Logbook

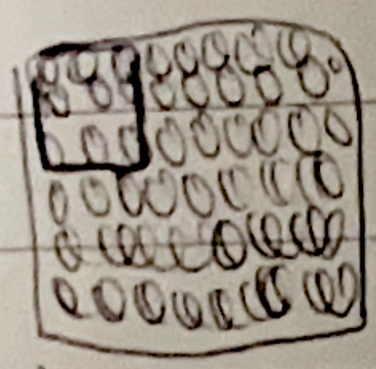
Anie Udofia, Gr. 9, ~~anieka~~ aniedofia8@gmail.com

Logbook must include:

- Schedule daily notes/ideas, background research contacts, references, experimental procedure/method, data collection sheets, observations/results, conclusion
- Aspects of the project performed differently
- All change!
- Extreme detail to ensure replication

MIT 6.S191, 2024 - Source of images

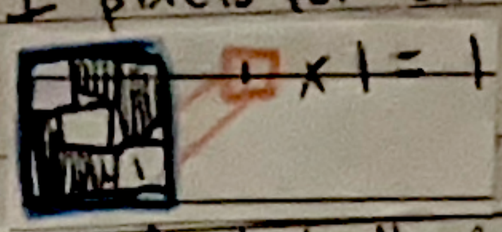
Convolutional Neural Networks (CNNs) are neural network models that process images of data through the extraction of features (important aspects of the input) in order to assign them to a particular class. These features have a set of weights executed on them, known as a filter, which can be of varying size. These filters are implemented on several different features to encourage algorithmical efficiency + ease instead of having individual pixels being processed



Input layer

Ex: Filter size is 3×3 → 9 different weights

↳ Move the filter by 1 pixels for a subsequent patch → convolution

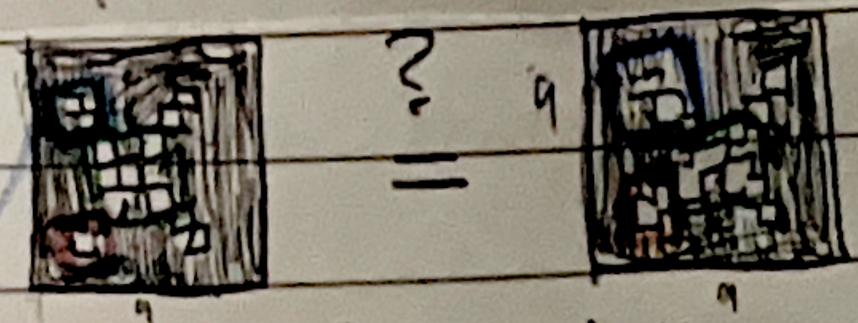


1 × 1 = 1

element wise multiply (for each element on the grid)

= 9

doesn't jam all the values into one large matrix of all the values (they are distributed among the filters in each layer)



? =

Teach model that these are both Xs

= features of the images

↳ Then we add a filter (3×3)

W = -1
W = 1

Convolution still allows for the preservation of spatial information because it

Abstract:

This scientific article explores the identification of malignant, cancerous tumors via the employment of Convolutional Neural Networks, a deep learning model engineered in Python code. The approach I present develops a machine learning algorithm equipped to attain a threshold of 80% percent within only one week of training, where each day, the CNN is fed CT scans through the means of user input displayed on a personally constructed website. On a daily basis, measures will be implemented to ensure the refinement of the network in order to guarantee optimal performance and results by leveraging the pattern recognition techniques of regularization, performance indicators, optimizers and dataset enhancement, each of which will be meticulously documented in the following chapters. The objective of my novel design* is to provide a solution to the alarming gap in the availability of rapid and proper resources for individuals affected by cancer. This continues to accelerate the process of diagnosis as well as offering medical professionals an additional analysis tool among the growing pool of diagnostic frameworks.

~~*~~ = change

* is to explore + audit the powerful capabilities of CNNs in assisting in the bridging of the alarming gap in the availability of rapid + proper resources for individuals affected by cancer. This continues to accelerate the process of diagnosis as well as offering medical professionals an additional innovative analysis tool among the growing pool of diagnostic frameworks

The similarity of the two initial X's is determined by the element wise multiplication that sums up every locations results (which becomes the output).

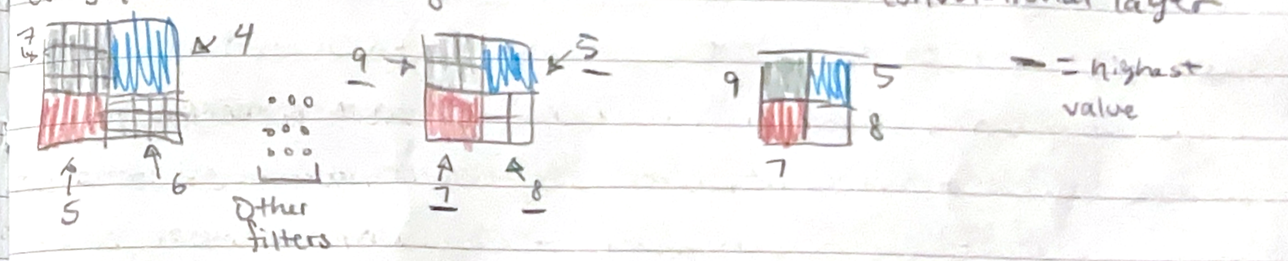
Equation of convolution (4x4 filter):

$$\sum_{i=1}^{4}\sum_{j=1}^{4} w_{ij} X_{i+p,j+q} + b$$

- summing over indices i and j from 1-4
- input from prev layers
- weights
- bias
- (weighted sum of filter's weights and the corresponding input values)

for neuron (p,q) within the hidden layer

Maxpooling - The process of extracting the highest, or maximum value of every patch created by the filters to the next convolutional layer



Pooling helps to minimize the dimensionality of the features, allowing for the lack of abundance of hyperparameters, and the preservation of spatial invariance

