UNSW SYDNEY

COMP9417 - ML vs Cancer Report

Team - MLHackers

Name	zID
Heeseob Park	z5375037
Seoyeon Park	z5248219
Elvin Lim	z5314041
Ayaan Adil	z5213315
Gordon Lam	z5060305

1. Introduction

The following report details the work of Predictive Solutions. The client has issued the task of providing a machine learning solution for efficiently classifying histologic images.

Histologic images are scans of human tissue magnified under the microscope. These images are traditionally used by pathologists in the course of detecting and analyzing tumors. When an abnormality is detected, it must be appropriately classified to ensure patients are directed to the most effective form of treatment.

Though this practice has been used extensively in both present and past, it comes with number of disadvantages due to its highly labored approach:

1) There are a limited number of professionals in the field who can readily perform this task relative to the demand for this skill. Excessive demand can overburden professionals through extensive classification backlogs and result in significant time taken to communicate findings to the patient
2) Detection and classification are vulnerable to high risk of false positive/negative as a consequence of human error. It is highly dependent on the skill and experience of the pathologist. Misclassification in this manner can lead to serious consequences for the patient.

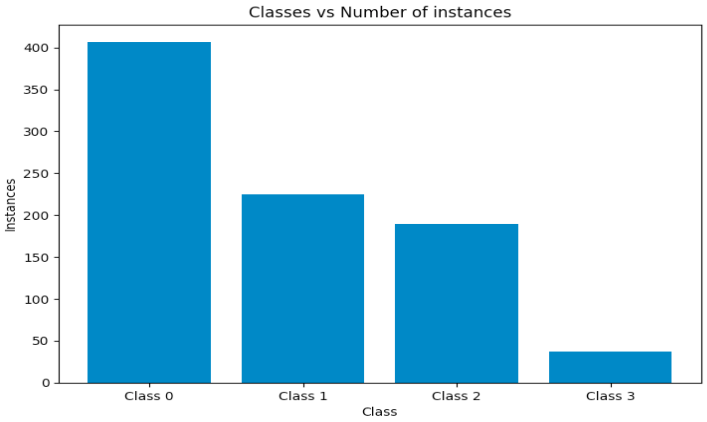
We therefore propose a solution that deploys image classification algorithms with the aim of addressing the issues identified above. To assist with this, the client has provided our team with training and test data which we will use as a basis to train our model and make our predictions.

2. Exploratory data analysis

The client has provided an initial dataset in the form of numpy files. We first convert the X_train_chunks file using a generator function into images that can be stored on a cloud drive as opposed to storing in memory which would not be technically feasible given its size.

X_train.npy	y_train.npy	X_test.npy
22.59GB	8KB	7.22GB

Conversion of the numpy matrix yields 858 images of the resolution 1024 x 1024 as a numpy file along with the associated classification labels. The labels separate the samples into 4 discrete classes 0,1,2,3 where 0 indicates that no tumor is present while classes 1-3 indicates the type of tumor present. The distribution of data is visualized in figure 1 and reveals that there is an uneven distribution of data across the training instances.



	Count
Class 0	407
Class 1	225
Class 2	189
Class 3	37

Figure 1. Count distribution of sample instances against classes and table

We can also display sample images across 4 classes as shown below:

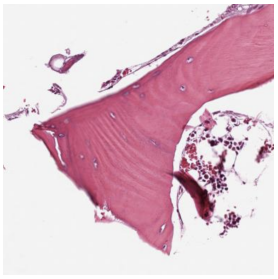
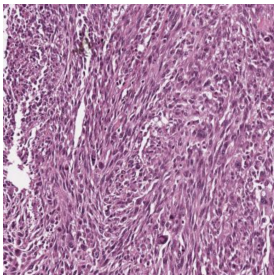
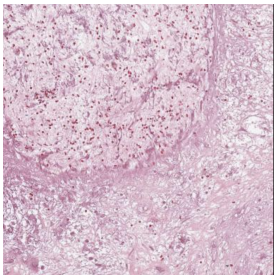
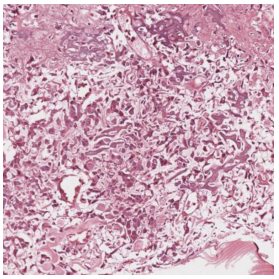
Class 0	Class 1	Class 2	Class 3
			

Figure 2. Example sample instances between classes

3. Data pre-processing

Class grouping

As observed in the exploratory analysis, the distribution of data appears to show a highly uneven distribution of images among the 4 classes with Class 1 having 10x the number of instances as Class 3. This imbalance could result in potential underfitting of our models if too few instances are provided for training and can affect our final test accuracy. As such we split the data into 2 groups with the intention of proportionally distributing Class 3 images proportionally across training phases. The data groupings are as below:

1. Set A: Class 0, Class 1, Class 2
2. Set B: Class 3

Cross validation

In the cross-validation phase, we split our data across 3 key phases post group allocation:

1. **Train**: Data used to train our model
2. **Validation**: Unseen data used for hyperparameters tuning and avoid overfitting
3. **Test**: Unseen data used to assess the performance and accuracy of the model

The allocation of data to each phase is documented as follow:

	Train	Validation	Test	Count images
Set A	80%	10%	10%	821
Set B	65%	20%	15%	37

Table 1. Test data allocation along 3 phases for data sets A and B

As outlined in the class grouping, the split of Set B data across the Training, Validation and Testing phases optimizes the distribution of class 3 data across 3 phases to ensure there are enough training samples for the model to learn from. We implement this split by applying a seeded random state on the sklearn function train_test_split() across both sets A and B and then afterwards, combining both sets back together. In total, 5 samples were created for training, validation and testing for each resolution x224 and x384.

