Fall 2024 CS53331/4331 Adversarial Machine Learning Assignment 0

Title	**The Deep Learning Stuff**
Due date	Friday, Sep 20th, before the class
First Name	Sruthi
Last Name	Mandalapu
Student ID	R11906160
Marks	50

Note: Please answer the following questions and submit them through Blackboard. Be sure to submit it to assignment 0. DO NOT write the report by hand and submit a scanned version. Just write the answers in a Word document and submit it. Both Word and PDF submissions are accepted.

Submission Instruction (3 documents)

You are required to submit three documents:

1. ***Report.*** Just fill out the above report and submit it as a Word or PDF document.
2. ***Ipynb file.*** The code that you have written. Preferably in an ipynb document. You can submit it as a .py file as well.
3. ***Txt file of the code.*** We need your code in the .txt file as well. Use whatever way you prefer. The fastest would be to download the file as a .py file and change the extension to .txt

Objectives

This assignment has three main objectives:

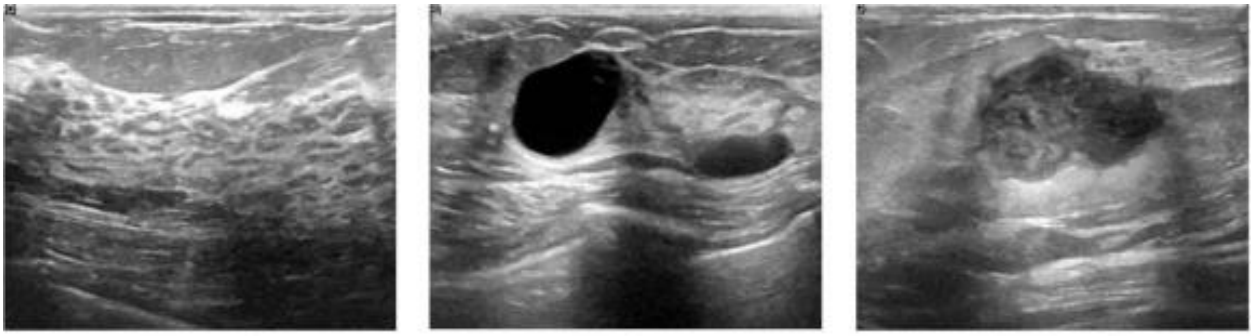
1. Get started with Notebook, Colab and other things that you will be using all over the course
2. Train a specified deep learning model and provide us with accuracies
3. Find a good performing model and provide us with accuracies

Get started

Download the assignment files from Blackboard. You will need the report (This file) and the .ipynb file where you will put your code.

Dataset

We will use Breast Ultrasound Images Dataset (Dataset BUSI). The dataset can be downloaded from here. It is a dataset for Breast Cancer detection. It includes breast ultrasound images among women in ages between 25 and 75 years old. This data was collected in 2018. The number of patients is 600 female patients. The dataset comprises 780 images with an average image size of 500*500 pixels. The images are in PNG format. The ground truth images are presented with original images. The images are categorized into normal, benign, and malignant. Below is a figure taken from their paper. It is recommended that you upload the dataset into your personal Google Drive to follow the Colab instructions as they are. Of course, if you prefer to use other than Colab, you will need a similar preprocessing.



Normal Benign Malignant

Instruction for Colab

To get started with Google Colab, simply go to Google Colab, sign in with your Google account, and create a new notebook. You can write and execute Python code directly in the notebook. To access your dataset stored in your Google Drive (previous step), first run the following code to mount your Drive:

```
from google.colab import drive
drive.mount('/content/drive')
```

Follow the authorization steps, and your Drive will be accessible at **/content/drive/My Drive/.** You can then load your dataset into the notebook by providing the correct file path. This part of the code is provided for you in the .ipynb file of Assignment 0. You will need to setup the drive connection and run the code.

To use the free GPU provided by Colab, you can change the runtime to access a GPU by clicking on **"Runtime" > "Change runtime type" and** selecting **"T4 GPU"** from the **Hardware accelerator** dropdown menu. You can always use higher GPU powers at a cost (Colab Pro is $10 per month), but you should be fine with the free version, considering that you start the assignment early enough.

