Distribution Shift in Medical Imaging Deep-Dive Explanation of the Autoencoder & Classifier Pipeline

This document is a detailed walkthrough of the project we discussed together in the chat: using an autoencoder and a DenseNet-121 classifier to analyze distribution shift in medical imaging data. It is written so that you can read it like a script or background document before your presentation. You can also quote or adapt sentences directly into your slides or speaker notes.

1. Project Overview and Intuition

The high-level goal of the project is to understand how well a chest X■ray classifier behaves when it is exposed to data from new hospitals or new patient populations. In practice, medical AI models are usually trained on data from one or a few institutions, but deployed in very different settings. If the new data is different enough from the training data, the model's performance can degrade, and this is known as *distribution shift*.

To study this, the project uses two key components:

1. A **convolutional autoencoder (AE)** trained on chest X■ray images from the NIH dataset (adult patients). 2. A **DenseNet■121 classifier with a custom head**, trained on the NIH data and then evaluated on other datasets.

The main idea is:

- The autoencoder learns what “typical” NIH images look like and reconstructs them with low error. - When we feed new datasets (Pediatric and CheXpert) through the autoencoder, the reconstruction error tends to increase if those datasets are different from NIH. - At the same time, we look at how the **classifier’s performance** (for example, balanced accuracy, AUC) changes when we move from NIH to Pediatric and CheXpert. - We then analyze the relationship between **autoencoder reconstruction error** and **classifier performance** to see whether reconstruction error is a good unsupervised indicator of out■of■distribution (OOD) data and performance degradation.

In addition, the project explores which types of distribution shift matter most: demographic differences (adult vs pediatric), pathology differences (normal vs abnormal), or institutional/technical factors (different scanners, protocols, or hospitals).

A central insight from your discussion is that **institutional and technical factors can be a dominant source of shift**, sometimes even more important than purely demographic or pathology differences.

2. Datasets and Types of Distribution Shift

The project uses three main datasets of chest X■ray images:

1. **NIH Chest X■ray (training dataset)** This is the source dataset. It contains adult chest X■rays (mostly from a US population) and is used to: - train the convolutional autoencoder, and - train the DenseNet■121 classifier (with a custom head) to distinguish between normal and abnormal images.

2. **Pediatric dataset** This dataset contains chest X■rays from children, usually from a different institution and a different country (Chinese pediatric population). It introduces both a **demographic shift** (adult $\rightarrow$ pediatric) and likely an **institutional / technical shift** (different hospital, scanner, protocols).

3. **CheXpert (Stanford Hospital) dataset** This dataset comes from a different US institution (Stanford). The patient population is closer to NIH in age, but there are important **institutional and technical differences**, as well as potentially different label definitions and disease prevalence. CheXpert is also heavily imbalanced for certain pathologies, which affects evaluation metrics.

The key notion here is **distribution shift**. You can think of it in three broad categories:

- **Demographic shift**: different age groups, genders, ethnicities, etc. In this project, the adult NIH dataset vs the pediatric dataset is a clear example.

- **Pathology shift**: different prevalence or severity of diseases, different label distributions (for example, many more abnormal cases vs normal ones). This is particularly relevant in CheXpert, which is known to have class imbalance.

- **Institutional / technical shift**: different X■ray machines, acquisition settings, image post■processing pipelines, or hospital protocols. These can change contrast, noise, resolution, and other visual properties in non■obvious ways.

The project’s findings support the idea that **institutional and technical factors are often the strongest driver of distribution shift** and can have a large impact on model performance, sometimes more than demographics alone.

3. Efficient Data Handling with HDF5 Files

Instead of working directly with thousands of separate image files on disk, the project uses the **HDF5** file format (often written as HDF5 or H5). HDF5 stands for *Hierarchical Data Format version 5*. It is specifically designed to store large numerical datasets efficiently and to organize them in a structured way.

