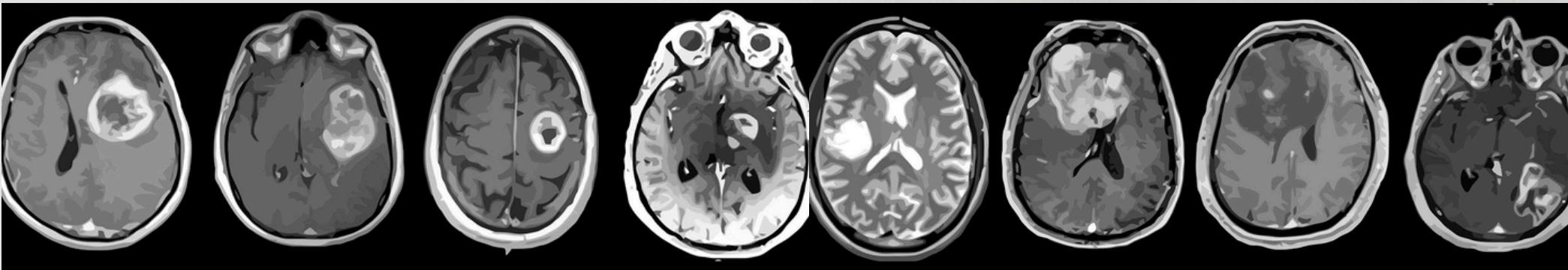


MACS 33002

video presentation link

*Analysis of ML Models
in Medical Imaging Classification*

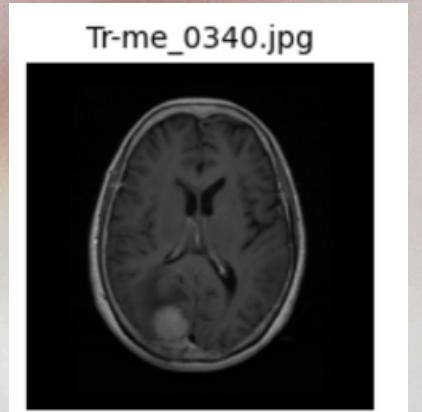


presented by
Helen Tian

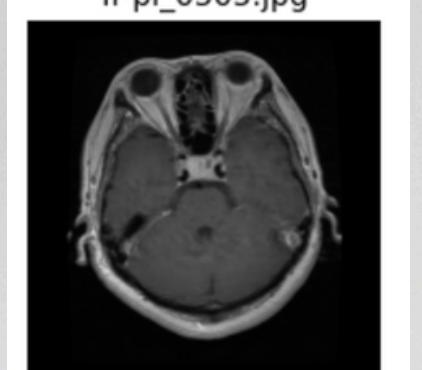
my motivation
combining interests: comp sci and medicine
hands-on learning opportunity for lab research

my research goal and question

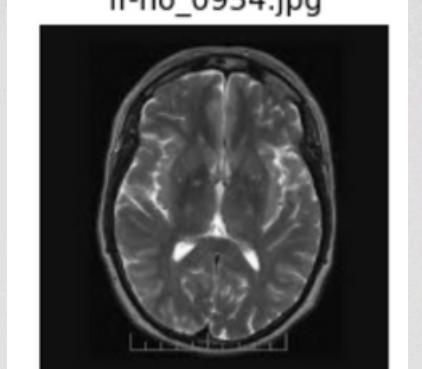
*how can we accurately classify MRI images
of brain tumors into categories of glioma,
meningioma, no tumor, and pituitary?*