Colab comes with many pre-installed libraries, but if you need to install additional Python packages, you can do so with pip. For example:

```
!pip install library_name
```

Remember to save your work frequently.

After you've completed your work in Google Colab, you can easily download your notebook from Google Colab, go to **"File" > "Download" > "Download.ipynb"**.

Other than Colab

If you don't prefer Colab or notebook, you always have the option to run it on your computer (especially if it has a GPU) or access HPCC resources at TTU (needs an account with my permission).

Additional resources

1. TensorFlow resource https://www.tensorflow.org/
2. PyTorch resources https://pytorch.org/get-started/pytorch-2.0/
3. Deep learning with Python https://dl-with-python.readthedocs.io/en/latest/
4. Get started with Colab https://colab.research.google.com/

Task 1 (10 pts)

Now, let's do some cool deep-learning stuff. For your first task, you will train a Convolutional Natural Network (CNN) model with the parameters in Table 1 and provide us with the results. You can use already developed models for Kears, TensorFlow, and PyTorch. Start with a simple CNN model (e.g., 2-layer CNN). For this task, you don't need to do hyper-parameter tuning, apply data augmentation, or fine-tune the layers of the models unless you wish to.

Table 1 Parameters for Task 1

Parameter	**Value**
Learning rate	0.0001
Epochs	100
Batch size	16
Model dimension	224*224*3
Optimization	Adam
Convolutional layers' activation function	Relu
Output layer activation function	Softmax

Results

(a) [5 pts] Fill in Table 2 with the values for the classification accuracy for the train set, validation set, and test set of images. You don't need to report other than accuracy. Test accuracy can be around 74% and that is okay for this task, we will fix it in the next task.

Table 2: Classification accuracy of the models on the BUSI dataset

Model	**Training set**	**Validation set**	**Testing set**
CNN	100%	70.5%	73.07%

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 20s 640ms/step - accuracy: 0.7067 - loss: 0.6444 - val_accuracy: 0.6346 - val_loss: 0.8771
Epoch 7/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 21s 642ms/step - accuracy: 0.7881 - loss: 0.5027 - val_accuracy: 0.6410 - val_loss: 0.7684
Epoch 8/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 22s 702ms/step - accuracy: 0.8211 - loss: 0.4475 - val_accuracy: 0.7244 - val_loss: 0.6890
Epoch 9/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 22s 698ms/step - accuracy: 0.8829 - loss: 0.3047 - val_accuracy: 0.7179 - val_loss: 0.6962
Epoch 10/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 22s 680ms/step - accuracy: 0.8993 - loss: 0.2636 - val_accuracy: 0.7436 - val_loss: 0.6955
Epoch 11/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 22s 678ms/step - accuracy: 0.9502 - loss: 0.1596 - val_accuracy: 0.7436 - val_loss: 0.7282
Epoch 12/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 24s 744ms/step - accuracy: 0.9684 - loss: 0.1170 - val_accuracy: 0.7436 - val_loss: 0.9202
Epoch 13/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 21s 658ms/step - accuracy: 0.9724 - loss: 0.0935 - val_accuracy: 0.7372 - val_loss: 0.9266
Epoch 14/100
```