Image format

PNG and JPG format was tested. The result was not different (not shown). So JPG is used since it is much smaller than PNG and has no additional alpha channel.

Image scaling

The initial size of the images in the dataset was 1024 X 1024 pixels which is computationally taxing to process as during the training phase, these individual pixels, channels and number of images would be transformed several times depending on the model used. For a single image of this size, the total feature that needs to be stored would be a function of its pixel height, pixel width and color channel size and will result in the following number of features: $\text{Pixelwidth} * \text{Pixelheight} * \text{No. channels} = 1024 * 1024 * 3 = 3{,}145{,}728 \text{ features}$!

We scaled the images down to a smaller resolution for two main reasons:

1. To improve efficiency of run-time during training and evaluation for all 858 instances
2. Accommodate input size requirements for pre-trained pytorch models for both model Vision Transformer (ViT) and Convolutional neural network architecture (CNN) e.g. ViT requires an input size of 384 x 384 and CNNs requires 224 x 224

No additional modifications to the aspect ratio or cropping was required.

Normalization of pixel (brightness)

The image pixels were normalized across the 3 channels for all images to ensure there is a similar distribution of image brightness. Normalization is crucial during training as we will be multiplying weights and adding bias to individual features and this may affect the activation of layers. This impacts the backpropagation process and the gradient optimiser i.e. without normalization, the convergence of loss to minimization would take much longer. We use the recommended means and standard deviations per channel from Pytorch as below:

```
Mean: [0.485, 0.456, 0.406]

Standard deviation: [0.229, 0.224, 0.225]
```

These values are calculated from millions of images from the ImageNet collection but for comparison, we also computed our own means/std within several samples (in appendix) to observe for any improvements in accuracy, but found that accuracy for the recommended normalization values worked best.

Data Augmentation

Two types of augmentation will be applied to the dataset aside from normalization. The first augmentation will be transforms.RandomResizedCrop(input) which will take a random sized crop of the image input size. This is included as a requirement to implement the models which we will later consider. The second augmentation is transforms.RandomHorizontalFlip() which entails the input matrix being randomly selected for horizontal flipping. This is done in the event of uneven distribution of images being presented in a particular horizontal direction and can contribute to better accuracy by treating direction agnostically during learning.

4. Methodology

In this section, we provide detail into the types of models and image classifications architectures used for our prediction model. We also define how to measure success in our model performance and briefly outline the hyperparameters that we will consider tuning in the validation phase.

Image Classification Architecture

For our architecture, we considered both convolutional neural networks (CNNs) and Vision Transformer (ViT) architecture. Both CNNs and ViTs have numerous documented use-cases for image recognition tasks particularly in the space of histologic image classification such as Esophageal cancer diagnosis using CNN (Hori, et al 2019) and ViT for Breast ultrasound imaging (Gheflati & Rivaz, 2022). In practice, ViTs have been known to outperform CNNs by almost 4 times for image recognition in terms of efficiency and accuracy. We will examine models using both architectures in our analysis and compare the difference in performance through evaluation metrics in later sections.

At first, we built the CNN model ourselves rather than using pre-trained models (1). We created 2 convolutional layers, 1 pooling layer, and 2 dense layers. We chose relu as our activation function for both convolutional layers to avoid the vanishing gradient problem and set the padding to preserve its size. Also, we used MaxPooling 2D to decrease the size of inputs. Again we used the relu activation function on the first dense layer. 100 outputs were passed to the second dense layer as the next dense layer's inputs.

Activation function, Loss and optimisation

We used the softmax activation function for the last layer to return 4 outputs which represent the 4 classes that we label images. We also elected to use the cross-entropy function as the loss function to satisfy the multi-classification problem (2). The cross entropy function is used to calculate a score that summarizes the average difference between actual and the predicted probability distribution for all classes from which it chooses the best class to fit the image. We used SGD for the optimization process to calculate the minimal loss parameters used to train the model. Note that we also opted to use pre-trained models which are preconfigured with their own activation and loss functions as seen in figure 3.

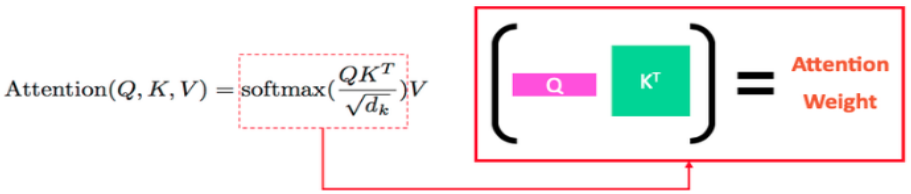


Figure 3. Attention mechanism for ViT with softmax as the activation function

Pre-trained models

The PyTorch and Torchvision module provides a host of different Convolutional Neural Network (CNN) and Vision Transformer (ViT) models. B_16_imagenet1k which have been pre-trained on the imageNet dataset (21k) and fine-tuned on ImageNet(1k)(Melas, L., 2020). The benefit of using pre-trained models is self-implicit, as it has already been trained on millions of data points and thus we can readily use the resulting weights and bias as a baseline for initial training as opposed to calculating these from scratch. Secondly, Pytorch provides a skeletal implementation of these models in their fine-tuning tutorial (Inkawhich, N., 2017). We decided to use the pytorch fine-tuning tutorial code as our starting point and modified it to suit our project purpose as it will allow us to focus more of our effort on hyperparameter tuning. In summary, we will consider pre-trained models in our comparison and similarly apply the same loss and optimiser as mentioned above (Cross-entropy loss and SGD).