Using HDF5 has several advantages for this project:

- **Single container file**: Instead of loading hundreds of thousands of individual PNG/JPEG images, you read from a single HDF5 file that contains the images in a contiguous block. This reduces filesystem overhead.

- **Hierarchical structure**: You can store datasets such as `/images`, `/labels`, or `/splits` inside one file. Each dataset can be indexed and accessed like a NumPy array.

- **Fast random access**: The HDF5 file is optimized for reading chunks of data, which is ideal for mini■batch training.

- **Metadata storage**: You can store additional information, such as dataset names, normalization statistics, or split definitions (train/validation/test), as attributes.

In the context of the project, HDF5 files simplify the data pipeline:

- The raw images from NIH, Pediatric, and CheXpert are preprocessed and stored into separate HDF5 files (or HDF5 groups). - For each dataset, the HDF5 file usually holds the pixel data and the labels, and possibly indices for train/validation/test. - During training and evaluation, the code reads mini■batches directly from HDF5 instead of repeatedly touching thousands of small files on disk, which would be slower and more error■prone.

4. Image Preprocessing and Normalization

Before feeding images into the autoencoder or the classifier, they are **preprocessed and normalized**. Although the exact code can vary, the common steps are:

1. **Resizing / resampling**: All images are resized to a fixed resolution (for example, 224×224 or 256×256 pixels). This is required because both the autoencoder and DenseNet■121 expect images of a fixed size.

2. **Grayscale or channel handling**: Chest X■rays are naturally grayscale, but many deep learning frameworks expect images with three channels (RGB). A typical approach is either: - keep a single channel and adjust the model input accordingly, or - replicate the single channel three times so that the input has shape (3, H, W).

3. **Normalization (scaling)**: All images are normalized so that pixel values have a consistent range. Two common approaches are: - Scaling pixel values to [0, 1] by dividing by 255. - Standardizing to zero mean and unit variance using dataset statistics. In your case, the important point for your presentation is: **yes, the images were normalized**, so that all datasets are on a comparable scale.

4. **Train / validation / test splits**: For the NIH dataset, separate splits are created to: - train the autoencoder (mostly on normal images), - train the classifier (normal vs abnormal), and - evaluate both models and compute reconstruction error and classification metrics.

This preprocessing ensures that differences in dynamic range or image size do not overwhelm the model and that comparisons between datasets are meaningful.

5. Convolutional Autoencoder: Architecture and Design Choices

The first core model in the project is a **convolutional autoencoder (AE)**. An autoencoder consists of two parts:

- An **encoder**, which compresses an input image into a smaller latent representation (a vector of features). - A **decoder**, which tries to reconstruct the original image from this compressed representation.

For images, the encoder and decoder are usually built using **convolutional layers**. This is because convolutional layers can exploit spatial structure and are much more parameter■efficient than fully connected layers.

In simplified terms, the architecture in this project looks like this:

- **Encoder** - Input: preprocessed chest X■ray (for example, shape 1×H×W or 3×H×W). - Several `Conv2D` layers with nonlinear activations (e.g., ReLU) and optional pooling or striding. - These layers gradually reduce the spatial resolution while increasing the number of feature channels. - The final encoder output is flattened or reshaped to produce a **latent vector** of size 128.

- **Latent (bottleneck) layer** - A single dense or convolutional layer with **128 units** (or channels), using the **tanh activation function**. - This vector is the compressed representation of the image; it is what you referred to as “the cells in the latent space”.

- **Decoder** - Mirrors the encoder structure using transposed convolutions or upsampling + convolutions. - Gradually reconstructs the image back to its original resolution. - The final layer typically uses a sigmoid or linear activation to output pixel values in the desired range (e.g. [0,1]).

Why 128 latent units?

The dimensionality of the latent vector (128) is a **design choice** or hyperparameter. It must be:

- **Small enough** to force the autoencoder to learn a compressed, meaningful representation rather than just memorizing images. - **Large enough** to retain essential information so that reconstruction remains reasonably accurate.