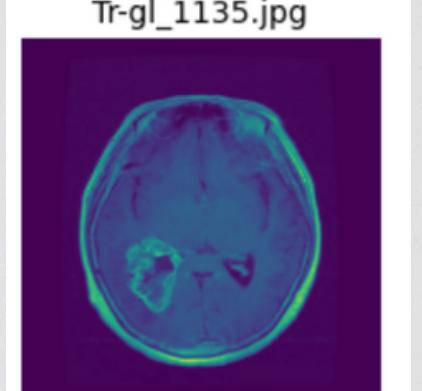
Tr-me_0340.jpg



Tr-pi_0505.jpg



Tr-no_0954.jpg

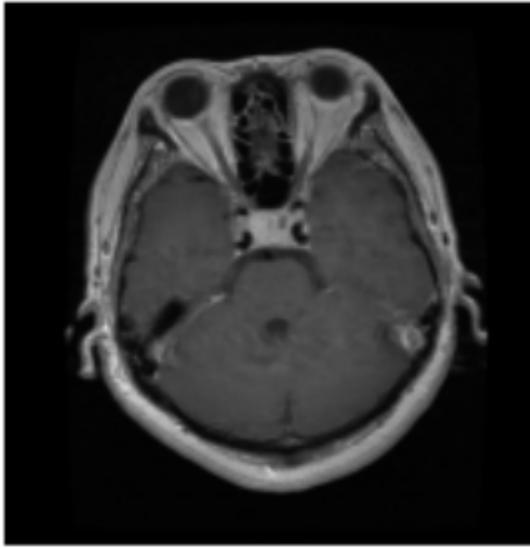


Tr-gli_1135.jpg

data collection and processing

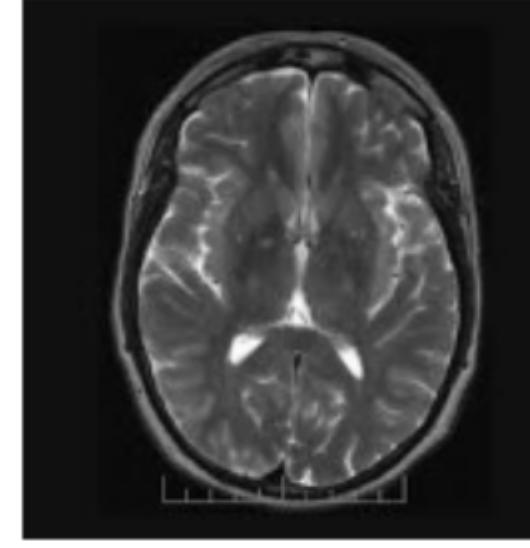
pituitary

Tr-pi_0505.jpg



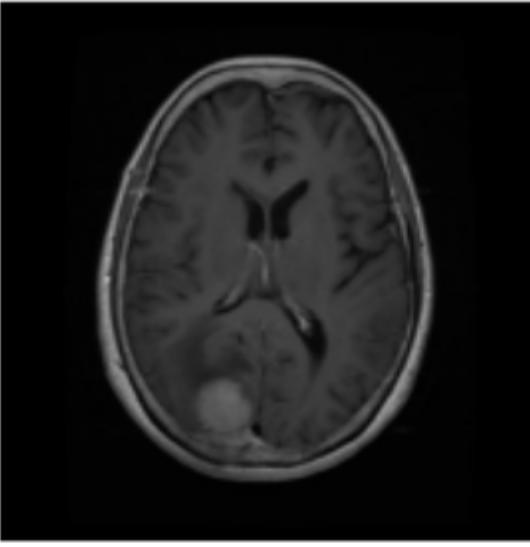
notumor

Tr-no_0954.jpg



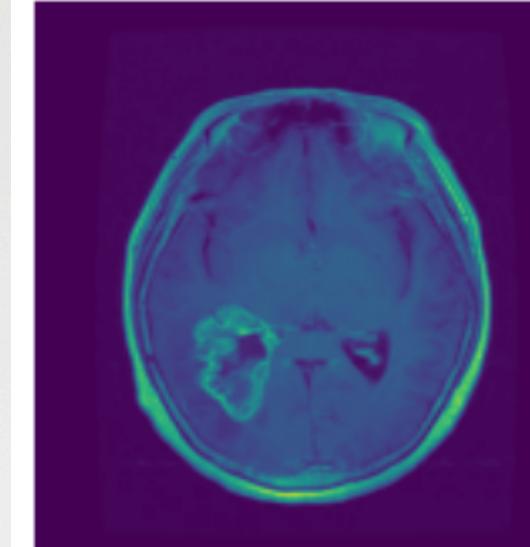
meningioma

Tr-me_0340.jpg



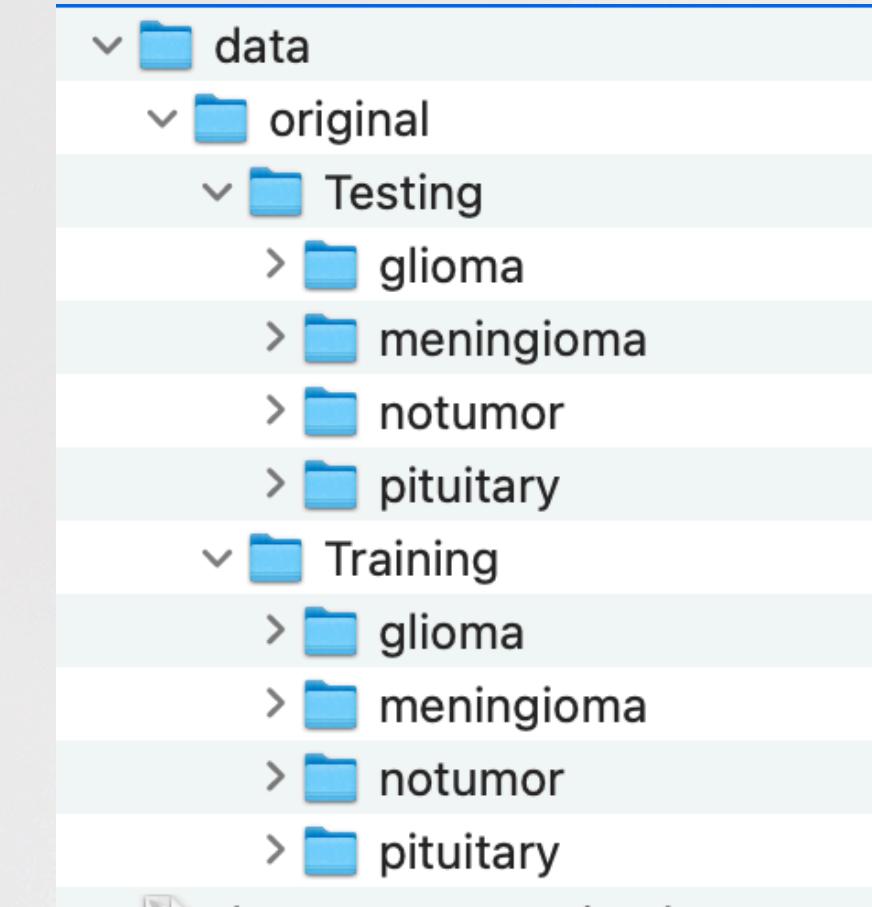
glioma

Tr-gl_1135.jpg



taken from [Kaggle](#) (combination of 3 datasets)

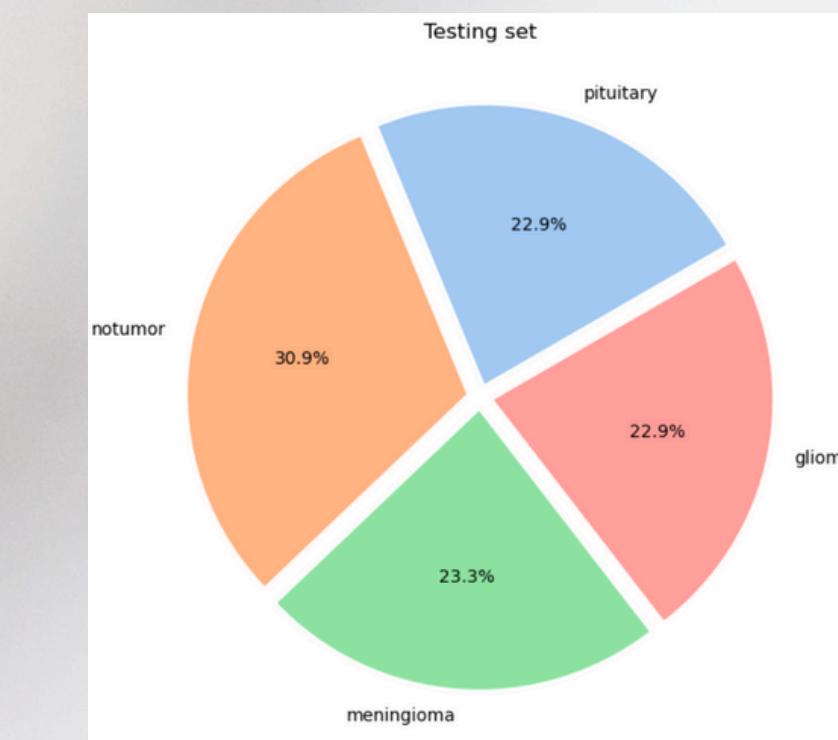
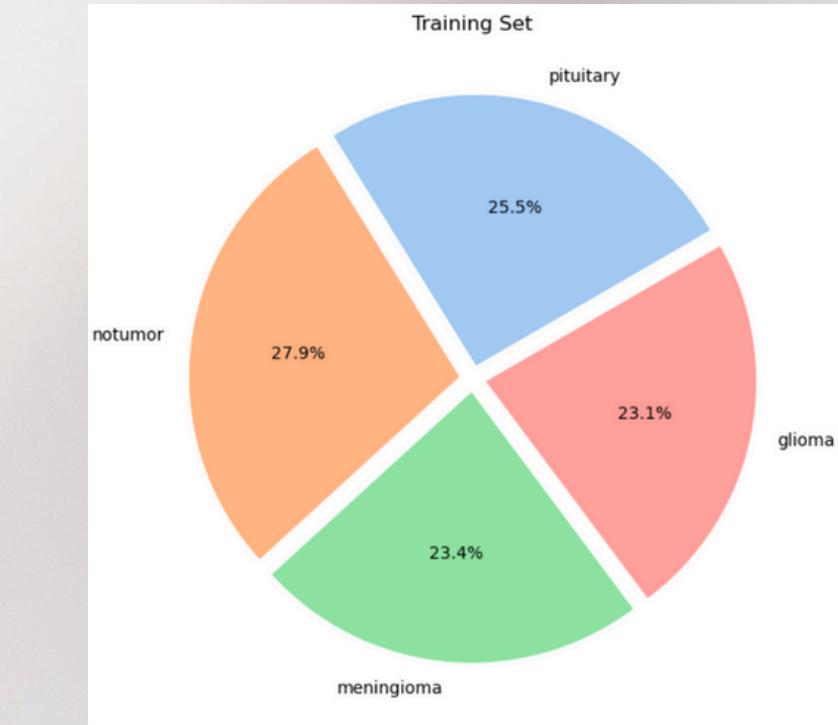
classified into 4 classes: glioma, meningioma, no tumor, and pituitary



81.3% training data

18.7% testing data

7023 images total
split across folders



data cleaning and transformation

```
def validify(img_files, img_dir):
    # Identify unreadable/missing images
    valid_images = []
    for file in img_files:
        try:
            img_path = os.path.join(img_dir, file)
            img = Image.open(img_path)
            valid_images.append(file)
        except Exception as e:
            print(f"Error with image {file}: {e}")

    return valid_images
```

```
def find_outliers(img_files, img_dir):
    valid_images = validify(img_files, img_dir)

    # Analyze the distribution of image sizes to detect outliers
    img_sizes = []
    for file in valid_images:
        img_path = os.path.join(img_dir, file)
        img = Image.open(img_path)
        img_sizes.append(img.size)

    img_size_distribution = np.array(img_sizes)
    img_size_mean = np.mean(img_size_distribution, axis=0)
    img_size_std = np.std(img_size_distribution, axis=0)

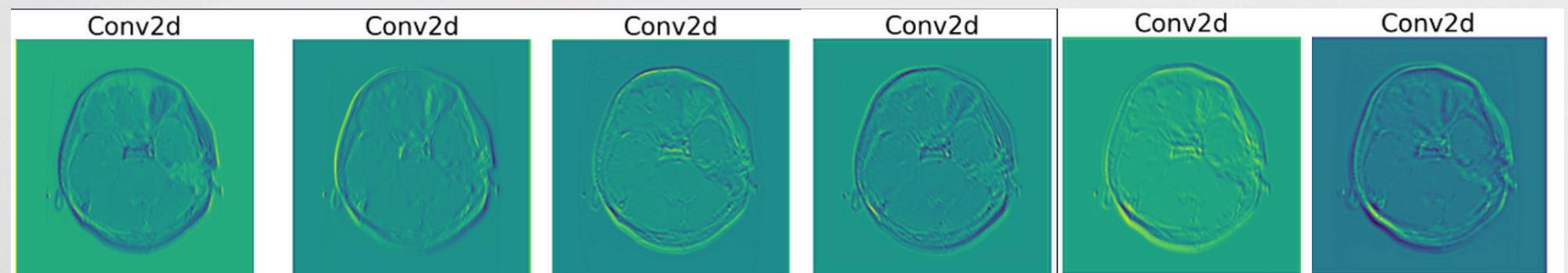
    outlier_indices = np.where(
        (img_size_distribution > img_size_mean + 3 * img_size_std) |
        (img_size_distribution < img_size_mean - 3 * img_size_std)
    )

    outlier_imgs = [valid_images[i] for i in outlier_indices[0]]
    return outlier_imgs
```

```
def transformation_def(img_files, img_dir):
    # Define standard image transformations
    # Respective order: standardize dimensions, convert to tensor, normalize pixel values
    img_transform = transforms.Compose([
        # transforms.Grayscale(num_output_channels=3),
        transforms.Resize(224, 224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.5], std=[0.5]),
    ])

    img_pil = [Image.open(os.path.join(img_dir, file)) for file in img_files]
    return img_pil, img_transform
```

```
def extract_conv_layers(pretrained_model):
    conv_weights = []
    conv_layers = []
    # Traverse through the model to extract convolutional layers and weights
    for module in pretrained_model.features.children():
        if isinstance(module, nn.Conv2d):
            conv_weights.append(module.weight)
            conv_layers.append(module)
    return conv_weights, conv_layers
```



