SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
FACULTY OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF COMPUTING TECHNOLOGIES
18CSP107L / 18CSP108L- MINOR PROJECT / INTERNSHIP

Comparison and Analysis of CNN Models on Breast Cancer Histopathological Images

Student 1 Reg No: RA1811003010684
Student 2 Reg No: RA1811028010052
Batch ID: B4
Guide name and Designation: Dr. G. Niranjana
(Associate Professor, Dept. of CTech., SRM IST)

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

Table of contents

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

Problem Statement

To build an algorithm which automatically identifies whether a patient is suffering from breast cancer or not by analyzing biopsy images and to verify the performance and results when CNNs are employed for this task (multiple models).

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

Abstract

- Over the last decade, the demand for early diagnosis of breast cancer has resulted in new research avenues. According to the World Health Organization (WHO), successful treatment can be provided to breast cancer patients if this fatal disease is diagnosed at an early stage.
- With the enormous advance in Image Processing and Machine Learning tools, research and development work in the field of biotechnology has gained good attention. Researchers have developed efficient algorithms to enhance the diagnosis process and to reduce the human intervention via CAD systems.
- Computer-Aided Diagnosis (CAD) tools are widely implemented in the healthcare sector to diagnose and detect different kinds of abnormalities. In the last few years, the use of the CAD system has become common to increase the accuracy in different research areas.
- Nowadays, Convolutional Neural Networks (CNNs) have been set up as an intense class of models of image acknowledgement issues. CNNs are a class of deep, feed-forward artificial neural networks that have been successfully applied to recognize images.
- In this project, we aim at providing a concrete analysis of the performance of various CNN models given the task of classifying cancer cells in breast biopsy images.

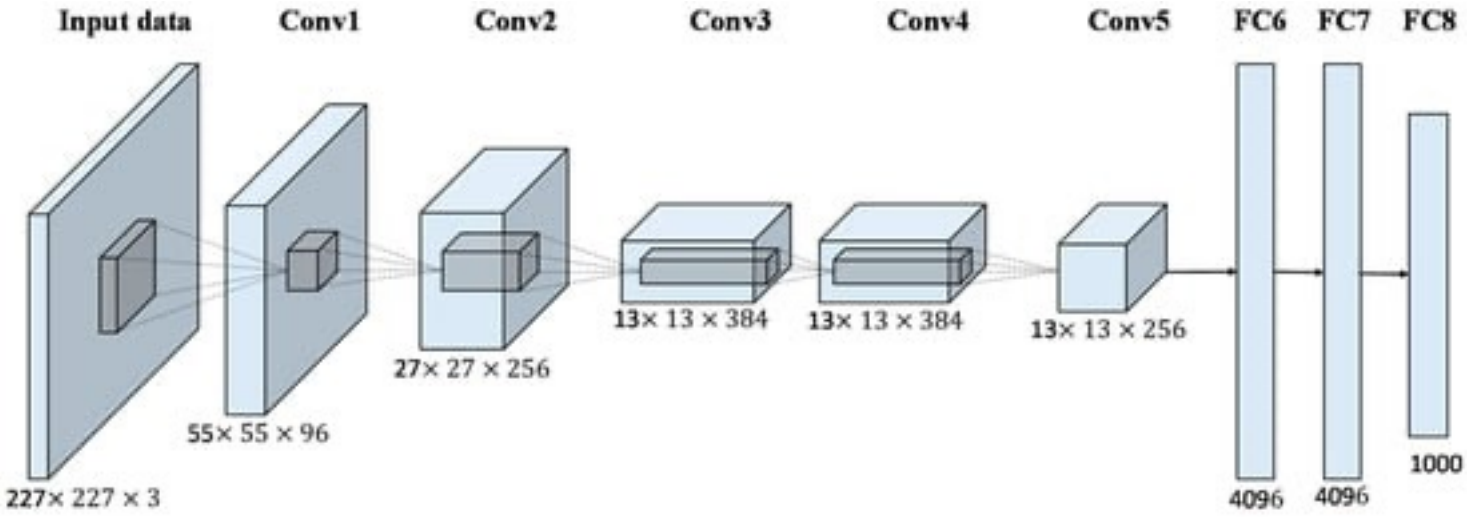
SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

Objectives

- To analyze medical biopsy images of cancer cells (from dataset).
- To classify the cancer cells as malignant or benign.
- To compare the performance and results achieved by various CNN models.
- To incorporate a visual layer to this model (UI).