```
Test Accuracy:  0.7307692170143127
```

(b) [3 pts] For the built model, plot the training and validation loss and accuracy (similar to the example in Figure 1).

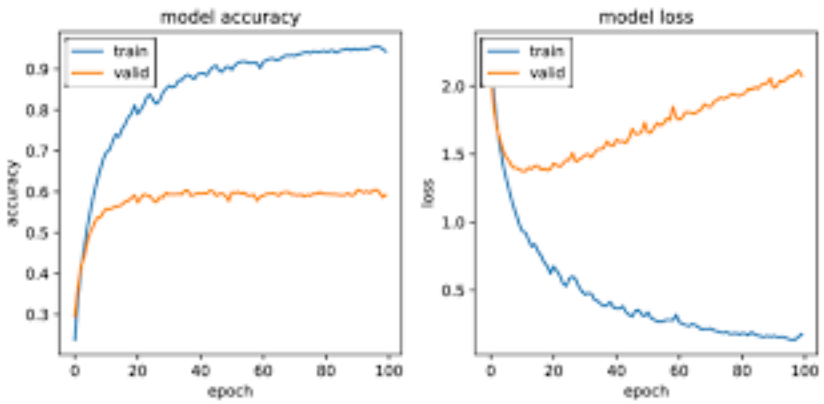
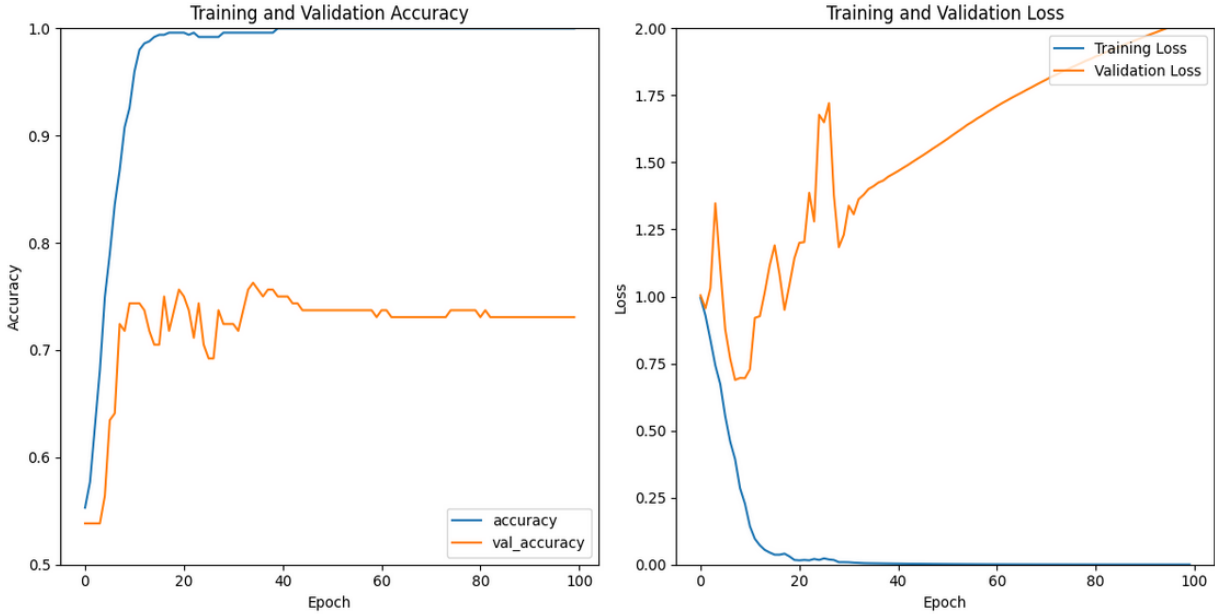


Figure 1. Example Loss and accuracy plots for a DL model. This is an example, and not the actual expected plot



(c) [2 pts] Briefly provide insights on the model's performance and any other observations regarding the models or the dataset.

- Overall, the model is producing 73% of test accuracy. It is procuring good results, but the performance can be improved by avoiding overfitting of the data samples. The model describes 2D convolutional layer with 32 filters, each of size 3x3, and ReLU activation. The input shape of images is defined with dimension (224, 224, 3) for the BUSI dataset. The output layer is defined with OUTPUT_DIM 3 units.
- MaxPooling2D((2, 2)): allows taking the maximum value from each 2x2 patch, reducing the spatial dimensions.
- Conv2D(64, (3, 3), activation='relu'): this third convolutional layer with 64 filters of size 3x3, MaxPooling2D((2, 2)) again allows in reducing the spatial dimensions.
- Flatten(): Converts the 2D output of the final convolutional layer into a 1D vector to prepare it for the fully connected layers.

- Dense(64, activation='relu'): Fully connected layer with 64 units and ReLU activation.
- Composing all the layers together, the model produced out significantly lesser validation accuracy than the training accuracy. The training accuracy is almost 100 percent, the performance for testing and validation can be improved by tuning the parameters or applying the techniques like regularization, early-stopping, bacth normalization for avoidance of overfitting.

Model: "sequential_10"

Layer (type)	Output Shape	Param #
conv2d_30 (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d_20 (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_31 (Conv2D)	(None, 109, 109, 64)	18,496
max_pooling2d_21 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_32 (Conv2D)	(None, 52, 52, 64)	36,928

Total params: 56,320 (220.00 KB)

Trainable params: 56,320 (220.00 KB)

Non-trainable params: 0 (0.00 B)

Model: "sequential_10"

Layer (type)	Output Shape	Param #
conv2d_30 (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d_20 (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_31 (Conv2D)	(None, 109, 109, 64)	18,496