In this project, 128 latent units strike a good balance. It is a common choice in practice for moderate■size images. You can confidently explain that 128 is a compromise between **compression** and **information preservation**.

Why use tanh in the latent layer instead of sigmoid?

The **tanh activation function** maps values to the range [-1, 1], while the **sigmoid** function maps values to [0, 1]. Using tanh in the latent layer has a few advantages:

- The latent representation is **zero■centered**, which often leads to more stable and efficient optimization. - It allows both positive and negative values, which can help the network encode more nuanced patterns. - For many autoencoder architectures, tanh in the bottleneck is a common, well■motivated choice.

So, yes: using tanh in the latent layer instead of sigmoid is a **good design decision** and you can explicitly state that it encourages a balanced, centered latent representation.

6. Training the Autoencoder: Loss, Optimizer, Batch Size, and Epochs

Once the architecture is defined, the autoencoder must be trained to reconstruct NIH images. Several training hyperparameters were chosen:

1. **Loss function: Mean Squared Error (MSE)** The autoencoder’s goal is to make the reconstructed image as close as possible to the original. A natural choice is **mean squared error**, which computes the average of (prediction – target)² over all pixels. - MSE penalizes larger deviations more strongly. - It is simple, differentiable, and widely used for reconstruction tasks.

2. **Optimizer: Adam** The **Adam optimizer** is used to update the model’s weights. Adam is popular because: - It adapts the learning rate for each parameter based on first and second moments of the gradients. - It often converges faster and more robustly than vanilla SGD. A typical learning rate is **0.001**, which is a good default for Adam in many vision tasks.

3. **Batch size: 32** During training, images are processed in **mini■batches of 32**. This is a compromise between: - having enough samples per batch to obtain a stable gradient estimate, and - keeping GPU memory usage manageable.

4. **Number of epochs: about 50** An **epoch** means one full pass through the training dataset. In this project, training is run for around 50 epochs. By that point, the reconstruction loss has largely converged and additional epochs would yield diminishing returns while increasing overfitting risk.

Mixed Precision Training

The project also uses **mixed precision training**. This means that:

- Some operations use 16■bit floating point (FP16), which is faster and uses less memory.
- Other critical operations (such as some reductions) still use 32■bit (FP32) to preserve numerical stability.

The advantages of mixed precision are:

- **Faster training**, especially on modern GPUs with tensor cores optimized for half precision. - **Reduced memory usage**, which allows larger batch sizes or larger models.

For your presentation, you can summarize it as: > "We enabled mixed precision so the model could train faster and more efficiently on GPU, without sacrificing accuracy."

7. Reconstruction Error: Per-Image and Per-Dataset Analysis

After the autoencoder is trained on NIH images, it is used as a **fixed feature extractor** to compute **reconstruction error** on all datasets (NIH, Pediatric, and CheXpert). The steps are:

1. For each image x in a dataset, pass it through the autoencoder to obtain a reconstruction $\hat{x}$. 2. Compute the **reconstruction error** (RE) for that image, typically using MSE over pixels: $$ RE(x) = \frac{1}{N} \sum_{i=1}^N (x_i - \hat{x}_i)^2 $$ where N is the number of pixels.

3. Collect these per■image reconstruction errors for each dataset.

This yields three distributions of reconstruction errors:

- **NIH RE distribution** (in■distribution data). - **Pediatric RE distribution**. - **CheXpert RE distribution**.

Using RE distributions to detect distribution shift

Once you have per■image RE values for all datasets, you can:

- Compute statistics such as mean, median, variance of RE for each dataset. - Plot histograms or density curves of RE for NIH vs Pediatric vs CheXpert.

If the Pediatric or CheXpert distributions are **shifted towards higher RE** compared to NIH, this is evidence that:

- The autoencoder finds these images “less familiar” or harder to reconstruct. - They are **out-of-distribution** relative to the NIH training data.

