Machine Learning for Automated Ultrasound Guided Infant Lumbar Puncture

Contents

S.No	Topic	Page No.
1.	Background	2
2.	About the Data Exploratory Data Analysis	2-3
3.	Data Augmentation	4-5
4.	Image Pre-Processing	6-7
5.	Data Modeling and Training	8-10
7.	Evaluation/Testing, Results and Inferences	11-15
8.	Future Scope and Plan for Object Localization	16
9.	Project Outcomes	17

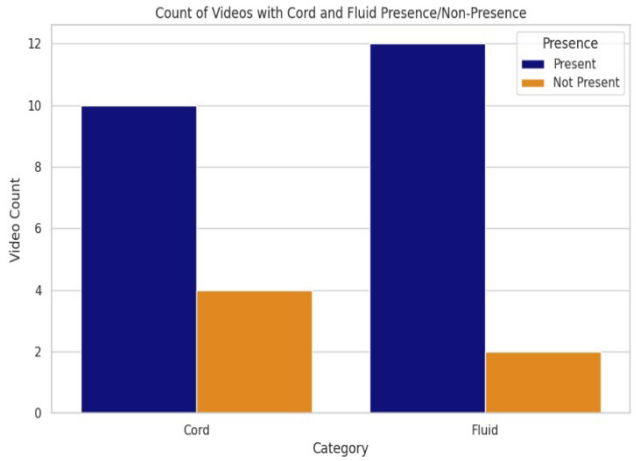
Background

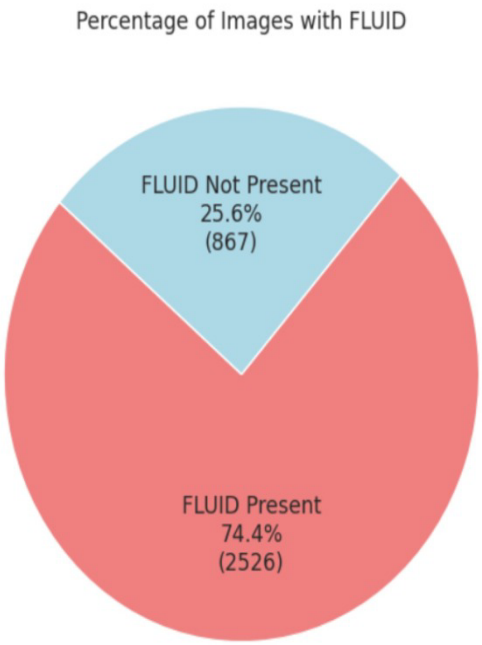
Lumbar puncture (LP) is a critical diagnostic test in febrile infants that is associated with high failure rates, especially among novice providers. Ultrasound (US) guidance can improve LP success but is underutilized due to lack of provider comfort and expertise in this skill. Automated identification of optimal interspaces on US has the potential to improve procedural success. Our aim was to develop an artificial intelligence (AI) algorithm using a database of ultrasound spinal anatomy videos to identify key anatomic structures and aid in infant LP performance.

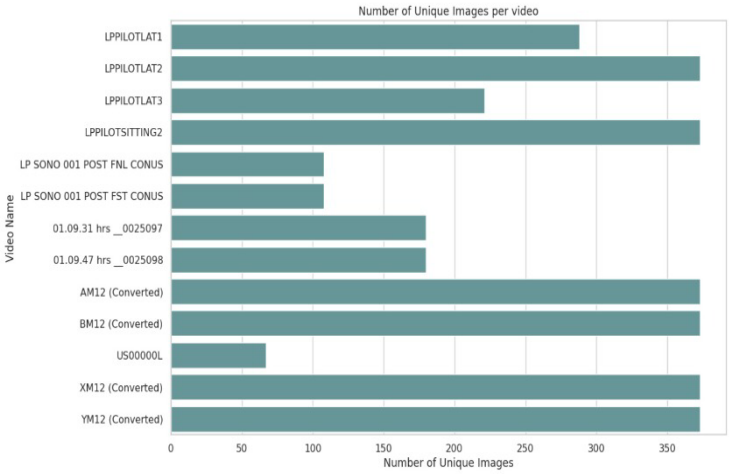
About the Data and Exploratory Data Analysis

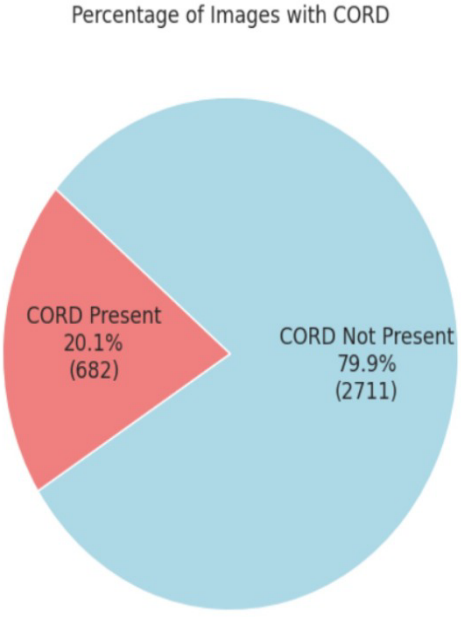
The data comprised of 13 real ultrasound spinal anatomy videos of patients. We facilitated expert annotation each video was broken down to frames. Each frame corresponds to an image. To put things into perspective, if the frame rate of a video is 30 frames per second and the video is 10 seconds, then we would have 300 images (30*10).