max_pooling2d_21 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_32 (Conv2D)	(None, 52, 52, 64)	36,928
flatten_10 (Flatten)	(None, 173056)	0
dense_20 (Dense)	(None, 64)	11,075,648
dense_21 (Dense)	(None, 3)	195

Total params: 11,132,163 (42.47 MB)

Trainable params: 11,132,163 (42.47 MB)

Non-trainable params: 0 (0.00 B)

Task 2 (20 pts)

Now, let's enhance the performance. You must train a DL model to achieve 85% or above testing accuracy. You are restricted to using the parameters provided in Table 3. You should start with pre-trained weights (e.g., on ImageNet, which is already available on Keras). It should result in a better performance. Using complex/deeper DL models, data augmentation, or fine-tuning the layers of the models is fine as long as you reach the desired accuracy.

Table 3 Parameters for Task 2 and 3

Parameter	Value
Learning rate	0.0001
Epochs	100
Batch size	16
Possible models to try (not limited to those)	VGG16/19 MobileNet EfficientNet RestNet50

[7 pts] Your algorithm and explain why you choose it

VGG16: Visual Geometry Group 16, an extension of the original VGG model, It contains the following layers: Input Layers, Convolutional Layers, Max-Pooling Layers, Fully Connected Layers (Relu Layer), Fully Connected Layer (Softmax Layer), Flatten Layer inclusion with drop-out rate. This model provides fine-tuning on domain-specific datasets. VGG-16 reduces the time during training the model and produces out optimization. As it is an image dataset, the BUSI images are classified with good accuracy, because this algorithm produces prominence in object detection and classification tasks achieving impactful results. The architecture pre-defined model outperforms built in Keras Model, this is because it flattens the last layer which is pooling layer storing the information. VGG16 produces robust architecture when explicating in terms of capturing complex patterns, which is widely used across different image datasets. The model view is described as:

- VGG16 consists of 13 convolutional layers, 5 MaxPooling layers, and 3 fully connected layers
- All convolutional layers use 3x3 kernels and RELU activation is used after every convolution.
- Input Layer: 224x224 RGB image, Input size: (224, 224, 3)
- Block1: Conv Layer 1-1 - 64 filters of size 3x3, Conv Layer 1-2 - 64 filters of size 3x3 – RELU Activation, MaxPooling - 2x2 window, dimensions to 112x112x64
- Block2: Conv Layer 2-1 - 64 filters of size 3x3, Conv Layer 2-2 - 64 filters of size 3x3 – RELU Activation, MaxPooling - 2x2 window, dimensions to 56x56x128