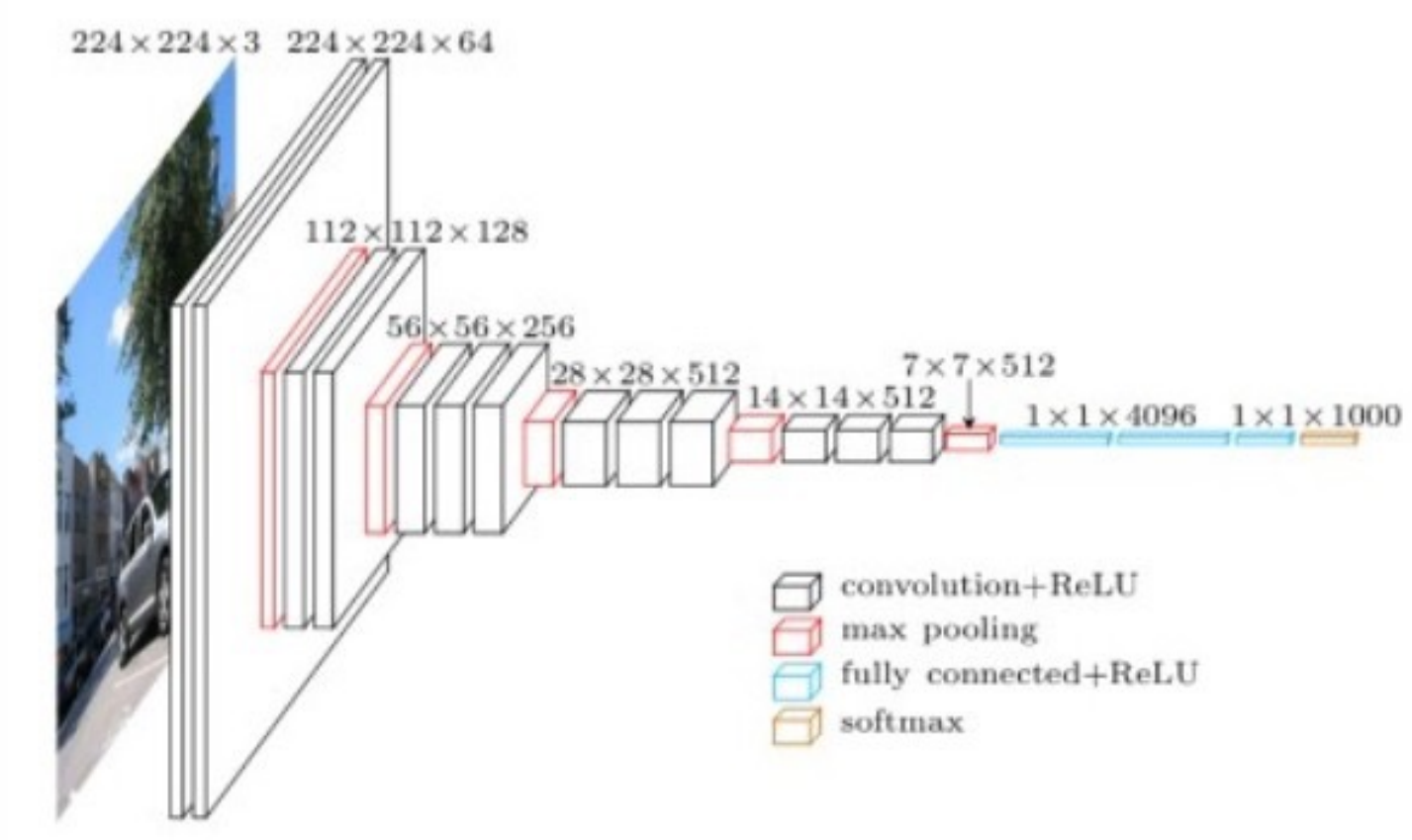
Architecture Diagram

AlexNet



Architecture Diagram

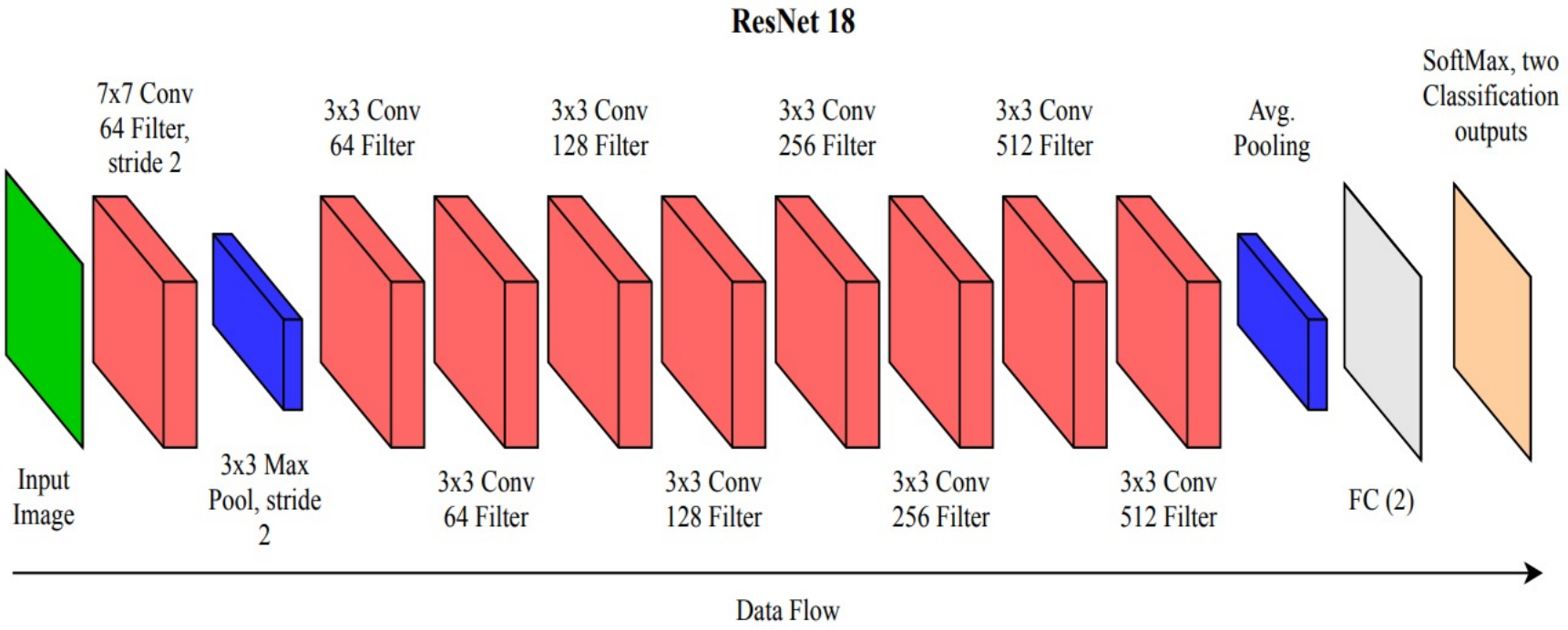
VGG



SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

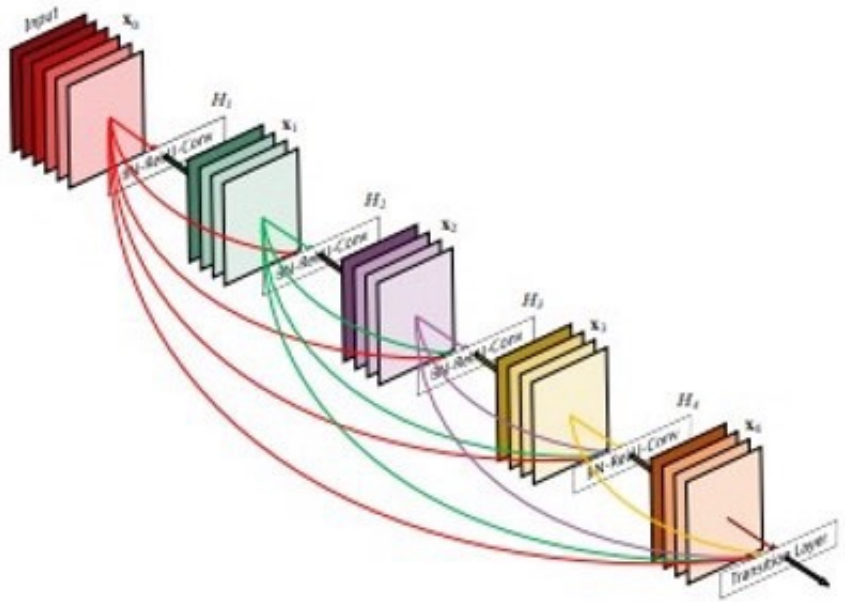
Architecture Diagram

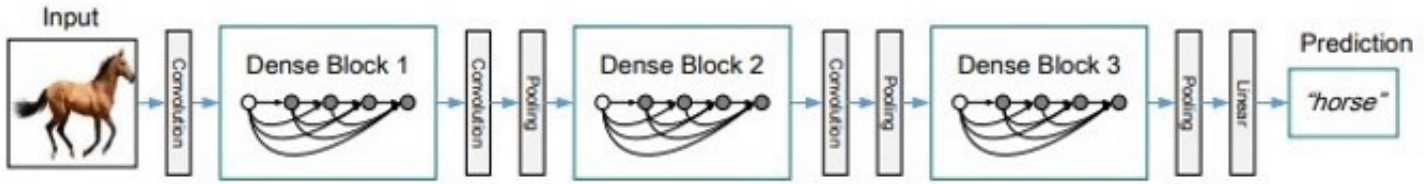
ResNet



Architecture Diagram

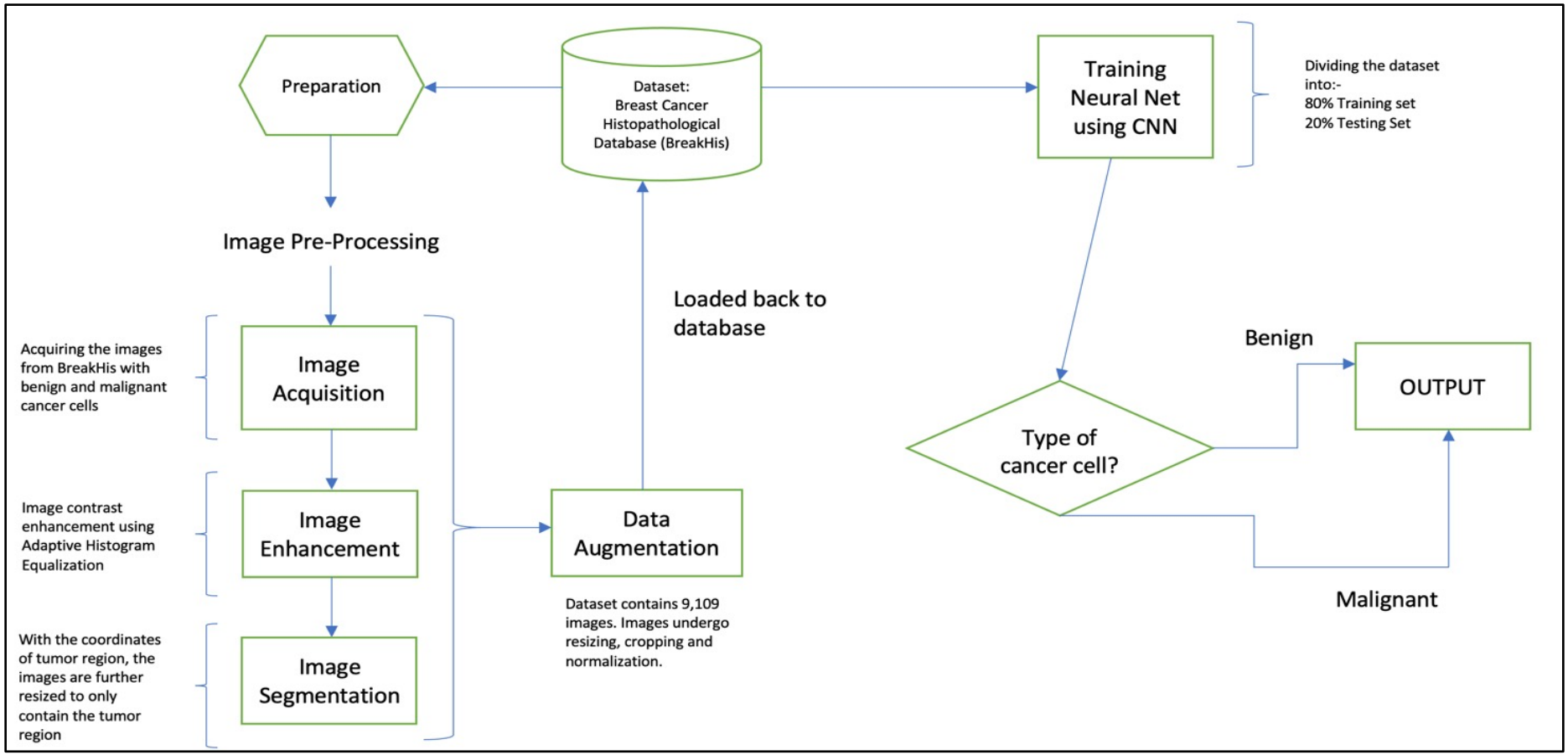
DenseNet





SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

Data Flow Diagram



Module Description & Implementation

Module 1 – Image Pre-processing

- Image Acquisition – Acquiring the images from BreakHis dataset with benign and malignant cancer cells.
- Image Enhancement – Image contrast enhancement with Adaptive Histogram Equalization.
- Image Segmentation – With the coordinates of tumor region, the images are further resized to only contain the tumor region

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

Module Description & Implementation

Module 1 – Image Pre-processing

```
#Defining our transforms
train_transforms = transforms.Compose([#transforms.RandomRotation(30),
                                       #transforms.RandomResizedCrop(224),
                                       transforms.Resize(256),
                                       transforms.CenterCrop(224),
                                       transforms.RandomHorizontalFlip(),
                                       transforms.ToTensor(),
                                       transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                                            std=[0.229, 0.224, 0.225])])

test_transforms  = 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])])

valid_transforms = 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])])


# loading and applyling above transforms on dataset using ImageFolder
train_dataset = datasets.ImageFolder(train_dir,transform=train_transforms)
test_dataset  = datasets.ImageFolder(test_dir ,transform=test_transforms)
valid_dataset = datasets.ImageFolder(valid_dir,transform=valid_transforms)
```

Module Description & Implementation