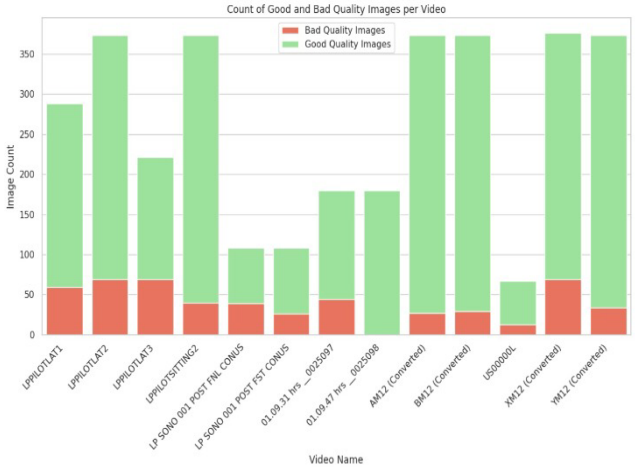
After segmenting all the videos into the respective frames, we ended up with a total of 3390 frames. These frames were then given to the expert (Doctor) to label. Each frame consisted of 3 labels, “Bad quality”, “Spinal Cord”, and “Spinal Fluid”. If any of these labels had the value of 1, that means the particular feature is present and 0 meant absent. In case of bad quality, if it is 1, that means the image cannot be used to make out whether the features spinal cord or fluid are present. Fundamentally, we treated this as a multilabel classification problem. Before we dive into any modeling aspect, it was crucial to understand the class distribution and address any imbalance class issues. So, for that we did exploratory analysis. Below are some results of the exploratory analysis.


Count of Videos with Cord and Fluid Presence/Non-Presence
Presence
Present
Not Present
Video Count
Cord
Fluid
Category


Percentage of Images with FLUID
FLUID Not Present
25.6%
(867)
FLUID Present
74.4%
(2526)


Number of Unique Images per video
LPPILOTLAT1
LPPILOTLAT2
LPPILOTLAT3
LPPILOTSITTING2
LP SONO 001 POST FNL CONUS
LP SONO 001 POST FST CONUS
01.09.31 hrs __0025097
01.09.47 hrs __0025098
AM12 (Converted)
BM12 (Converted)
US00000L
XM12 (Converted)
YM12 (Converted)
Video Name
Number of Unique Images

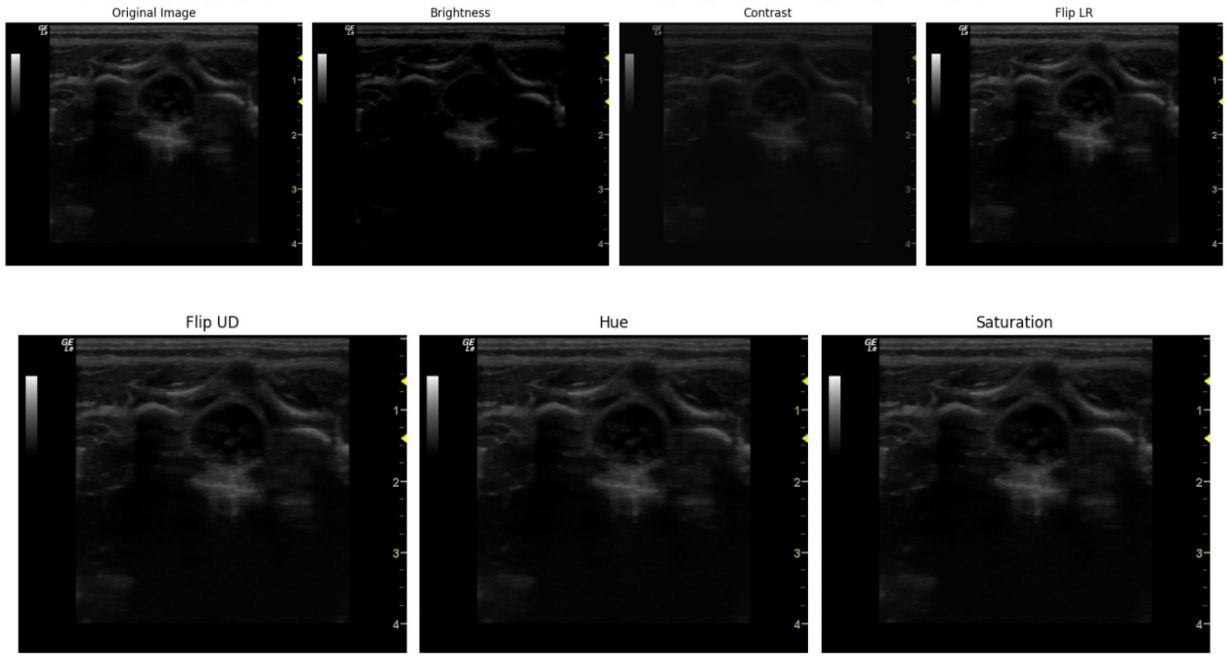

Percentage of Images with CORD
CORD Present
20.1%
(682)
CORD Not Present
79.9%
(2711)


Count of Good and Bad Quality Images per Video
Bad Quality Images
Good Quality Images
Image Count
LPPILOTLAT1
LPPILOTLAT2
LPPILOTLAT3
LPPILOTSITTING2
LP SONO 001 POST FNL CONUS
LP SONO 001 POST FST CONUS
01.09.31 hrs __0025097
01.09.47 hrs __0025098
AM12 (Converted)
BM12 (Converted)
US00000L
XM12 (Converted)
YM12 (Converted)
Video Name

Data Augmentation

From the exploratory analysis in the previous section, we noticed a certain degree of imbalance in our data. Besides that, we realized that the number of images in total and the number of images in each class would probably be insufficient for a deep learning architecture to generalize. To cater to this, we augmented the data. Here are the augmentation techniques we carried out using OpenCV and PyTorch.