model exploration

VGG19

```
VGG19Model (VGG19Model)
└── VGG (backbone_model)
    └── Sequential (features)
        └── Conv2d (0)
            └── ReLU (1)
        └── Conv2d (2)
            └── ReLU (3)
        └── MaxPool2d (4)
        └── Conv2d (5)
            └── ReLU (6)
        └── Conv2d (7)
            └── ReLU (8)
        └── MaxPool2d (9)
        └── Conv2d (10)
            └── ReLU (11)
        └── Conv2d (12)
            └── ReLU (13)
        └── Conv2d (14)
            └── ReLU (15)
        └── Conv2d (16)
            └── ReLU (17)
        └── MaxPool2d (18)
        └── Conv2d (19)
            └── ReLU (20)
        └── Conv2d (21)
            └── ReLU (22)
        └── Conv2d (23)
            └── ReLU (24)
        └── Conv2d (25)
            └── ReLU (26)
        └── MaxPool2d (27)
        └── Conv2d (28)
            └── ReLU (29)
        └── Conv2d (30)
            └── ReLU (31)
        └── Conv2d (32)
            └── ReLU (33)
        └── Conv2d (34)
            └── ReLU (35)
        └── MaxPool2d (36)
    └── AdaptiveAvgPool2d (avgpool)
        └── Sequential (classifier)
            └── Linear (0)
            └── ReLU (1)
            └── Dropout (2)
            └── Linear (3)
            └── ReLU (4)
            └── Dropout (5)
            └── Linear (6)
    ====
    Total params: 139,586,628
    Trainable params: 119,562,244
    Non-trainable params: 20,024,384
    Total mult-adds (G): 628.57
    ====
    Input size (MB): 19.27
    Forward/backward pass size (MB): 3804.23
    Params size (MB): 558.35
    Estimated Total Size (MB): 4381.85
    =====
```

ResNet50

```
=====
Layer (type (var_name))
=====

ResNet50Model (ResNet50Model)
└── ResNet (backbone_model)
    └── Conv2d (conv1)
    └── BatchNorm2d (bn1)
    └── ReLU (relu)
    └── MaxPool2d (maxpool)
    └── Sequential (layer1)
        └── Bottleneck (0)
        └── Bottleneck (1)
        └── Bottleneck (2)
    └── Sequential (layer2)
        └── Bottleneck (0)
        └── Bottleneck (1)
        └── Bottleneck (2)
    └── Sequential (layer3)
        └── Bottleneck (0)
        └── Bottleneck (1)
        └── Bottleneck (2)
    └── Sequential (layer4)
        └── Bottleneck (0)
        └── Bottleneck (1)
        └── Bottleneck (2)
    └── AdaptiveAvgPool2d (avgpool)
    └── Sequential (fc)
        └── Linear (0)
        └── ReLU (1)
        └── Dropout (2)
        └── Linear (3)
    ====
    Total params: 25,610,308
    Trainable params: 2,102,276
    Non-trainable params: 23,508,032
    Total mult-adds (G): 130.86
    ====
    Input size (MB): 19.27
    Forward/backward pass size (MB): 5690.62
    Params size (MB): 102.44
    Estimated Total Size (MB): 5812.33
    =====
```

EfficientNet

```
=====
Layer (type (var_name))
=====

EfficientNetV2Model (EfficientNetV2Model)
└── EfficientNet (backbone_model)
    └── Sequential (features)
        └── Conv2dNormActivation (0)
        └── Sequential (1)
    else
        └── Sequential (2)
    False
        └── Sequential (3)
    lse
        └── Sequential (4)
    else
        └── Sequential (5)
    False
        └── Sequential (6)
    else
        └── Sequential (7)
    se
        └── Conv2dNormActivation (8)
        └── AdaptiveAvgPool2d (avgpool)
        └── Sequential (classifier)
            └── Flatten (0)
            └── Dropout (1)
            └── Linear (2)
            └── GELU (3)
            └── Dropout (4)
            └── Linear (5)
    ====
    Total params: 117,563,232
    Trainable params: 328,960
    Non-trainable params: 117,234,272
    Total mult-adds (G): 391.39
    ====
    Input size (MB): 19.27
    Forward/backward pass size (MB): 17633.22
    Params size (MB): 470.25
    Estimated Total Size (MB): 18122.74
    =====
```

EfficientNet scales uniformly at width, depth, and resolution. It's also the simplest (least computationally expensive model).

- 328,960 trainable parameters

Conv2dNormActivation

- Conv2d
- Batch normalization
- Activation (Swish activation function: $Y = X * \text{sigmoid}(X)$)