Module 2 – Data Augmentation

- Dataset contains 9,109 images.
- Images undergo horizontal flip.

- Images are then loaded back into the dataset.

Module 3 – Training Neural Net using CNN

- Dividing the dataset into : 80% training set & 20% testing set

Module Description & Implementation

Module 4 – Algorithm Used

- Our input is a training dataset that consists of *N* images, each labelled with one of 2 different classes.
- Then, we use this training set to train a classifier to learn what every one of the classes looks like.
- In the end, we evaluate the quality of the classifier by asking it to predict labels for a new set of images that it has never seen before. We will then compare the true labels of these images to the ones predicted by the classifier.

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

Module Description & Implementation

Training the CNN Models :-

```
def train_cnn_model(model_name, n_epochs=1):
  if model_name=='alexnet':
    # print out the model structure
    print("#################################################################################")
    model = models.alexnet(pretrained=True)
    print("Model Name: ",model_name,end="\n\n")
    print(model)
    print("Last layer input features: ",model.classifier[6].in_features)
    print("Last layer output features: ",model.classifier[6].out_features)
    # Freeze training for all layers
    for param in model.parameters():
        param.requires_grad = False

    n_inputs = model.classifier[6].in_features

    # add last linear layer (n_inputs --to---> 14 painting classes)
    # new layers automatically have requires_grad = True
    last_layer = nn.Linear(n_inputs, len(classes))

    model.classifier[6]= last_layer

    # if GPU is available, move the model to GPU
    if train_on_gpu:
        model=model.to(device)

    # check to see that your last layer produces the expected number of outputs
    print("Last layer output features (after changes): ",model.classifier[6].out_features)

    # specifying optimizer and learning rate