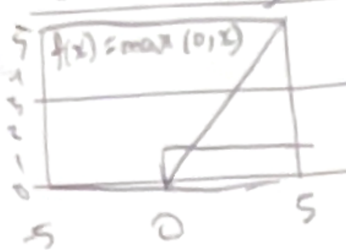
Non-Linearity

- CNNs often use ReLU activation functions due to their maximum operation feature that maintains a continuous positive gradient for inputs that return positive and a reaction of zero for negative values (not strictly 0 and 1 but >1 and 0)
- Dismantles (solves) vanishing gradient*
- Leaves room for nuance + mirrors perceptron

Note to self

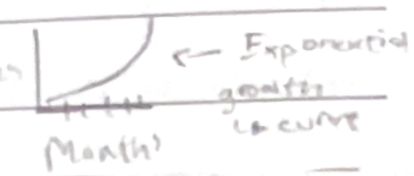
* = problem during backprop when the weights become very small as they are fed backwards from the gradient of the loss function (how does a small change in a weight affect the final loss). Symoid, the activation function, causes this because it squishes values between 0 and 1 so they have even smaller derivatives for input values. This problem is easily combatted by ReLU because ReLU employs a system of values higher than 0. Vanishing gradient causes some weights to not improve at all because the weight updates gradually become unnoticeable, halting learning and bringing about a lot of frustration and unhappiness.

(but better) ★ ReLU is applied to every pixel, replacing all negative values (less than 0) to zero itself



↳ My assumption is that this is to reduce the gravity of unnecessary weights that would have carried negative values to remain at equilibrium, and to not overinflate over weights that now seem more important when they are not

- ReLU is applied after every convolution layer to each individual unit, only offering non-negative values
- Nonlinearity allows networks to be accustomed to the data of the real world, where values don't always prove to be straight lines



Classification

Once the advanced level features of the previous layers have been outputted, the output is flattened (1D vector, more space) to be interpreted by the final fully connected layer (Dense), which used said features for image classification.

- The classification of the input image is expressed in the form of a probability (how likely this image belongs to — class)

↳ How likely this image depicts the presence of cancer

- In order to convert the output into a probability, we use the sigmoid function, converting the set of values into probabilities between 0 and 1

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

softmax can help inform us of the distance between two probability distributions (between the predicted output and true, or accurate output), so it is more relevant to the mind process of the machine (at least (this is it does).

negative values are the same as 0 because there is no activation of weights to carry into further layers, so replacing negatives with zero makes more sense also because it eliminates unnecessary exceptions (No activation = 0 and #<0, etc.). Simplicity is more sought after to be more efficient, especially with the creation of a model within only the span of a week.

The model currently includes Grad CAM in the code because it visualizes the predictions of the model while employing the gradient of the output with respect to the activations of the CNN layer. Research says it is easier to use compared to the boxes because it is more precise

equally used in loss quantification to our overall loss → activation = 'softmax'

*For model use .h5 file bc → useful for scientific + numerical data (weights, hyperparams) (Greeks for Greeks) > Load H5 files in Python

↳ Told me to use .keras instead because ".h5 format is legal. Please use the .keras format instead"

Methods:

① Image-Preprocessing:

- Hardest part is actually uploading images to use on my computer to be classified by the CNN model. The process is very tedious and requires a lot of troubleshooting. Currently, I am trying to change my directories and file environments to accomodate to the demanded paths of the program (VS code). I already had the files and the folders set, but the paths I tried to use did not work due to not being set to work in my intended environment. I navigated to my directory → cd "C:\Users\——\vscode\CNN\ImageDownloader" and adjusted my files to include this path (I think I made the path recognize the files by specifying through Powershell actually). Every time I tried to run my script: python download-images.py, it gave me one error code after another (realised it was the way I formatted my file [folder → text; Python file = .py, not an additional PY file which turned to a py.py file], so I fixed that and went to my Image Downloader Folder, created a Text Document, opened it with notepad and created a set of code to open the file of the images I downloaded from The Cancer Imaging Archive → LIDC-IDRI). I was able to see the names and amount of files I had under my directory by using the dir function in Powershell, after activating the ...\CNN\ImageDownloader

When I tried to run my script, it showed another error message regarding my URLs, saying, "500 Server Error: for url: .., Error downloading: ..," and I believe this might be due to invalid URLs or limited access. This is only the image pre-processing

Thankfully, I have finally found a dataset that does not require me to go through the unnecessary hassle of transcribing the images to a path to download that I do not understand what to do.

and this catalogue is: IQ-OTH/NCCD- Lung Cancer Dataset.

The dataset includes labeled CT scans of lung cancer, so my assumption is that I must find another data set where the images are unlabeled. My goal right now is to feed my CNN % of the files tomorrow (there are 1100 images) because it is at its early stage of classification and I want it to be easier for me to understand what is going on. There are a total of 110 cases divided into 3 major categories: benign (b), malignant (m) and normal (n) (40m, 15b, 55n + cases). Description says they were collected in DICOM format, but on my files, showed up as JPG

- 561 malignant images
- 416 normal images
- 120 benign images

File (must adjust code accordingly, instead of being made for DICOM files, must focus on JPG files).

600-700 images per day

Last day (day 7) is the determinant of the model's accuracy.

For my CNN, I am using Streamlit as the website for my model, acting as the backend of my interface, while the more decorated Readymag page introduces the user to my CNN: ONCO SCAN

The Readymag houses the network address that the user can then go to (there are 2 websites joint together: front-end, back-end). Streamlit doesn't allow you to directly edit your application on the website: you have to write a script of code describing the website (appearance), option for image upload, and the model (I used vscode Notepad). Then I run the script with Windows Powershell (as admin, then I select the directory and environment I used).