1. *Random Brightness Adjustment:*
   This function adjusts the brightness of the image by multiplying all pixel values by a brightness_factor, which is randomly chosen between 0.5 (darker) and 2.0 (brighter). cv2.convertScaleAbs is used to scale the image and convert it to 8-bit per channel. It also ensures that the pixel values remain within the valid range by applying an absolute value.
2. *Random Contrast Adjustment:*
   Similar to brightness adjustment, this function alters the contrast of the image. Contrast is the difference in luminance or color that makes an object distinguishable. A contrast_factor is randomly selected in the same range as brightness_factor, and it scales the pixel values, which changes the contrast of the image.
3. *Random Left-Right Flip:*
   This function flips the image horizontally (left-right) with a 50% chance. Flipping does not affect the pixel intensities but changes the orientation of the image, simulating a mirrored view.
4. *Random Up-Down Flip:*
   This function similarly flips the image vertically (up-down) with a 50% chance. This simulates cases where the camera may be inverted, or the object of interest appears upside down.
5. *Random Hue Adjustment:*
   Hue represents the color aspect of an image. This function first converts the image to the HSV (Hue, Saturation, Value) color space.It then adjusts the hue by adding a small random value, allowing for slight changes in color that can mimic different lighting conditions or variations in object colors.
6. *Random JPEG Quality Adjustment:*
   JPEG quality adjustment simulates the effect of different compression levels on an image. High compression can result in loss of detail and artifacts. The function randomly selects a JPEG quality factor between 50 (higher compression, lower quality) and 100 (lower compression, higher quality), encodes the image to JPEG format with this quality setting, and then decodes it back to simulate the effect of JPEG compression.
7. *Random Saturation Adjustment:*
   Saturation refers to the intensity of colors. This function adjusts the saturation by scaling the Saturation channel in the HSV color space. A saturation_factor is chosen randomly and applied, allowing the colors to become more vivid or more muted, which helps the model learn from images with various color intensities.



We identified and used the above 7 methods to the existing images to come up with modified images. We then augmented the data with these new images. So, our dataset was basically just scaled by a factor of 8. To give you an idea of how the number of frames in a video changed after augmentation, below is the before and after statistics of each video. As you can see the data has just been scaled by a factor of 8 so the visualizations, we obtained during the initial EDA are also relatively the same. Only the values have been scaled by a factor of 8.

Before

```
Subfolder 'LPPILOTSITTING2' contains 373 image(s).
Subfolder 'LPPILOTLAT2' contains 373 image(s).
Subfolder 'US00000L' contains 67 image(s).
Subfolder 'LPPILOTLAT1' contains 288 image(s).
Subfolder 'LPPILOTLAT3' contains 221 image(s).
Subfolder 'LPSONO001POSTFNLCONUS' contains 108 image(s).
Subfolder 'LPSONO001POSTFSTCONUS' contains 108 image(s).
Subfolder '01.09.3hrs-0025097' contains 180 image(s).
Subfolder '01.09.47hrs-0025098' contains 180 image(s).
Subfolder 'AM12' contains 373 image(s).
Subfolder 'BM12' contains 373 image(s).
Subfolder 'XM12' contains 371 image(s).
Subfolder 'YM12' contains 373 image(s).
```

After

```
Subfolder 'LPPILOTSITTING2' contains 2984 image(s).
Subfolder 'LPPILOTLAT2' contains 2984 image(s).
Subfolder 'US00000L' contains 536 image(s).
Subfolder 'LPPILOTLAT1' contains 2304 image(s).
Subfolder 'LPPILOTLAT3' contains 1768 image(s).
Subfolder 'LPSONO001POSTFNLCONUS' contains 864 image(s).
Subfolder 'LPSONO001POSTFSTCONUS' contains 864 image(s).
Subfolder '01.09.3hrs-0025097' contains 1440 image(s).
Subfolder '01.09.47hrs-0025098' contains 1440 image(s).
Subfolder 'AM12' contains 2984 image(s).
Subfolder 'BM12' contains 2984 image(s).
Subfolder 'XM12' contains 2968 image(s).
Subfolder 'YM12' contains 2984 image(s).
```

Image preprocessing

We performed image processing for several important reasons:

Image Enhancement: Ultrasound images can be noisy and have poor contrast, making it challenging to visualize anatomical structures accurately. Image processing techniques can enhance the quality of these images by reducing noise, improving contrast, and sharpening edges, making it easier for healthcare professionals to interpret the images.

Noise Reduction: Ultrasound images can suffer from various types of noise, such as speckle noise, which can degrade image quality and make it difficult to identify structures. Image processing can employ filtering techniques to reduce noise and create smoother images.

Feature Extraction: Image processing can extract important features or measurements from ultrasound images. This is crucial for quantifying and analyzing anatomical structures, such as measuring the size of organs, blood flow velocity, or identifying specific lesions or abnormalities.

Real-time Image Improvement: During live ultrasound imaging, healthcare professionals often need real-time feedback to adjust the probe's position or settings. Image processing can provide continuous improvements to the image quality, aiding in the accuracy of medical procedures and diagnostics.

Image Visualization: Image processing can provide different visualization techniques to highlight specific structures or tissue properties. For example, color mapping can be used to display blood flow direction and velocity, while grayscale inversion may help emphasize certain features.

Overall, image processing in ultrasound videos plays a critical role in improving image quality, aiding in diagnosis, and enhancing the utility of this non-invasive medical imaging technique in various clinical applications. In our case we also hoped it would aid our deep learning models in identifying the classes better.

