Applications of Deep Learning in Healthcare: Breast Cancer Classification using Deep Learning models

Rishav Agarwal (ra3141)
Rachana Dereddy (rd2998)

COMS 6998 009

Practical Deep Learning
System Performance

Final Project Presentation

Executive Summary

The project employs the use of various Deep Learning (Neural Network) models in order to classify Breast Cancer Image appropriately in its two classes of “Non-IDC” and “IDC”. The work proceeds with the Kaggle Breast Histopathology Images, preprocesses it and employs four different neural net models to find out the one that works best. These models were implemented in two different GPU system scenarios: V100 and P100, so that these models can be compared across systems. Our work starts with implementing the VGG16 model, moves on to AlexNet model, and then we create our own model. Finally, we compare all of these models to the method of Transfer Learning using the ResNet50 model. The model that learns best and provides best solution turns out to be the Transfer Learning model with an accuracy close to 85%

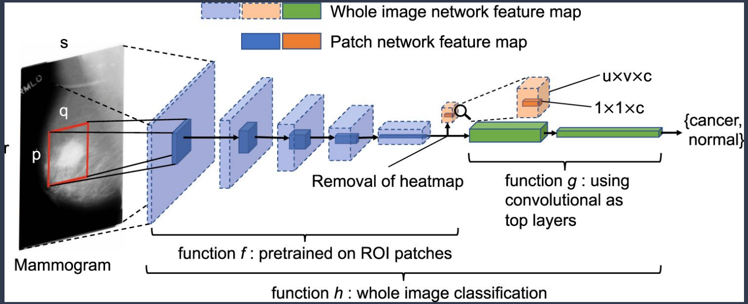
Problem Motivation

If the current scenario of the pandemic is any indication, healthcare industry is a sector that will always be the most important part of this world. Talking about the automation of this industry is not a luxury anymore but rather a necessity.

Deep Learning is a field that contributes heavily to the healthcare sector because of its application via Computer Vision techniques.

We realize that there has been much improvement in the healthcare industry, the most important one should take place in the sector of Cancer image classification, especially, Breast Cancer image classification.

Background Work


Whole image network feature map
Patch network feature map
s
q
p
r
Mammogram
u×v×c
1×1×c
{cancer, normal}
Removal of heatmap
function g : using convolutional as top layers
function f : pretrained on ROI patches
function h : whole image classification

As we dwelled into discussing the applications of Deep Learning in the healthcare industry, the concepts of Computer Vision took the lead. The two domains that grabbed our attention was the application in the detection of Breast Cancer.

1) **Classification of breast cancer histology using deep learning (2018, June): https://link.springer.com/chapter/10.1007/978-3-319-93000-8_95**

2) **SD-CNN: A shallow-deep CNN for improved breast cancer diagnosis (2018): https://doi.org/10.1016/j.compmedimag.2018.09.004**

Dataset

Link to the Dataset:
https://www.kaggle.com/paultimothymooney/breast-histopathology-images

- The original dataset consisted of 162 whole mount slide images of Breast Cancer (BCa) specimens scanned at 40x.
- From that, 277,524 patches of size 50 x 50 were extracted (198,738 IDC negative and 78,786 IDC positive).
- Each patch's file name is of the format: u*X*y*Y*class*C.png —> example* *10253*idx5*x1351*y1101*class0.png* .

  *where u is the patient ID (10253*idx5),

- X is the x-coordinate of where this patch was cropped from,
- Y is the y-coordinate of where this patch was cropped from,
- C indicates the class where 0 is non-IDC and 1 is IDC.