- Block3: Conv Layer 3-1 - 64 filters of size 3x3, Conv Layer 3-2 - 64 filters of size 3x3, Conv Layer 3-3 - 64 filters of size 3x3 – RELU Activation, MaxPooling - 2x2 window, dimensions to 28x28x256

- Block4: Conv Layer 4-1 - 64 filters of size 3x3, Conv Layer 4-2 - 64 filters of size 3x3, Conv Layer 4-3 - 64 filters of size 3x3 – RELU Activation, MaxPooling - 2x2 window, dimensions to 14x14x512
- Block5: Conv Layer 5-1 - 64 filters of size 3x3, Conv Layer 5-2 - 64 filters of size 3x3, Conv Layer 5-3 - 64 filters of size 3x3 – RELU Activation, MaxPooling - 2x2 window, dimensions to 7x7x512
- Output Layer: 1000 neuron Layer (Trained on imagenet) – Uses Softmax Activation

Results

(a) [5pts] Fill in Table 4 with the values for the classification accuracy for the train set, validation set, and test set of images. You don't need to report other than accuracy.

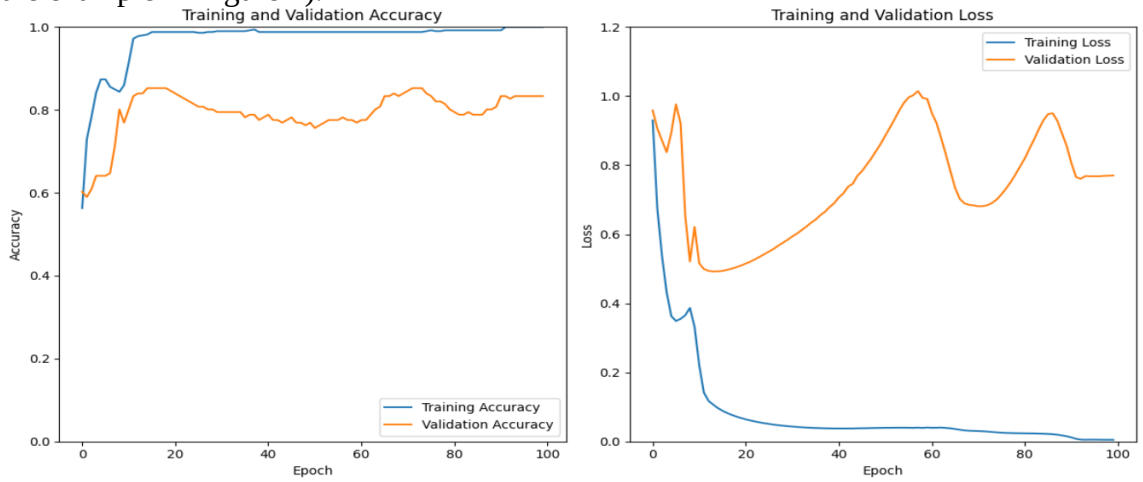
Table 4: Classification accuracy of the models on the BUSI dataset

Model	Training set	Validation set	Testing set
VGG16	100%	78.85%	83.33%

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 323s 10s/step - accuracy: 0.9875 - loss: 0.0644 - val_accuracy: 0.8205 - val_loss: 0.5347
Epoch 25/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 295s 9s/step - accuracy: 0.9875 - loss: 0.0617 - val_accuracy: 0.8141 - val_loss: 0.5424
Epoch 26/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 323s 10s/step - accuracy: 0.9868 - loss: 0.0594 - val_accuracy: 0.8077 - val_loss: 0.5500
Epoch 27/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 1308s 42s/step - accuracy: 0.9868 - loss: 0.0571 - val_accuracy: 0.8077 - val_loss: 0.5578
Epoch 28/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 333s 10s/step - accuracy: 0.9879 - loss: 0.0552 - val_accuracy: 0.8013 - val_loss: 0.5672
Epoch 29/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 311s 10s/step - accuracy: 0.9879 - loss: 0.0535 - val_accuracy: 0.8013 - val_loss: 0.5754
Epoch 30/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 296s 9s/step - accuracy: 0.9917 - loss: 0.0520 - val_accuracy: 0.7949 - val_loss: 0.5842
Epoch 31/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 292s 9s/step - accuracy: 0.9917 - loss: 0.0505 - val_accuracy: 0.7949 - val_loss: 0.5936
Epoch 32/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 222s 7s/step - accuracy: 0.9917 - loss: 0.0494 - val_accuracy: 0.7949 - val_loss: 0.6018
```

```
Test accuracy: 0.8333333134651184
```