ST = same thing

What I will do for code:

import streamlit (st), pandas (pd), tensorflow (tf), sequential, load_model, Dense, Flatten (tensorflow keras.models, .layers), numpy (np), cv2, os, matplotlib? → already using cv2 and st, so no need (for visualization)

def create_model (input_shape): ↵ IMAGE_HEIGHT, IMAGE_WIDTH, = ENTER * more space

model = Sequential(), model.add (Flatten(input_shape = input_shape)), model.add (Dense (128, activation = 'relu'), model.add (Dense (1, activation = 'sigmoid')) # Cancer or no cancer [or] activation = 'relu' if I wanna do malignant, benign, normal. under cancer present/non-present

model.compile (optimizer = 'adam', loss = 'binary_crossentropy', metrics = 'accuracy'), return model

CONSTANTS

IMAGE_HEIGHT, IMAGE_WIDTH = 150, 150

Because cv2 has default, but in b RGB (kept getting warning messages, search on why this) ✱

EPOCH = 10 (but it can be from min = 1 to max = 100)

BATCH_SIZE = 32

MODEL_FILE = 'lung_cancer_detection_model.keras'

✱ I have both a python script + a streamlit app script; streamlit app goes more into detail, but not sure why I have the python script, I will just keep it, think it will still be helpful

base_data_dir = os.path.join (os.getcwd(), 'data')

train_data_dir = os.path.join (base_data_dir, 'train')

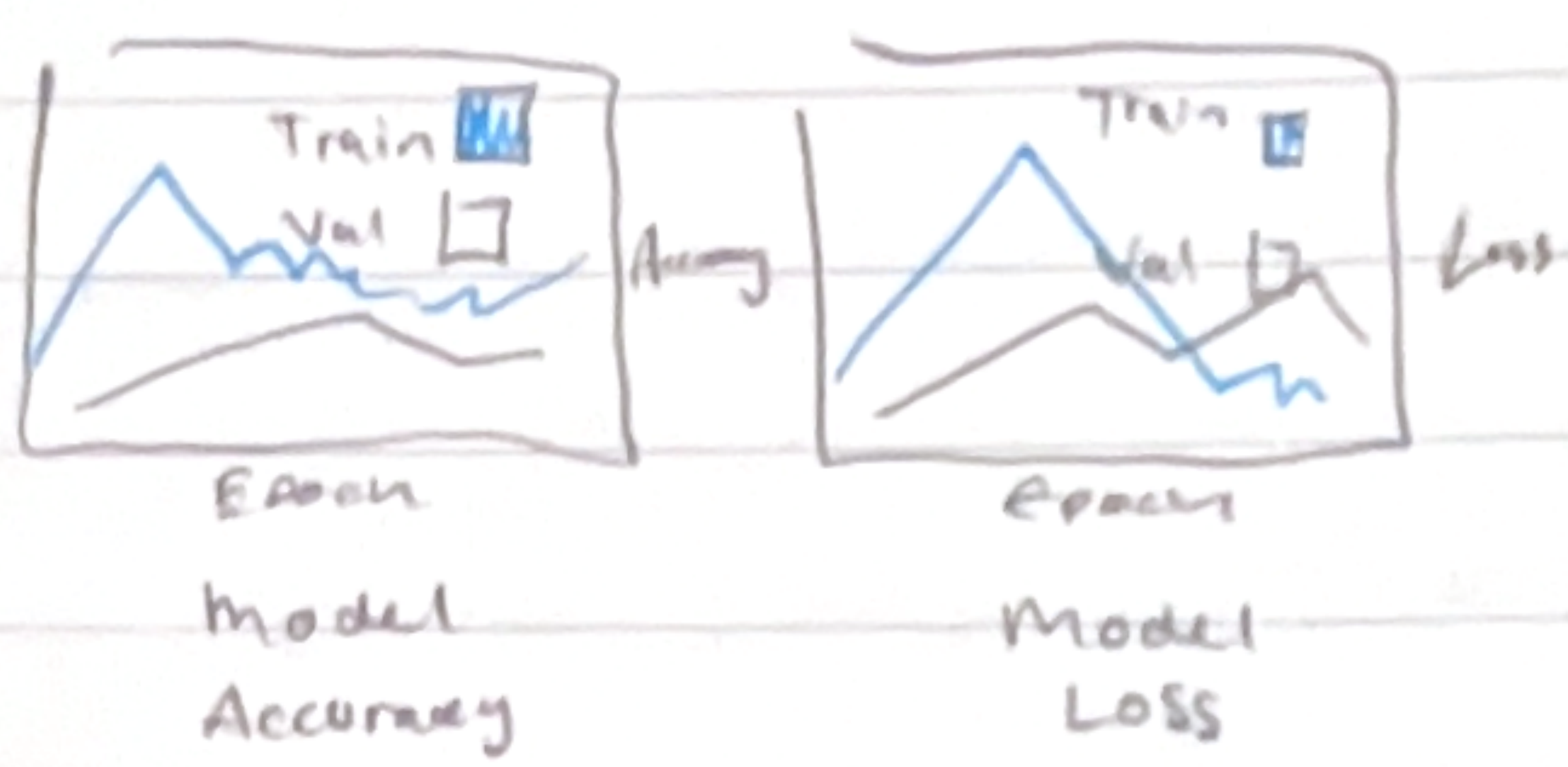
val_data_dir = ↓ (↓ , 'val')

test_data_dir = ↓ (↓ , 'test')

ST

(new line = indent 2) train_datagen = ImageDataGenerator (rescale = 1./255, rotation_range = 20, width_shift_range = 0.2, height = 0.2, shear_range = 0.2, zoom = 0.2, horizontal_flip = True, fill_mode = 'nearest'), val_datagen = ImageDataGenerator (ST)