Feature extraction

Feature extraction in the context of image classification refers to the process of reducing dimensionality of features into smaller groups that add the most weight to the model. This is achieved within the layers of the model which create image maps (subset of image). Between CNN and ViT architect, the method at which these 'features' are extracted will vary from between the layers used. For example in the figure 4, convolutions in CNN will identify fixed sizes of the image which will later be given more weight during the training process for more prominent and common features. In ViT, the attention mechanism is embedded in the reshaped layer and is the driver for feature recognition.

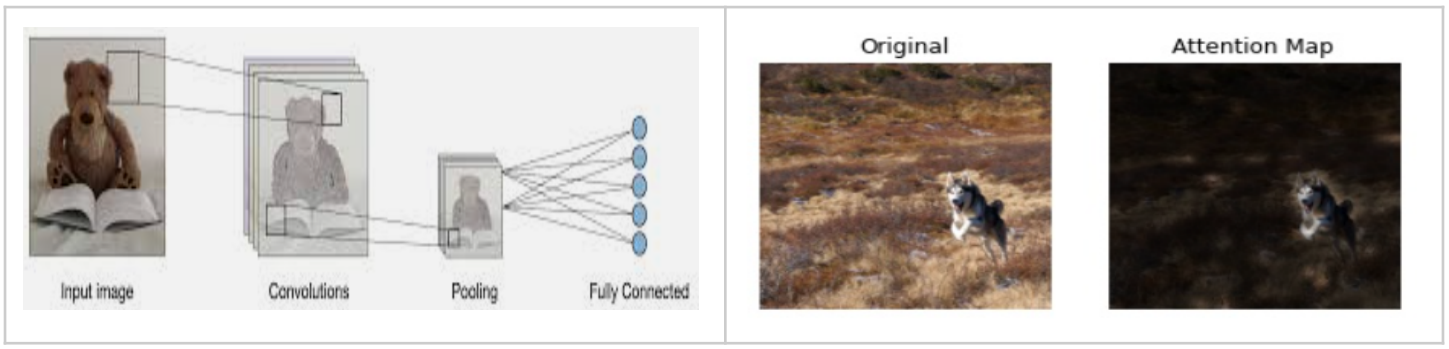


Figure 4. CNN (Left) and ViT(Right) example of feature extraction

In our approach, we will initially run the model including all features initially by setting the feature_extract flag as False. Where we suspect there is potential under/overfitting (i.e. significantly poorer validation and test accuracy vs training accuracy) after the model has been trained, we will evaluate feature extraction on different layers to reduce features in the final model.

Model Evaluation

To understand the differences between different models and outputs, we have produced several evaluation metrics that are essential to ensure that the outcomes of our predictions are in line with addressing the problem space and minimizing both type I (False Positive) and type II (False Negative) errors.

- Accuracy - This is calculated by dividing the number of correct predictions by the total number of predictions. We used it as an initial estimate to get an idea of how well the overall model is performing. However, given the uneven distribution of samples, accuracy can be a misleading measure as it does not provide measurement on an individual class level.
- Recall (type II error) - Recall was chosen as another measure because the cost of false negative misclassification in this case is quite high. If an image belonging to classes 1,2 or 3 is misclassified as class 0, it can lead to the patient not receiving any treatment for the tumor and worsen long term prognosis. For this reason we are interested in tracking this metric.
- F1 score - This metric is useful for evaluating multiclass classification problems that have an uneven class distribution as it is sensitive to type I and type II errors. Models are rated less favorably when there is a high number of either errors.
- F1 score (class) - Measure F1 score on an individual class basis to give a better indication on its class prediction performance. This is especially of great interest as this can be used to assess the impact of separating Class 3 data into two groups prior to distributing them across the 3 phases.

During the testing phase, these metrics will be implemented through the specifically designed test_model_eval() function across each model output. We use the insight gained at this stage to determine which model we will use.

Hyperparameters

Hyperparameter tuning is an essential stage in improving model efficiency and unlocking the potential of a model to perform better. Tuning will occur post the model selection stage after it is scored using the evaluation metric mentioned above. We focus our attention on the following hyperparameters:

- **Batch Size** - The number of images selected randomly to undergo stochastic gradient descent. Increasing batch number can improve the gradient descent process but is computationally expensive.
- **Learning Rate** - The learning rate controls the rate of change in the model's weight post update of weights during the learning process.
- **Momentum** - Momentum allows our descent algorithm to build inertia in a direction of a search space by controlling the amount of "history" added into the update equation.
- **Feature Extraction** - Feature extraction was a hyperparameter which when set to true, results in updates in only the reshaped layer parameter of the model.
- **Epoch size** - One pass on all training instance in SGD optimiser

The approach for testing hyperparameters will be to vary each parameter during the learning process with a number of different values and ultimately assess validation accuracy for performance.

