Chest tumors CT scan Datasets Project
Category: Healthcare

Israel Cohen **Kfir Inbal**

Abstract

This article revolves around the problem of developing a quick chest cancer diagnosis using machine learning. In this project, an encoder-decoder architecture is utilized to clean the data, after that, we used CNN algorithm to classify 3 types of chest cancer against the normal condition.

1. Introduction

There exists various types of chest cancer. In our data, we have used 3 types of chest cancer: Adenocarcinoma, Large cell carcinoma and Squamous cell carcinoma.
These types are most common around people who smoke frequently.
Adenocarcinoma, is the most common form of lung cancer accounting for 30 percent of all cases overall and about 40 percent of all non-small cell lung cancer (NSCLC) occurrences. Adenocarcinomas of the lung are found in the outer region of the lung in glands that secrete mucus and help us breathe. It appears that smoking is a significant factor in the initiation and progression of lung adenocarcinoma. However, 10–40% of cases occur in patients with no reported smoking history, suggesting the involvement of other risk factors, including environmental exposure and genetic susceptibility.
Next is Large Cell Adenocarcinoma. This type of lung cancer grows and spreads quickly and can be found anywhere in the lung. It usually accounts for 10

to 15 percent of all cases of NSCLC. There has been found a correlation of increasing risk of large cell cancer with both the frequency and number of years of cigarette smoking.
Lastly is Squamous cell carcinoma. This type of lung cancer is found centrally in the lung, where the larger bronchi join the trachea to the lung, or in one of the main airway branches.
Squamous cell lung cancer is responsible for about 30 percent of all non-small cell lung cancers, and is generally linked to smoking.
Symptoms of all these types include coughing, hoarseness, weight loss and weakness. Each of these types of cancer is diagnosed by CT Scans analysis.

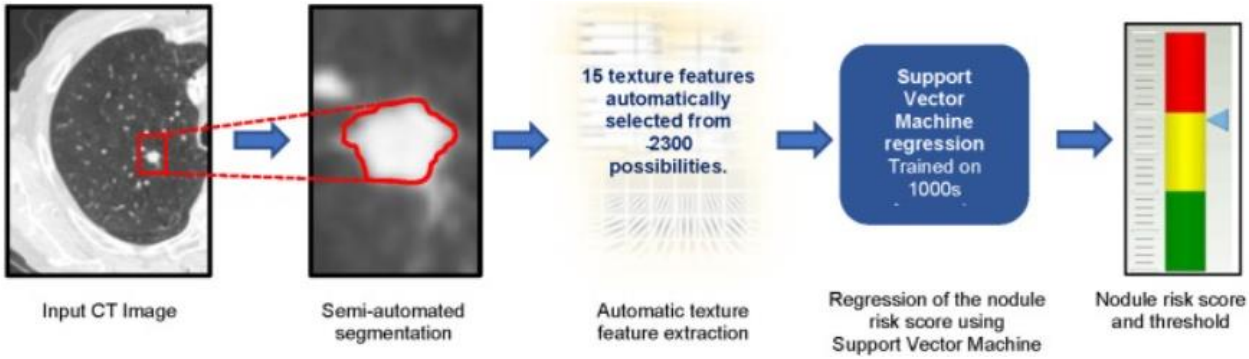
Ideally, NSCLC should be diagnosed in early stages, when there are more and better treatment options. However, since many symptoms are common to other conditions, a diagnosis isn't made until the later stages.
In addition, CT scans require an expert's examination, and with the growing population, more and more people do CT scans, making it a load of images to examine for the experts and so the standby time prolongs for the results to arrive for the patients.
If the reason above is not enough, NSCLC tends to be more aggressive and can spread quickly, making it even more dangerous to wait for diagnosis. Thus, a rapid and accurate identification of these cancers using machine learning is indispensable and highly important.

In this project, we developed a Convolutional Neural Network (CNN) algorithm to classify these 3 types of NSCLC using a Chest CT-Scan images dataset from Kaggle.
We first used an encoder-decoder architecture to clean the data from noise-like marks on the images to achieve more accurate images, and input its latent layer to binary classification of type Forward neural network (FNN).
We achieved a high accuracy of 97% and the NN can be used to give first identification of cancer.

After that, we took the images without encoder-decoder processing and got accuracy of 76-74% at best in multiclass classification.

2. Related works:

A. We saw an award competition winners work to classify lung cancer in a completely another way, their methodology based on segmentation learning as shown in the picture below:



Their data was Lung-RADS (which is a quality assurance tool designed to standardize lung cancer screening CT reporting and management recommendations, reduce confusion in lung cancer screening CT interpretations, and facilitate outcome monitoring) that gaurantied a high quality of the data (make it easier to learn).