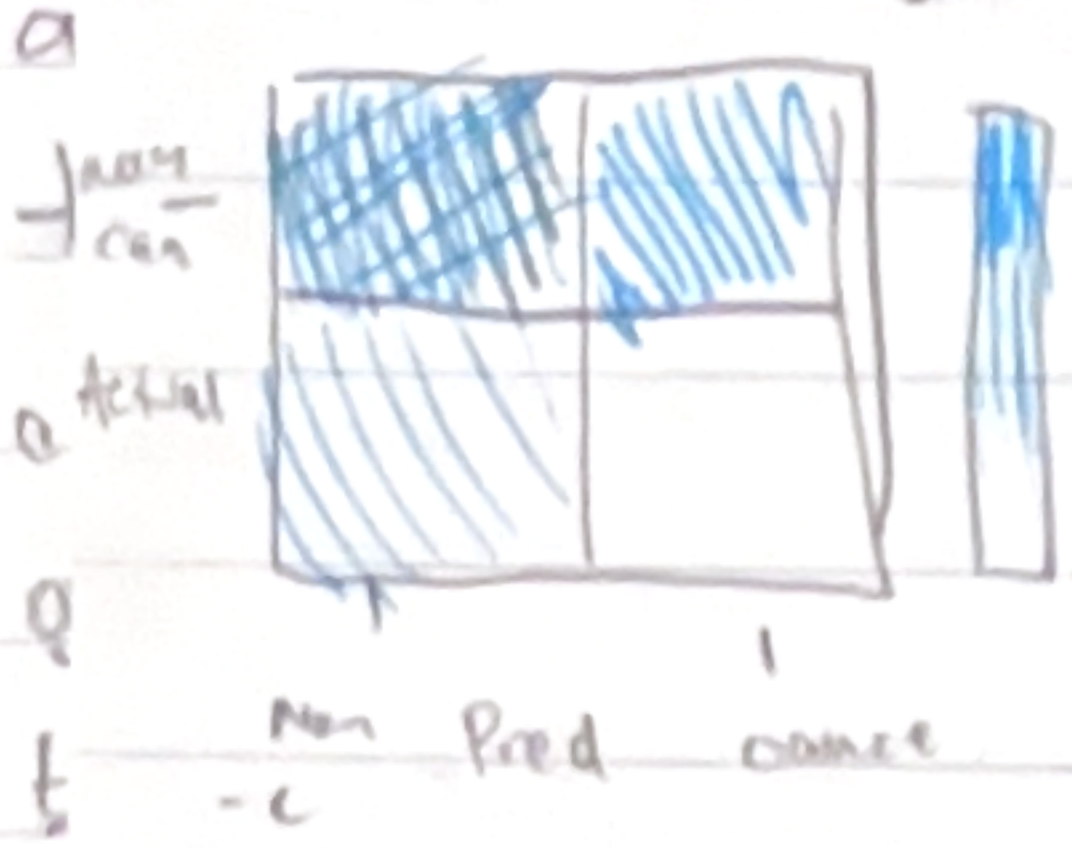
function + flow_from_directory (train_dir, target_size = (224, 224),
batch_size = BATCH_SIZE (32), class_mode = binary
(does it already know its from base dir?)
preprocess img = load_img (img_path, target_size = ST)
image_array = img_to_array (img), image_array = np.expand_dims(image_array,
axis=0) * Pilot said to as well
, image_array = preprocess_input (image_array)
matplotlib for training = Train Accuracy, Val Accuracy, Train Loss, Val Loss



* maybe I should add emojis to make interface more intriguing and more fun, less boring and with the (business + tonic to make site more appealing..) ☺

(Google says measure performance in this)

scikit-learn + Plot_lib for overall = Precision, Recall, F1, Accuracy



F1 = ___
Precision = ___
Recall = ___
Accuracy = ___

Prediction = model.predict (processed_image)
result = "Cancerous" if prediction [0][0] > 0.5 else 'Non-Cancerous' & cause bias towards non cancerous cause of bias always activated and there are less normal so more appear to normal offset thing
st.subheader ("Prediction Result ☺")
st.write (f"The model predicts that the image is: {result}")
no clue what last conv_layer_name so maybe I should do a backward math thing like, last_conv_layer_name = dense_layer [-2] or smth similar :).

cache! = if st.button ("Clear Cache"):
st.cache_data.clear()
st.success ("Cache cleared successfully!")

btw changed my mind about datasets being only one. Found 3 other datasets on Kaggle to use as well, and I have them downloaded. I have a folder of data on my device so I can update my data after every ...

Day 4: Observations

Windows Powershell model shows table =

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 150528)	0
dense (Dense)	(None, 128)	19 267 712
dense_1 (Dense)	(None, 1)	129

Total params = 57 803 525 (220.5 MB)
Trainable params = 19 267 841 (73.5 MB)
Non-trainable params = 0
Optimizer params = 38 535 684 (147 MB)

So far right now: 33 steps / epoch

Accuracy: (ends as) 1.0000, 0.9876, 0.9968, 0.9983, 0.9984 0.7265 ~ biggest to smallest

Loss: (begins as) 0.0000e+10, 0.1982, 11.0870, 0.1000, 11.0470, 0.0033, 7.0332e-19, 5.3226e-21, 3.2982e-21, 2.4744e-21, 3.7319e-21, 1.7108e-21, 6.5832e-21

Val-Accuracy: 0.9962, 0.6921

Val-Loss: 1.0174, 1.0199, 1.0200, 0.4470, 0.7194, 0.5785, 0.5665, 0.5660

For 33 steps

Notes: 100% accuracy might be an indicator of overfitting (20 epochs) * some repeat and last is correct value

(Research says that CNNs are prone to overfitting.. "due to their high complexity and large number of parameters." [AI and LinkedIn comments]), especially since the loss value is so low. Training loss became as low as 0%, demonstrating the model's ability to learn the training data, but could also be another case of overfitting with all the hyperparameters. The validation accuracy is a good start, showing the model identified patterns that were similar in the training data; can make the strong connections we are looking for. Validation loss could be better though, but the model's reaction to unseen data is promising, and I am very excited to see how this will gooo

Today, I fed my model 20 480 images (20 epochs, 32 steps per epoch (batch size = 32) or 20 x 32 x 32 = 20480) → But the images are repeated so ~ 2000 images

finish all my data of my streamlit app + computer.)