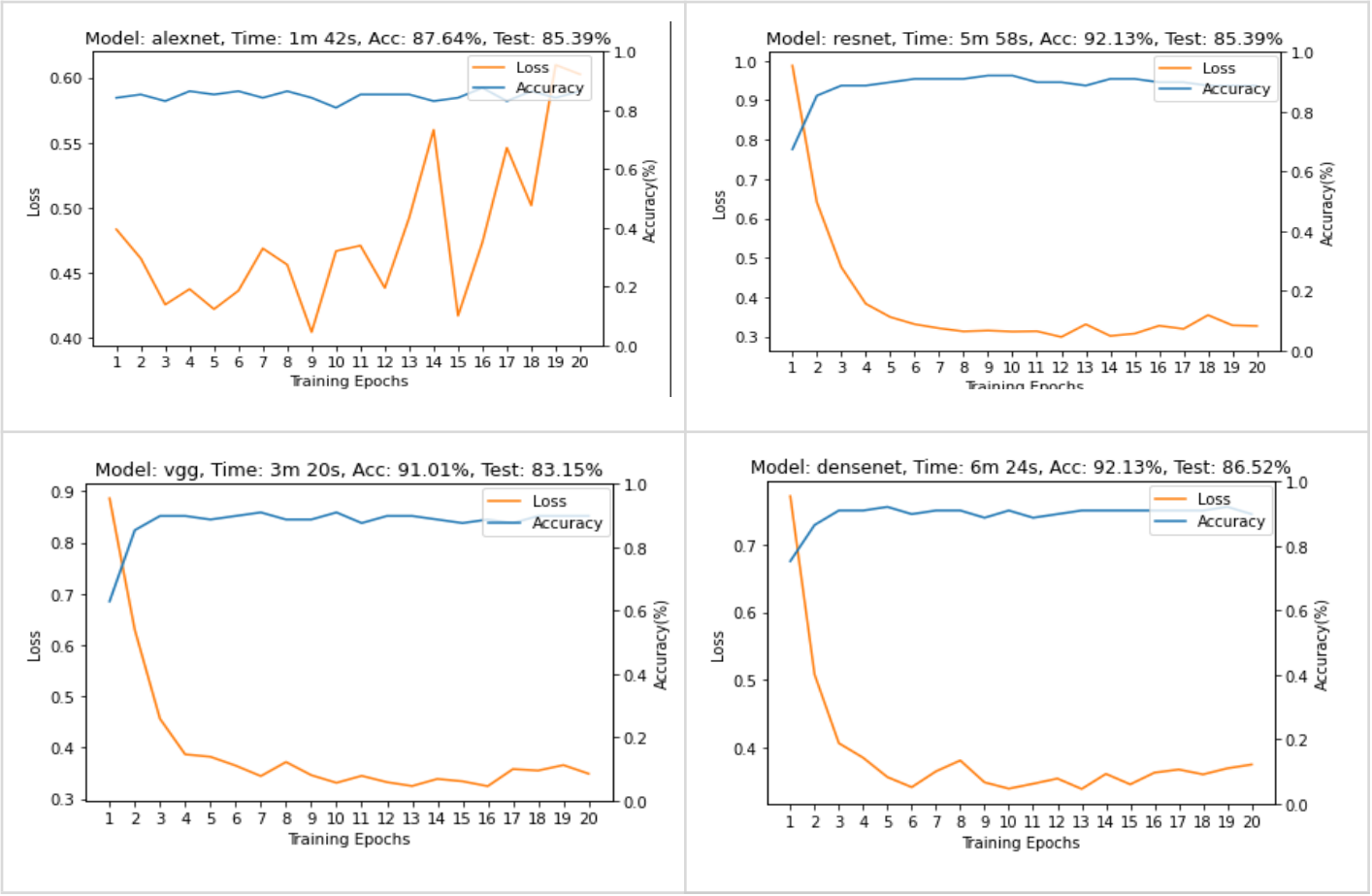
5. Results

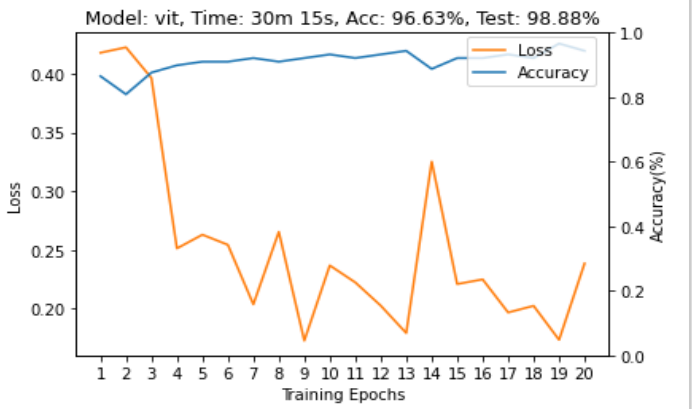
Model training process

For the training process and evaluation process, each model had undergone the follow steps:

1. Initialized global variables
    a. Batchsize = 43 for CNN and 5 for ViT
    b. Classes = 4
    c. Epochs = 20
    d. Feature extract = False (Using all features initially)
2. Initialized pre-trained model for both CNN and ViT through initialize_model() and setting Pre-trained = True (Used pretrained weights for each model)
3. Pre-processed data - Normalization/Augmentation (see pre-preprocessing section)
4. Fit model to GPU for training
5. Declared the parameters that required optimisation (All if feature extract is set to false)
6. Fitted parameters to SGD optimizer and initialized cross-entropy loss function.
7. Executed train_model() to train pre-processed data and immediately checking loss/accuracy against training and validation datasets
8. Execute test_model_eval() to retrieve the evaluation metrics for the validation/test phase

Performance overview





Training parameters:

Model	Batch size	Training size	Input size
alexnet	43	684	224
resnet	43	684	224
vgg	43	684	224
densenet	35	684	224
vit	5	684	384

Figure 5: Run-time of candidate models loss and accuracy values over 20 epochs.
Note: 'Acc' refers to validation accuracy and 'Test' refers to test accuracy.