Their machine used 2 model types to train over 2 types of data:
Model A was trained on size-matched data, and model B was trained on unmatched data.
They achieved average AUC for the classifier trained on dataset A - 0.70, whereas they used the size alone on the same dataset they got an AUC of 0.50.
The AUC was 0.91 for the classifier trained on dataset B.
This indicates that the classifier can learn morphological features that can discriminate between benign and malignant nodules and, moreover, that such features add approximately 0.2 AUC points to using size-alone.

Their goal was two subjects (that pretty much aligned to ours).
First, to reduce the variability in assessing and reporting the lung cancer risk between interpreting physicians.
Second, CADx could improve classification performance by supporting the less experienced or non-specialised clinicians in assessing the risk of a particular nodule being malignant.

As we wrote before, our data wasn't size-matched data, so as they explained, harder to achieve good results.

B. In another work, using dataset of different types of NSCLC categorized into three groups: ADC, SCC and an "Other" category that comprised all other NSCLC histological subtypes, including large cell and mixed histology.
Their image pre-processing included manual tumor identification, isotropic rescaling, and density normalization of input CT data.
They used VGG-16 model of CN for feature extraction and image classification.
As opposed to our project, their approach was to use transfer learning.
Meaning they used weights of the last convolutional and pooling layers using radiographic data for their learning process with CNN.
Another key to take into consideration from this work is their discussion.
They concluded that CNNs are effective at natural image recognition tasks, can be implemented to distinguish between the most common histopathologic subtypes in NSCLC. So with enough data, split into labels, CNN can distinguish subtle differences between images and thus to classify and predict classes. With this in mind, we head on to our project, knowing our methods are based.

3. Data details:

The data was collected from the kaggle site, and originated from many databases.
It has 1000 images, divided to train, validation and test folders at 70 10 20 percent respectively.

Preprocessing:

We have cropped the black frame from 4 sides and tried to work only over the rib cage in the image, because we found out later that all those black pixels (as they are a lot of zeroes) ruin our learning process.
The cropped caused a loss of exactly 31 images, as we want to throw away as many black pixels as we can, and the high threshold caused a little loss of data.
After that, we did an augmentation of horizontal flip and so the amount of images in our data went up to 1938.
We have done a transformation over every image to grayscale, and did CenterCrop of 350X350 pixels at FNN, and resize at CNN to 120, 120.

4. METHODS:

A. Denoising autoencoder (DAE)

DAEs are used to solve the risk of the Network to learn the “Null Function”, which means that the output equals the input.

We used the denoising autoencoder algorithm that takes a partially corrupted input and recovers the original undistorted input.

B. Feedforward Neural Network(FNN)

The FNN was the first type of Artificial Neural Network, thus we used this method as a base-line for Non-deep learning network.

The method is based on a Multi-layer perceptron type of FNN in which each neuron in one layer is feeded forward to the subsequent layer.

For that, the data was split into two categories: with lung cancer and normal(without cancer). Before applying the FNN on the data, we first used the DAE on the data. Then plotted out a group of images before DAE and after DAE(as you can see at Figure 5).

As well as plotting a T-sne graph. After that - we applied the FNN algorithm on the data.

C. Convolutional Neural Network(CNN)

The CNN method uses convolutional layer, pooling layer, and fully connected layer.

Each convolutional layer contains a series of filters known as convolutional kernels. The filter is a matrix of integers that are used on a subset of the input pixel values, the same size as the kernel.

Kernels scan the images at first and then shrinks their dimensions until it becomes a 1-dimensional vector. From this it continues to NN and then with the output prediction. With the results, we do the multi-classification.

5. Methods technical details

A.

Binary classification results - confusion measurement test output:

	precision	recall	f1-score	support
0	0.96	1.00	0.98	136
1	1.00	0.82	0.90	34
accuracy			0.96	170
macro avg	0.98	0.91	0.94	170
weighted avg	0.97	0.96	0.96	170

Figure 1: FNN binary classification results report

Train last epoch result:

Train: Epoch 014: | Loss: 0.08255 | Acc: 96.544

Validation: Epoch 014: | Loss: 0.03417 | Acc: 98.235

B.

Our approach:

a. Encoder-Decoder algorithm:

Our binary classification starts with an encoder-decoder algorithm:

The hyper params are:

learning_rate = 0.6 (as we used SGD, we wanted a high LR to make it faster)

epochs_num = 25

BATCH_SIZE = 1 (make it run longer, but more accurate)

Randomization:

We used random_split to make our data randomly assigned and add shuffle to the train and validation before running our program to get a broader guarantee of results integrity.
99% of the data went to the train, 1% to validation (test for encoder-decoder is irrelevant).

Algorithm layers:

Our latent_dims assigned a vector size of 900.

Encoder:
self.linear1 = nn.Linear(122500, 950)
self.linear2 = nn.Linear(950, 920)
self.linear3 = nn.Linear(920, latent_dims)

Decoder:
self.linear1 = nn.Linear(latent_dims, 920)
self.linear2 = nn.Linear(920, 950)
self.linear3 = nn.Linear(950, 122500)

Optimization:

Used SGD.

