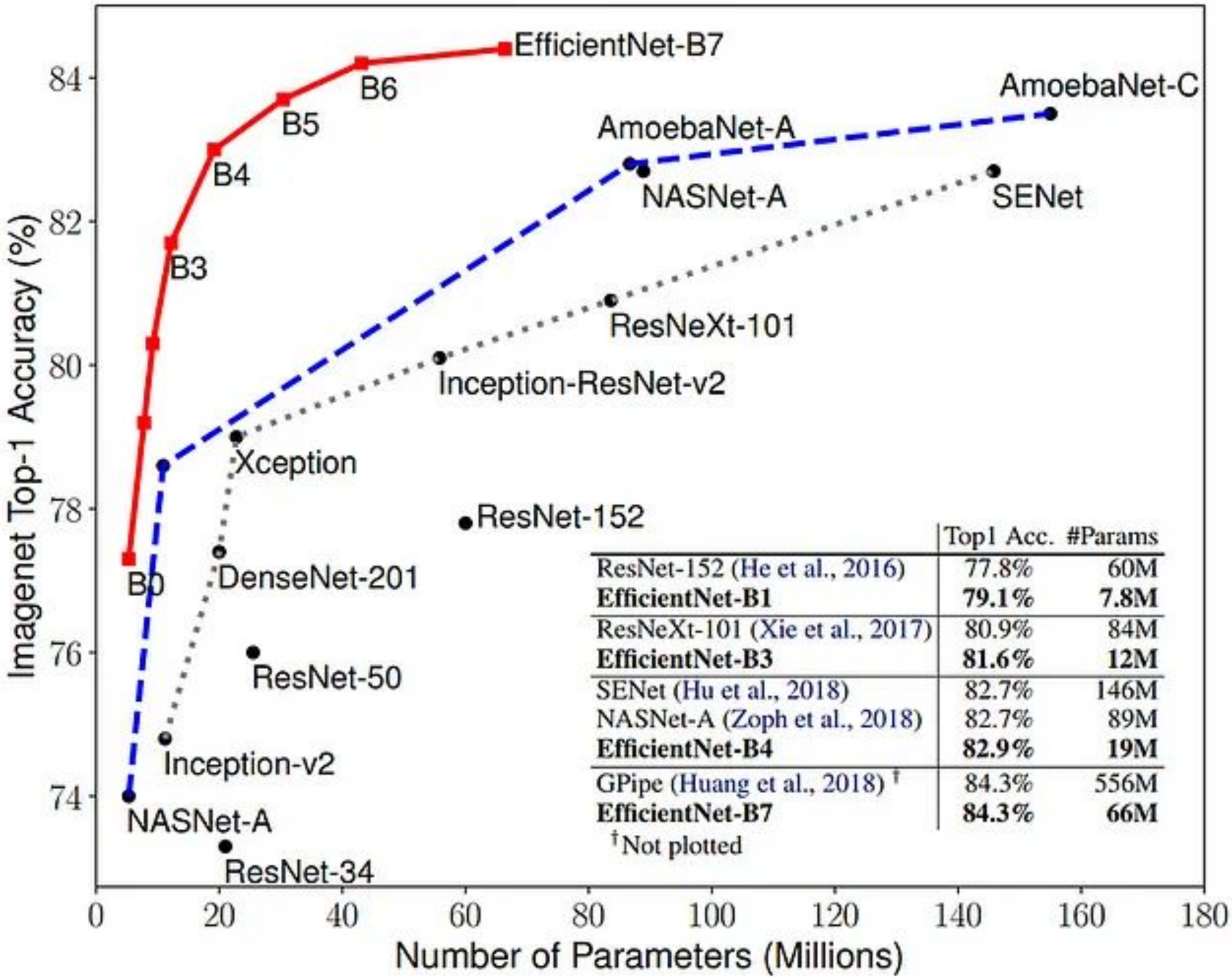
Things that might not be worthwhile

- Training entire CNN from scratch (not freezing any layers)
- AlexNet
- Oversampling (maybe)

Appendix

EfficientNet Notes

- Pre-trained on ImageNet dataset
- Faster training:
  - “EfficientNet-B0 achieves 77.1% top-1 accuracy on ImageNet with only 5.3M parameters, while ResNet-50 achieves 76.0% top-1 accuracy with 26M parameters. Additionally, the B-7 model performs at par with Gpipe, but with way fewer parameters (66M vs 557M)” viso.ai



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) [†]	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

[†]Not plotted

EfficientNet Details

- Currently: only training fully-connected layers of ResNet, DenseNet, EfficientNet on our classifications
    - Freezing Convolutional Layer and deep layers to speed up training
    - Parameters of frozen layers are trained on ImageNet images, which may not be the best (strawberries, cars, etc.)