After experimenting with a lot of image processing techniques, we identified the following 2 techniques to be useful. All the preprocessing was carried out using OpenCV in Python.

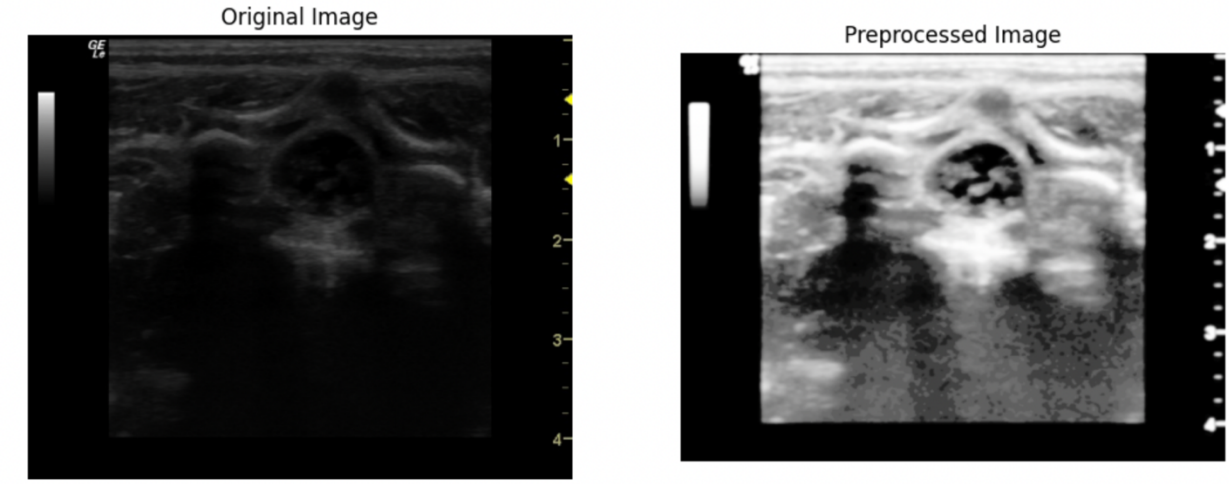
Histogram Equalization

Histogram equalization is a method for adjusting image intensities to enhance contrast. The histogram of an image shows the distribution of pixel intensities (from dark to light). In histogram equalization, the idea is to spread out the most frequent intensity values, which typically improves the visibility of features in the image.

The process involves the following steps:

1. Calculate the histogram of the image pixel intensities.
2. Compute a cumulative distribution function (CDF) from the histogram.
3. Use the CDF to map the pixel intensities of the original image to new values that produce a more uniform histogram.

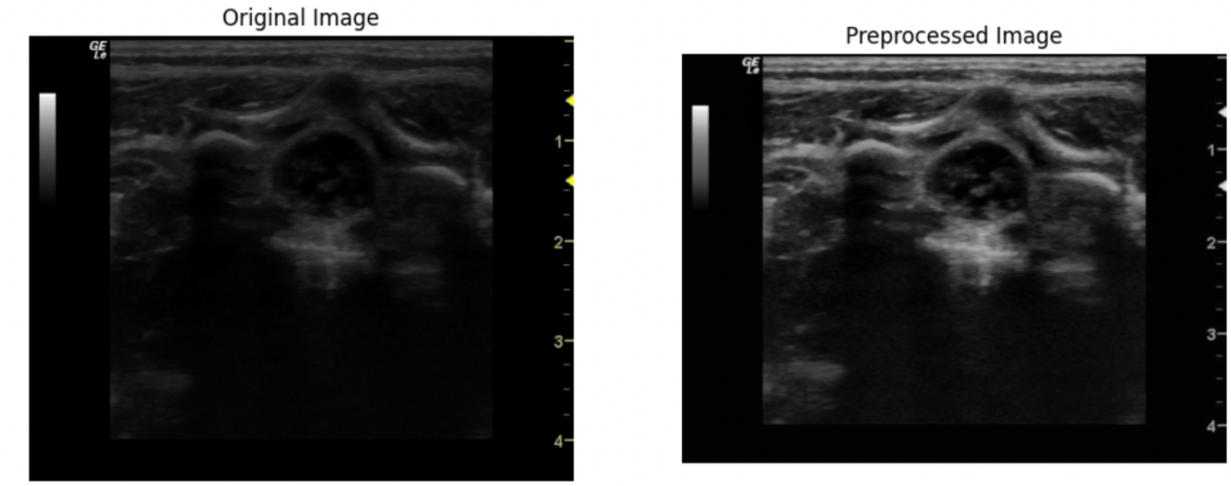
This stretching of the histogram results in an image with enhanced contrast, often making features that were indistinct in the original image more visible. Below is an example showing the effect of histogram equalization.



Contrast Enhancement

Contrast enhancement refers to any technique that increases the contrast between the features in an image. The goal is to make the distinctions between different parts of the image more pronounced. It can involve:

1. Linear Contrast Stretching: This technique rescales the pixel values of the image so that they spread across a desired range of values, which enhances the overall contrast.
2. Adaptive Methods: Methods that adjust the contrast based on local pixel statistics, such as adaptive histogram equalization (AHE) or its variant, contrast-limited adaptive histogram equalization (CLAHE), which prevents over-amplification of noise that regular histogram equalization might cause.
3. Non-linear Mapping: This approach might apply a non-linear function to the image pixels to increase the contrast. Common functions include logarithmic transformations and power-law (gamma) transformations. Below is an example showing the effect of histogram equalization.