    optimizer = optim.Adam(model.classifier.parameters(), lr=0.001)

  elif model_name=='vgg19':
    # print out the model structure
    print("#################################################################################")
    model = models.vgg19(pretrained=True)
    print("Model Name: ",model_name,end="\n\n")
    print(model)
    print("Last layer input features: ",model.classifier[6].in_features)
    print("Last layer output features: ",model.classifier[6].out_features)
    # Freeze training for all layers
    for param in model.parameters():
        param.requires_grad = False
```

```
    n_inputs = model.classifier[6].in_features

    # add last linear layer (n_inputs --to---> 14 painting classes)
    # new layers automatically have requires_grad = True
    last_layer = nn.Linear(n_inputs, len(classes))

    model.classifier[6]= last_layer

    # if GPU is available, move the model to GPU
    if train_on_gpu:
        model=model.to(device)

    # check to see that your last layer produces the expected number of outputs
    print("Last layer output features (after changes): ",model.classifier[6].out_features)

    # specifying optimizer and learning rate
    optimizer = optim.Adam(model.classifier.parameters(), lr=0.001)

elif model_name=='resnet152':
    # print out the model structure
    print("#################################################################################")
    model = models.resnet152(pretrained=True)
    print("Model Name: ",model_name,end="\n\n")
    print(model)
    print("Last layer input features: ",model.fc.in_features)
    print("Last layer output features: ",model.fc.out_features)
    # Freeze training for all layers
    for param in model.parameters():
        param.requires_grad = False

    n_inputs = model.fc.in_features

    # add last linear layer (n_inputs --to---> 14 painting classes)
    # new layers automatically have requires_grad = True
    last_layer = nn.Linear(n_inputs, len(classes))

    model.fc= last_layer

    # if GPU is available, move the model to GPU
    if train_on_gpu:
        model=model.to(device)
```