Metric evaluation and model selection

		alexnet	resnet	vgg	densenet	ViT
Evaluation Metrics	**Test Acc.**	85.39%	85.39%	83.15%	86.52%	**98.88%**
	Recall	92.68%	87.80%	80.49%	95.12%	**100.00%**
	F1	83.12%	81.74%	79.80%	83.77%	**98.91%**
	F1 class 0	91.49%	94.00%	91.26%	93.62%	**100.00%**
	F1 class 1	89.36%	88.00%	82.61%	84.62%	**98.18%**
	F1 class 2	80.00%	63.64%	69.57%	84.62%	**100.00%**
	F1 class 3	0.00%	0.00%	0.00%	0.00%	**92.31%**
Diagnostics	**Valid. Acc.**	87.64%	92.13%	91.01%	92.13%	**96.63%**
	Time taken	**1m 42s**	5m 58s	3m 20s	6m 24s	30m 15s
	Best epoch	9	12	13	**6**	9

Table 2: Evaluation of vgg, densenet & vit model against defined metrics and run-time diagnostics

The evaluation table in table 2 summarizes the key metric findings across all candidate models and clearly shows ViT to be the superior model across most of the metrics considered. Taking a closer examination of the recall and F1 metrics, we can see that ViT was highly effective in correctly classifying all classes of test images with an astonishing overall F1 of 98.91%. For the purposes of hypertuning - we will focus our attention on this model and discuss our rationale on selecting the ViT model in the discussion

Hyper parameter optimization (fine-tuning):

We conduct experimentation for each hyperparameter and use previous parameters in the model selection stage as a control. The parameter that produces the highest accuracy while controlling for all other variables

will be used in the final model. Control parameters (for elected ViT model): Batch Size = 5, Learning Rate = 0.01, Momentum = 0.9, Feature Extraction = False.

Batch size

The batch size of data trained during the optimisation process was hypothesized to improve the epoch accuracy sooner than smaller batch sizes; however, no meaningful insights could be inferred. A second run was performed with the complete training set as seen in figure 6 and the results support the hypothesis. For a larger batch size, the accuracy improved faster than smaller batch sizes without trade off in efficiency but there was no significant difference in performance between batch sizes 5 and 7 (upper GPU limit). Based on these results, we conclude that a batch parameter of 7 would be used for the final model.

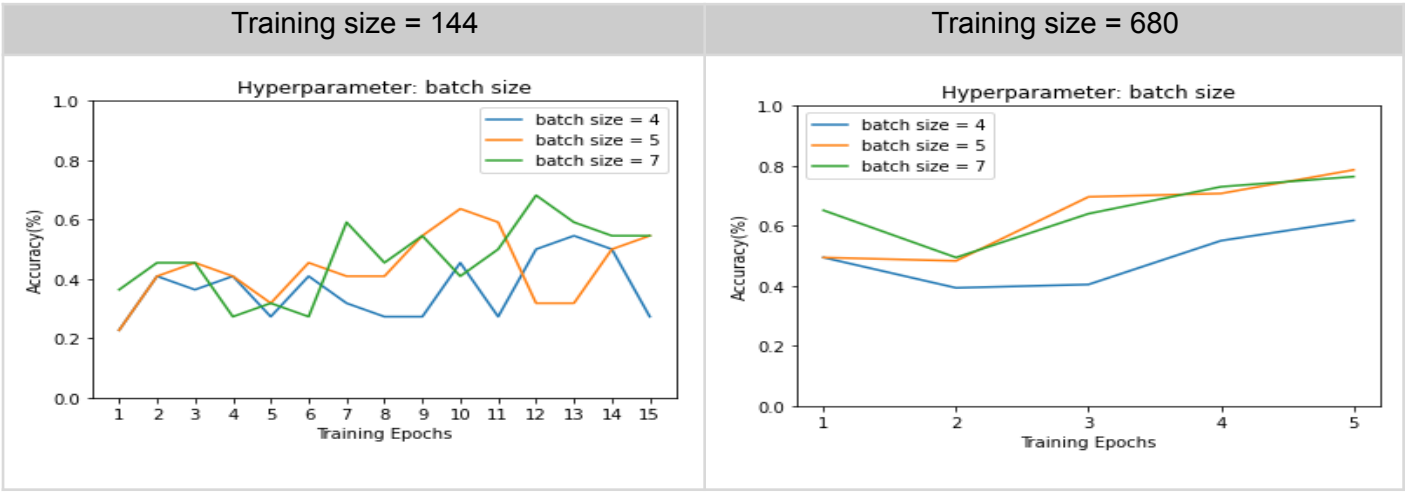


Figure 6: Measuring change in accuracy for different batch sizes. Training size = 144 for left image and 680 for right

Learning rate

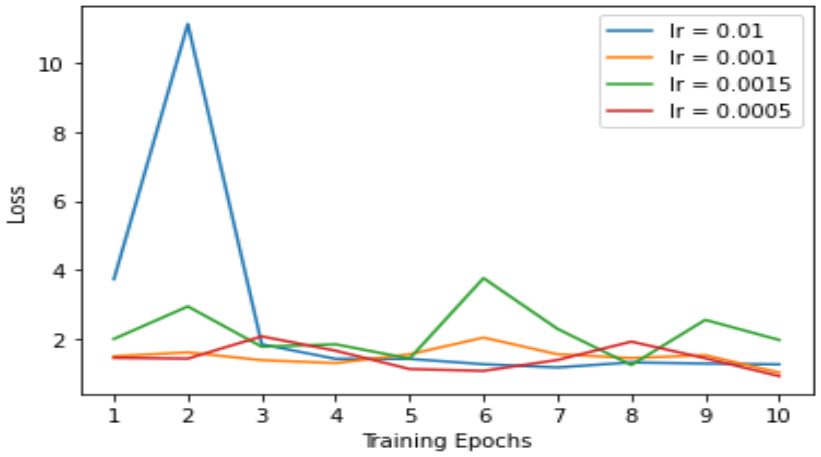


Figure 7. ViT Model performance when tuned with different learning rates

When set at batch_size = 5, we observe that the epoch loss steadily reaches a plateau for smaller learning rates. When a learning rate of 0.01 was used (the largest) we observed stark jumps initially before falling quickly - which indicates a faster convergence. All learning rates appear to be relatively on par with smaller values doing marginally better. For the purpose of the prediction run we will set our learning rate to 0.001.