Activation function:

Used Leakyrelu at every layer except the last, which used sigmoid.

Loss function:

nn.MSELoss() (*comparison between pixel of output layer to input layer)*

Tuning hyper params:

Our method was a lot of a simple try and fail, in accordance to the Measurement results.
Batch size was decided to be 1, as we don’t have a lot of data, and the images batch did some averaging over a group of images.
LR was made to be on the higher side, as we saw that with SGD it learned otherwise very slowly.
Epoch decided while checking out validation and train accuracy output, as we don't get into overfitting, so we took the last epoch before that happened.

Our layers size, and SGD instead of Adam optimizer was decided while trying to use GPU, as GPU memory is limited and we did those changes to be able to run the program while using GPU.
So our layer size is almost at the maximum available size, as the input vector is set in stone (image size), after that we tried to diminish layers to get to latent.
Latent layer size was decided to be 900 to represent an image of 30X30 pixels, as it felt like the minimum lower we can go to be able to represent the characteristic of the cancer types.

b. T-sne algorithm:

Used the encoder output of 200 pictures encoded from the latent layer, and built a T-sne map of 2d.

c. FNN:

Hyper parameters:
Learning rate = 0.001
Epochs = 7
Batch size = 1

Input Data:
Input is 1700 images as encoder latent layer output.
80% goes to train, 20% to test.

Optimization:
Adam

Loss function:
BCEWithLogitsLoss.
It's contain One Sigmoid Layer + BCELoss (solved numerically unstable problem)

FNN layers:
From first layer to last:
self.layer_1 = nn.Linear(900, 200)
self.layer_2 = nn.Linear(200, 20)
self.layer_out = nn.Linear(20, 1)

Activation function:
Using Leakyrelu at every layer.

Resulting measurement:
Using confusion_matrix.

Tuning hyper params:
Our method was a simple try and fail, in accordance to the
Measurement results.

Our layers size, and SGD instead of Adam optimizer was decided while trying to use GPU, as GPU memory is limited and we did those changes to be able to run the program while using GPU.
So our layer size is almost at the maximum available size, as the input vector set in stone (image size), after that we tried to diminish layers to get to latent

We didn't use GPU here, as the program ran fast enough, so we used Adam as it's considered fast and in general better than SGD.
Epoch decided while seeing measurement results good enough.
Lr was decided to be lower than before, as Adam is fast, so we made the LR a little bit more slower than it was, and saw it work good enough according to the binary train and measurement at test output.

d. CNN multi-class classification

	precision	recall	f1-score	support
adenocarcinoma	0.73	0.71	0.72	240
large.cell.carcinoma	0.61	0.81	0.69	102
squamous.cell.carcinoma	1.00	1.00	1.00	104
normal	0.77	0.65	0.71	180
accuracy			0.76	626
macro avg	0.78	0.79	0.78	626
weighted avg	0.77	0.76	0.76	626

Figure 2: CNN classification results report

Hyper-parameters:

lr = 0.001

Number of epochs= 15

Batch Size = 8

Input data:

Input is 1938 images after preprocessing(cropped edges and horizontal augmentation).
60%~ train, 32%~ test, 7%~ valid.

Optimization:

SGD (Stochastic Gradient Descent).

Loss function:

nll_loss

Tuning hyper-parameters:

In this architecture, our method of tuning was again, try and fail, in accordance with the measurement results.
Batch size was decided to be 8 although we also succeeded getting the save accuracy percentage(76) with batch size 1. We used a relatively small size batch size because we don’t have a lot of data, and the images batch did some averaging over a group of images.
LR was made to be a little lower than in FNN, deciding with try and fail concept.
Epoch decided while checking out validation and train accuracy output, as we don't get into overfitting, so we took the last epoch before that happened.

Layers:

Started with kernel of size 1, in order to duplicate our data (4 times), and make it able to distinguish some features combined together.

4 layers, each contains:

Conv2d(in_channels= , out_channels= , kernel_size=),

BatchNorm2d(num_features),

LeakyReLU()

Conv2d:

Applies a 2D convolution for the given input from the input channels.

The values for function Conv2d are put as follows:

In_channels from 1 to 32(multiply by 4 for each layer)

Out_channels from 4 to 64(multiply by 4 for each layer)

Kernel_size from 1 to 7(add by 2 for each layer)

BatchNorm2d:

Applies Batch Normalization over a 4D input.

The values for function BatchNorm2d are put as follows:

BatchNorm2d size(the num_features) from 4 to 64(multiply by 4 for each layer)

In addition, for each layer from the second layer to the fourth layer we added:

nn.AvgPool2d(kernel_size=2 , stride=2)

Which applies a 2D average pooling over an input signal composed of several input planes. As we said, the average pooling would hopefully help to ignore some noise (line or letters) in our data.

