# Integrating Metadata, Applying Augmentation, and Fine-Tuning

In [1]:

```
import os
import copy
import pandas as pd
import torch
import torchvision
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader, WeightedRandomSampler
import torchvision.transforms as transforms
import torchvision.models as models
from PIL import Image
from sklearn.preprocessing import StandardScaler, LabelEncoder
import numpy as np
from sklearn.metrics import roc_auc_score
import torch.nn.functional as F
```

In [2]:

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device.type}")
```

```
Using device: cuda
```

In [10]:

```
CKPT_PATH = 'model.pth.tar'
N_CLASSES = 14
CLASS_NAMES = [ 'Atelectasis', 'Cardiomegaly', 'Effusion', 'Infiltration', 'Mass',
                'Pneumothorax', 'Consolidation', 'Edema', 'Emphysema', 'Fibrosis',
DATA_DIR = 'images'
TEST_IMAGE_LIST = 'labels/test_list.txt'
BATCH_SIZE = 64
TRAIN_LIST = "labels/train_list.txt"
VALID_LIST = "labels/val_list.txt"
IMAGE_DIR = "images"
```

In [11]:

```
# Load metadata
metadata = pd.read_csv("Data_Entry_2017.csv")

# Drop rows with missing age or gender (if any)
metadata = metadata.dropna(subset=["Patient Age", "Patient Gender"])

# Normalize age
scaler = StandardScaler()
metadata["age_scaled"] = scaler.fit_transform(metadata[["Patient Age"]])

# Encode gender as binary (Female=0, Male=1)
label_encoder = LabelEncoder()
metadata["gender_encoded"] = label_encoder.fit_transform(metadata["Patient Gender"]

# Create metadata dictionary
patient_info = {
    row["Image Index"]: (row["age_scaled"], row["gender_encoded"])
    for _, row in metadata.iterrows()
```

```python
}
print(f"Metadata loaded for {len(patient_info)} images.")
```

```
Metadata loaded for 112120 images.
```

In [12]:
```python
class ChestXrayDataSet(Dataset):
    def __init__(self, data_dir, image_list_file, metadata, transform=None):
        image_names, labels = [], []
        with open(image_list_file, "r") as f:
            for line in f:
                items = line.split()
                image_name = items[0]
                label = [int(i) for i in items[1:]]
                image_name = os.path.join(data_dir, image_name)
                image_names.append(image_name)
                labels.append(label)

        self.image_names = image_names
        self.labels = labels
        self.metadata = metadata
        self.transform = transform

    def __getitem__(self, index):
        image_name = self.image_names[index]
        image = Image.open(image_name).convert('RGB')
        label = self.labels[index]

        base_name = os.path.basename(image_name)
        age, gender = self.metadata.get(base_name, (0.0, 0.0))

        if self.transform:
            image = self.transform(image)

        age_tensor = torch.tensor([age], dtype=torch.float32)
        gender_tensor = torch.tensor([gender], dtype=torch.float32)

        return image, torch.FloatTensor(label), age_tensor, gender_tensor

    def __len__(self):
        return len(self.image_names)
```

In [13]:
```python
class DenseNet121WithMetadata(nn.Module):
    def __init__(self, out_size):
        super(DenseNet121WithMetadata, self).__init__()
        # Load pretrained DenseNet
        self.densenet = models.densenet121(pretrained=True)

        # Get the feature size
        self.feature_size = self.densenet.classifier.in_features

        # Remove the original classifier
        self.densenet.classifier = nn.Identity()

        # Global pooling
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
```

```
        # Create a new classifier that takes image features + metadata
        self.classifier = nn.Sequential(
            nn.Dropout(0.2),
            nn.Linear(self.feature_size + 2, 1024),  # +2 for age and gender
            nn.BatchNorm1d(1024),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(1024, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(512, 256),
            nn.BatchNorm1d(256),
            nn.ReLU(),
            nn.Linear(256, out_size)
        )

    def forward(self, x, age, gender):
        # Extract features
        features = self.densenet.features(x)
        features = torch.relu(features)
        features = self.avgpool(features)
        features = torch.flatten(features, 1)

        # Combine metadata
        metadata = torch.cat([age, gender], dim=1)
        combined = torch.cat([features, metadata], dim=1)

        # Classify
        output = self.classifier(combined)
        return output
```

In [14]:
```
def compute_AUCs(gt, pred):
    """Computes Area Under the Curve (AUC) from prediction scores.