Technical Challenges

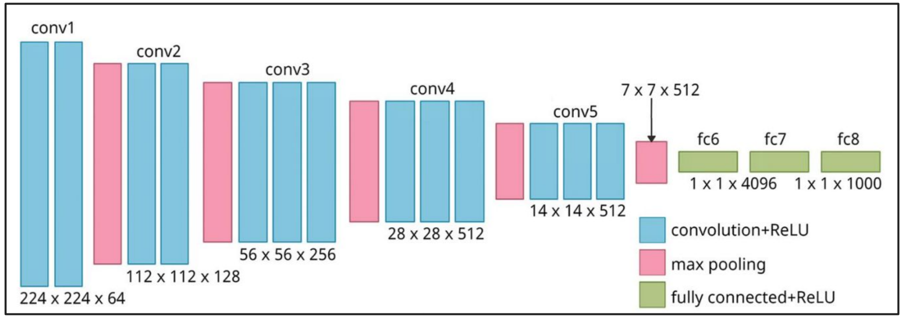
- One of the most important and cumbersome technical challenges that we faced was handling such a huge dataset.
- The solution employed by us was to load the datasets in small segments as Zip in the Google CLoud Console environment, and then unzip it in real-time. We also used the method of Google Drive extraction to eases the load on the local machine.
- Then the other challenge that we faced was how much the data was imbalanced with Class0 taking being the majority class. This problem was solved by using Random Sampling.

Approach (Pre-Processing and Initial Approach)

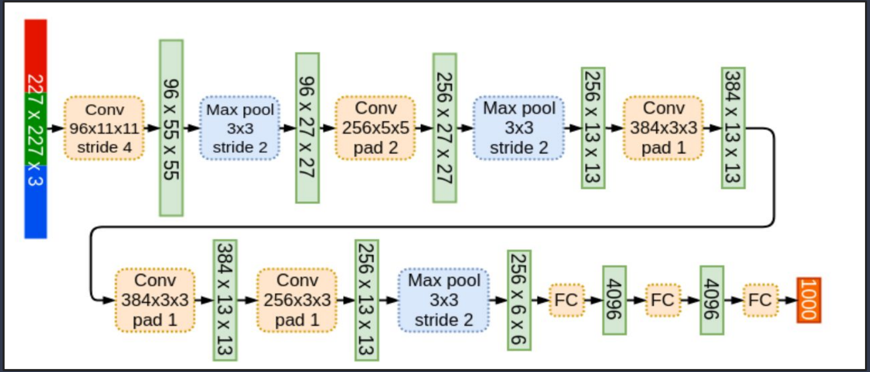
- The dataset that we loaded had a total of 92057 Class0 data points and 36658 Class1 data points.
- As this is highly imbalanced, we proceeded to do random sampling using the function random.sample() and brought each class to the size of the lower sized class, that is, to 36658, so a total of 72000+ data points.
- Post this, we proceeded to load the images and convert them into array for further processing. While creating the X-list, we proceeded to add the labels beside each of the images, which can then be stored in the Y-list.
- We then divided the X and Y to train and test data with a ratio of 80:20, and then proceeded with our models.
- We then decided to employ 4 different models to understand which one works best and also compare it across two different GPU environments namely P100 and V100.

Solution 1: Use VGG16 Model

The **VGG16** CNN model is one of the most prominent vision frameworks which has been employed in countless technology. It has a convolution layers of a 3x3 channel in the first step, while utilizing a cushioning max pooling technique of 2x2 channel in the second step.