6. Results Analysis

C. Visualization

Encoder-Decoder results:

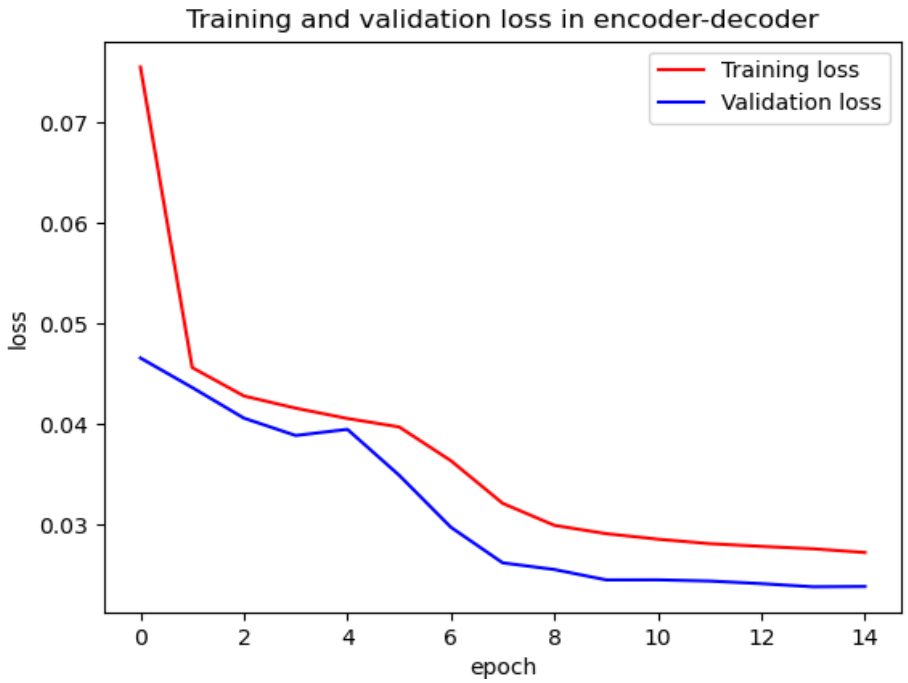


Figure 3: Training and validation loss results from Encoder-decoder

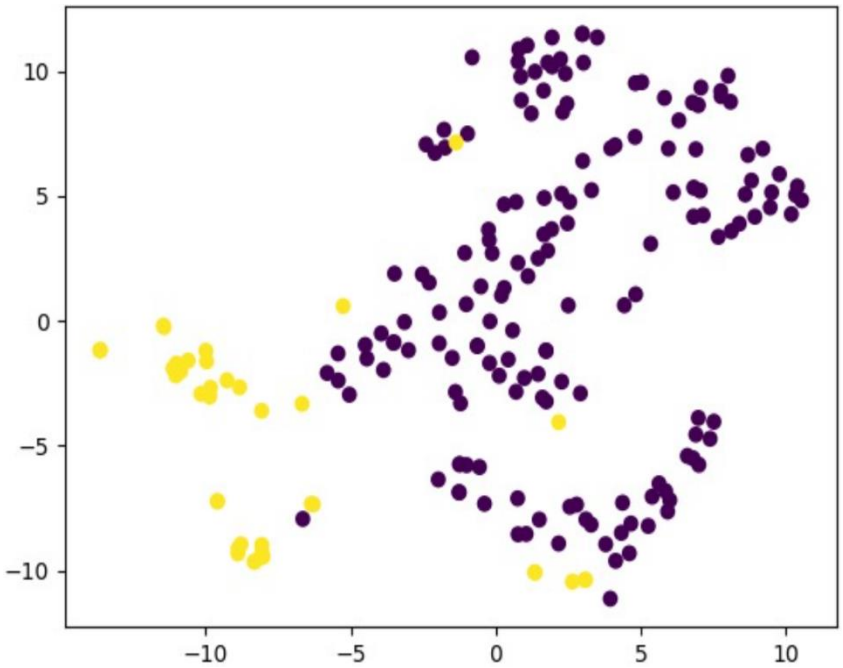
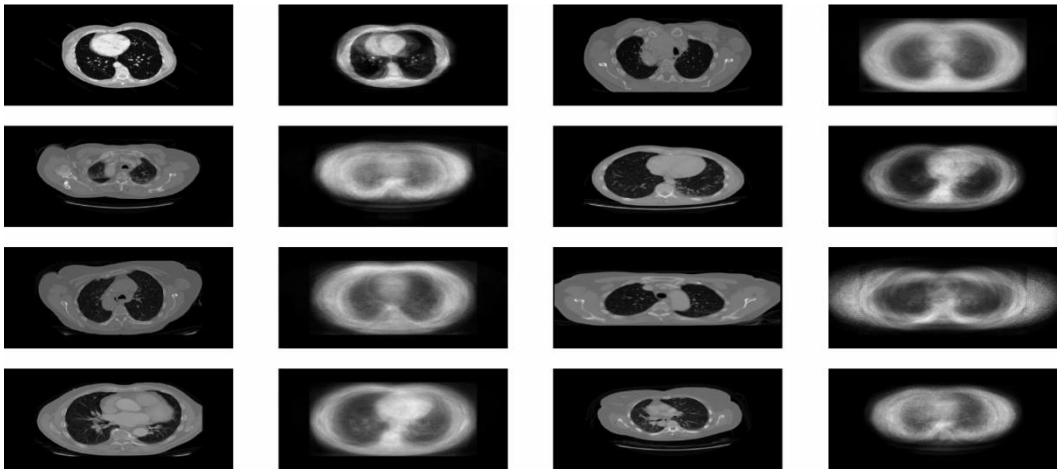