Data Modeling and Training

Now that our data is prepared, we need decided to implement with machine learning and deep learning models. In machine learning, we used **K-nearest neighbors** and **Decision trees**. Even after doing feature extraction, these models were not able to go beyond 65 percent in accuracy so we decided to drop these right off the bat. As next step, we proceeded with some state-of-the-art Deep Learning architectures like AlexNet, VGG16, ResNet18, ResNet34 and DenseNet. Based on our literature survey, these architectures have been known to perform pretty well for vision-based tasks. However, they require tons of data, something that we don't have in our case. Despite of that, we came up with augmentation techniques, transfer learning and impressive hyperparameter tuning optimization techniques to circumvent all these issues. Before we dive into that below is a short description of each of the deep learning architectures used for our task.

AlexNet
AlexNet was one of the pioneering convolutional neural networks (CNNs) that significantly advanced image classification. It consists of five convolutional layers followed by three fully connected layers. Introduced techniques like ReLU activation, dropout, and data augmentation, leading to improved performance on ImageNet, a large image dataset.

VGG16
VGG16 is part of the VGG family of CNN architectures. It is characterized by its simplicity and uniform architecture, consisting of 16 weight layers (13 convolutional and 3 fully connected).
VGG16 achieved strong results on ImageNet and has been widely used as a base model for transfer learning.

ResNet18, **ResNet34**
ResNet, short for Residual Network, introduced the concept of residual blocks. Residual blocks contain skip connections, allowing gradients to flow more easily during training and preventing the vanishing gradient problem. ResNet18 and ResNet34 have 18 and 34 layers, respectively, and have been widely adopted for various computer vision tasks due to their depth and effectiveness.

DenseNet
DenseNet, or Densely Connected Convolutional Networks, proposed a unique architecture where each layer receives input from all previous layers. This dense connectivity promotes feature reuse and gradient flow, resulting in efficient and accurate models.
DenseNet has different versions, such as DenseNet-121 and DenseNet-169, which vary in terms of depth and complexity.

Each of these architectures has made significant contributions to the field of deep learning, particularly in image classification and computer vision tasks. Researchers and practitioners often choose these models based on their specific requirements and the complexity of the task at hand.
Initially we were training the above architectures from scratch but due to lack of data they gave accuracies around 75 percent and f1-scores around 0.71. After augmentation these number did boost further but we were not convinced. After extensive research we decided to use **transfer learning** which is another impressive technique to deal with small data.

Transfer learning is a machine learning technique where a model trained on one task is adapted or fine-tuned for a different but related task. Instead of training a model from scratch, transfer learning leverages the knowledge and features learned from a pre-trained model, which has typically been trained on a large and diverse dataset. So, we used architectures that were already pretrained on ImageNet dataset, so these architectures already have a good understanding of different types of images.

Other than that, we also setup a framework to implement RNN and transformer models but the complexity of these models surpassed the complexity of our task, so they were not useful. Besides the ResNet18 architecture already by far surpassed the expectations of the medical experts (Discussed more in the evaluations).

Specifications of Training / Hyperparameters

We used 70 percent of the data in training, 10 percent for validation and 20 percent of the data for testing. In terms of frames, we had 2373 images in training, 339 in validation and 678 in testing. To maintain the ratio of classes and address imbalance we did a stratified train-valid-test split. It must be noted that the supposedly optimum train-valid-test split was achieved after a lot of experimentation. The testing dataset was kept common across all the models for unbiased evaluation.

Learning rate:

In deep learning, the learning rate is a hyperparameter that determines the size or step size of adjustments made to the model's weights during training. It plays a crucial role in controlling the convergence and stability of the training process. To find the appropriate learning rate we performed random search combined with grid search. This was done in combination with the number of epochs. Eventually we ended up with a learning rate of **0.01**.

Epochs:

The number of epochs is a hyperparameter that you specify before training a model. It determines how many times the model will iterate through the entire training dataset. Setting the number of epochs is important because it affects how long the training process will run. If you choose too few epochs, the model may not have enough opportunities to learn from the data, and the training process might terminate prematurely, resulting in underfitting (poor performance).If you choose too many epochs, the model may overfit the training data, meaning it will learn to perform exceptionally well on the training data but poorly on new, unseen data. After experimentation we realized that 20 epochs would be optimum for the model to converge on.

Loss Function:

The purpose of a loss function is to provide a measure of the error or "loss" between the predicted values and the actual values. The goal during training is to minimize this loss function, as a lower loss indicates better model performance. We used the **binary cross entropy loss** because we have a binary classification task, where each data sample belongs to one of two classes (e.g., positive or negative, 1 or 0)

Optimizer:

Optimizers are a crucial component of the training process in machine learning and deep learning,

as they determine how the model's parameters are updated during each iteration of training. Different optimizers employ various techniques and update rules to iteratively refine the model's parameters. We used the **Adam** (short for Adaptive Moment Estimation) optimizer as it was the most suitable for our task.

All the experimentations with the above hyperparameters were carried out on the validation dataset instead of the test dataset to avoid any data leakage. We initially set up the google cloud platform and used a VM on there to train the models. However due to resource constraints we eventually moved to colab pro where we used the **V100 GPU's**.