(b) [3 pts] For the built model, plot the training and validation loss and accuracy (similar to the example in Figure 1).



(c) [5 pts] Briefly provide insights on the model's performance and any other observations regarding the models or the dataset. Does the model have any obvious problem?

- Overall, the model procures 84% test accuracy procuring improved in performance with pretrained model. The accuracy can be improved by avoidance of outliers by overfitting techniques. Here, the model uses VGG16 had a prominent difference considering the User-defined model to that of pre-trained model. This architecture procures more than ten percent difference on comparable to model in Task1.
- This model is loaded with ImageNet weights (weights='imagenet'). include_top=False removes the fully connected layers at the top - keeping only the convolutional base.
- It acts as a feature extractor – defined by layer.trainable = False.
- Flatten layer allows to convert the output from the convolutional layers into a 1D vector. Coming to Dense Layer, it produces out number of classes (OUTPUT_DIM=3), with a softmax activation for multi-class classification.
- Here, Adam Optimizer is used with Learning rate = 0.0001. Loss function is defined with a Categorical Cross Entropy.
- The main pre-trained features of VGG16 is implicated like textures, shapes, which are trained by using ImageNet
- The model performs good in constraint of accuracy showing up more than 80%, but the performance can be improved by removing outliers applying some techniques like regularization, early stopping or batch-normalization.

Task 3 (20 pts)

Now, let's fix the problem with the previous model. Most models in the last task were overfitting (training accuracy got to 100% so quickly, and validation accuracy started to decrease). Fix that problem without changing the batch size, number of iterations, or learning rate. Use the same model and just add any technique that avoids overfitting. Keep the parameters in Table 3 the same.

[7 pts] Your algorithm and explain why you choose it

L2 Regularization: This is one of the overfitting techniques allowing to minimize the loss and improves the efficiency of the model. The model usually performs well on training data but poorly on unseen data. Regularization techniques reduce the complexity of the model, making it better at generalizing from the training data. Coming to L2 regularization: the loss function is evaluated by: Loss=Original Loss+Lambda*(sum of coefficients)^2. The main prominence for L2 regularization is providing generalization providing a multicollinearity in the data. It tunes the overall impact on complexity on model training, by multiplying the value by scalar which is called "regularization rate". This lambda is called as regularizing parameter, and coefficients is termed as "Weights".

- L2 regularization in Keras by specifying it in the layers using the kernel_regularizer argument.
- The higher the lambda, the more the model will penalize large weights. The regularized weights is given for the model to run efficiently.
- This is different from early stopping and other techniques. The resulting model from early stopping is very unlikely for the good model, where ideally l2 regularization thoroughly trained by specific regularizing factor.
- Learning rate equally effects the lambda factor, more lambda compared to learning rate procures poor predictions. Simillarly, more learning rate comparable to lambda factor which overfits the model. The state of equilibrium across learning rate and regularization rate would ultimately provides good accuracy.

Results

(a) [5pts] Fill in Table 5 with the values for the classification accuracy for the train set, validation set, and test set of images. You don't need to report other than accuracy.

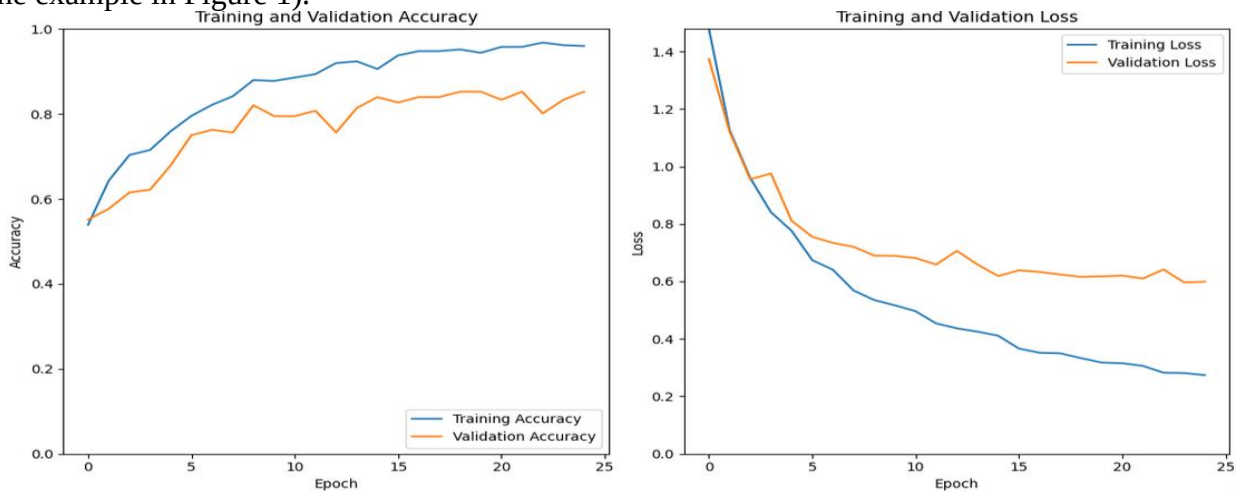
Table 5: Classification accuracy of the models on the BUSI dataset