Solution 2: Use AlexNet Model

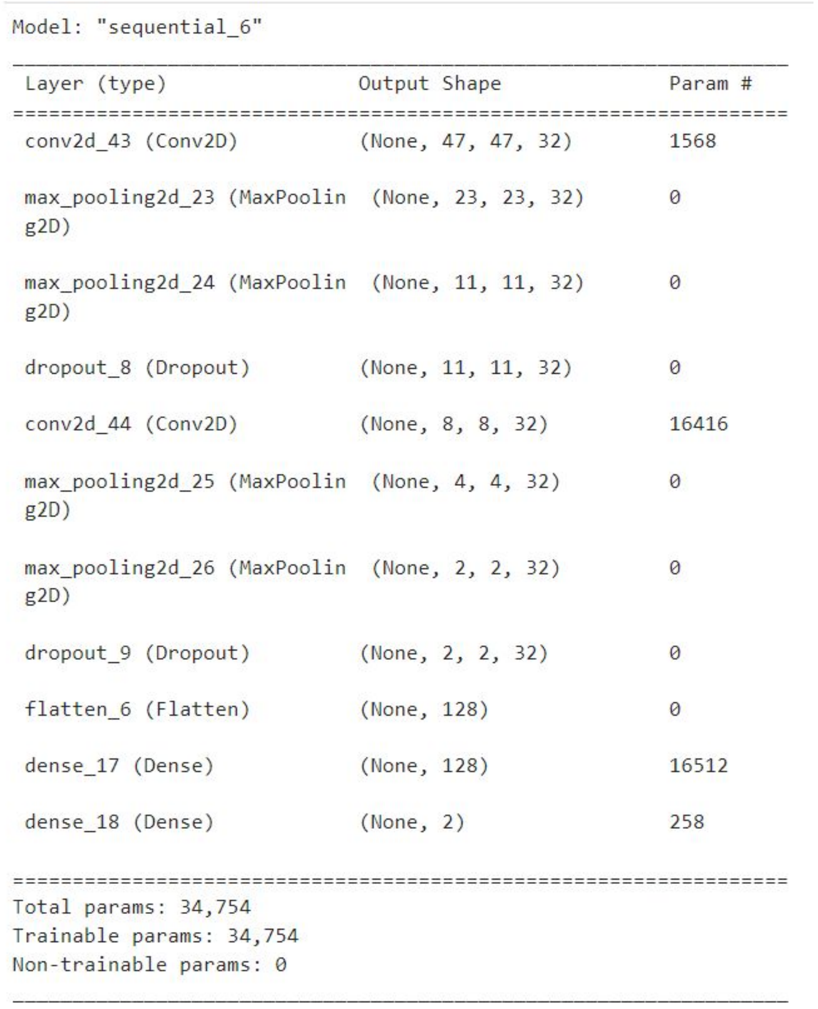


AlexNet architecture consists of 5 convolutional layers, 3 max-pooling layers, 3 fully connected layers, and 1 softmax layer. Each convolutional layer consists of convolutional filters and a nonlinear activation function ReLU. Input size is fixed due to the presence of fully connected layers. AlexNet overall has 60 million parameters.

The 11 layers of AlexNet are:
Layer C1: Convolution Layer (96, 11×11)
Layer S2: Max Pooling Layer (3×3)
Layer C3: Convolution Layer (256, 5×5)
Layer S4: Max Pooling Layer (3×3)
Layer C5: Convolution Layer (384, 3×3)
Layer C6: Convolution Layer (384, 3×3)
Layer C7: Convolution Layer (256, 3×3)
Layer S8: Max Pooling Layer (3×3)
Layer F9: Fully-Connected Layer (4096)
Layer F10: Fully-Connected Layer (4096)
Layer F11: Fully-Connected Layer (1000)

Solution 3: Created our Model

We also used a baseline model with 2 convolutional layers. 2 maxpool layers, 2 dropout layers and 2 dense layers. We used this as a starting point to our model, and used binary cross entropy as our loss, adam as our optimizer, and the metrics is accuracy.