Evaluation/Testing, Results and Inferences

For quantitative evaluation, we primarily used accuracy, precision, recall and f1-score. This was supplemented with qualitative evaluations where we displayed all the misclassified images and understand from the experts whether the model is making errors in places where humans would do as well. So first we look at the scores of each of the models and then zone into the best model to see if it is as per with the expert standards.
The following evaluation metrics were used:

Accuracy: Accuracy measures the overall correctness of a classification model and is defined as the ratio of correctly predicted instances to the total instances.
Formula: (TP + TN) / (TP + TN + FP + FN)

Precision: Precision quantifies the ability of a model to make accurate positive predictions and is calculated as the ratio of true positives to the total predicted positives.
Formula: TP / (TP + FP)

Recall: Recall, also known as sensitivity or true positive rate, measures the model's ability to identify all relevant instances and is calculated as the ratio of true positives to the total actual positives.
Formula: TP / (TP + FN)

F1 Score: The F1 score is the harmonic mean of precision and recall and provides a balanced measure of a model's performance in binary classification tasks.
Formula: 2 * (Precision * Recall) / (Precision + Recall)

Metrics	**AlexNet**	**VGG16**	**ResNet18**	**ResNet34**	**DenseNet121**
Accuracy	0.9784	0.9798	0.9882	0.9843	0.9897
Precision	0.9775	0.9720	0.9852	0.9836	0.9863
Recall	0.9647	0.9733	0.9825	0.9730	0.9849
F1-score	0.9710	0.9726	0.9838	0.9783	0.9856
TN	1253	1264	1280	1282	1294
FP	17	21	11	12	10
FN	27	20	13	20	11
TP	737	729	730	720	719
FP (Cord)	2	8	0	1	2
FN (Cord)	14	1	3	5	2
FP (Fluid)	5	13	0	6	7
FN (Fluid)	9	2	10	11	4
FP (Quality)	10	0	11	5	1
FN (Quality)	4	17	0	4	5
Misclassified images No.	26	23	11	23	12

Results table

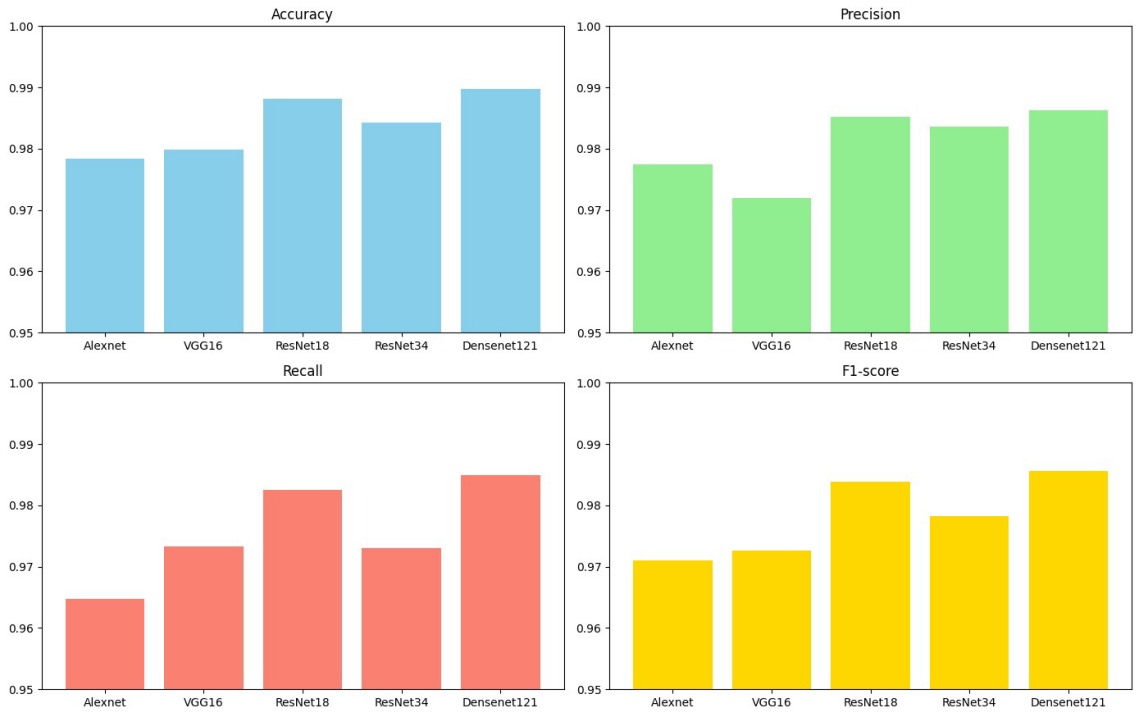
FP=False positives
FN = False Negatives
TN = True Negatives
TP = True Positives
Quality = Bad Quality
Recall = Sensitivity
FN (cord) and FP (Fluid) are highlighted because they are important to the Doctor (expert)

Inferences

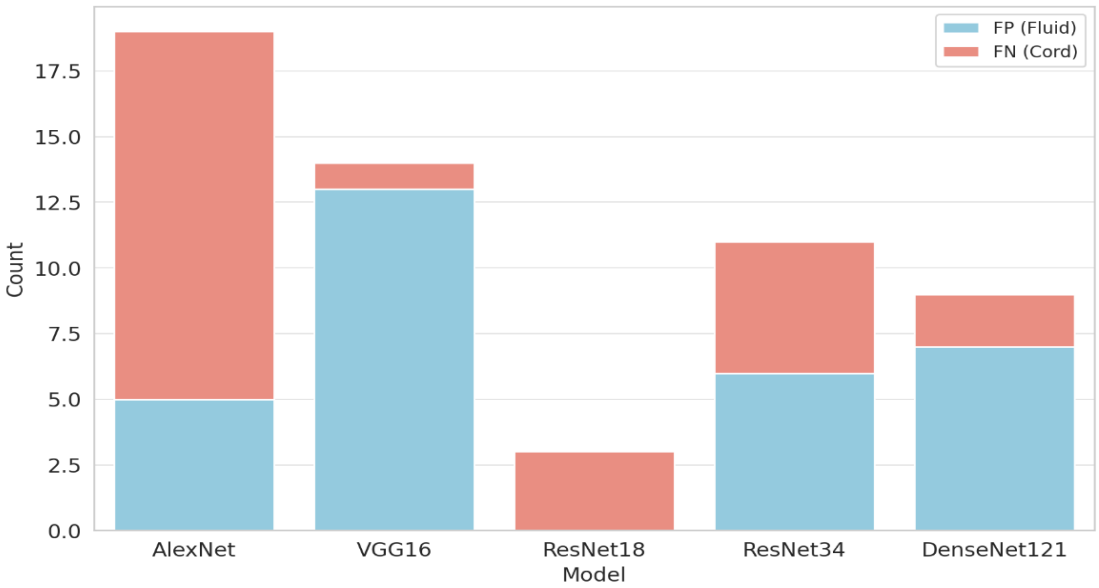
The comprehensive table above provides a comparison of different deep learning models across various metrics commonly used in classification tasks. Let's go through each metric and the corresponding results for the models:

Accuracy: DenseNet121 has the highest accuracy at 0.9897, indicating it correctly predicted the most instances out of all the models.
Precision: Here, DenseNet121 leads again with a precision of 0.9863, implying it has a lower rate of false positives.
Recall (Sensitivity): DenseNet121 has the highest recall at 0.9849, meaning it is best at capturing the actual positive cases.
F1-score: DenseNet121 has the highest F1-score at 0.9856, indicating a balanced trade-off between precision and recall.
TN (True Negative): ResNet34 has the most, with 1282.
FP (False Positive): DenseNet121 has the fewest at 10.
FN (False Negative): The number of incorrect predictions where an instance is negative. DenseNet121 has the fewest at 11.
TP (True Positive): The number of correct predictions that an instance is positive. AlexNet seems to have the truest positives at 737.
Misclassified images No.: This number reflects the total count of images that were misclassified. ResNet18 has the lowest number at 11, which shows that it made the fewest errors in classification overall.

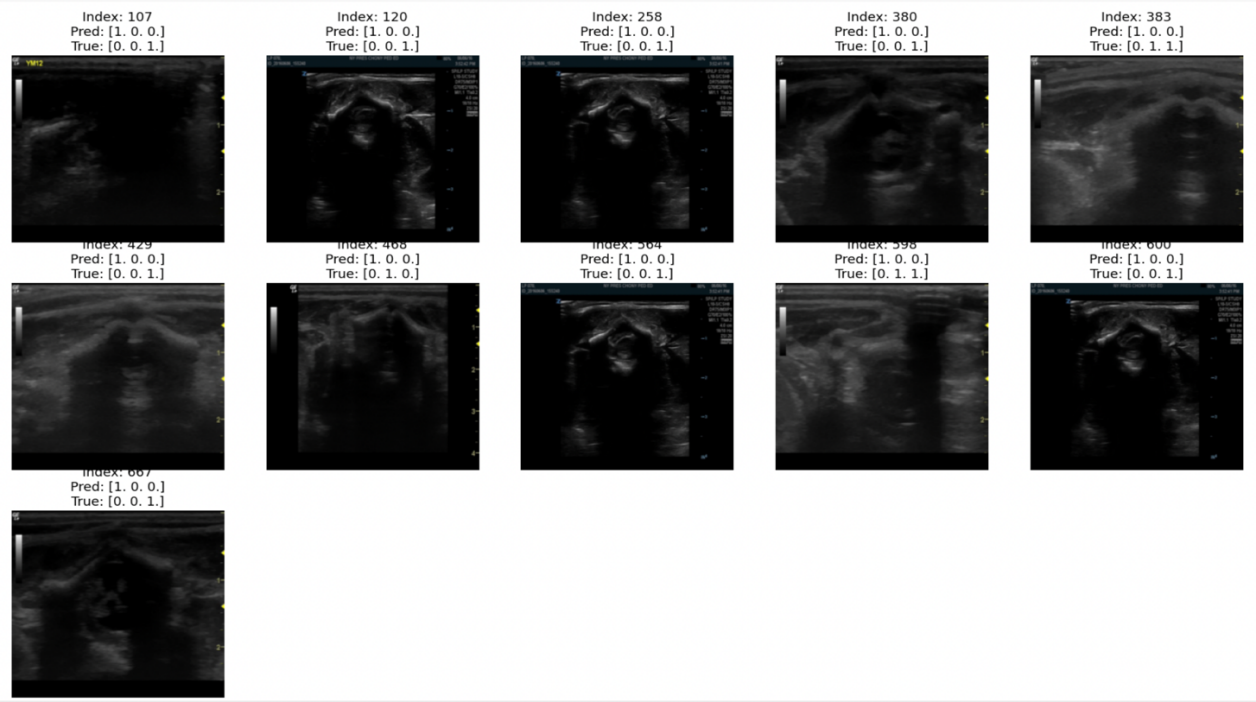
Right after DenseNet121, ResNet18 is strong contender. It is just marginally behind DenseNet121. However, this does not convey the whole story. After consulting the expert, we realized that it is essential to minimize the false positives for fluid and the false negatives for spinal cord. That is why we have highlighted that in the table. Now when you look at the visualization 2 below you will notice that ResNet18 would be a clear winner as it has 0 False positives in the case of fluids and just 3 false negatives for cord. Although this is 1 more than DenseNet121, it is certainly acceptable. Hence, we concluded that ResNet18 would perhaps be the best model.