    Args:
        gt: Pytorch tensor on GPU, shape = [n_samples, n_classes]
          true binary labels.
        pred: Pytorch tensor on GPU, shape = [n_samples, n_classes]
          can either be probability estimates of the positive class,
          confidence values, or binary decisions.

    Returns:
        List of AUROCs of all classes.
    """
    AUROCs = []
    gt_np = gt.cpu().numpy()
    pred_np = pred.cpu().numpy()
    for i in range(N_CLASSES):
        AUROCs.append(roc_auc_score(gt_np[:, i], pred_np[:, i]))
    return AUROCs
```

In [15]:
```
# Data transformations
train_transforms = transforms.Compose([
    transforms.Resize(256),
```

```
    transforms.RandomResizedCrop(224),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

valid_transforms = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

In [18]:

```
train_dataset = ChestXrayDataSet(IMAGE_DIR, TRAIN_LIST, metadata=patient_info, tran
valid_dataset = ChestXrayDataSet(IMAGE_DIR, VALID_LIST, metadata=patient_info, tran

# Class counts in the same order as CLASS_NAMES
class_counts = [
    313,  # Atelectasis
    141,  # Cardiomegaly
    341,  # Effusion
    580,  # Infiltration
    111,  # Mass
    151,  # Nodule
    45,   # Pneumonia
    141,  # Pneumothorax
    136,  # Consolidation
    62,   # Edema
    86,   # Emphysema
    117,  # Fibrosis
    114,  # Pleural_Thickening
    17    # Hernia
]

# Total samples and class weights
total_count = sum(class_counts)
class_weights = [total_count / count for count in class_counts]

# Keep weights on CPU for computing sample weights
class_weights_cpu = torch.tensor(class_weights, dtype=torch.float)

sample_weights = []
for data_tuple in train_dataset:
    label = data_tuple[1].float()
    weight = torch.sum(class_weights_cpu * label).item()
    sample_weights.append(weight)

sample_weights = torch.tensor(sample_weights, dtype=torch.float)

sampler = WeightedRandomSampler(weights=sample_weights, num_samples=len(sample_weig

trainloader = DataLoader(train_dataset, sampler=sampler, batch_size=64, pin_memory=
validloader = DataLoader(valid_dataset, batch_size=64, shuffle=False, pin_memory=Tr
```

In [19]:

```
new_model = DenseNet121WithMetadata(N_CLASSES).to(device)

for param in new_model.densenet.parameters():
    param.requires_grad = False

# Unfreeze last dense block and transition
for param in new_model.densenet.features.denseblock4.parameters():
    param.requires_grad = True

for param in new_model.densenet.features.transition3.parameters():
    param.requires_grad = True

for param in new_model.classifier.parameters():
    param.requires_grad = True

trainable_params = [p for p in new_model.parameters() if p.requires_grad]
print(f"Number of trainable parameters: {sum(p.numel() for p in trainable_params):,

criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(
    filter(lambda p: p.requires_grad, new_model.parameters()),
    lr=1e-4, weight_decay=5e-4
)
```

```
c:\Users\aroce\miniforge3\envs\572\Lib\site-packages\torchvision\models\_utils.py:20
8: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be remov
ed in the future, please use 'weights' instead.
  warnings.warn(
c:\Users\aroce\miniforge3\envs\572\Lib\site-packages\torchvision\models\_utils.py:22
3: UserWarning: Arguments other than a weight enum or `None` for 'weights' are depre
cated since 0.13 and may be removed in the future. The current behavior is equivalen
t to passing `weights=DenseNet121_Weights.IMAGENET1K_V1`. You can also use `weights=
DenseNet121_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Number of trainable parameters: 4,399,374
```