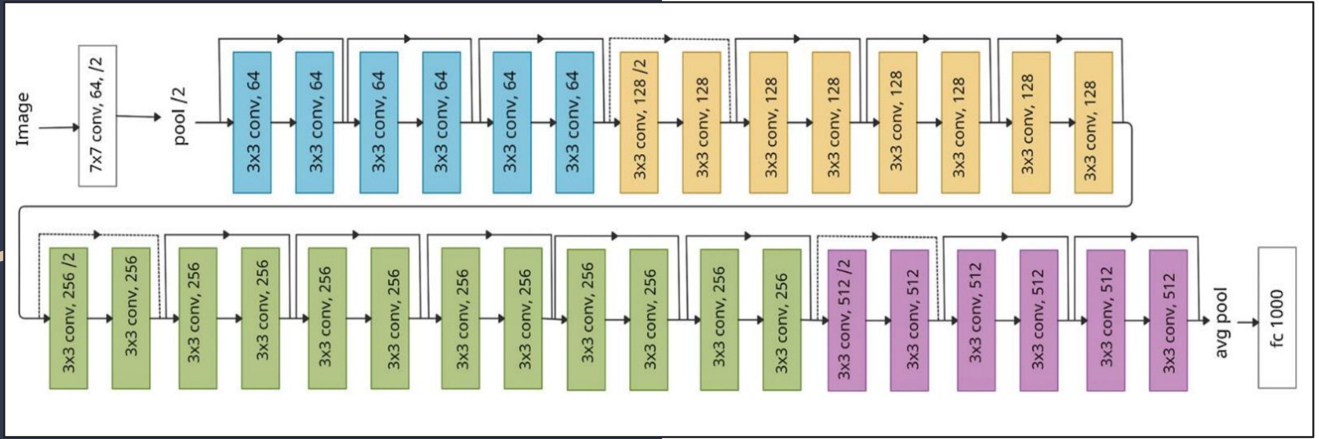


```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_43 (Conv2D)          (None, 47, 47, 32)        1568

 max_pooling2d_23 (MaxPoolin  (None, 23, 23, 32)       0
 g2D)

 max_pooling2d_24 (MaxPoolin  (None, 11, 11, 32)       0
 g2D)

 dropout_8 (Dropout)         (None, 11, 11, 32)        0

 conv2d_44 (Conv2D)          (None, 8, 8, 32)          16416

 max_pooling2d_25 (MaxPoolin  (None, 4, 4, 32)         0
 g2D)

 max_pooling2d_26 (MaxPoolin  (None, 2, 2, 32)         0
 g2D)

 dropout_9 (Dropout)         (None, 2, 2, 32)          0

 flatten_6 (Flatten)         (None, 128)               0

 dense_17 (Dense)            (None, 128)               16512

 dense_18 (Dense)            (None, 2)                 258

=================================================================
Total params: 34,754
Trainable params: 34,754
Non-trainable params: 0
_________________________________________________________________
```

Solution 4: Use Transfer Learning (ResNet50)

Transfer Learning is the method of taking the required parts of a pre-trained model and then using that to learn and predict on different objects. This can be done if the problem statement is similar. For example, our model as pre-trained on the ImageNet dataset of Cifar-10 and then after making the first layer as untrainable we trained the remaining layers with our Breast Cancer Image dataset. This not only saves on computation but also gives better results.

A Residual Neural Network (**ResNet**) is one of the standard architectures of artificial neural networks which expands its similarity to the pyramidal cells in the cerebral cortex of human body. Residual neural networks do this by using skip associations, or easy direction to directly jump towards a higher level layer. The initial stages of ResNet models contained a three layer architecture which included the nonlinearities (ReLU) on the sides and keep all the standardization in between.