In the presentation, you can summarize this as:

> "We used the trained autoencoder to reconstruct images from all datasets and computed the reconstruction error per image. > By comparing the error distributions across NIH, Pediatric, and CheXpert, we could detect distribution shift: > higher reconstruction error indicates that the data is more out-of-distribution."

8. Classifier Design: DenseNet■121 with a Custom Head

The second major component of the project is the **classifier**. Here, a **DenseNet■121** architecture is used, augmented with a **custom classification head**.

DenseNet■121 is a convolutional neural network architecture that connects each layer to every subsequent layer within a block. This "dense" connectivity pattern: - improves gradient flow, - encourages feature reuse, and - allows good performance with relatively fewer parameters compared to some other deep CNNs.

In the project, the role of the classifier is:

1. To learn to **distinguish normal vs abnormal** chest X■rays using NIH data. 2. To be evaluated on NIH (in■distribution) and then on Pediatric and CheXpert (out■of■distribution) to quantify performance degradation.

There are two conceptually equivalent ways to combine the autoencoder and DenseNet■121:

1. **Use the autoencoder’s latent features as input to the classifier**. - The autoencoder encoder produces a 128■dimensional vector for each image. - DenseNet■121 (or a lighter classifier) with a custom head is trained on these features to predict normal vs abnormal.

2. **Use DenseNet■121 directly on the images (pretrained or trained from scratch)**, and treat the autoencoder as a separate OOD detector.

In your verbal explanation during the chat, we essentially followed the first interpretation: > The autoencoder compresses the image into features; then those features are used to train a DenseNet■121 classifier with a custom head on NIH; > finally, we evaluate this classifier on Pediatric and CheXpert.

What does the custom head do?

The "custom head" refers to the final layers of the classifier that are specifically tailored to the binary classification task. Typically, this consists of:

- A global pooling layer (if not already present in DenseNet■121). - One or more dense (fully connected) layers. - A final output layer with a single neuron and sigmoid activation for binary classification (or two neurons with softmax for two classes).

This head is trained to output the probability that an image is **abnormal** (or **normal**, depending on the label convention).

9. Training and Evaluating the Classifier

The workflow for the classifier is as follows:

1. **Training on NIH data** - Inputs: images (or autoencoder features) from NIH. - Labels: normal vs abnormal. - Loss function: typically binary cross■entropy for a two■class problem. - Optimizer: Adam or another suitable choice. - The classifier learns to map NIH inputs to the correct label.

2. **In■distribution evaluation (NIH test split)** - After training, evaluate the classifier on a held■out NIH test set. - Compute metrics such as: - Accuracy - Balanced accuracy (important if some classes are less frequent), - ROC AUC (area under the receiver operating characteristic curve), - Precision, recall, F1■score as needed.

3. **Out■of-distribution evaluation (Pediatric and CheXpert)** - Without retraining, apply the trained NIH classifier to Pediatric and CheXpert test sets. - Compute the same metrics as above. - Any significant drop in performance (especially in balanced accuracy or AUC) indicates **performance degradation due to distribution shift**.

Why balanced accuracy and AUC are important (especially for CheXpert)

CheXpert and similar clinical datasets often have **class imbalance**: for example, far more abnormal images than normal ones (or vice versa), or some rare pathologies.

- **Accuracy** can be misleading when the dataset is imbalanced. A model that always predicts the majority class can still achieve a high accuracy. - **Balanced accuracy** averages the recall of each class and gives a better sense of performance across both normal and abnormal cases. - **AUC (Area Under the ROC Curve)** measures how well the model ranks positive vs negative cases, independent of a specific threshold.

In your project, you correctly emphasized that:

> "Due to class imbalance in the test sets (especially CheXpert), **balanced accuracy** and **AUC** are critical metrics for evaluating the model."