Momentum

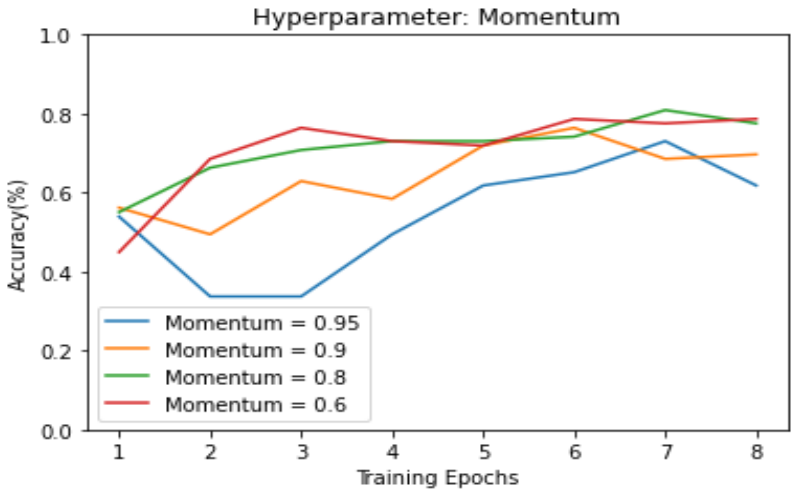
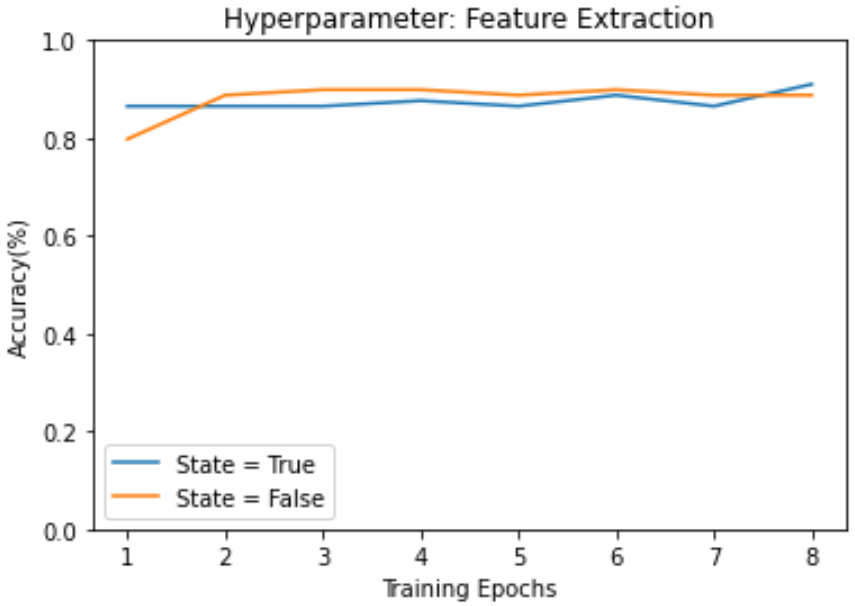


Figure 8. ViT Model performance when tuned with different momentum

We observe that during testing of different values, lower momentum values show much better accuracy improvement up to a point between 0.6 and 0.8 where the difference becomes marginal. For this parameter, we take the average and use 0.7

Feature Extraction




```
Test Accuracy: 84.27%
Recall: 90.00%
Weighted f1: 83.03%
f1 for class 0: 91.36%
f1 for class 1: 81.48%
f1 for class 2: 88.24%
f1 for class 3: 22.22%
```

```
Epoch 19/19
train Loss: 0.3354 Acc: 0.8735
valid Loss: 0.3491 Acc: 0.8876

Training complete in 20m 28s
Best val Acc: 0.898876
```

Figure 9. ViT Model performance with feature extraction set to true and evaluation metrics

The overall effect on accuracy is worse off when we set feature extraction to **true** (meaning only the reshaped layer parameter of the model is updated). Whilst we saw huge improvements to the run-time for training (about 33% faster), the overall accuracy never exceeded the 90% mark. In fact, even after extended to 20 epochs (as shown in figure 1), it still suffered a significant drop across all F1 measures. Thus, we decide to keep feature extraction to false for the final prediction run.

6. Discussion

Model comparison, evaluation and selection

The overall performance of our ViT model as demonstrated during the testing phase far outperforms competing CNNs models with respect to its accuracy as seen in table 2 in the results section. The primary reason to why the ViT was selected however was for two main reason:

1. High validation and test accuracy - As we want to know if the model is appropriately fitted, we compare the high validation and test accuracy score to see if there was evidence of overfitting. For each of the pre-trained CNN models, the test score reported lower accuracies to their training and validation values but conversely, ViT reported much better values for both which is a positive indicator for good fitting.
2. F1 Score - For all classes, the ViT score achieved the best result. Most surprising however was how poorly CNN's model performed for Class 3 predictions - 0% for all CNN models. This could be attributed to uneven distribution of data (37 instances in class 3) amongst classes which we had earlier attempted to address during preprocessing. In retrospect, we may need to allocate a higher proportion of class 3 samples into the training data to see improvements. For ViT however, our allocations worked very well and produced very good predictions.