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

Module Description & Implementation

Training the CNN Models :-

```
# check to see that your last layer produces the expected number of outputs
print("Last layer output features (after changes): ",model.fc.out_features)

# specifying optimizer and learning rate
optimizer = optim.Adam(model.fc.parameters(), lr=0.001)



lif model_name=='densenet201':
 # print out the model structure
 print("#####################################################################")
 model = models.densenet201(pretrained=True)
 print("Model Name: ",model_name,end="\n\n")
 print(model)
 print("Last layer input features: ",model.classifier.in_features)
 print("Last layer output features: ",model.classifier.out_features)
 # Freeze training for all layers
 for param in model.parameters():
     param.requires_grad = False

 n_inputs = model.classifier.in_features

 # add last linear layer (n_inputs --to---> 14 painting classes)
 # new layers automatically have requires_grad = True
 last_layer = nn.Linear(n_inputs, len(classes))

 model.classifier= last_layer

 # if GPU is available, move the model to GPU
 if train_on_gpu:
     model=model.to(device)

 # check to see that your last layer produces the expected number of outputs
 print("Last layer output features (after changes): ",model.classifier.out_features)

 # specifying optimizer and learning rate
 optimizer = optim.Adam(model.classifier.parameters(), lr=0.001)
```

```
# specify loss function (categorical cross-entropy)
criterion = nn.CrossEntropyLoss()


model_filename = 'tumor_' + model_name + '.pt'
print('File name for saved model: ',model_filename)



print("\n\n\nTraining the model...\n")


a = time.time()  #Start-time for training

# initialize tracker for minimum validation loss
valid_loss_min = np.Inf # set initial "min" to infinity


train_losses,valid_losses, accuracies=[],[],[]

for epoch in range(1, n_epochs+1):
    c = time.time()  #Start-time for epoch

    # to keep track of training loss
    training_loss = 0.0
```

Module Description & Implementation

Training the CNN Models :-

```
##################
# train the model #
##################

# model by default is set to train
for batch_i, (images, labels) in enumerate(train_dataloader):    #Getting one batch of training images and their corresponding true labels
    # move tensors to GPU if CUDA is available
    if train_on_gpu:
        images = images.to(device)
        labels = labels.to(device)

    # clear the previous/buffer gradients of all optimized variables
    optimizer.zero_grad()

    # forward pass: compute predicted outputs by passing inputs to the model
    outputs = model.forward(images)

    # calculate the batch loss
    loss = criterion(outputs, labels)    #(y_hat, y)  or (our-prediction, true-label)

    # backward pass: compute gradient of the loss with respect to model parameters
    loss.backward()

    # perform a single optimization step (parameter update)
    optimizer.step()

    # update training loss
    training_loss += loss.item()

    ###End of training
```

Module Description & Implementation

Validating the CNN Models :-

```
####################
# validating the model #
####################

#validation loss and accuracy
validation_loss = 0.0
accuracy = 0.0

model.eval() #model is put to evalution mode i.e. dropout is switched off

with torch.no_grad():  #Turning off calculation of gradients (not required for validaiton)  {saves time}
    for images, labels in valid_dataloader:   #Getting one batch of validation images

        if train_on_gpu:   #moving data to GPU if available
            images = images.to(device)
            labels = labels.to(device)

        outputs = model.forward(images)
        batch_loss = criterion(outputs, labels)
        validation_loss += batch_loss.item()

        # Calculating accuracy
        ps = torch.exp(outputs)   #Turning raw output values into probabilities using exponential function

        #getting top one probablilty and its corresponding class for batch of images
        top_p, top_class = ps.topk(1, dim=1)

        #Comparing our predictions to true labels
        equals = top_class == labels.view(*top_class.shape)   #equals is a list of values
        #incrementing values of 'accuracy' with equals
        accuracy += torch.mean(equals.type(torch.FloatTensor)).item()   #taking average of equals will give number of true-predictions
                                        #equals if of ByteTensor (boolean), changing it to FloatTensor for taking mean...
```

```
    train_losses.append(training_loss/len(train_dataloader))
    valid_losses.append(validation_loss/len(valid_dataloader))
    accuracies.append(((accuracy/len(valid_dataloader))*100.0))
    d = time.time() #end-time for epoch

    print(f"Epoch {epoch} "
          f"Time: {int((d-c)/60)} min {int(d-c)%60} sec "
          f"Train loss: {training_loss/len(train_dataloader):.3f}.. "
          f"Validation loss: {validation_loss/len(valid_dataloader):.3f}.. "
          f"Validation accuracy: {((accuracy/len(valid_dataloader))*100.0):.3f}% "
          )

    training_loss = 0.0

    # save model if validation loss has decreased
    if ( validation_loss/len(valid_dataloader) <= valid_loss_min):
        print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model ...'.format(valid_loss_min , validation_loss/len(valid_dataloader)))
        torch.save(model, model_filename) #Saving model
        valid_loss_min = validation_loss/len(valid_dataloader)   #Minimum validation loss updated


    #After validation, model is put to training mode i.e. dropout is again switched on
    model.train()

    ###end of epoch

b = time.time()  #end-time for training
print('\n\n\tTotal training time: ' , int((b-a)/(60*60)), "hour(s) " , int(((b-a)%(60*60))/60),"minute(s) ", int(((b-a)%(60*60))%60) , "second(s)")
return train_losses, valid_losses, accuracies,b-a
```

Module Description & Implementation

Testing the CNN Models :-

```
def test_cnn_model(model):
  # track test loss
  # over 14 painting classes
  test_loss = 0.0
  class_correct = list(0. for i in range(2))
  class_total = list(0. for i in range(2))
  classes_accuracies=[]
  model.eval() # evaluation mode (switching off dropout)
  counter = 0
  criterion = nn.CrossEntropyLoss()
  # iterate over test data
  for images, labels in test_dataloader: #geeting one batch of data from testloader

      # move tensors to GPU if CUDA is available
      if train_on_gpu:
          images = images.to(device)
          labels = labels.to(device)

      # forward pass: compute predicted outputs by passing inputs to the model
      output = model(images)

      # calculate the batch loss
      loss = criterion(output, labels)

      # update  test loss
      test_loss += loss.item()*images.size(0)

      # convert output probabilities to predicted class
      ps, pred = torch.max(output, 1)

      # compare model's predictions to true labels
      for i in range(len(images)):
          class_total[labels[i]] += 1
          if pred[i] == labels[i]:
              class_correct[pred[i]] += 1
      counter += 1
```