Model	Training set	Validation set	Testing set
VGG16/L2 Regularizer	95.4%	81.36%	85.25%

```
Epoch 8/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 236s 7s/step - accuracy: 0.8032 - loss: 0.6281 - val_accuracy: 0.7564 - val_loss: 0.7205
Epoch 9/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 241s 8s/step - accuracy: 0.8594 - loss: 0.5725 - val_accuracy: 0.8205 - val_loss: 0.6900
Epoch 10/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 230s 7s/step - accuracy: 0.8594 - loss: 0.5416 - val_accuracy: 0.7949 - val_loss: 0.6889
Epoch 11/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 222s 7s/step - accuracy: 0.8518 - loss: 0.5301 - val_accuracy: 0.7949 - val_loss: 0.6814
Epoch 12/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 225s 7s/step - accuracy: 0.8602 - loss: 0.5157 - val_accuracy: 0.8077 - val_loss: 0.6587
Epoch 13/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 219s 7s/step - accuracy: 0.8989 - loss: 0.4604 - val_accuracy: 0.7564 - val_loss: 0.7066
Epoch 14/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 217s 7s/step - accuracy: 0.9048 - loss: 0.4484 - val_accuracy: 0.8141 - val_loss: 0.6584
Epoch 15/100
```

```
Test accuracy: 0.8525640964508057
```

(b) [3 pts] For the built model, plot the training and validation loss and accuracy (similar to the example in Figure 1).



(c) [5 pts] Briefly provide insights on the model's performance and any other observations regarding the models or the dataset.

- Overall, the model had improved the accuracy than in before stages producing 86% of accuracy. Here the L2 regularization technique had been a prominent approach to figure the abnormal data in BUSI images making it as a generalized view. This regularized data is helps reduce the model's complexity, preventing it from memorizing the training data. It enforces the model to learn smaller weights, which improves its ability to generalize to unseen data.
- Here Learning Rate is used similar to that for Task1 and Task2, the lambda parameter is tuned that was closer to the learning rate.
- In the current model this overfitting technique prevents from having larger weights, making the model more stable.
- With L2 regularization however the training loss might increase slightly, due to added penalty term, but test loss should be stabilized or decreased.
- To use this technique in Keras, regularizers module need to be imported from tensorflow.keras. And this need to applied within the layers of models.

- In terms of observations, looking through the accuracy – Validation accuracy steadily increases and the validation set is exactly projecting just like training set. The Validation loss should decrease reflecting better performance for the unseen data. Coming to the test accuracy compared to the previous implementations for Task1 and Task2, the test accuracy procured better results than before.

Submission Instruction (3 documents)

You are required to submit three documents:

1. ***Report.*** Just fill out the above report and submit it as a Word or PDF document.
2. ***Ipynb file.*** The code that you have written. Preferably in an ipynb document. You can submit it as a .py file as well.
3. ***Txt file of the code.*** We need your code in the .txt file as well. Use whatever way you prefer. The fastest would be to download the file as a .py file and change the extension to .txt