Day 2: Observations

Windows powershell model shows table=

Layer (type)	Output shape	Param #
flatten (Flatten)	(None, 150528)	0
dense (Dense)	(None, 128)	19 267 712
dense_1 (Dense)	(None, 1)	129

TP = 19 267 843
Tr P = 19 267 841
N-TP = 0
OP = 2

- However, when I click a file for the image to guess, it gives me an error message on how my "layer sequential has never been called and thus has no defined output". The configuration of my layer doesn't follow the format that the computer can understand, because I never called it.
Now, my app code isn't giving the input image in the specific shape the model is expecting from the flattened output of the last CONV layer. The image dimensions aren't the same (oversight on my behalf) → "expected axis-1 of input image to have value 150528, but received input with shape (1, 67500)".
It wants = 150528, but it get 167500 (too big)
Trying to solve issue by allowing the model to handle grayscale images (Helpful but didn't really work)
Adding model summary to do calculations when I need to debug
Instead of having the app be codedependant on windows powershell results I can deploy the app using a Github repository I created
But it needs the lung_cancer_detection_model.h5 in the Github repository and the file is too big, so I need to download Git and Git LFS.
After download, I cloned my repository and now I added the model file with Git LFS to my repository

Accuracy: 0.7603, 0.9062, 0.9744, 0.9983, 0.9664, 0.9375, 0.9858, 1.0000, 0.9275, 0.6915

Loss: 0.0047, 10.9854, 9.5724, 0.1273, 4.6205, 7.7494, 6.8155, 0.6646, 0.0125

Val-A: 0.8675, 0.9255, 0.9125, 0.9045, 0.8728, 0.8125

Val-L: 0.6785, 2.005, 0.7154, 8.4534, 7.2965, 0.0142, 0.4744

Day 3:

TP = 4 828 481 (18.42 MB)
Tr P = 4 828 481 (18.42 MB)
N-TP = 0
OP = NP (didn't even show)

Layer (type)	Output shape	Param #
conv2d	(None, 178, 178, 32)	896
max_pooling2d	(None, 89, 89, 32)	0
conv2d_1	(None, 87, 87, 64)	18 496
max_pooling2d_1	(None, 43, 43, 64)	0
conv2d_2	(None, 41, 41, 128)	73 856
global_average_pooling2d	(None, 128)	0
flatten	(None, 128)	0
dense	(None, 128)	16 512
dense_1	(None, 1)	129

Epochs = 20, Batch size = 32

- Problem still persist where my app did not recognize my data folder path's existence. Attempted many strategies to fix problem, like rewriting my code, rearranging the order of my code, and I am now trying to add my data with Git LFS because my code and model.keras file are all on a Git Hub repository, so I think it can't recognize my data because it is found locally on my device. I revisited what the repository of other models looked like, and they had their data included in their repository.

Accuracy: 0.6351, 0.8750, 0.9688 (×3), 0.9751, 0.9911, 0.9943, 0.9958, 0.9962, 0.9968, 0.9988, 0.9968, 1.0000, 0.9943, 0.9813, 1.0000, 0.6125, 0.5345, 0.6712, 0.5547

Loss: 0.6647, 0.4774, 0.0838, 0.0695, 0.0289, 0.0175, 0.0125, 0.0127, 0.0073, 0.0343, 0.0103, 6.9016e-05, 0.0123, 2.0424e-04, 0.0129 (×2), 0.1027, 0.0376, 0.0015

Val-Accuracy: 0.8125 (×3), 0.7812 (×9), 0.7500 (×6), 0.7188, 0.8438 (×4)

Val-Loss: 0.8933, 1.2010, 5.7849, 6.0613, 7.0139, 10.6828, 8.2998, 8.1908, 4.5915, 6.9065, 12.6269, 8.4280, 9.4978, 7.4797, 8.2694, 6.9891, 7.4927 (×2), 12.5460

Notes: Ok now I did that and got my results, updated my dataset to the next one, got a low score of 0.6351 to start but model hung around 0.8000 - 0.9000, so not much of a better, as data switch might have changed CNN's outcome.

Day 4: Observations

C = ...
F = flatten D = dense

layer (type)	output shape	Param
C	N, 148, 148, 32	896
M	N, 74, 74, 32	0
C-1	N, 72, 72, 32	18 496
M-1	N, 36, 36, 64	0
C-2	N, 34, 34, 128	73 856
M-2	N, 17, 17, 128	0
F	N, 36992	0
D	N, 128	4 735 104
D-1	N, 1	129

TP = 4 828 481

Tr P = 4 828 481

N-TP = 0

LR ≈ 0.0010000000474974513
≈ 0.001

Epochs = 20, Batch size = 32

Accuracy: 0.5729, 0.9688, 0.9793, 1.0000, 0.9824, 0.9688, 0.9812, 0.9375, 0.9815, 0.9688, 0.9791, 1.0, 0.9937, 1.0, 0.9833, 0.4687, 0.5917, 0.6688, 0.6738, 0.544

Loss: 0.7762, 0.1472, 0.0942, 0.0104, 0.0444, 2.9284e-04, 0.0562, 0.2623, 0.0845, 0.3892, 0.0600, 0.0544, 0.0572, 0.0083, 0.0230, 0.0336, 0.0527, 0.0165, 0.0158, 0.0329

Val-Accuracy: 0.8125, 0.9375, 0.7812, 0.8438, 0.7500, 0.9062, 0.7500, 0.8438, 0.7812, 0.8438, 0.8125 (x2), 0.7500, 0.7188, 0.8125, 0.7500 (x2), 0.7812 (x2), 0.7188

Val-Loss: 2.5952, 0.6397, 7.5135, 3.4479, 10.0655, 4.2414, 9.3846, 6.1287, 11.8976, 6.2950, 5.5708, 5.0023, 7.9780, 9.6719, 7.0406, 7.6519, 14.8102, 8.6712, 15.9307, 11.6931

Notes: I implemented the learning rate displayer, showing the model's rate of convergence, and since my learning rate = 0.001, it allows for steady convergence toward the minimum of the lost function, stopping overshooting. I'm using Adam, so this value is its default. Also, I always would see error message = Error during prediction: The layer sequential has never been called and thus has no defined input - and I think it is because I never added the preprocess image function to my lung_cancer_detection.py script, even though I did to my streamlit_app.py, so it could be causing the error?