10. Performance Degradation and Correlation with Reconstruction Error

Once we have:

- Per■dataset reconstruction error distributions (from the autoencoder), and - Per■dataset classifier performance metrics (e.g., AUC, balanced accuracy),

we can analyze **how these quantities are related**.

The central hypothesis is:

> Datasets that are more out■of■distribution (higher reconstruction error) will show greater classifier performance degradation (lower AUC / balanced accuracy).

To test this idea conceptually, we can:

1. Compute a summary statistic of reconstruction error for each dataset, such as mean RE or median RE. 2. Compute a performance metric for each dataset, such as AUC. 3. Look at the relationship between these two. For example, if we have three points: - (NIH mean RE, NIH AUC) - (Pediatric mean RE, Pediatric AUC) - (CheXpert mean RE, CheXpert AUC)

and we see that as mean RE increases, AUC decreases, then reconstruction error is a **useful predictor** of performance degradation.

If we had more datasets, we could compute a **correlation coefficient** (Pearson or Spearman) between RE and performance. Even with a small number of datasets, we can still qualitatively observe the trend: higher RE $\rightarrow$ lower performance.

In your words from the chat: > "It is expected that as soon as the error increases, the performance will drop." The correlation analysis formalizes this expectation.

The key takeaway for your presentation: - **Autoencoder reconstruction error is a strong indicator of out-of-distribution data**, and - It **correlates with classifier performance degradation** across datasets.

11. Key Findings and Insights (What You Should Emphasize in the Talk)

From the detailed discussion, the key insights you can confidently state are:

1. **Institutional and technical factors are a dominant source of distribution shift.** - Differences in scanners, image acquisition protocols, and hospital pipelines can significantly change image appearance. - These changes can affect model performance even when the patient population and pathologies are similar. - In the project, you observed that datasets from different institutions showed clear differences in reconstruction error and classifier performance.

2. **Autoencoder reconstruction error is a strong indicator of out-of-distribution data.** - The AE is trained only on NIH data, so it is best at reconstructing NIH-like images. - When you apply it to Pediatric or CheXpert, the reconstruction error increases. - This increase in error correlates with a drop in classifier performance, making reconstruction error a useful unsupervised OOD signal.

3. **Class imbalance in test datasets (especially CheXpert) makes balanced accuracy and AUC critical.** - With imbalanced data, raw accuracy is not sufficient. - Balanced accuracy and AUC provide a more reliable view of classifier behavior across classes.

4. **Multi-institutional validation is essential for clinical deployment.** - A model that works well on a single dataset (e.g. NIH) may fail substantially on another institution’s data. - Validating across multiple institutions (like NIH + Pediatric + CheXpert) is necessary before using such models in real clinical workflows. - This project is a good demonstration of why such validation is mandatory.

You can summarize your key message along these lines:

> "Our experiments show that autoencoder reconstruction error is a practical and informative signal for detecting distribution shift in chest X■ray datasets. We also find that institutional and technical differences between datasets are a major cause of performance degradation. This highlights the importance of multi■institutional validation and careful evaluation beyond a single training dataset."

12. Retraining the Model: Whole Network vs Custom Head

During the chat, you asked a very practical question: **If we see performance degradation on new data, do we need to retrain the whole model, or only the head?**

The answer is: - It depends on how severe the shift is and how many new labeled examples you have.

In many real situations:

- If the new dataset is not too different and you have limited labeled data, you can: - **Freeze most of the network** (for example, the convolutional backbone or the autoencoder encoder), and - Retrain or fine■tune only the **custom head** (the final layers that map features to labels).

This is called **fine-tuning** and is efficient: it adapts the model to the new distribution without training everything from scratch.

- If the new dataset is very different (for example, new imaging modality, very different image statistics), or if you have a lot of new labeled data, you may choose to: - **Fine■tune deeper layers**, or - Even retrain the entire model from scratch with combined data (old + new).