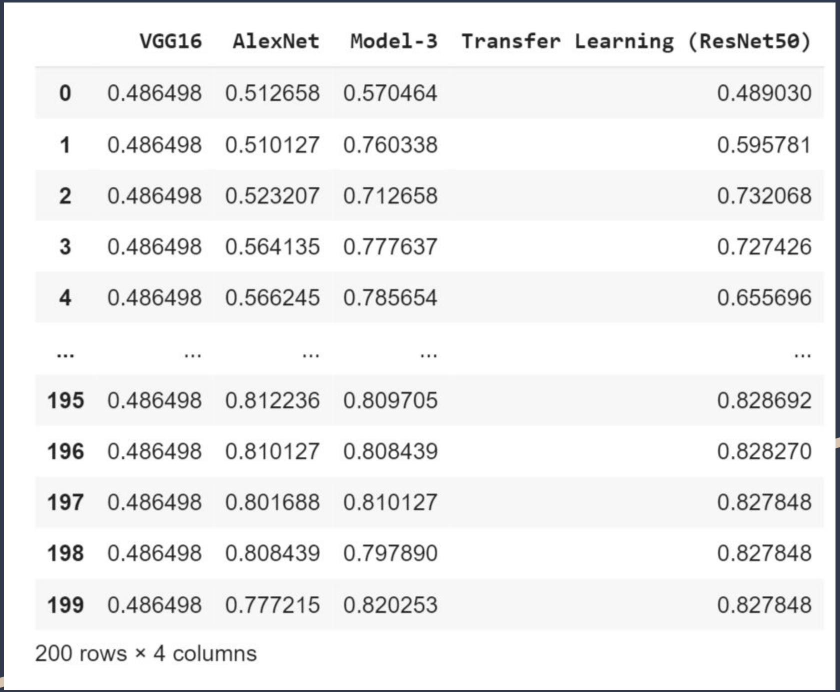


Implementation and Experimental Flow

To way we organized our flow is as follows:

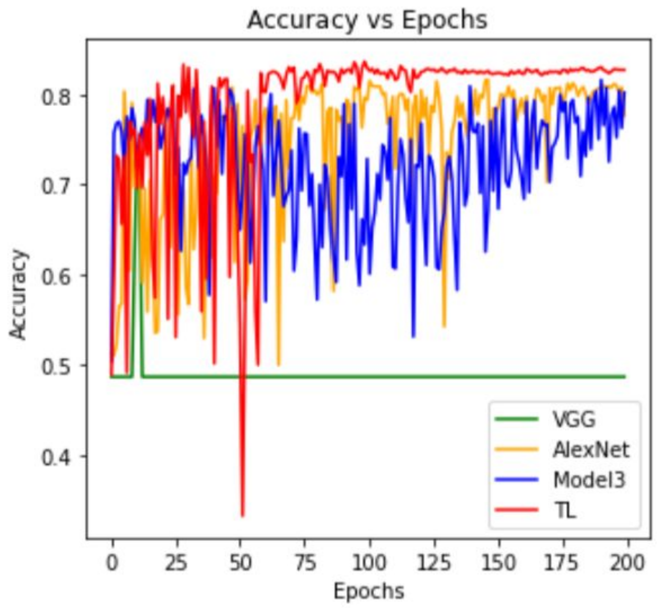
- Loading of the dataset takes place, followed by the pre-processing, including random sampling.
- Then we create the train-test split and proceed with our models. First we implement, the VGG16 model, followed by the AlexNet model.
- Post that we created our own model and see if we can get better accuracy. Finally, we concluded with the implementation of Transfer Learning with ResNet50.
- We took accuracy and loss as the metric for comparison between these models across two system environments: P100 and V100.

Results (P100)



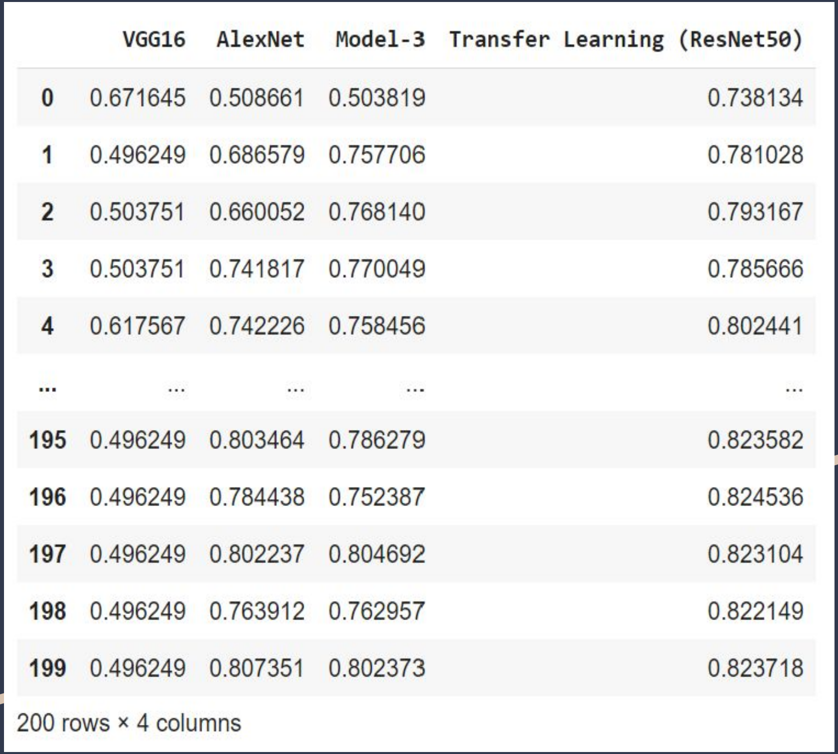
	VGG16	AlexNet	Model-3	Transfer Learning (ResNet50)
0	0.486498	0.512658	0.570464	0.489030
1	0.486498	0.510127	0.760338	0.595781
2	0.486498	0.523207	0.712658	0.732068
3	0.486498	0.564135	0.777637	0.727426
4	0.486498	0.566245	0.785654	0.655696
...	...	...	...	...
195	0.486498	0.812236	0.809705	0.828692
196	0.486498	0.810127	0.808439	0.828270
197	0.486498	0.801688	0.810127	0.827848
198	0.486498	0.808439	0.797890	0.827848
199	0.486498	0.777215	0.820253	0.827848