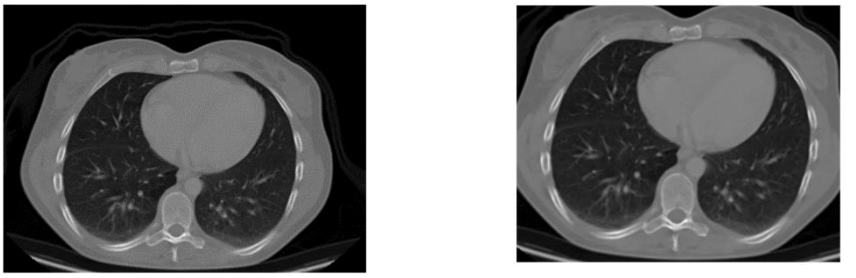
Figure 4: T-sne algorithm (binary classification):

Purple = cancerous, Yellow = normal



Figre 5: 15 images examples of DAE

Figure 6: The DAE cleared the white mark(right-up) from the original image(left)

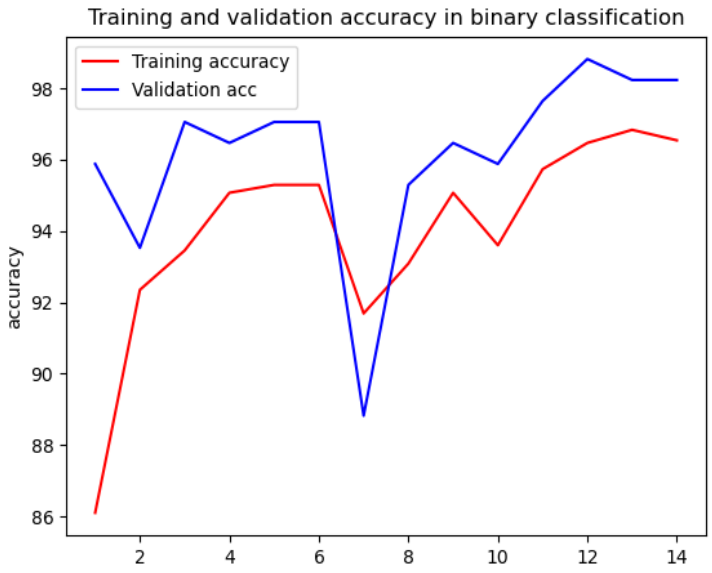


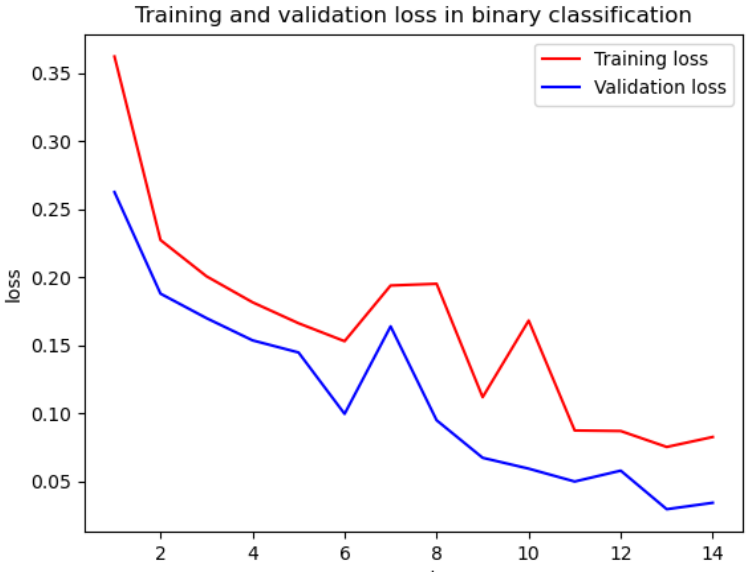
Uncropped image

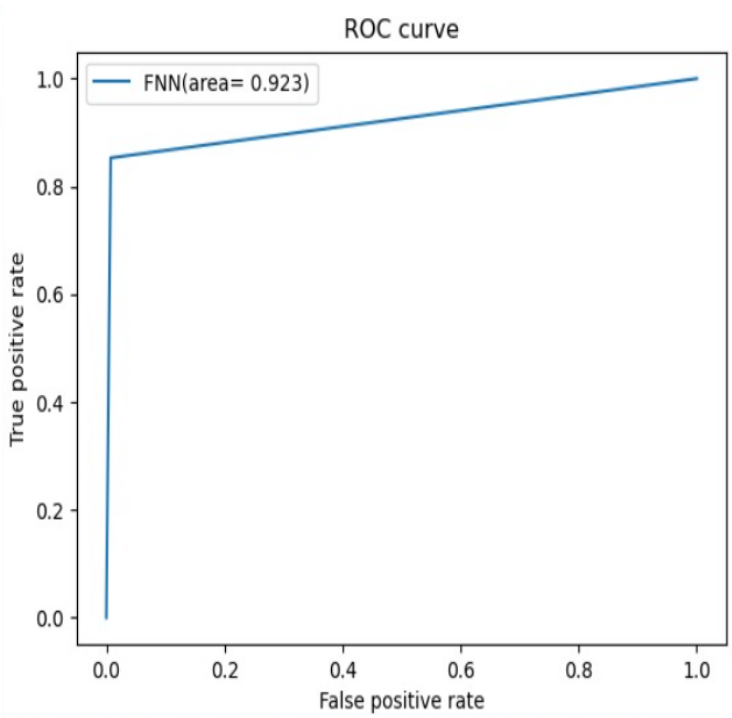
Cropped image

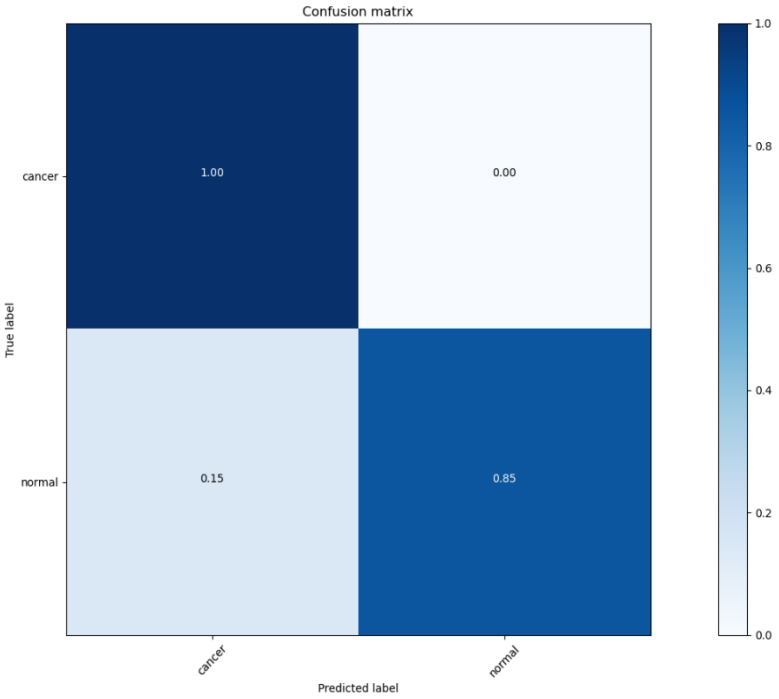
Figure 7: Cropping images example

FNN - Binary classification results:

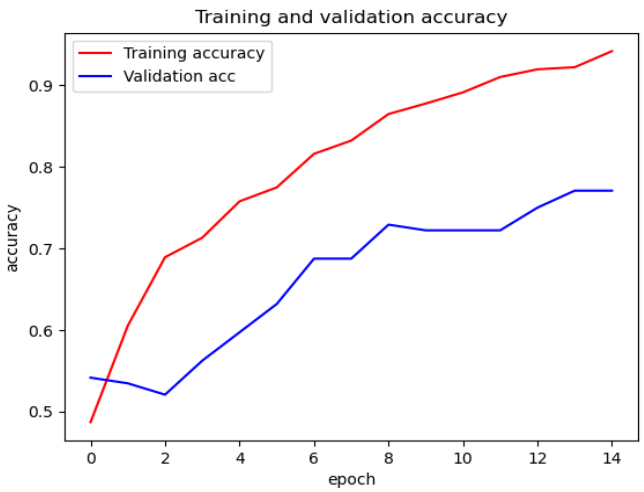


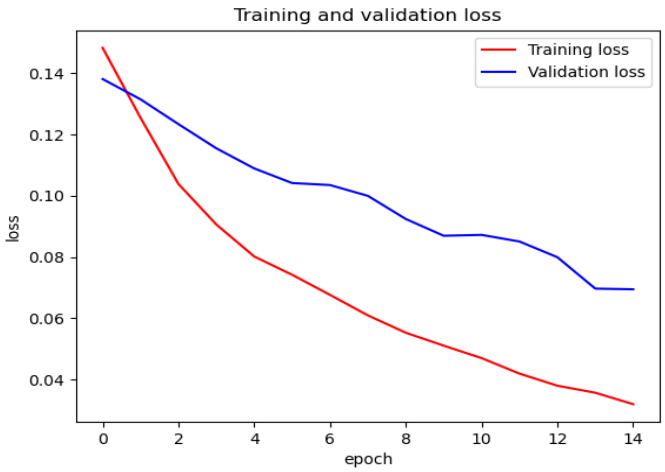


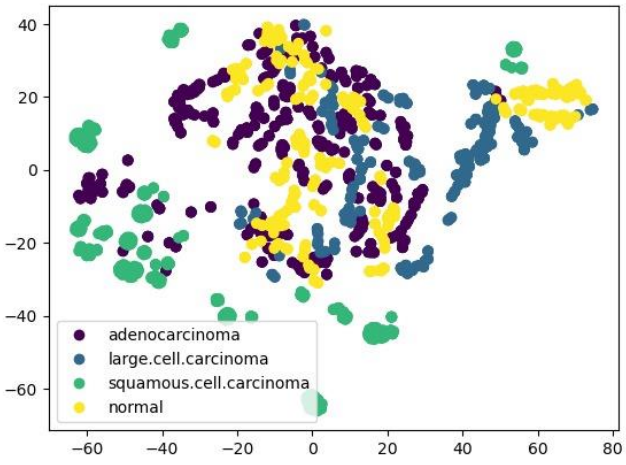


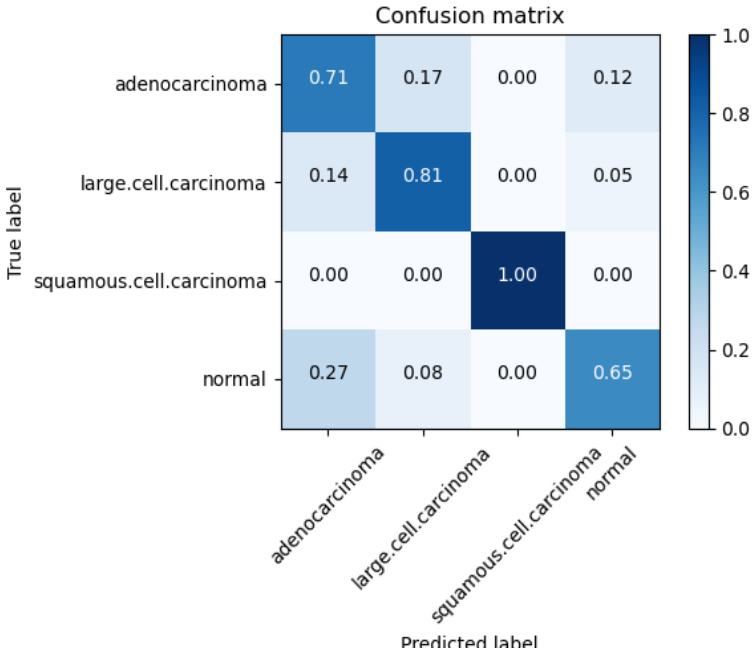


CNN multi-class classification results:









*Note: We didn't make an ROC curve for the CNN classification because we failed to do so(in the given time), and the built-functions for making ROC curve in Python are built for binary classification only, so customizing it to work with 4 classes was to complex for the time we had left.

The binary classification model did a much better job, the reason for that in our opinion is because it's pretty much easy to differentiate as we can see from T-SNS's we have shown above.
Trying to do classification with 3 cancer types is much harder, as it's not always that big a change as you can see.
Simply put, the reason to the success of the binary FNN algorithm over the CNN multi-classification one, is that the first had to decide between image that held certain characteristic other than the normal, added touch over the normal ones, and the CNN, had to differentiate between the unique characteristic, which sometime can be pretty close to be identified as from being from the other cancer group.
From the confusion matrix, we saw that the squamous cancer is the easiest to identify, by many test run we did, and it's indeed made sense, as its uniqueness is very bold, it's a cancer that hold like most of the space of the lungs, the other to cancers are a bit more tricky in compare, and that's mainly the reason the relatively failure of the CNN prediction to the Binary FNN classification. Easier problems equal higher success.

Performance appraisal:

In conclusion, both out model did a great job, furthermore, they did it with a relatively small images data (less than 4000), which mean with more data, a greater success could be achieved,
Our binary model got a fantastic ROC curve, and fenomenal results in confusion matrix - reached almost 98% accuracy and recall.
Our multi-class CNN algorithm did a great Job too, as according to our results, it can identify with a high certainty (at 75+-% accuracy at our current very best results) anyone of the 3 cancers, while being able to tell if it's not a cancer at all.

Try and failed:

1. Tried doing normalization like that:
   transform = transforms.Compose([transforms.Resize((180, 180)), transforms.ToTensor(),transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5)),transforms.Grayscale(1)])
   And it didnt work well, as our decoder output looked just bad and far from the original.