*I spend less time on the training and more about trying to fix the errors of code
↳ Tried testing, but after a while, caused errors about input shape; immediately stopped the testing; prolonge testing till Day 7
↳ Tried MobilenetV3 but caused the accuracy to plummet again... (5 last epochs)

Day 5: Observations

★ 5 epochs but 20 in total

5, 5, 5, 5

None = N

Layer (type)	Output Shape	Param #
C	N, 148, 148, 32	896
m	N, 74, 74, 32	0
C-1	N, 72, 72, 32	18 496
m-1	N, 72, 72, 32	0
C-2	N, 34, 34, 128	73 856
m-2	N, 17, 17, 128	0
F	N, 36992	0
D	N, 128	4 735 104
D-1	N, 1	129

← Why is it the same? Its identical? What can I do to change my params?

Accuracy: 0.5599, 0.5616, 0.7500 (x2), 0.9522, 0.9375 (x2), 0.9178, 0.9835, 0.9688, 0.9873, 1.0000, 0.7601, 0.9062, 0.9748, 1.0000, 1.0000, 0.9957, 1.0000 (x2)

Loss: 1.0510, 1.0392, 0.4745 (x2), 0.2625, 0.3046, 0.095, 0.2185, 0.0513, 0.1753, 0.0390, 0.1797, 0.0728, 0.0143, 0.0919, 0.0443, 0.0467, 0.0207, 0.0297, 12.4574

Val-Accuracy: 0.2188, 0.4375, 0.6875, 0.8438 (x7), 0.7812, 0.8125 (x4), 0.7812, 0.8438, 0.7812, 0.8438

Val-Loss: 0.7585, 0.7638, 4.1309, 1.5703, 5.1304, 6.4283, 8.3524, 10.6428, 14.4882, 12.7373, 1.9509, 4.1532, 5.5797, 6.0472, 4.9655, 4.9677, 3.7146

Notes: I think the results are lower than before because I didn't do my epochs all together (I did 5 epochs, and following those 5, I did three more 5 instead of doing 20 epochs all together). *couldn't do altogether because I've run out the streamlit. Also because I changed the dataset on this day to a newer one that the model did not initially recognize. *written hour later But wouldn't the model have made new connections and changed params? Maybe tomorrow I need to implement another strategy to improve model performance so that my desired result will come to pass (brainstorm on page after day 7). Storage is full, or almost full for LFS, tried Drive before, didn't work, now know it's because I did not have any access to the Cloud (Model and data weren't accessible; no free access). Maybe I can ask my mother to install more LFS but was hoping I could have found a free option instead.
Removed LR displayer to leave room for other parts of control panel *seeing if mobile still works today too

Day 6: Observations

TP = 4 377 764

Tr-P = ~~WON'T SHOW~~ 528 193 Nvm I found it

N-TP = 4 049 571

★I also noticed training ... a lot longer now (more than 42s → 157s/step) → 57,645,28...

input_layer_1	None, 224,224,3	0
efficientnetb0	None, 7,7,1280	4 049 571
global_average_pooling	None, 1280	0
dense	None, 256	327 936
dropout	None, 256	0
dense_1	None, 1	257

(x1)

Accuracy: 0.7291, 0.7295, 0.7500, 0.7574, 0.6562, 0.7630, 0.8438, 0.7500, 0.8438, 0.7984, 0.7812, 0.8125, 0.8280, 0.8150, 1.0000, 0.4834, 0.5319, 0.5772

Loss: 0.8382, 0.1462, 0.7258, 0.7230, 0.6338, 0.6850, 0.1044, 0.2487, 0.7826, 0.3289, 0.9133, 0.2286, 0.1409, 0.8897, 0.2595, 0.3112, 0.2003

Val-Accuracy: 0.7246, 0.7293, 0.7650, 0.8438, 0.7824, 0.7545, 0.8668, 0.8349, 0.8695, 0.8654, 0.8766, 0.9845, 0.4975, 0.7625, 0.8838, 0.6753, 0.5765, 0.7675, 0.815, 0.6277

Val-Loss: 0.2905 (x3), 0.3154, 0.3075, 0.3730, 0.3335, 0.3755 (x1), 0.3673, 0.3147, 0.7128 (x4), 0.1275, 0.7895 (x2), 0.1923

→ I added (appl) emojis to make it look more fun; noticed streamlit had emojis during configuration

Notes: Turns out the repository didn't update all the images. Now that I have updated every image in the dataset from the original to contain images from Dataset 3, the website is taking time to load on my computer and its accuracy is lower than what is was on Day 1. Plan is to finally do the test section on Day 7 because my own (test) is when I just try out the app and when my friends do it. Or maybe it is going slower because I'm listening to YT on my browser + its memory usage is 403 MB... probably not right lol. I also added the early stopping instead of manually stopping the training myself. It is slower because of my limited space on Git LFS too. Never always been able to see the actual image, so changing it from 150×150 to 274 × 224 (x3), mirroring those of predeveloped CNNs from tensorflow. ★ Later in day = Purchased 50 GB of Git LFS and I am using EfficientNetB0 as a form of transfer learning to reach my goal.