Sequential

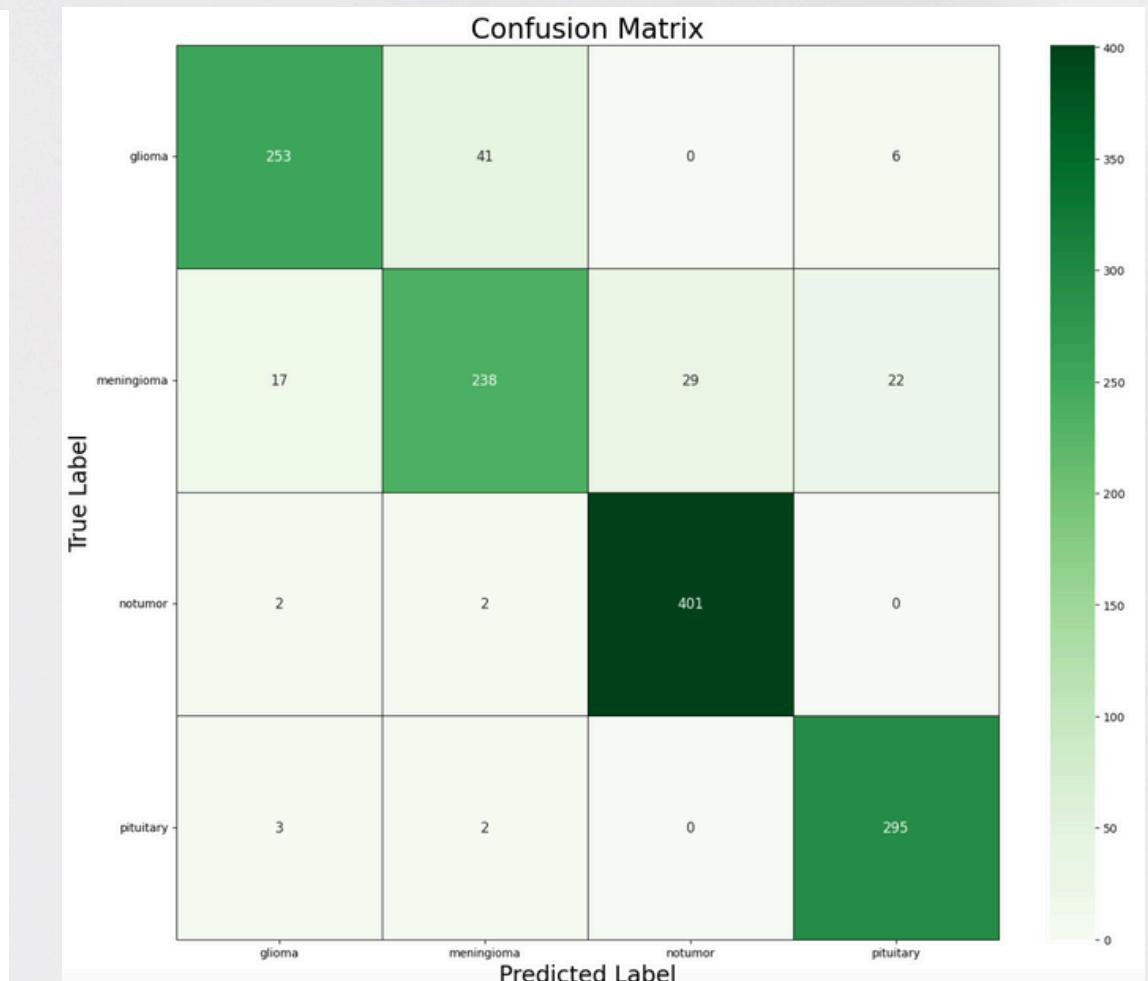
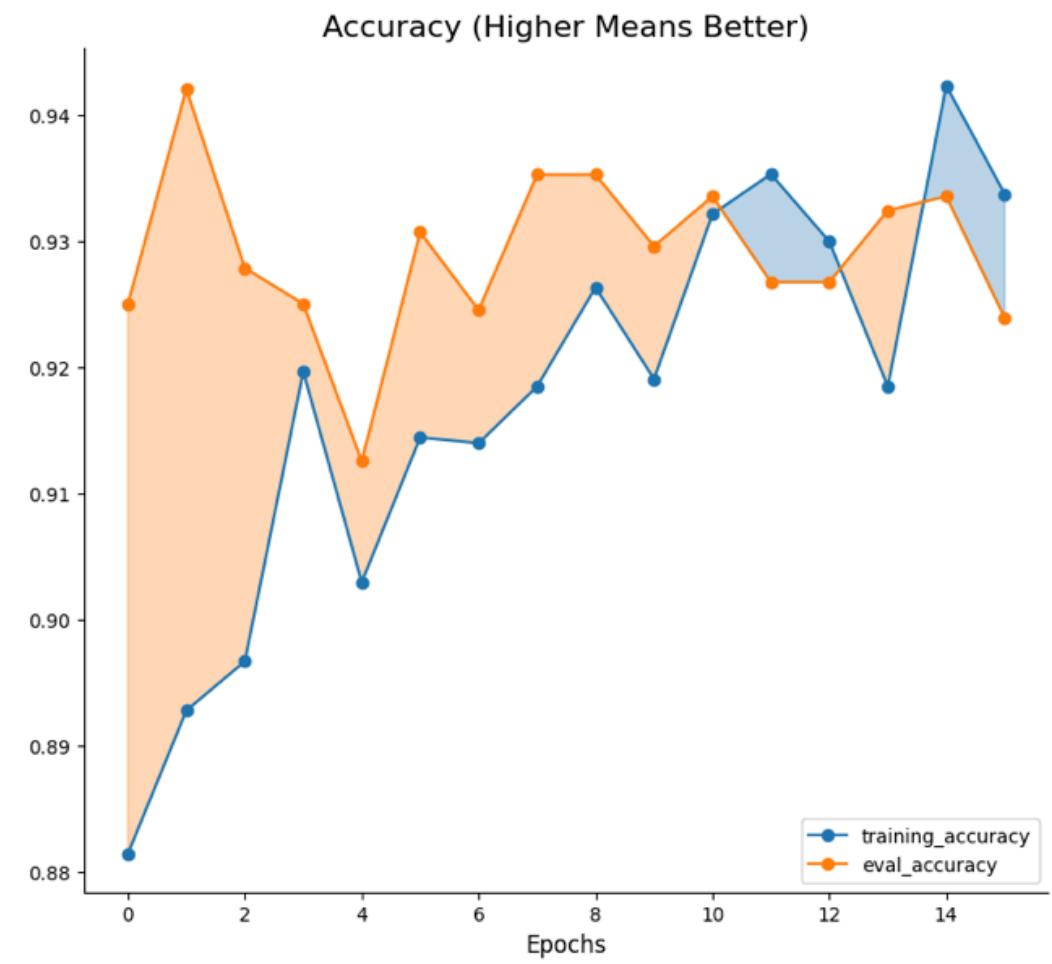
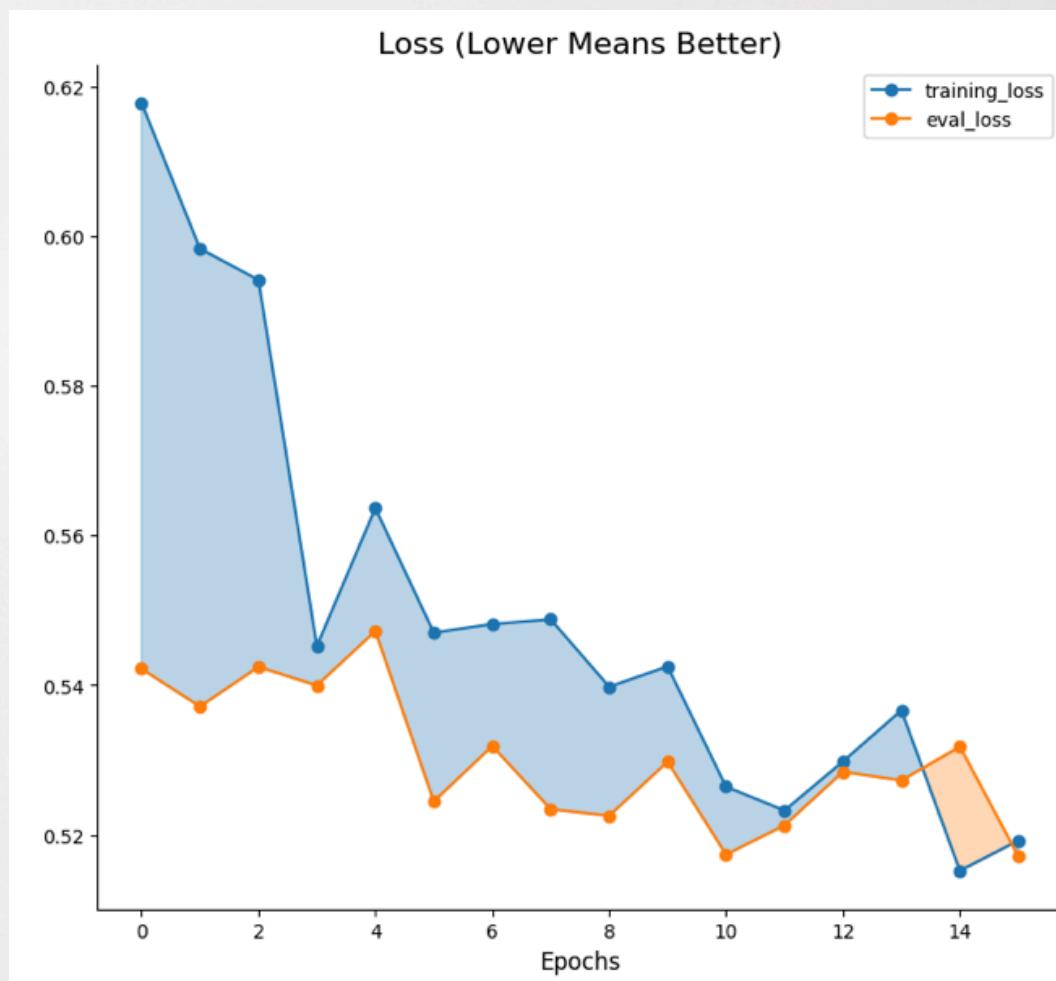
- To stack layers to form a feed-forward network

supervised learning

trained on 10%, 20%, and 30% of our data to see which one produced the best accuracy and stable model