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

Module Description & Implementation

Module 5 – Performance Metrics

Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

$$precision = \frac{TP}{TP + FP}$$

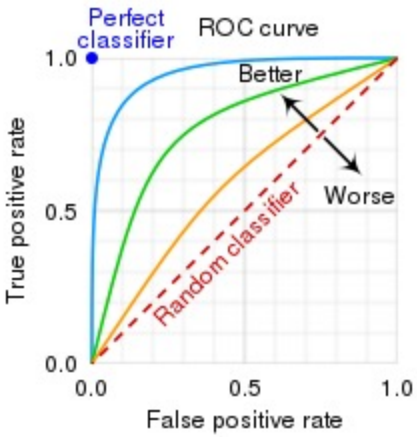
$$recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

$$specificity = \frac{TN}{TN + FP}$$

Medical Image analysis terms



ROC Curve

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

Module Description & Implementation

Performance Metrics :-

```
#calculate avg test loss
test_loss = test_loss/len(test_dataloader.dataset)
print('Test Loss: {:.6f}\n'.format(test_loss))

for i in range(2):
    classes_accuracies.append(100 * class_correct[i] / class_total[i])
    if class_total[i] > 0:
        print('Test Accuracy of %5s: %2d%% (%2d/%2d)' % (
            classes[i], 100 * class_correct[i] / class_total[i],
            np.sum(class_correct[i]), np.sum(class_total[i])))

    else:
        print('Test Accuracy of %5s: N/A (no training examples)' % (classes[i]))

print('\nTest Accuracy (Overall): %2d%% (%2d/%2d)' %
                        (100. * np.sum(class_correct) / np.sum(class_total),np.sum(class_correct), np.sum(class_total)))
```

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

Module Description & Implementation

Performance Metrics :-

```
def get_classification_scores(model,model_name, time_taken):

  print("#################################################################")
  labels_all=[]
  pred_all=[]


  for images, labels in test_dataloader:
      with torch.no_grad():

        if train_on_gpu:
            images = images.to(device)
            labels = labels.to(device)

        output = model(images)
        ps, pred = torch.max(output, 1)

        labels=labels.cpu()
        labels = labels.numpy()
        pred=pred.cpu()
        pred = pred.numpy()


        labels_all+=labels.tolist()
        pred_all+=pred.tolist()

  v_labels = labels_all
  v_pred = pred_all

  # define example

  v_encoded_labels = array(v_labels)
  v_encoded_labels = to_categorical(v_encoded_labels)

  v_encoded_pred = array(v_pred)
  v_encoded_pred = to_categorical(v_encoded_pred)

  precision = precision_score(v_labels,v_pred,average='weighted')
  recall = recall_score(v_labels,v_pred,average='weighted')
  f1 = f1_score(v_labels,v_pred,average='weighted')
  jaccard = jaccard_score(v_labels,v_pred,average='weighted')
  roc_auc_area = roc_auc_score(v_encoded_labels,v_encoded_pred,average='weighted')

  tn, fp, fn, tp = confusion_matrix(v_labels, v_pred).ravel()
  specificity = tn / (tn+fp)
  print("Model = ", model_name)
  print("precision = ", precision)
  print("recall = ",recall)
  print("f1 = ",f1)
  print("jaccard = ",jaccard)
  print("ROC_AUC score = ",roc_auc_area)
  print("Specificity = ", specificity)
  print("Training Time (mins)= ", time_taken / 60.0)
  print("_____________________________________________________________\n\n")
```

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

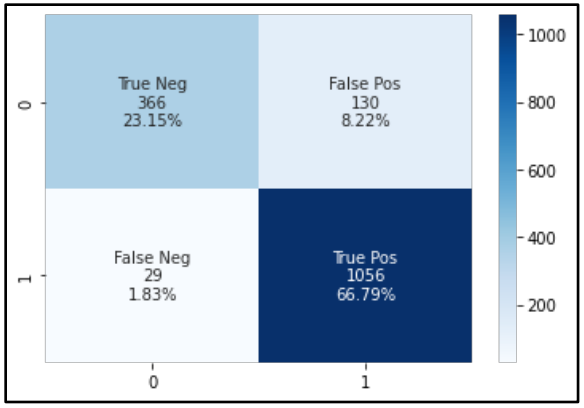
Results

	Test Loss	Test Accuracy of Benign (%)	Test Accuracy of Malignant (%)	Test Accuracy – Overall (%)
AlexNet	0.2484	73	97	89
VGG19	0.3056	68	97	88
ResNet152	0.2155	76	97	90
DenseNet201	0.2609	73	96	89

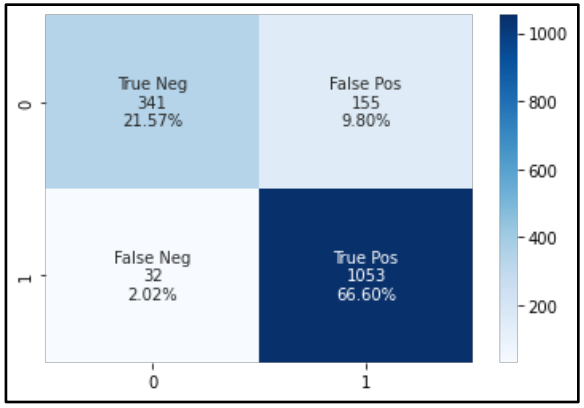
SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