In [20]:

```
def trainer(model, criterion, optimizer, trainloader, validloader, epochs=20, patie
    train_loss, valid_loss, valid_accuracy = [], [], []
    best_auroc = 0.0
    best_model_weights = None
    counter = 0

    for epoch in range(epochs):
        model.train()
        epoch_train_loss = 0.0

        for images, labels, ages, genders in trainloader:
            images, labels = images.to(device), labels.to(device)
            ages, genders = ages.to(device), genders.to(device)

            optimizer.zero_grad()
            outputs = model(images, ages, genders)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
```

```
            epoch_train_loss += loss.item() * images.size(0)

        train_loss.append(epoch_train_loss / len(trainloader.dataset))

        # Validation Phase
        model.eval()
        epoch_valid_loss = 0.0
        all_labels = torch.FloatTensor().to(device)
        all_outputs = torch.FloatTensor().to(device)

        with torch.no_grad():
            for images, labels, ages, genders in validloader:
                images, labels = images.to(device), labels.to(device)
                ages, genders = ages.to(device), genders.to(device)

                outputs = model(images, ages, genders)
                loss = criterion(outputs, labels)
                epoch_valid_loss += loss.item() * images.size(0)

                all_labels = torch.cat((all_labels, labels), 0)
                all_outputs = torch.cat((all_outputs, outputs), 0)

        # Calculate metrics
        predictions = (all_outputs > 0.5).float()
        correct = (predictions == all_labels).sum().item()
        total = all_labels.numel()
        accuracy = correct / total

        aurocs = compute_AUCs(all_labels, all_outputs)
        mean_auroc = np.mean(aurocs)

        valid_loss.append(epoch_valid_loss / len(validloader.dataset))
        valid_accuracy.append(accuracy)

        if verbose:
            print(f"Epoch {epoch+1}/{epochs} - Train Loss: {train_loss[-1]:.4f}, Va
            print(f"Accuracy: {accuracy:.4f}, Mean AUROC: {mean_auroc:.4f}")

            if (epoch + 1) % 5 == 0 or epoch == epochs - 1:
                for i, auroc in enumerate(aurocs):
                    print(f"  {CLASS_NAMES[i]}: AUROC = {auroc:.4f}")

        # Early stopping and model saving
        if mean_auroc > best_auroc:
            best_auroc = mean_auroc
            best_model_weights = copy.deepcopy(model.state_dict())
            print(f"  New best model with AUROC: {best_auroc:.4f}")
            counter = 0
        else:
            counter += 1
            if counter >= patience:
                print(f"  Early stopping triggered after {epoch+1} epochs")
                break

    # Load the best model weights
    if best_model_weights is not None:
```

```
        model.load_state_dict(best_model_weights)
        print(f"Loaded best model with AUROC: {best_auroc:.4f}")

    return {
        'train_loss': train_loss,
        'valid_loss': valid_loss,
        'valid_accuracy': valid_accuracy,
        'best_auroc': best_auroc
    }
```