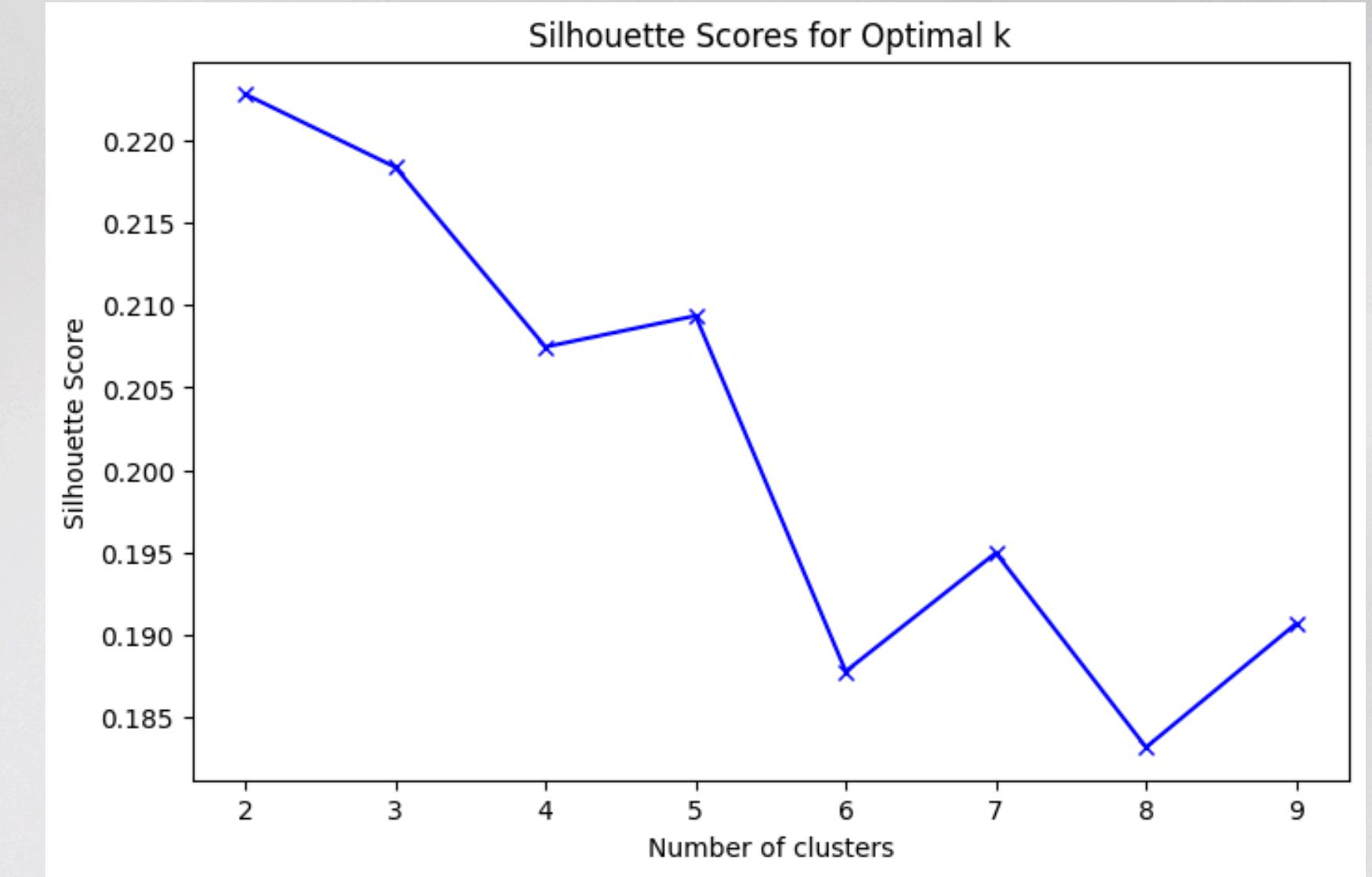
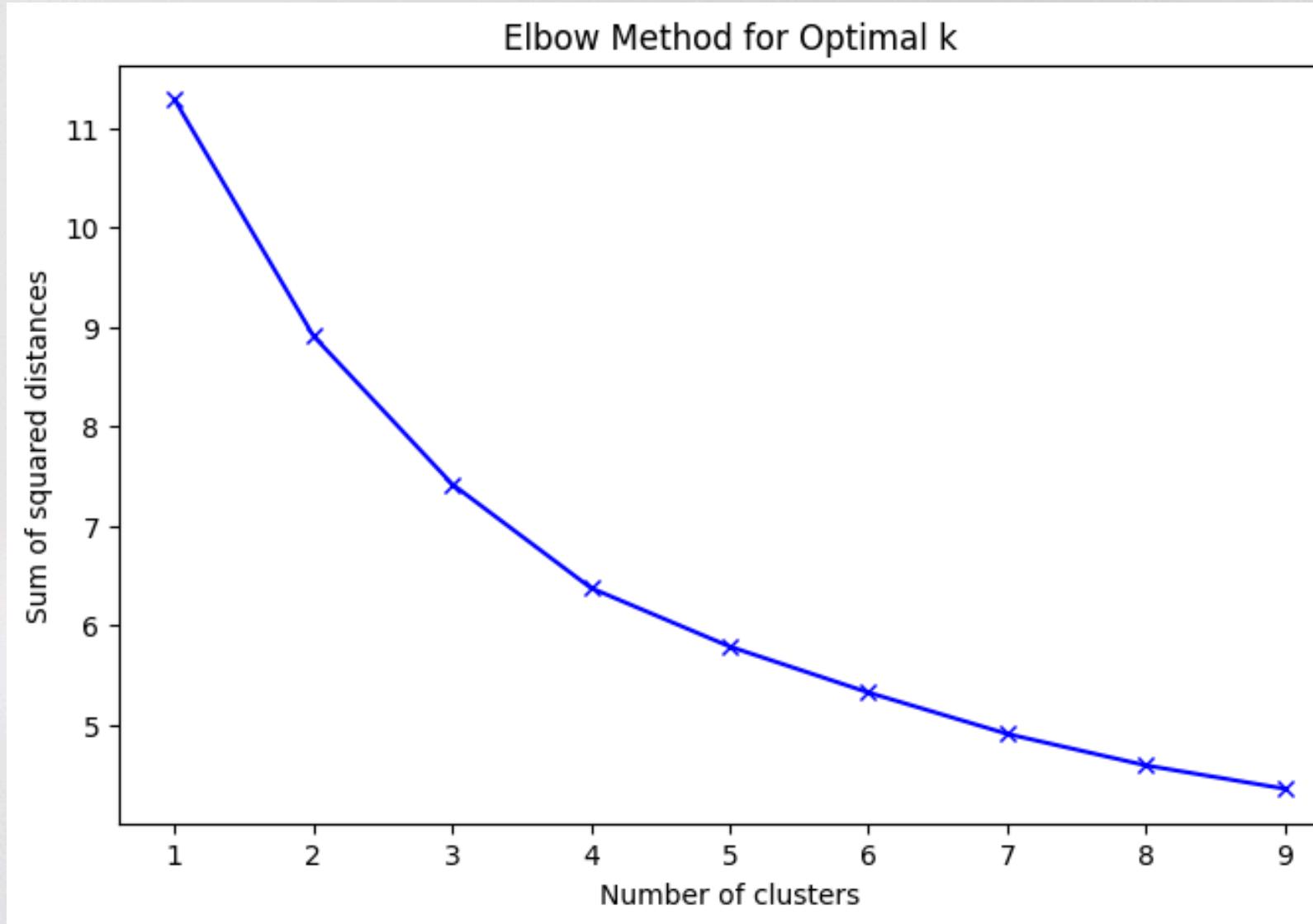
30% did the best

please note that while we found using 30% of our data was better, we will continue with 10% to optimize time efficiency



kmeans clustering

```
0: {'avg_intensity': 32.86749, 'std_dev': 39.523575}  
1: {'avg_intensity': 44.899525, 'std_dev': 49.508053}  
2: {'avg_intensity': 63.047462, 'std_dev': 63.520638}  
3: {'avg_intensity': 47.403687, 'std_dev': 41.16208}
```