200 rows × 4 columns



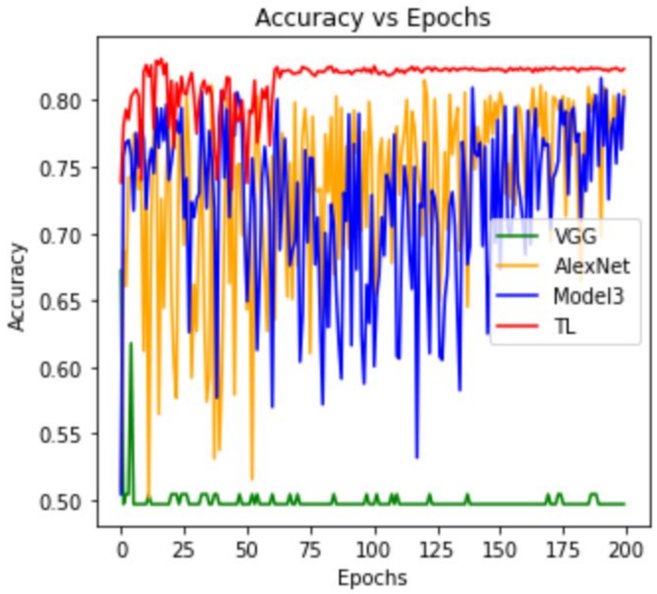
- As we can see from the table provided, VGG16 performs the worst with an accuracy of 50%, while other models go upto 80%.
- The Transfer Learning model with ResNet50 performs the best with an accuracy of 82.78% and is the suggested model for these classification.

Results (V100)



	VGG16	AlexNet	Model-3	Transfer Learning (ResNet50)
0	0.671645	0.508661	0.503819	0.738134
1	0.496249	0.686579	0.757706	0.781028
2	0.503751	0.660052	0.768140	0.793167
3	0.503751	0.741817	0.770049	0.785666
4	0.617567	0.742226	0.758456	0.802441
...	...	...	...	...
195	0.496249	0.803464	0.786279	0.823582
196	0.496249	0.784438	0.752387	0.824536
197	0.496249	0.802237	0.804692	0.823104
198	0.496249	0.763912	0.762957	0.822149
199	0.496249	0.807351	0.802373	0.823718

200 rows × 4 columns



- As we can see from the table provided, VGG16 performs the worst with an accuracy of 50%, while other models go upto 80%.

- The Transfer Learning model with ResNet50 performs the best with an accuracy of 82.4% and is the suggested model for these classification.

Conclusion and Discussion

- One of the key contributions was the preprocessing technique and the Transfer Learning Technique

- Transfer Learning works best for these type of classification problem, and the model that showed the best result was the ResNet model with around 83% accuracy, across both the GPU environments.

THANK YOU

COMS 6998 009

Practical Deep Learning System Performance

Final Presentation