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 [3]:
```
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 [4]:
```
# 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()
```

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

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

In [5]:
```
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 [6]:
```
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))
```

```python
        # 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 [7]:
```python
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 [8]:
```python
# Data transformations
train_transforms = transforms.Compose([
    transforms.Resize(256),
```

```python
    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.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

In [9]:

```python
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 = [
    959,  # Atelectasis
    276,  # Cardiomegaly
    972,  # Effusion
    1591, # Infiltration
    313,  # Mass
    461,  # Nodule
    140,  # Pneumonia
    469,  # Pneumothorax
    397,  # Consolidation
    140,  # Edema
    208,  # Emphysema
    244,  # Fibrosis
    349,  # Pleural_Thickening
    35    # 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 [10]:

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

for param in new_model.densenet.parameters():
```

```python
    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 [11]:
```python
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 [12]:
```
new_model.to(device)
trainer(new_model, criterion, optimizer, trainloader, validloader, epochs=50, patie
```

```
Epoch 1/50 - Train Loss: 0.6402, Valid Loss: 0.6068
Accuracy: 0.8925, Mean AUROC: 0.6280
  New best model with AUROC: 0.6280
Epoch 2/50 - Train Loss: 0.5222, Valid Loss: 0.5258
Accuracy: 0.8938, Mean AUROC: 0.6634
  New best model with AUROC: 0.6634
Epoch 3/50 - Train Loss: 0.4505, Valid Loss: 0.4634
Accuracy: 0.8933, Mean AUROC: 0.6885
  New best model with AUROC: 0.6885
Epoch 4/50 - Train Loss: 0.4044, Valid Loss: 0.4157
Accuracy: 0.8943, Mean AUROC: 0.7031
  New best model with AUROC: 0.7031
Epoch 5/50 - Train Loss: 0.3756, Valid Loss: 0.3851
Accuracy: 0.8939, Mean AUROC: 0.7193
  Atelectasis: AUROC = 0.7252
  Cardiomegaly: AUROC = 0.7403
  Effusion: AUROC = 0.7053
  Infiltration: AUROC = 0.5934
  Mass: AUROC = 0.6501
  Nodule: AUROC = 0.7299
  Pneumonia: AUROC = 0.5447
  Pneumothorax: AUROC = 0.6838
  Consolidation: AUROC = 0.7013
  Edema: AUROC = 0.8709
  Emphysema: AUROC = 0.7417
  Fibrosis: AUROC = 0.6867
  Pleural_Thickening: AUROC = 0.7210
  Hernia: AUROC = 0.9761
  New best model with AUROC: 0.7193
Epoch 6/50 - Train Loss: 0.3530, Valid Loss: 0.3615
Accuracy: 0.8942, Mean AUROC: 0.7309
  New best model with AUROC: 0.7309
Epoch 7/50 - Train Loss: 0.3376, Valid Loss: 0.3426
Accuracy: 0.8949, Mean AUROC: 0.7342
  New best model with AUROC: 0.7342
Epoch 8/50 - Train Loss: 0.3204, Valid Loss: 0.3268
Accuracy: 0.8944, Mean AUROC: 0.7413
  New best model with AUROC: 0.7413
Epoch 9/50 - Train Loss: 0.3020, Valid Loss: 0.3190
Accuracy: 0.8946, Mean AUROC: 0.7490
  New best model with AUROC: 0.7490
Epoch 10/50 - Train Loss: 0.2934, Valid Loss: 0.3094
Accuracy: 0.8939, Mean AUROC: 0.7477
  Atelectasis: AUROC = 0.7266
  Cardiomegaly: AUROC = 0.8291
  Effusion: AUROC = 0.7314
  Infiltration: AUROC = 0.6225
  Mass: AUROC = 0.6874
  Nodule: AUROC = 0.7339
  Pneumonia: AUROC = 0.5596
  Pneumothorax: AUROC = 0.7498
  Consolidation: AUROC = 0.7344
  Edema: AUROC = 0.8642
  Emphysema: AUROC = 0.8082
  Fibrosis: AUROC = 0.7209
  Pleural_Thickening: AUROC = 0.7557
```

```
  Hernia: AUROC = 0.9445
Epoch 11/50 - Train Loss: 0.2789, Valid Loss: 0.3042
Accuracy: 0.8949, Mean AUROC: 0.7454
Epoch 12/50 - Train Loss: 0.2666, Valid Loss: 0.2985
Accuracy: 0.8964, Mean AUROC: 0.7460
Epoch 13/50 - Train Loss: 0.2528, Valid Loss: 0.2946
Accuracy: 0.8944, Mean AUROC: 0.7467
Epoch 14/50 - Train Loss: 0.2414, Valid Loss: 0.2931
Accuracy: 0.8946, Mean AUROC: 0.7411
Epoch 15/50 - Train Loss: 0.2278, Valid Loss: 0.2899
Accuracy: 0.8940, Mean AUROC: 0.7410
  Atelectasis: AUROC = 0.7228
  Cardiomegaly: AUROC = 0.8316
  Effusion: AUROC = 0.7276
  Infiltration: AUROC = 0.6224
  Mass: AUROC = 0.7090
  Nodule: AUROC = 0.7416
  Pneumonia: AUROC = 0.6333
  Pneumothorax: AUROC = 0.7679
  Consolidation: AUROC = 0.6882
  Edema: AUROC = 0.7984
  Emphysema: AUROC = 0.7951
  Fibrosis: AUROC = 0.6933
  Pleural_Thickening: AUROC = 0.6903
  Hernia: AUROC = 0.9533
Epoch 16/50 - Train Loss: 0.2182, Valid Loss: 0.2879
Accuracy: 0.8945, Mean AUROC: 0.7433
Epoch 17/50 - Train Loss: 0.2105, Valid Loss: 0.2899
Accuracy: 0.8948, Mean AUROC: 0.7389
  Early stopping triggered after 17 epochs
Loaded best model with AUROC: 0.7490
```

Out[12]:
```
{'train_loss': [0.6401657053402492,
  0.5221529744352613,
  0.4504772332736424,
  0.40440248795918055,
  0.3756065801211766,
  0.35303262659481593,
  0.33764585648264206,
  0.320412438426699,
  0.30202747651508877,
  0.2933856477056231,
  0.27887831483568465,
  0.26655664801597595,
  0.2528457125595638,
  0.2413997313805989,
  0.22780898877552577,
  0.21820709884166717,
  0.21054484988961902],
 'valid_loss': [0.6067854873339336,
  0.5258200287818908,
  0.4633747879664103,
  0.41570135911305744,
  0.3850545903046926,
  0.3615047554175059,
  0.34262948791186015,
  0.3268141750494639,
  0.3189827088514964,
  0.30938701272010805,
  0.3042289062341054,
  0.29847488284111023,
  0.29459155559539796,
  0.2930800994237264,
  0.2899391531944275,
  0.2878683694203695,
  0.28992931842803954],
 'valid_accuracy': [0.8925,
  0.8938095238095238,
  0.8933333333333333,
  0.8942857142857142,
  0.8939285714285714,
  0.8941666666666667,
  0.8948809523809523,
  0.8944047619047619,
  0.8946428571428572,
  0.8939285714285714,
  0.8948809523809523,
  0.8964285714285715,
  0.8944047619047619,
  0.8946428571428572,
  0.8940476190476191,
  0.8945238095238095,
  0.8947619047619048],
 'best_auroc': np.float64(0.7490224347009925)}
```