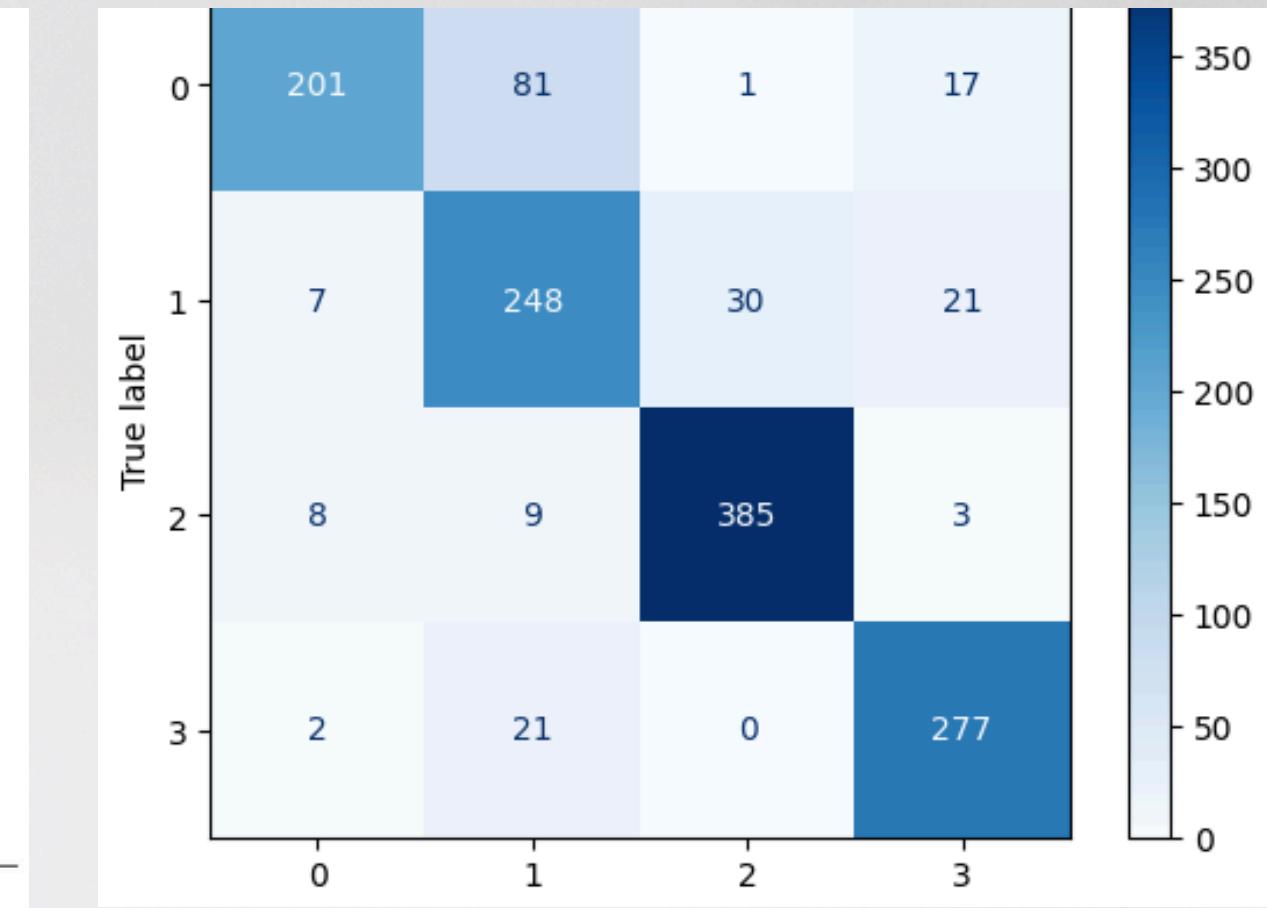
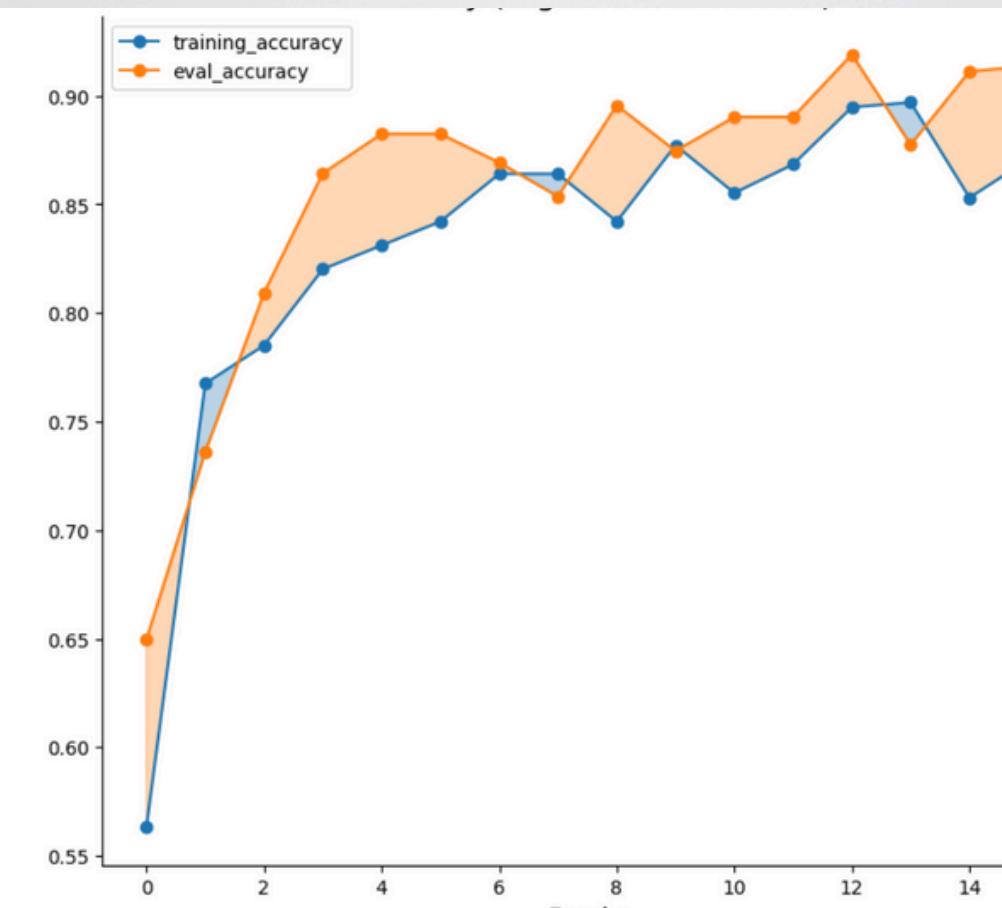
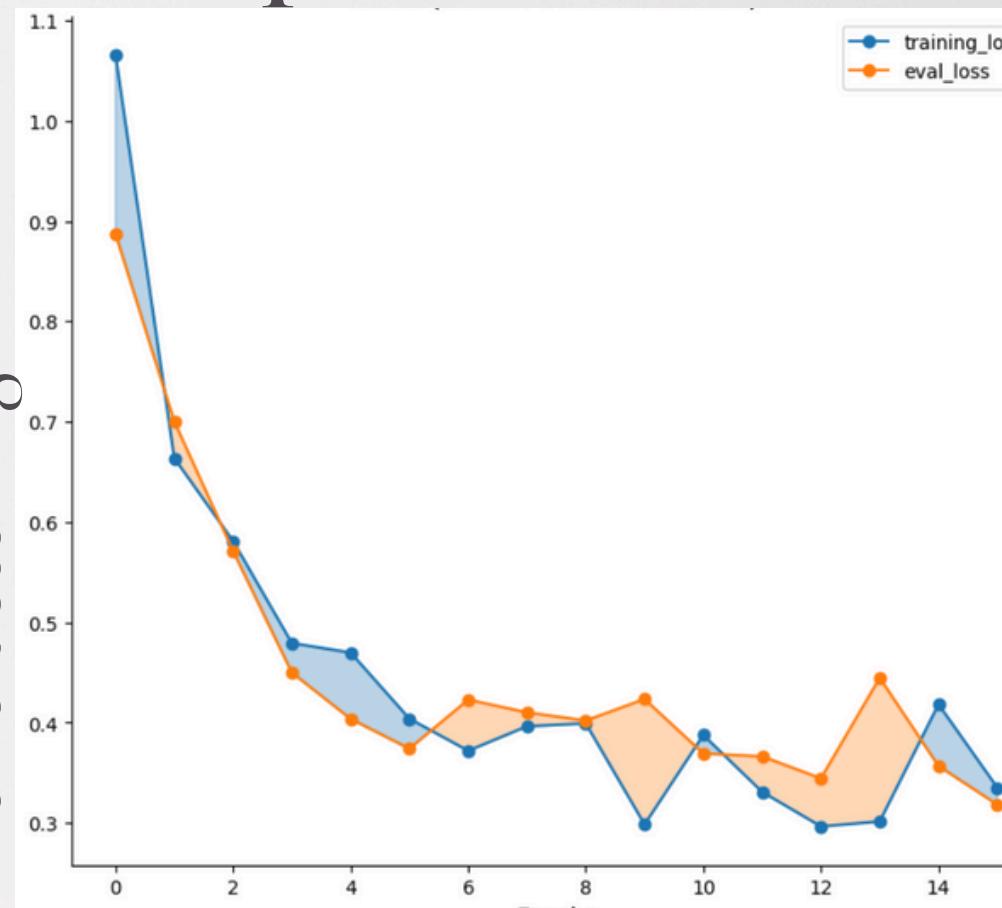


- plotting the sum of squared distances of samples to their closest cluster center
- "elbow" point is where the rate of decrease sharply shifts