★ Did I forget what I did yesterday lol

★ Because of streamlit; needs to be a light-weight model

I added a st.button to show the model layer names because errors keep occuring about input shape and attribute error (button acts as debugging tool) ★ Used Reinvent for first half of training (0)

Day 7: Observations

* LAST DAY + Testing *

TP = 4 377 764

Tr-P = 528 193

N-TP = 4 649 571

OP = 0

★ not sure if I still want it want data to weight more

★ Still takes some time to load

input_layer_1	None, 224,224,3	0
efficientnetb0	None, 7,7,1280	4 049 571
global_average_pooling	None, 1280	0
dense	None, 256	327 936
dropout	None, 256	0
dense_1	None, 1	257

Accuracy: 0.8239, 0.8859, 0.9088, 0.9065, 0.9026, 0.9057, 1.0000, 0.9728, 0.8954, 0.9132, 0.9426, 0.9625, 0.8995, 0.8885, 0.9455, 0.9977 (x3)

Loss: 0.6620, 0.2970, 0.2200, 0.6097, 0.3689, 0.3631, 0.2400 (x2), 0.7041, 0.6959, 0.6938, 0.6994, 0.7041 (x2), 0.7052, 2.4095

Val-Accuracy: 0.8962, 0.8808, 0.8583, 0.9077, 0.9945, 0.9029, 0.8954, 0.9554, 0.9785, 0.9375, 0.9125

Val-Loss: 0.3128, 0.2479, 0.4165 (x3), 0.9549, 0.1275, 1.0259 (x4), 0.1279 (x3), 1.1027, 0.2284 (x2), 0.5075, 0.2364

Test-Accuracy: 0.9034! (x4), 0.9839 (x5), 0.9436 (x27), 0.9825 (x2), 0.9275 (x3), 0.9340 (x2), 1.0000

Test-Loss: 0.2350 (x4), 0.6936 (x5), 0.2954, 0.9823, 0.1026, 0.2843 (x3), 1.7289, 0.1299, 0.5642 (x2), 0.7726

Out of 10

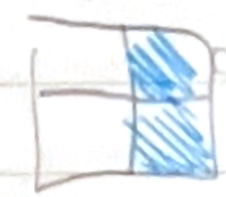
Overall Result: Hypothesis was correct; CNN should promising results, it did good

F1 Score: 61.27%, 81.33%, 81.19%, 78.26%, 76.55%, 76.47%, 80.38%, 81.69%, 82.42%, 84.54%

Out of 10

Precision: 81.72%, 80.39%, 74.77% (x2), 75.73%, 79.41%, 82.86% (x2), 80.73%, 79.25%

Accuracy: 89.23%, 85.44%, 86.14%, 84.92%, 74.27% (x2), 82.05%, 79.45% (x2)



Recall: 1.0000 (x4), 80.95%, 82%; (x1) 78.26%,

Databricks log-in Account

Email: Williams.Udofia@havates.ca

P/word: 2K18@hava20214good

Service account details (email) = my-app-account@modelanddata.iam.gserviceaccount.com

Ways to improve score (Google Search):

1. Higher value of weights to cancerous images (cancerous images are the exception) → class_labels = [0, 1] # cancer has stronger activation (1), than no-cancer (0), or should I do other way around?

2. # of Layers... Maybe a random generator hyperparameter tuner (but I think it will take too long maybe) → could also end up lowering performance

3. Cross Validation where I further divide my data to x and y (symbolizing features and labels from HOG) then split dataset into k subsets (no clue what that is). Each fold is trained on the model followed by an additional validation metric

4. HOG (Histogram of Oriented Gradients), which uses edges and shapes to capture distinguishing features of C-scans and NC-scans, helpful to improve performance of CNN when performance is low

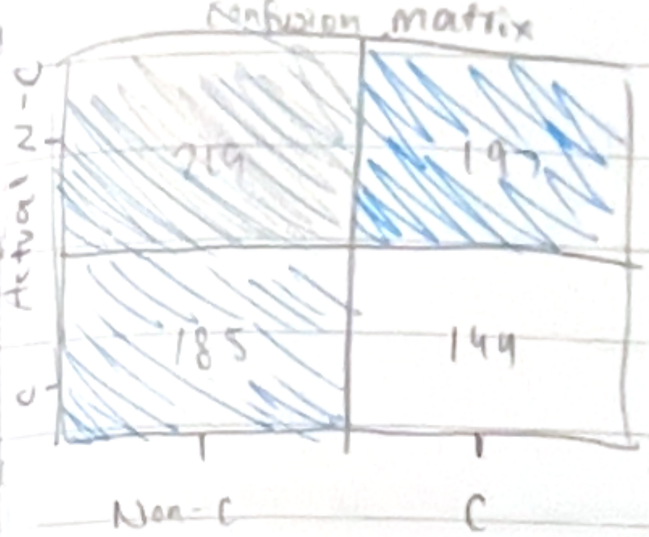
5. Transfer Learning, where I find a CNN structure to implement so as to train the CNN on how to see lung cancer images in particular, as it already has had practice on other types of images → Ex = DenseNet (121, 169, 201, etc.), VGG16, ResNet, ResNet, etc.

↳ I'll do DenseNet; saw Grad CAM implementation already + works good on datasets with not too many images; faster and intuitive

↳ DenseNet is too heavy and took too long to load, sprouted many errors, and laced my app with problems. Now I switched to EfficientNetB0 and it is significantly more lightweight; allows for a smoother run time on Streamlit (load?) and still has a high accuracy (F1 score) when implemented into code, already 61.27% from the get-go

Trying out test feature

Currently, when testing: (Day 4)