Results

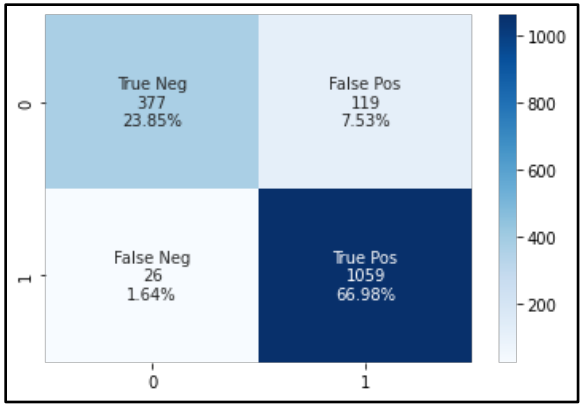
AlexNet Confusion Matrix



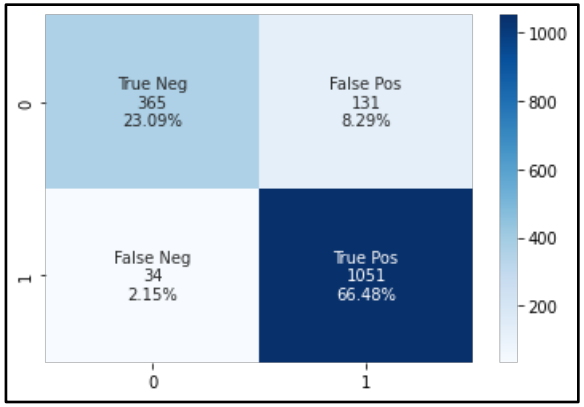
VGG19 Confusion Matrix



ResNet152 Confusion Matrix



DenseNet201 Confusion Matrix



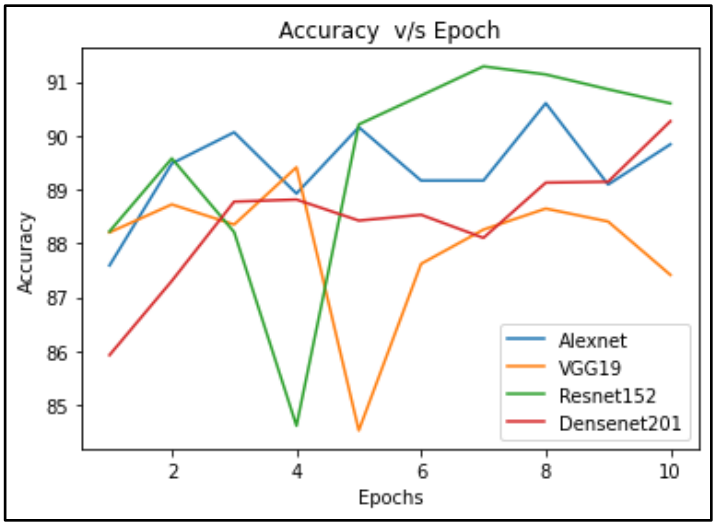
SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

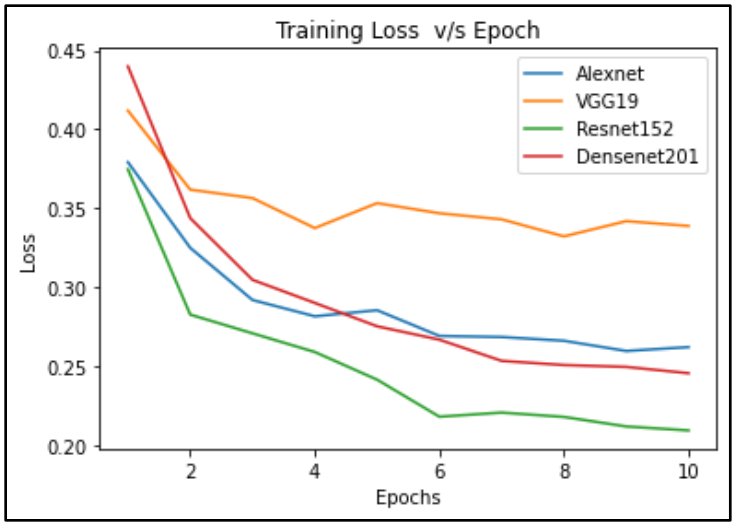
Results

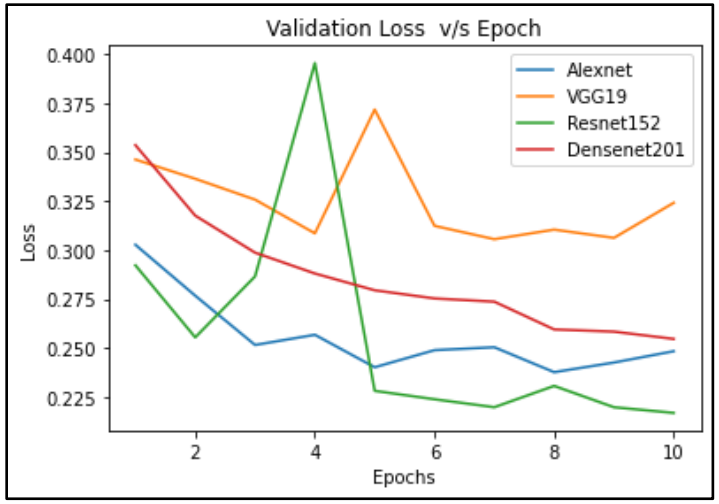
	Accuracy	Precision	Recall	F1 Score	Jaccard Score	ROC_AUC Score	Specificity	Training Time (mins.)
AlexNet	89%	0.9017	0.8994	0.8959	0.8151	0.8555	0.7379	25.6478
VGG19	88%	0.8850	0.8817	0.8765	0.7853	0.8290	0.6875	38.3050
ResNet152	90%	0.9104	0.9082	0.9054	0.8302	0.8680	0.7600	43.4959
DenseNet201	89%	0.8972	0.8956	0.8922	0.8092	0.8522	0.7358	37.7008