In [21]:

```
new_model.to(device)
trainer(new_model, criterion, optimizer, trainloader, validloader, epochs=50, patie
```

```
Epoch 1/50 - Train Loss: 0.6150, Valid Loss: 0.5667
Accuracy: 0.9497, Mean AUROC: 0.6157
  New best model with AUROC: 0.6157
Epoch 2/50 - Train Loss: 0.4905, Valid Loss: 0.4570
Accuracy: 0.9508, Mean AUROC: 0.6442
  New best model with AUROC: 0.6442
Epoch 3/50 - Train Loss: 0.4239, Valid Loss: 0.3713
Accuracy: 0.9498, Mean AUROC: 0.6760
  New best model with AUROC: 0.6760
Epoch 4/50 - Train Loss: 0.3857, Valid Loss: 0.3343
Accuracy: 0.9508, Mean AUROC: 0.6898
  New best model with AUROC: 0.6898
Epoch 5/50 - Train Loss: 0.3634, Valid Loss: 0.2942
Accuracy: 0.9499, Mean AUROC: 0.7046
  Atelectasis: AUROC = 0.6619
  Cardiomegaly: AUROC = 0.6726
  Effusion: AUROC = 0.7833
  Infiltration: AUROC = 0.5958
  Mass: AUROC = 0.6043
  Nodule: AUROC = 0.5362
  Pneumonia: AUROC = 0.7455
  Pneumothorax: AUROC = 0.8035
  Consolidation: AUROC = 0.7213
  Edema: AUROC = 0.8163
  Emphysema: AUROC = 0.6511
  Fibrosis: AUROC = 0.7172
  Pleural_Thickening: AUROC = 0.6729
  Hernia: AUROC = 0.8821
  New best model with AUROC: 0.7046
Epoch 6/50 - Train Loss: 0.3533, Valid Loss: 0.2782
Accuracy: 0.9492, Mean AUROC: 0.7105
  New best model with AUROC: 0.7105
Epoch 7/50 - Train Loss: 0.3408, Valid Loss: 0.2593
Accuracy: 0.9489, Mean AUROC: 0.7114
  New best model with AUROC: 0.7114
Epoch 8/50 - Train Loss: 0.3341, Valid Loss: 0.2519
Accuracy: 0.9481, Mean AUROC: 0.7204
  New best model with AUROC: 0.7204
Epoch 9/50 - Train Loss: 0.3192, Valid Loss: 0.2443
Accuracy: 0.9465, Mean AUROC: 0.7174
Epoch 10/50 - Train Loss: 0.3180, Valid Loss: 0.2333
Accuracy: 0.9471, Mean AUROC: 0.7218
  Atelectasis: AUROC = 0.6849
  Cardiomegaly: AUROC = 0.7513
  Effusion: AUROC = 0.8118
  Infiltration: AUROC = 0.6088
  Mass: AUROC = 0.6355
  Nodule: AUROC = 0.5418
  Pneumonia: AUROC = 0.6996
  Pneumothorax: AUROC = 0.8191
  Consolidation: AUROC = 0.7208
  Edema: AUROC = 0.8056
  Emphysema: AUROC = 0.7062
  Fibrosis: AUROC = 0.7173
  Pleural_Thickening: AUROC = 0.6455
  Hernia: AUROC = 0.9564
```

```
  New best model with AUROC: 0.7218
Epoch 11/50 - Train Loss: 0.3075, Valid Loss: 0.2285
Accuracy: 0.9457, Mean AUROC: 0.7198
Epoch 12/50 - Train Loss: 0.3004, Valid Loss: 0.2241
Accuracy: 0.9474, Mean AUROC: 0.7236
  New best model with AUROC: 0.7236
Epoch 13/50 - Train Loss: 0.2943, Valid Loss: 0.2213
Accuracy: 0.9444, Mean AUROC: 0.7223
Epoch 14/50 - Train Loss: 0.2920, Valid Loss: 0.2191
Accuracy: 0.9477, Mean AUROC: 0.7256
  New best model with AUROC: 0.7256
Epoch 15/50 - Train Loss: 0.2847, Valid Loss: 0.2201
Accuracy: 0.9416, Mean AUROC: 0.7226
  Atelectasis: AUROC = 0.6794
  Cardiomegaly: AUROC = 0.7554
  Effusion: AUROC = 0.8127
  Infiltration: AUROC = 0.6143
  Mass: AUROC = 0.6082
  Nodule: AUROC = 0.5192
  Pneumonia: AUROC = 0.7357
  Pneumothorax: AUROC = 0.8308
  Consolidation: AUROC = 0.7113
  Edema: AUROC = 0.8265
  Emphysema: AUROC = 0.7140
  Fibrosis: AUROC = 0.6902
  Pleural_Thickening: AUROC = 0.6559
  Hernia: AUROC = 0.9635
Epoch 16/50 - Train Loss: 0.2762, Valid Loss: 0.2225
Accuracy: 0.9428, Mean AUROC: 0.7286
  New best model with AUROC: 0.7286
Epoch 17/50 - Train Loss: 0.2716, Valid Loss: 0.2296
Accuracy: 0.9390, Mean AUROC: 0.7343
  New best model with AUROC: 0.7343
Epoch 18/50 - Train Loss: 0.2660, Valid Loss: 0.2138