144 True C (P)

219 True NC (P)

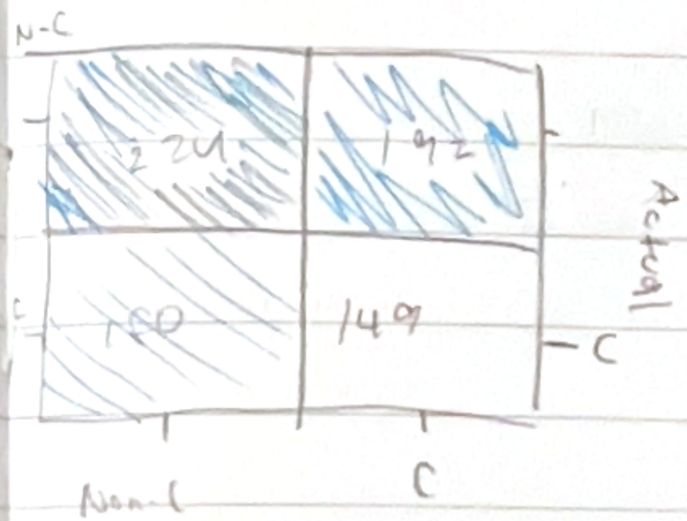
185 wrong NC (N)

197 wrong C (N)

→ Possibly points towards a bias the model has towards C-CT scans

Accuracy (overall) = $\frac{144+219}{144+219+197+185} = \frac{363}{745}$ ≈ 0.48724 ≈ 0.49 or 49%

Precision = $\frac{144}{144+197}$ (34%) = 0.421 ≈ 42.1%...



149 True C

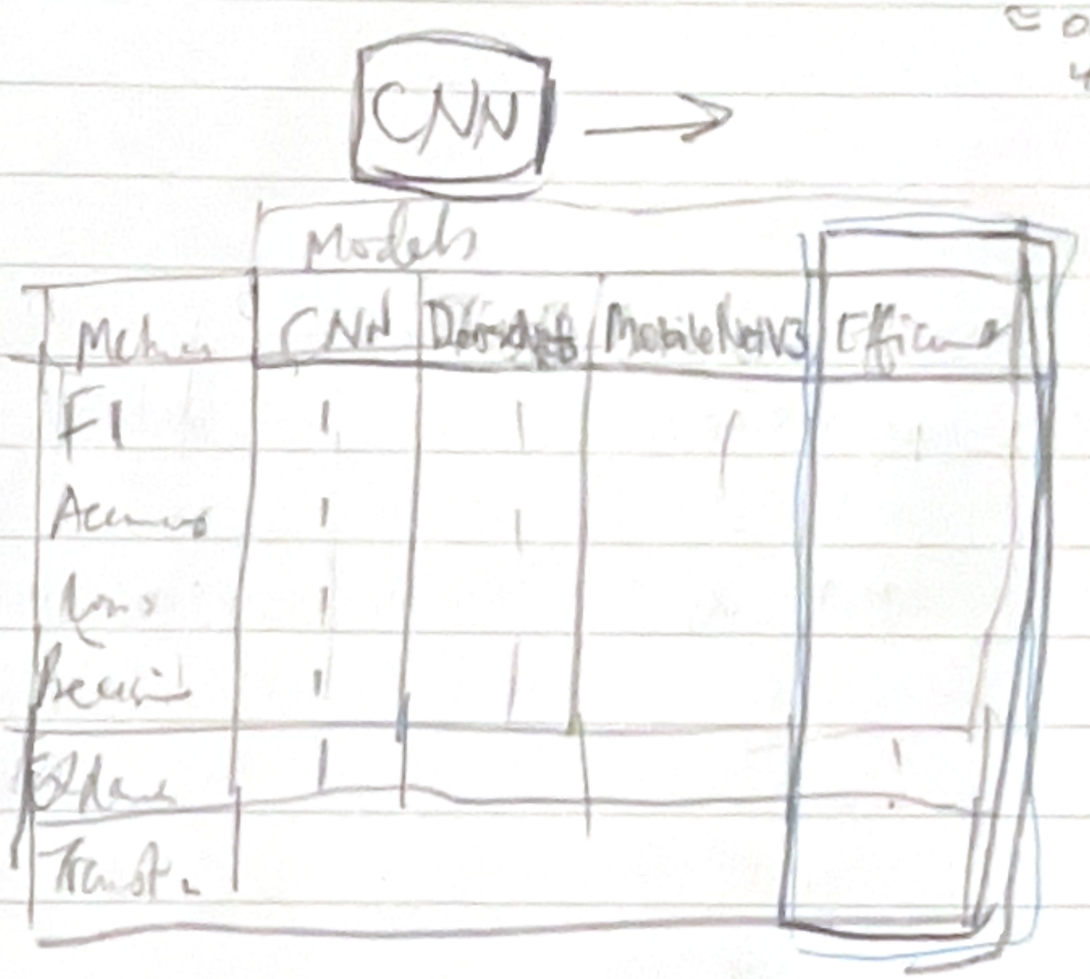
224 True NC

192 wrong C

180 Wrong NC

$\frac{149+224}{745} = \frac{373}{745} \approx 0.5006$ Accuracy ↑ or 50%

precision = $\frac{149}{149+192} = 0.4368$ ≈ 0.44 or 44%



CNN Names

Words of names m

ONCOSCAN (alr exists)

PULMOSCAN (alr exists)

PULMOCNN

OncoVision

Pulmo AI

ONCOCNN

PULMO AI (alr exists!)

Endings	First
1. SCAN ×	1. ONCO
2. NET ✓	2. PULMO
3. AI	3. LUNG
4. VISION ×	4. RESPIRA
5. CNN ×	5. BRONCHI
	6. TRACHEA
	7. BREATH
	8. LARYNGO-
	9. PNEUM-

PULMO Net × alr exists.

It looks like latin on the site help

Pulmo CNN? RESPIRASCAN

OncoLung NET RespiraScan

ThoraxSense ONCO AI ✓

LUMASCAN?

Analysis

- Custom CNN model, ONCO AI, attains a accuracy score of 80%, displaying the amazing abilities of CNNs and their ability to learn at an incredible rate, proving my proposed hypothesis to be correct. 8/10
tries, model reached and surpassed the 80% threshold
- I employed (1) Normal, 3 layer, then 4 layer CNN structure, then (2) the VGG-16, but then I quickly replaced it with the DenseNet 121 because I searched up online which structure is more compatible with CT scans, or better at accessing CT scans, and all pointed in favor to Densenet