Comparison of different methods, their feature and performances

As mentioned in the methodology, we attempted to construct the CNN architecture from scratch which would be beneficial in terms of tuning, feature selection/extraction and future enhancements. When testing our model however, we only achieved a test accuracy of 59% which suggests that a more intricate architecture needs to be considered (3). For this reason, we concluded to use the pre-trained models as opposed to building it ourselves and to take advantage of the huge imageNet database used to pretrain the weights in the model.

Ensemble methods

We constructed an ensemble model with pytorch as the pre-trained ViT model had the best accuracy among models we tried. We trained 5 additional VIT models on the same image dataset but each time redistributing the images randomly into training, validation and test sets to account for overfitting. Each model achieved 96%, 96%, 97%, 94%, 95% accuracy respectively. We use the max voting ensemble method(4). Max voting is one of the ensemble methods which return the most common class among various model's outputs. To check the model's performance, we insert each class dataset to the ensemble model and test whether the model classifies each class well or not. Our results perfectly predict class 1 and class 2 datasets and achieve great results for class 0 and class 3 as seen in figures 10 and 11 respectively.

```
results class 0: [0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
results class 1: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
results class 2:  [2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
results class 3: [3 1 3 3 3 2 1]
```

Figure 11. Result output Class 0, 1, 2 and 3. Shows images in class 0 and 3 with imperfect predictions to other classes

Final model

On comparing our ensemble model to a single trained ViT model, we find that ViT5 (model for 5th dataset) performed better by achieving a higher prediction accuracy. As the results below show, the ViT5 model perfectly classifies class 0 whereas our ensemble model misclassified one class 0 instance. In addition, ViT5 perfectly works on class1 and 2 data sets. Furthermore, ViT5 classifies class3 instances better than our ensemble model. We have therefore selected ViT5 as our best of breed model.

```
Real predictions class 0: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0]
Real predictions class 1: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1]
Real predictions class 2: [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
Real predictions class 3: [3, 3, 3, 3, 3, 2, 1]
```

Figure 12. Result output Class 0,1,2,3

Discussion of future improvements

Given more time, the team would look to further improve the model by employing weighted ensemble methods to cover possible shortfalls of using only ViTs. An initial attempt was made without weighing the contribution of different models and this resulted in decreases in accuracy as other participating models did not provide good results inherently. The team used hard voting for our ensemble classifier. Hard voting is the voting calculated on predicted output class. Whereas, soft voting is based on probability of the output class. In practice soft voting is more widely used than hard voting as it deals with probability. This makes our prediction more accurate as we multiply a relatively big number if we have bigger probability vice versa. Thus, for further improvement we will try soft max with unequal weights. By doing so, we could consider the probability and penalize the models depending on their performances. Thus we would improve our model by choosing soft voting as classifier and assigning different weights to each classifier [4].

6. Conclusion

The results of our experiment conclude that the pre-trained ViT models yield the best performance for histologic image classification on average offering accuracies above 95% when tested against curated test data sets. The ViT model was able to produce an outstanding value whereas CNNs models failed terribly in this respect. We also retrained models by shuffling data and this showed that the model was not overfitted. In our evaluation of different models, ViT achieved the highest F1 score among all other classes. This is important to get as accurate as possible as the risks and impact of misclassification are much higher for more advanced classes of tumors. Therefore we recommend using pre-trained ViT as the preferred classification model to support the identification of cancers/tumors in histopathic images as it offers (1) high degree of accuracy and (2) can reduce capacity burdens of manual imaging and review by pathologists. Time constraints limited our capacity to explore further modeling, tuning and ensembling options for both CNN and ViT models which could have led to better prediction outcomes for this task.

Appendix

Manual calculations of sample means/std per channel

	Mean per channel	Std. per channel
samples_x384_all_1	[0.8507, 0.6785,0.7583]	[0,1255, 0.2430, 0.1892]
samples_x384_all_2	[0.8476, 0.6784, 0.7568]	[0.1283, 0.2436, 0.1882]
samples_x384_all_3	[0.8484, 0.6749, 0.7561]	[0.1275, 0.2433, 0.1892]
samples_x384_all_4	[0.8481, 0.6741, 0.7554]	[0.1284, 0.2439, 0.1896]
samples_x384_all_5	[0.8478,0.6723,0.7548]	[0.1271,0.2422,0.1877]

Table 1. Pytorch mean and standard deviation calculated per sample given

Multi-class Cross-Entropy Loss

$$cost(W) = -\sum_{j=1}^{k} y_j log(p_j)$$

The above is the formula for cross entropy. 'j' here refers to the class, p_j and refers to the probability obtained from the activation function (Soft max in this case). y_j are the truth values.

To generally describe each architecture:

Pre-trained vs scratch comparison

We illustrate the performance between pre-trained and scratch models as discussed earlier in the methodology and clearly show that pre-trained models achieve better overall stability during gradient descent and on average, better accuracy.

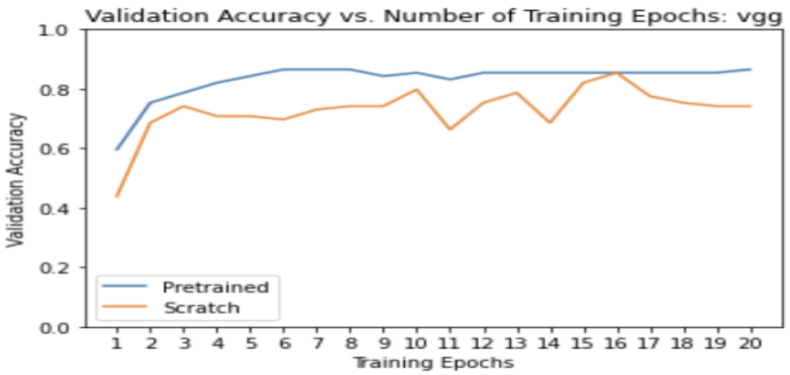


Figure 1. Validation accuracy for pre-trained model (vgg) vs scratch model

Pytorch

With the decision of using pre-trained models came the idea to use Pytorch to find pre-trained models. Pytorch provides a multitude of pre-trained models which can be experimented with easily. It is based on the Torch library and is designed to provide flexibility and high speeds for deep neural network implementation.