Accuracy: 0.9414, Mean AUROC: 0.7333
Epoch 19/50 - Train Loss: 0.2584, Valid Loss: 0.2289
Accuracy: 0.9355, Mean AUROC: 0.7263
Epoch 20/50 - Train Loss: 0.2567, Valid Loss: 0.2186
Accuracy: 0.9388, Mean AUROC: 0.7278
  Atelectasis: AUROC = 0.7016
  Cardiomegaly: AUROC = 0.7802
  Effusion: AUROC = 0.8256
  Infiltration: AUROC = 0.6017
  Mass: AUROC = 0.6202
  Nodule: AUROC = 0.5364
  Pneumonia: AUROC = 0.7641
  Pneumothorax: AUROC = 0.8482
  Consolidation: AUROC = 0.7133
  Edema: AUROC = 0.8630
  Emphysema: AUROC = 0.6608
  Fibrosis: AUROC = 0.6840
  Pleural_Thickening: AUROC = 0.6496
  Hernia: AUROC = 0.9404
Epoch 21/50 - Train Loss: 0.2496, Valid Loss: 0.2115
Accuracy: 0.9440, Mean AUROC: 0.7298
Epoch 22/50 - Train Loss: 0.2419, Valid Loss: 0.2105
```

```
Accuracy: 0.9429, Mean AUROC: 0.7262
Epoch 23/50 - Train Loss: 0.2376, Valid Loss: 0.2170
Accuracy: 0.9384, Mean AUROC: 0.7243
Epoch 24/50 - Train Loss: 0.2318, Valid Loss: 0.2111
Accuracy: 0.9402, Mean AUROC: 0.7336
Epoch 25/50 - Train Loss: 0.2234, Valid Loss: 0.2089
Accuracy: 0.9417, Mean AUROC: 0.7215
  Atelectasis: AUROC = 0.6988
  Cardiomegaly: AUROC = 0.7512
  Effusion: AUROC = 0.8352
  Infiltration: AUROC = 0.6037
  Mass: AUROC = 0.6480
  Nodule: AUROC = 0.5403
  Pneumonia: AUROC = 0.5983
  Pneumothorax: AUROC = 0.8237
  Consolidation: AUROC = 0.7522
  Edema: AUROC = 0.8707
  Emphysema: AUROC = 0.7142
  Fibrosis: AUROC = 0.6718
  Pleural_Thickening: AUROC = 0.6107
  Hernia: AUROC = 0.9821
  Early stopping triggered after 25 epochs
Loaded best model with AUROC: 0.7343
```

```
Out[21]: {'train_loss': [0.6150076894563892,
  0.4905262444403622,
  0.42387798437904584,
  0.3856518385443697,
  0.3633594352131266,
  0.35329419158976155,
  0.34079630753420936,
  0.33410104391199413,
  0.31920621483010475,
  0.31799494007309015,
  0.30749466005956555,
  0.3003865083427762,
  0.2942822369618837,
  0.2919568213676514,
  0.28467885411000315,
  0.2761934056013576,
  0.27163809748266926,
  0.26600651799457076,
  0.25842904915091447,
  0.2566547337243406,
  0.2495752162475455,
  0.24194228084862796,
  0.23764604326230043,
  0.23177117166365171,
  0.2234396956398951],
 'valid_loss': [0.5666714046796163,
  0.4569898856480916,
  0.371339679479599,
  0.3343385257720947,
  0.2942291405200958,
  0.278245933453242,
  0.25931656138102216,
  0.25193688142299653,
  0.24427994549274445,
  0.23331503105163573,
  0.22846508328119913,
  0.22407191602389018,
  0.22131952675183614,
  0.21914870476722717,
  0.2200619997183482,
  0.22248999659220378,
  0.22964911778767905,
  0.21384362451235453,
  0.22886449948946636,
  0.21856773742039998,
  0.2115029188791911,
  0.21054298710823058,
  0.21701028501987457,
  0.21108198753992716,
  0.20893272856871287],
 'valid_accuracy': [0.9497142857142857,
  0.9507619047619048,
  0.9498095238095238,
  0.9507619047619048,
  0.9499047619047619,
  0.9492380952380952,
```

```
 0.9488571428571428,
 0.9480952380952381,
 0.9464761904761905,
 0.9471428571428572,
 0.9457142857142857,
 0.9474285714285714,
 0.9443809523809524,
 0.9477142857142857,
 0.9416190476190476,
 0.9427619047619048,
 0.939047619047619,
 0.9414285714285714,
 0.9355238095238095,
 0.9387619047619048,
 0.944,
 0.9428571428571428,
 0.9383809523809524,
 0.9401904761904762,
 0.9417142857142857],
'best_auroc': np.float64(0.734284917878435)}
```