2. At the Encoder-Decoder algorithm, we tried first to use image size close to the original image (as we said in the presentation, we calculated the mean of the overall images size), as well as big layers in NN, to minimize data loss.
   Without using GPU, it turns out, with a pretty good computer, to take like 2-3 days.
   In order to make it much more efficient (but less accurate as we would like to, as we felt we had not enough time) we changed from using CPU to GPU, and so, resize to much smaller images and smaller size of fully connected layer.

3. We have tried using Adam optimiser instead of SGD, and it didn't work well with running over GPU, as well as using second and third layers higher than 1000 length vector caused a bug, as the GPU didn't have enough memory to calculate all that data simultaneously.
   For the same reason, we chose batch = 1 (which help to accuracy overall), and images was cropped to 350X350 instead of average images size as was intended, as that took too much memory (first layer originally should have been at size of 684X492 = 336528 as layer vector length, and using fully connected to the second layer of length 1000 made it too big to handle).
   Learning rate change from 0.0001 to 0.6 for the change from Adam to SGD, and with 10 epochs - it managed to achieve loss of only 0.02, so I changed that to 30 epochs with LR of 0.4, and achieve loss of train: 0.01+ with validation loss of: 0.01+.
   We saw Decorator output better matched to original images.

4. We failed to achieve good results with our CNN at first, so we found out we had 2 problems:
   A. The original images contained a black frame, which means a lot of zeroes in the images data.
   We solved it by cutting each image's black frame before resizing.
   B. We tried using rotation augmentation of 0-5 degrees, and we got an accuracy of 60+- at best. So we thought that it's too similar to our original images, and so, causing over-fitting. We changed to horizontal flip augmentation only and achieved an accuracy of 74%.

5. We have tried using resize to 350X350, but it gave less good results compared to e.g. 180X180. We thought the cause may be that some images' size is smaller than 350X350, so the resize actually "stretched" these images and it might have caused some problems with image pixels precision.

6. High batch number, caused averaging images and so lead to less good results.

The said above explain why we didn't use Encoder-Decoder algorithm with CNN, as we tried to do multiclass classification, and so, we needed precision of images pixel, and the Decoder output diminished that because of the kind of under-fit presentation of the images in not big enough vectors.

7. Discussion

If we had more time, we would like to:

1. Add segmentation algorithm to enter image to the CNN, and get it as output with the cancer being marked.
2. Check out more in depth augmentation methods (as we saw that rotation of 0-5 degrees didn't do good to the train, probably because it's too close to the original image, and it achieved overfitting fast, or been tested over too much similar images.)
3. We would like to use latent output of encoder decoder to CNN, currently we circle the problem of image containing some lines and words with using average pooling in the CNN, but maybe we could achieve better results with encoder output in the overall cleaning of the data.
4. Clean the data in a better way, as we saw that CNN had some difficulties to get a high value of precision of large cell carcinoma. From looking at images data, we saw a lot of images not in good proportion, like very wide while being very low and having bad quality images overall our data. If we would filter those, we assume we would get much better results in CNN test measurements.
5. We would like to check out other versions of CNN types, to check out that maybe some would give better results than what we currently achieved.
6. We could try out in much more depth various layers in our own CNN, as we used CNN from a machine learning course and didn’t change it very much, (number of layers, strides, etcetera).
7. We would like to give the computer time to calculate the binary classification without the GPU memory restriction, so that we could get better images from encoder-decoder algorithm, and probably, better results in: binary classification (using encoder output), CNN (if encoder latent output would be used as input to CNN layer), T-sne separation.
8. Try out more linear classification and do a comparison between the results.
9. Made a better automatic hyper params tuning.

Work needed in the project:

1. In addition to the above, we really need to do an in depth search in google data over CT-chest scan, in order to get much more images to train on.
2. Use Pickle to save the very best weights of the CNN.
   We saw that the CNN can achieve very good results, but with the very same hyper params, it can achieve very bad results (e.g. one run can get 74% accuracy, other can get 30%, that mean that the CNN can learn a locally best prediction instead of the very best, and that is probably because we used batch of size 1 and low LR, so we have a lot of "noise" in the learning progress where it can be stack)

Conclusions:

We can help humanity with our computer knowledge through image processing, which is truly amazing.
According to our CNN and Binary classification results, we saw that we can help with a very high certainty to identify lung cancers, and with the mentioned improvements in the previous section, we might achieve our goal in helping hospitals around the world to function better.

8. Bibliography:

Data originated from:

https://www.kaggle.com/mohamedhanyyy/chest-ctscan-images

Related work:

A: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6037965/

B: https://www.nature.com/articles/s41598-021-84630-x

Help links:

https://towardsdatascience.com/a-simple-cnn-multi-image-classifier-31c463324fa

https://www.healthline.com/health/lung-cancer/large-cell-carcinoma

Appendices

Link for our Github repository:

https://github.com/Kfirinb/ImgProcessingProjData