Google Colab

We were unable to handle the massive amount of data for training and testing on our local machines. Hence, we decided to use Google Colab. Google Colab allows users to run python code on the web browser. It allows us to use data stored on the cloud and this meant we didn't have to download the entire 8 GB dataset. It allows a large amount of free compute power, this was very helpful in training our models.
More importantly, Google Colab allows users to access free GPUs.
Gpu resolves the memory issue and speeds up the running progress.

(1) CNN model architecture

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 256, 256, 6)       168

 conv2d_3 (Conv2D)           (None, 256, 256, 16)      880

 max_pooling2d_1 (MaxPooling  (None, 128, 128, 16)     0
 2D)

 flatten_1 (Flatten)         (None, 262144)            0

 dense_2 (Dense)             (None, 100)               26214500

 dense_3 (Dense)             (None, 4)                 404

=================================================================
Total params: 26,215,952
Trainable params: 26,215,952
Non-trainable params: 0
```

(2) Loss function, optimizer in codes for implementation of CNN

```
model.compile(loss = 'sparse_categorical_crossentropy',
              optimizer = 'sgd',
              metrics = ['accuracy']
             )
```

(3) Our CNN model's result

```
Epoch 1/30
5/5 [==============================] - 11s 2s/step - loss: 2.7889 - accuracy: 0.3300 - val_loss: 1.1696 - val_accuracy: 0.5900
Epoch 2/30
5/5 [==============================] - 8s 2s/step - loss: 1.2095 - accuracy: 0.5100 - val_loss: 1.2808 - val_accuracy: 0.5100
Epoch 3/30
5/5 [==============================] - 7s 1s/step - loss: 1.3849 - accuracy: 0.4150 - val_loss: 1.1552 - val_accuracy: 0.5100
Epoch 4/30
5/5 [==============================] - 7s 2s/step - loss: 1.5275 - accuracy: 0.4150 - val_loss: 1.2806 - val_accuracy: 0.6200
Epoch 5/30
5/5 [==============================] - 7s 1s/step - loss: 1.3503 - accuracy: 0.5200 - val_loss: 1.3430 - val_accuracy: 0.2500
Epoch 6/30
5/5 [==============================] - 7s 2s/step - loss: 1.3003 - accuracy: 0.5050 - val_loss: 1.2647 - val_accuracy: 0.5000
```

(4) Further improvement example

For example let's say we have below probability distributions for each class (class0 - class3).

classifier 1: [0.6, 0.1, 0.0, 0.3]

classifier 2: [0.4, 0.2, 0.1, 0.3]

And let's assign [0.7, 0.3] as weights for each classifier. Thus , classifier1 is much more important than classifier2. Then the probabilities for each class will be calculated like below.

class1: $0.7 \times 0.6 + 0.3 \times 0.4 = 0.54$

class2: $0.7 \times 0.1 + 0.3 \times 0.2 = 0.13$

class3: $0.7 \times 0.0 + 0.3 \times 0.1 = 0.03$

class4: $0.7 \times 0.3 + 0.3 \times 0.3 = 0.30$

Therefore, we conclude that class 1 is our class as it has the highest probability which is 0.54.

References

Anon., 2022. Vision Transformer (ViT). [Online]

Available at: https://huggingface.co/docs/transformers/model_doc/vit

[Accessed 5 April 2022].

Boesch, G., 2022. Vision Transformers (ViT) in Image Recognition – 2022 Guide. [Online]

Available at: https://viso.ai/deep-learning/vision-transformer-vit/

[Accessed 4 April 2022].

Brownlee, J., 2020. Understand the Impact of Learning Rate on Neural Network Performance. [Online]

Available at: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

[Accessed 10 April 2022].

Inkawhich, N., 2017. FINETUNING TORCHVISION MODELS. [Online]

Available at: https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html

[Accessed 31 March 2022].

Korstanje, J., 2021. The F1 Score. [Online]

Available at: https://towardsdatascience.com/the-f1-score-bec2bbc38aa6

[Accessed 10 April 2022].

Kumar, A., 2020. Hard vs Soft Voting Classifier Python Example. [Online]

Available at: https://vitalflux.com/hard-vs-soft-voting-classifier-python-example/

[Accessed 1 April 2022].

Massa.F & Massa, F., 2017. How to extract features of an image from a trained model. [Online]

Available at: https://discuss.pytorch.org/t/how-to-extract-features-of-an-image-from-a-trained-model/119

[Accessed 31 March 2022].

Melas, L., 2020. PyTorch-Pretrained-ViT. [Online]

Available at: https://github.com/lukemelas/PyTorch-Pretrained-ViT

[Accessed 5 April 2022].

Wang, P., 2022. vit-pytorch. [Online]

Available at: https://github.com/lucidrains/vit-pytorch

[Accessed 5 April 2022].

Yoshio, T. et al., 2019. Diagnostic outcomes of esophageal cancer by artificial intelligence using convolutional neural networks. Gastrointestinal Endoscopy, 89(1), pp. 25-32.

Gheflati, B. & i Rivaz, H, 2022. Vision Transformers for classification of breast ultrasound images. Department of Electrical and Computer Engineering, arXiv:2110.14731v2