Viz1: Performance metrics comparison of different models



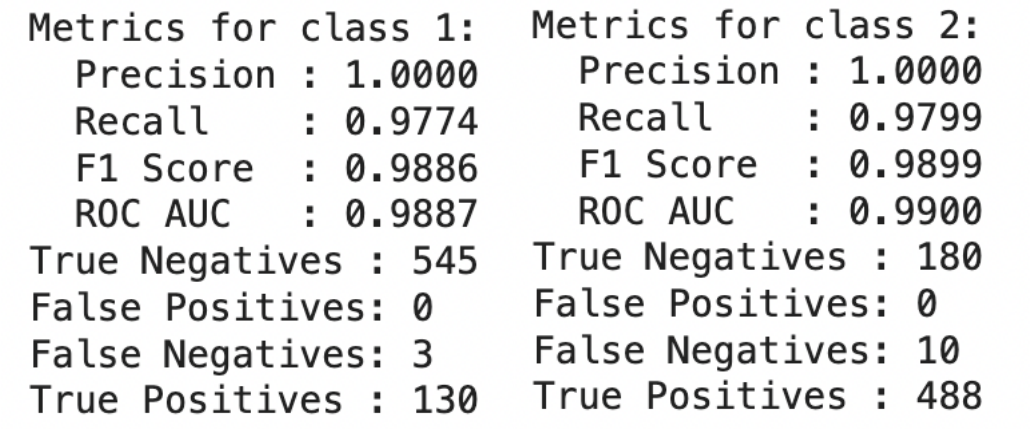
Viz 2: Comparison of False positives (FP) for Fluid and False Negatives (FN) for cord



Pred: [1. 0. 0.]
True: [0. 0. 1.]
Pred/True: [class 0 (bad quality), class 1(cord), class 2(fluid)]
1 indicates presence of feature and 0 indicates absence

It's interesting to note that the models can make out that if an image is bad quality, then spinal cord and spinal fluid will both be absent. There is absolutely no case where the model predicts an image to be bad quality and also predicts the presence of cord and fluid as '1'(present). We have already done a good chunk of error analysis for all the models but let us dive deeper into the ResNet18 model. Now we visualize all the misclassifications for this model to see whether it meets the expert standards.

On showing these visualizations to the Doctor (expert), we came to know that some of the images annotated as 'not bad quality' i.e bad quality as '0' were wrong annotations. So basically, some of the false positives for 'bad quality' were indeed positives. Hence our false positives for 'bad quality' have even reduced further now. So, it is impressive that the model is in some manner able to pick up such small granularities and can correct experts to some extent. According to the Doctor's requirement False negatives need to be minimized for spinal cord and false positives need to be minimized for spinal fluid. It's impressive how we have managed to achieve 0 false positives for spinal fluid and just 3 false negatives for spinal cord. Although only ~20% of the data contains spinal cord, our model is still able to achieve a pretty impressive recall of 0.9774 for the spinal cord. To give a more detailed idea into the metrics of spinal cord and spinal fluid specifically, please refer the image below:



Class 1 = spinal cord
Class 2 = spinal fluid

More metrics can be found in the Jupyter notebooks of our GitHub repository.
Looking at these impressive results the ResNet18 model was approved as it was at par with the expert standards.

Future Scope and plan for Object Localization

Now that we have an excellent image classifier model which tells us about the presence of a particular feature, the next step would be to think about ways to use this model to provide us with auxiliary features to aid in localization of features like spinal cord and spinal fluid. To extract the features from this model, we would just need to remove the final layer i.e classification head. This vector would be the feature vector which can then be used to localize spinal cord and spinal fluids with better accuracy. To give intuition in layman's terms: If our model already has a good understanding of what features are present in an image, then this understanding should be transferred or used to better localize these features. Although we have not been able get heatmaps to understand where our model is focusing while making a prediction, this can be explored further to bring it to fruition.

For object localization i.e getting the region of interest for spinal cord and spinal fluid we can use annotation tools like 3D slicer, ITK-SNAP, or Labelbox. These tools allow for precise labeling of medical images. Labeling format would need to be decided (e.g. bounding boxes, segmentation masks)

Segmentation: If the objective is to precisely localize the spinal cord and fluid, segmentation might be the most suitable method. This involves delineating the exact outline of these structures in each frame.

Bounding Boxes: For a simpler approach, bounding boxes can be drawn around the spinal cord and fluid. However, this might be less precise than segmentation.

We would need to ensure the labels are stored in a format compatible with your machine learning framework. Common formats include JSON, XML, or CSV files that store the coordinates and types of the labels. Once we have the annotated images, which include the bunding boxes and segmentation masks for the spinal cord and spinal fluid, the next step would be on preparing the data for training a machine learning/deep learning model. For segmentation masks, ensure they are in a format that your model can interpret, like binary masks where the target area is marked distinctly from the background. For bounding boxes, convert the coordinates into a format required by your chosen model architecture (e.g., top-left corner, width, and height). For segmentation tasks, models like U-Net or Mask R-CNN are commonly used. For object detection with bounding boxes, models like YOLO (You Only Look Once), SSD (Single Shot MultiBox Detector), or Faster R-CNN can be appropriate. Evaluate the model on the test set to assess its performance. Use metrics relevant to your task, like Intersection over Union (IoU) for segmentation or mean Average Precision (mAP) for object detection.

Project Outcomes

- Develop an entire pipeline of approach 1 (Image classification)
- Maximize accuracy, f1 score of approach 1.
- Make detailed evaluations and inferences on the outcome of all models and shortcomings.
- Based on the evaluations, discuss future.
- Develop guided annotation procedure.
- Get hands on experience dealing with medical data and annotation procedures.
- Propose a plan for region of interest approach.

We have successfully achieved all the project outcomes that were set at the start of the project.