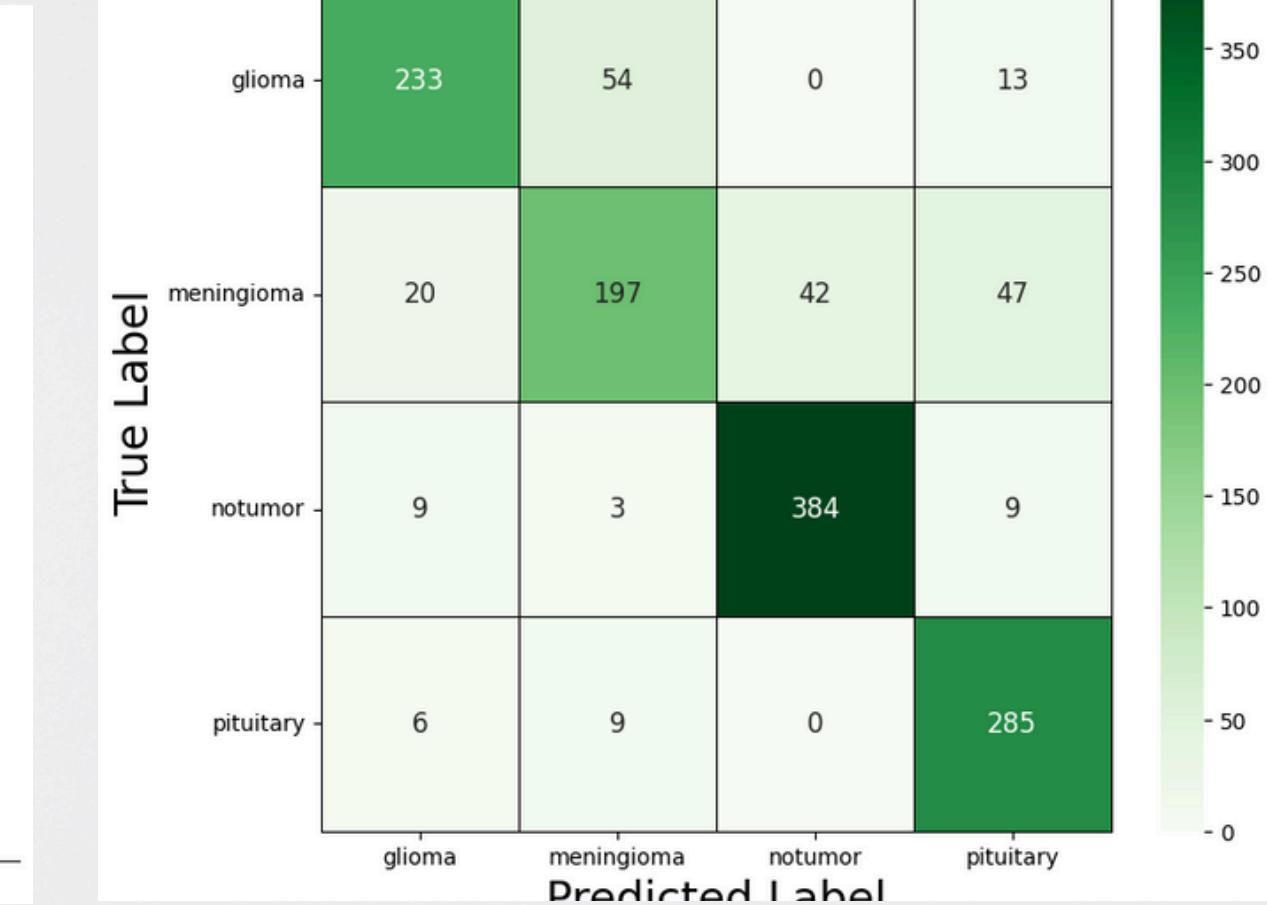
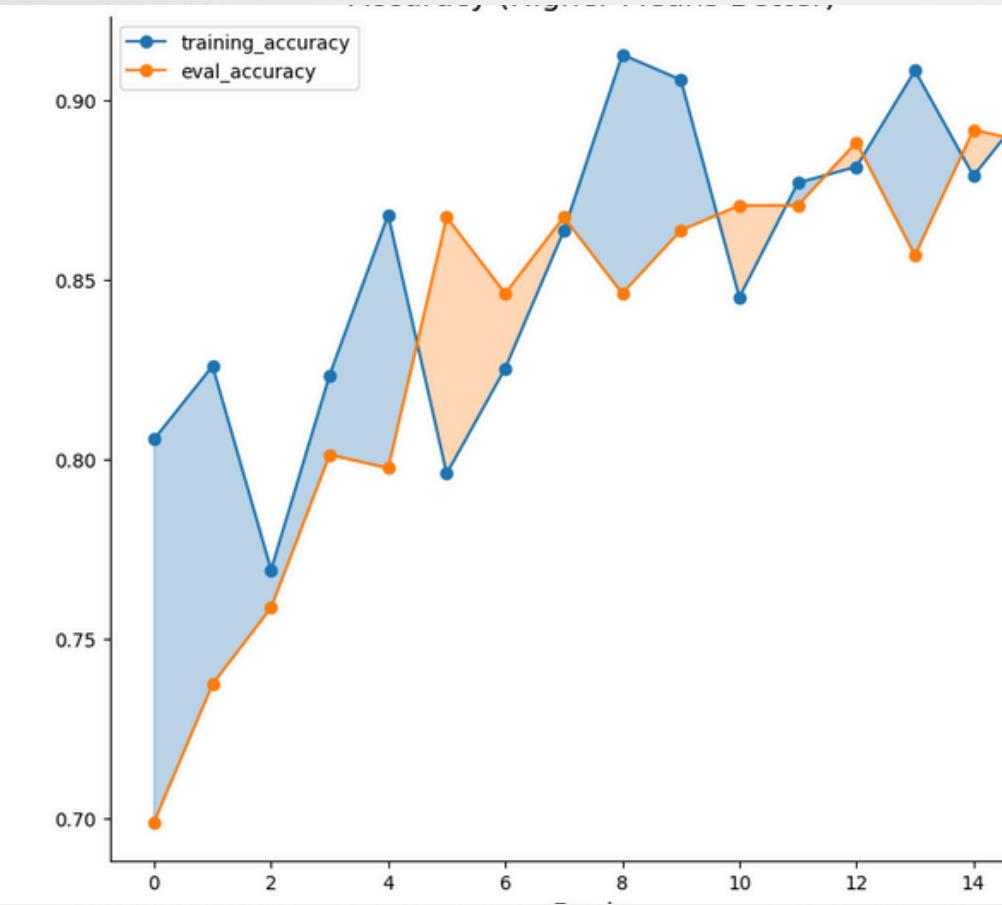
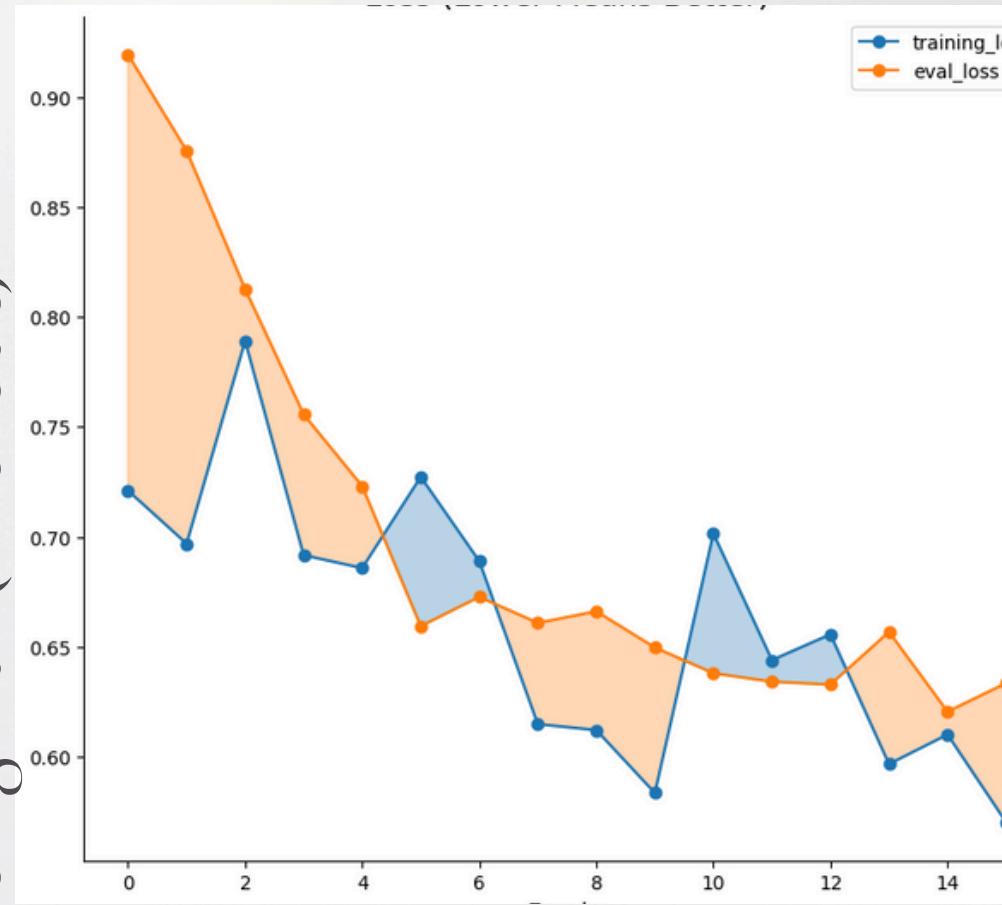
- $a(i)$ = average distance in same cluster (cohesion)
- $b(i)$ = average distance in nearest cluster (separation)
- $s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$

unsupervised learning

with clustering



original (without)



please note that while we found using 30% of our data was better, we will continue with 10% to optimize time efficiency

parameter fine-tuning

using GridSearchCV, the best parameters are:

dropout_rate: 0.0, weight_decay: 0.0, momentum: 0.95, lr: 1e-2

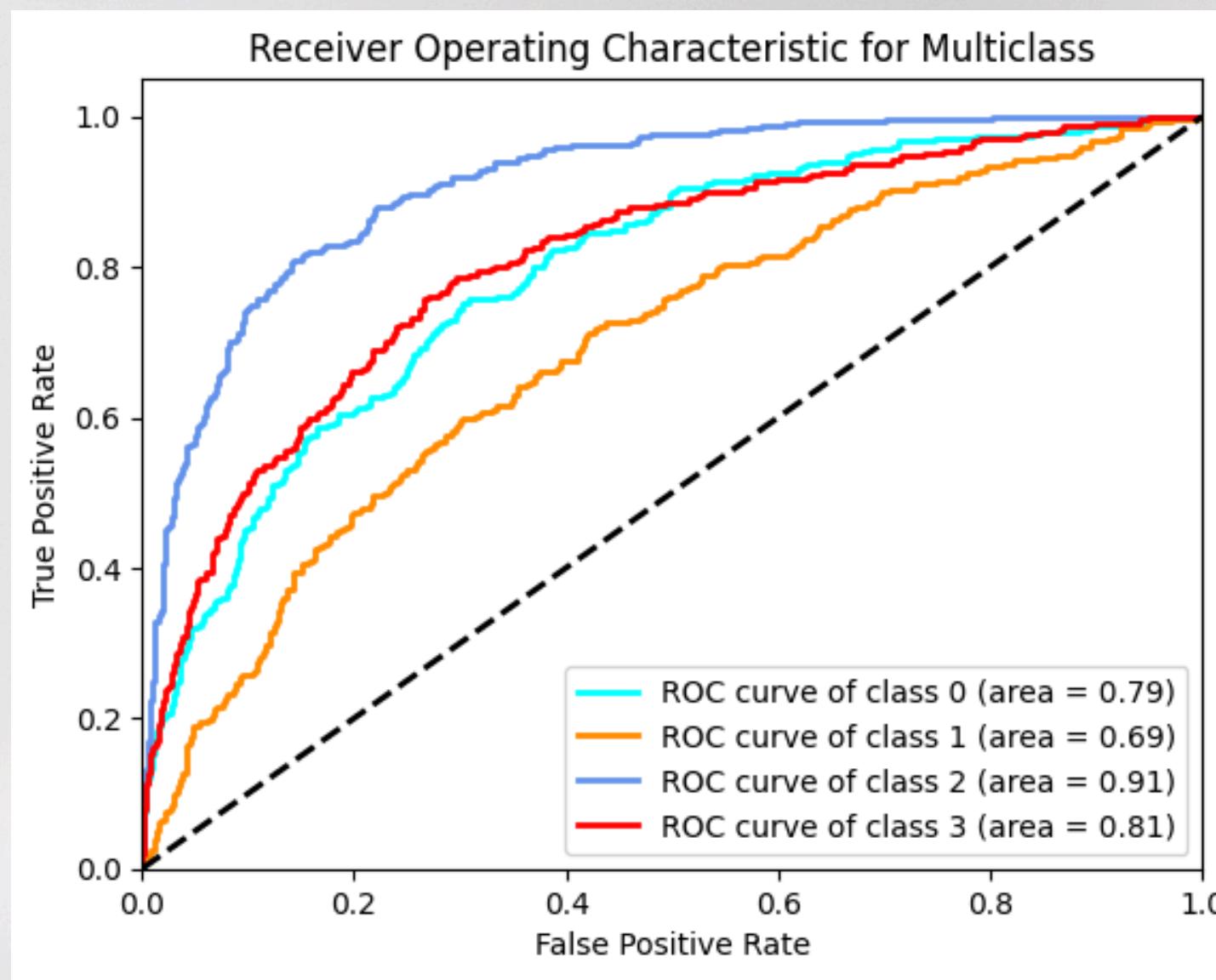
These are the parameters we explored:

```
param_grid = {  
    'dropout_rate': [0.0, 0.1, 0.2],  
    'weight_decay': [0.0, 0.001, 0.01],  
    'momentum': [0.9, 0.95],  
    'lr': [1e-4, 1e-3, 1e-2]}
```

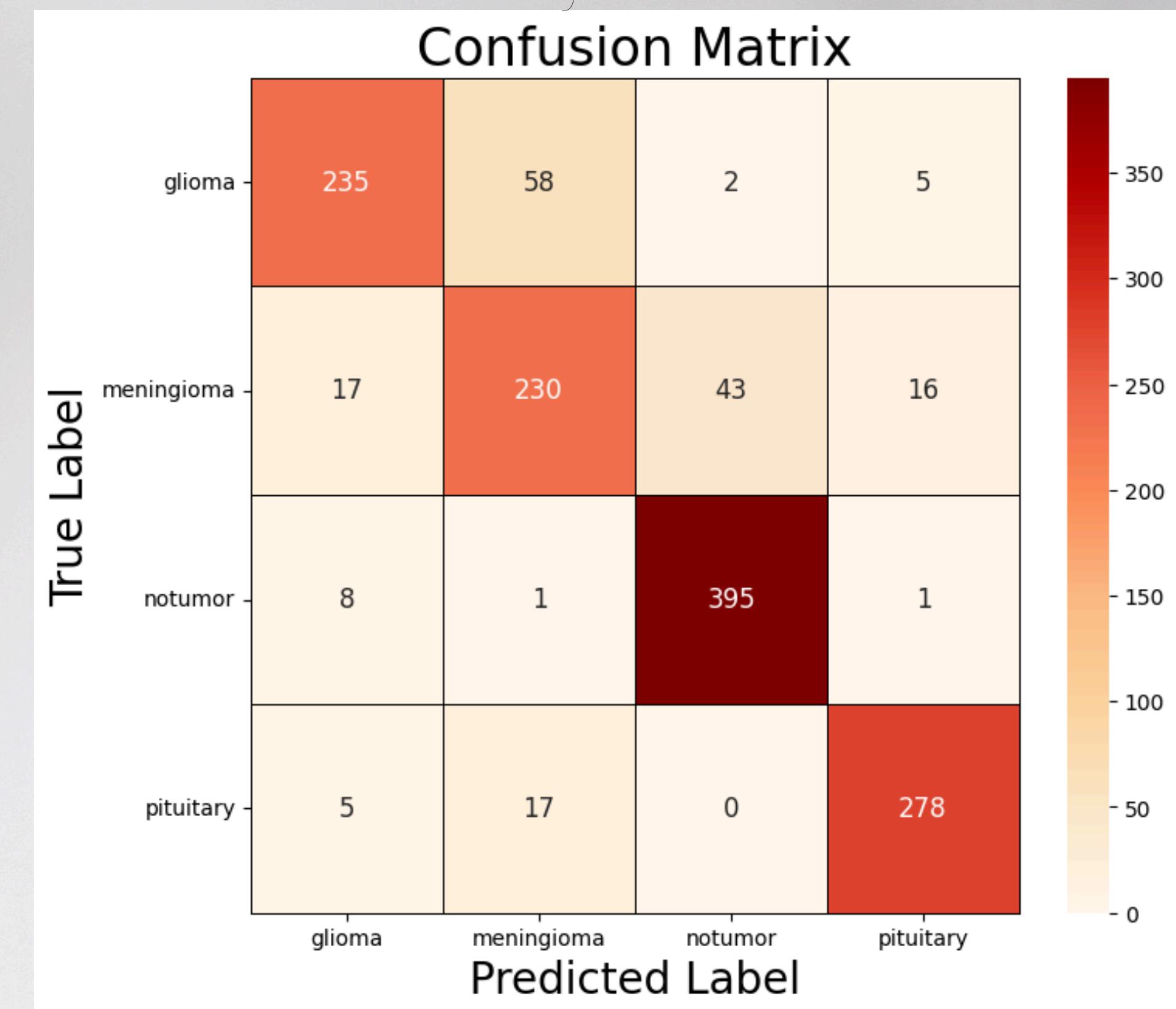
```
Sample 1:  
    True Label: 0  
    Predicted Label: 1  
    Confidence Scores: [0.2502424  0.27024752  0.24365504  0.23585504]  
Sample 2:  
    True Label: 0  
    Predicted Label: 3  
    Confidence Scores: [0.2572371  0.23838969  0.19616205  0.3082111 ]  
Sample 3:  
    True Label: 1  
    Predicted Label: 0  
    Confidence Scores: [0.26567176  0.2408866  0.23631462  0.25712708]
```

- dropout_rate: regularization technique that randomly drops fraction of neurons during training
- weight_decay: regularization technique that penalizes large weights in the model
- momentum: helps accelerate gradients in the right direction and dampens oscillations
- learning rate: step size taken during optimization
- low confidence scores indicate that the model is unsure about its prediction

final model evaluation



Classification Report:				
	precision	recall	f1-score	support
0	0.92	0.67	0.78	300
1	0.69	0.81	0.75	306
2	0.93	0.95	0.94	405
3	0.87	0.92	0.90	300
accuracy			0.85	1311
macro avg	0.85	0.84	0.84	1311
weighted avg	0.86	0.85	0.85	1311



answering the research question

a multi-class classification task that used a pre-loaded EfficientNet Model

- explored how including unsupervised clustering techniques affected the model
- explored different parameters and how to fine-tune them to prevent over-fitting

analysis for the future

want to further analyze this dataset to investigate the feasibility of using symmetry-based methods to estimate the tumor volume from the 2-d MRI scans (a regression task)

thank you!