In the context of your project, you can say:

> "When we observe performance degradation on the Pediatric or CheXpert datasets, one potential remedy is to fine-tune the classifier’s custom head on new labeled data from those institutions. > If the shift is very large, more extensive retraining may be needed, but fine-tuning is often an effective first step."

13. Mixed Precision and Practical Engineering Considerations

You also asked about **mixed precision training**. In practical terms:

- Mixed precision uses 16■bit floats (FP16) where possible and 32■bit floats (FP32) where necessary. - This speeds up training and reduces memory usage on compatible GPUs. - Frameworks like PyTorch and TensorFlow provide automatic mixed precision tools that handle most details for you.

For your talk, it’s enough to explain that:

> "We enabled mixed precision to speed up training and make better use of GPU memory, which is especially helpful for large models like DenseNet■121 and for processing large image datasets."

On the engineering side, you also made good choices with:

- Using HDF5 for data storage (efficient I/O). - Normalizing all images to keep pixel value ranges comparable. - Carefully splitting datasets into train and test sets to avoid data leakage.

These details show that the project is not just theoretically interesting, but also **implemented in a robust and scalable way**.

14. How to Present This Confidently in Your Talk

Even if much of the implementation was done by someone else, you can present the project confidently by focusing on:

1. **The story, not every line of code.** - Explain the problem (distribution shift in medical imaging). - Explain the tools (autoencoder, DenseNet■121 classifier). - Explain the measurements (reconstruction error, AUC, balanced accuracy). - Explain the findings (institutional shift, RE as OOD signal, importance of multi-institutional validation).

2. **Key design choices and their motivation.** - 128■dimensional latent space: compromise between compression and information retention. - tanh in the latent layer: zero■centered, stable, expressive. - Adam + MSE for the autoencoder: standard, robust choices for reconstruction. - Mixed precision: faster training and lower memory usage.

3. **Intuitive explanations.** When you talk about reconstruction error, you can say something like: - "If the autoencoder reconstructs an image badly, it means the image is very different from what it has seen during training." - "So high reconstruction error is a sign that the data is out-of-distribution."

4. **Clear conclusions.** - Autoencoder RE correlates with classifier performance degradation. - Institutional / technical differences are a key driver of shift. - Multi■institutional validation is mandatory before clinical deployment.

If you want to improvise near the end of your presentation, you can add a forward■looking statement like:

> "In future work, we could use reconstruction error as part of a monitoring system in hospitals. > Whenever the distribution of incoming images changes significantly, the system could alert us that the model may need retraining or recalibration."

With this document and the earlier slides, you now have:

- A clear mental model of what each component does. - A detailed narrative for each phase of the project. - The language to explain key decisions and findings in a confident, expert way.

15. Final Recap

To summarize everything in one place:

- You trained a **convolutional autoencoder** on NIH chest X■ray images to learn a compact latent representation (128■dimensional, tanh activation). - You used **HDF5** to store and load large datasets efficiently, with images normalized for consistent input scaling. - You trained a **DenseNet■121 classifier with a custom head** on NIH data to distinguish normal vs abnormal images. - You evaluated the classifier on NIH (in■distribution) and on Pediatric and CheXpert (out■of■distribution). - You computed **per■image reconstruction errors** from the autoencoder on all datasets and compared their distributions to detect distribution shift. - You observed that higher reconstruction error on Pediatric and CheXpert correlates with **lower classifier performance**, confirming that RE is a strong OOD indicator. - You noted that **institutional and technical differences** are often the dominant source of shift, more so than demographics alone. - You highlighted that **class imbalance** in CheXpert makes **balanced accuracy and AUC** the most appropriate evaluation metrics. - Finally, you emphasized that **multi-institutional validation** is essential for safe clinical deployment of such models, and that fine■tuning or retraining may be necessary when performance degrades on new data.

This gives you a full, end■to■end understanding of the project that you can now use for your presentation, exam, or written report.