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

Results

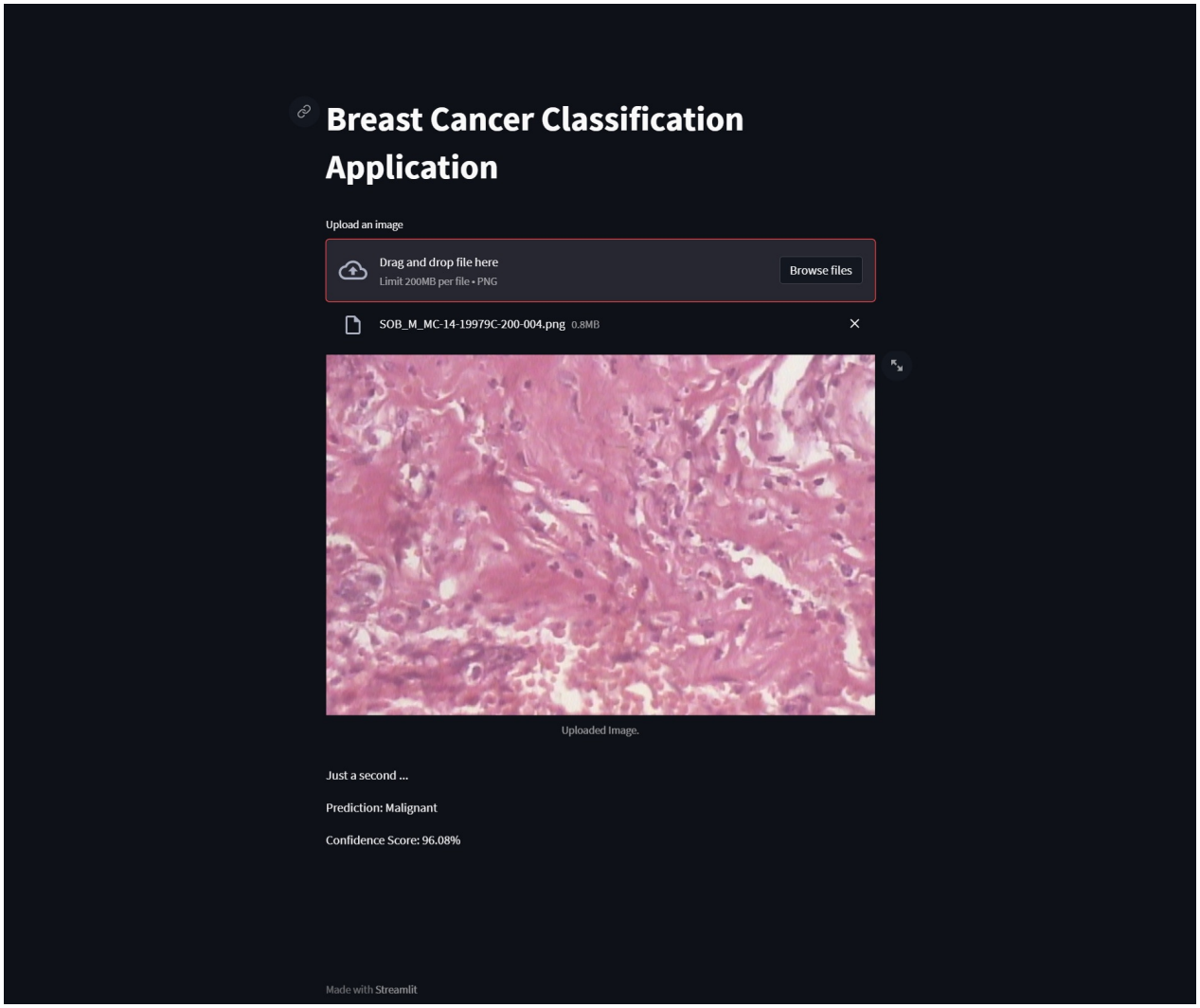






SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

Screenshots



SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

References

- https://www.mdpi.com/2072-4292/9/8/848
- https://debuggercafe.com/implementing-vgg11-from-scratch-using-pytorch/
- https://deepai.org/publication/tbnet-pulmonary-tuberculosis-diagnosing-system-using-deep-neural-networks
- https://towardsdatascience.com/creating-densenet-121-with-tensorflow-edbc08a956d8
- Elazab N, Soliman H, El-Sappagh S, Islam SMR, Elmogy M. Objective Diagnosis for Histopathological Images Based on Machine Learning Techniques: Classical Approaches and New Trends. *Mathematics*. 2020; 8(11):1863. https://doi.org/10.3390/math8111863
- Alzubaidi, L., Zhang, J., Humaidi, A.J. *et al.* Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *J Big Data* 8, 53 (2021). https://doi.org/10.1186/s40537-021-00444-8

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

Paper Submission Proof

[IJETI] Submission Acknowledgement Inbox ×

ojs.taeti@gmail.com 8:18 PM (0 minutes ago)
to me

Dear Anamitra Bhar:

Thank you for submitting the manuscript, "Comparison and Analysis of CNN Models on Breast Cancer Histopathological Images" to International Journal of Engineering and Technology Innovation. With the online journal management system that we are using, you will be able to track its progress through the editorial process by logging in to the journal web site:

Manuscript URL: https://ojs.imeti.org/index.php/IJETI/authorDashboard/submission/8966
Username: anamitrabhar

If you have any questions, please contact me. Thank you for considering this journal as a venue for your work.

The editorial office

The following message is being delivered on behalf of IJETI journal. _____

Reply Forward

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be **University** *u/s 3 of UGC Act, 1956*

Thanks