An Improved Densenet Deep Neural Network Model for Tuberculosis Detection Using Chest X-Ray Images

Spring 2025: Neural Networks & Deep Learning- Mini Project

Name: Veera Manikanta Kumar Allada

Student ID: 700756934

Github Link: https://github.com/maniallada9/Neural-Networks-deep-Learning

```
[1]  # Mount Google Drive
     from google.colab import drive
     drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
def prepare_dataset():
    # Google Drive mounted path
    MONT_PATH = "/content/drive/MyDrive/DataSet/MontgomerySet/CXR_png
    SHENZHEN_PATH = "/content/drive/MyDrive/DataSet/ChinaSet_AllFiles
    TBX11K_PATH = "/content/drive/MyDrive/TB_Dataset/TBnNormal"

    OUTPUT_PATH = "/content/filtered_dataset"
    TB_DIR = os.path.join(OUTPUT_PATH, "TB")
    NORMAL_DIR = os.path.join(OUTPUT_PATH, "Normal")

    os.makedirs(TB_DIR, exist_ok=True)
    os.makedirs(NORMAL_DIR, exist_ok=True)

    def copy_resize(img_path, label, count):
        target_dir = TB_DIR if label == "TB" else NORMAL_DIR
        try:
            img = Image.open(img_path)
            img.verify()  # Check image validity
            img = Image.open(img_path).convert("RGB").resize((224, 22
            img.save(os.path.join(target_dir, f"{label}_{count}.jpg")
        except (UnidentifiedImageError, OSError) as e:
            print(f"Skipped {img_path}: {e}")

    image_records = []

    # Montgomery: TB images have '_1' in name, Normal have '_0'
    for img in Path(MONT_PATH).rglob("*.png"):
```

```
            elif "_1.png" in name:
                label = "TB"
            else:
                continue
            image_records.append((img, label))

        # Shenzhen: Same logic
        for img in Path(SHENZHEN_PATH).rglob("*.png"):
            name = img.name
            if "_0.png" in name:
                label = "Normal"
            elif "_1.png" in name:
                label = "TB"
            else:
                continue
            image_records.append((img, label))

        # TBX11K - already separated
        tbx_tb_path = os.path.join(TBX11K_PATH, "PULMONARY_TUBERCULOSIS")
        tbx_normal_path = os.path.join(TBX11K_PATH, "NORMAL")

        for img in Path(tbx_tb_path).rglob("*.jpg"):
            image_records.append((img, "TB"))
        for img in Path(tbx_normal_path).rglob("*.jpg"):
            image_records.append((img, "Normal"))

        # Shuffle and limit
        random.shuffle(image_records)
        tb_images = [img for img in image_records if img[1] == "TB"][:2500]
        normal_images = [img for img in image_records if img[1] == "Normal"][:2500]

        print(f"Total TB images found: {len(tb_images)}")
        print(f"Total Normal images found: {len(normal_images)}")
```

```
def get_dataloaders(data_dir, batch_size=32):
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
    ])

    dataset = datasets.ImageFolder(data_dir, transform=transform)
    train_size = int(0.8 * len(dataset))
    test_size = len(dataset) - train_size
    train_dataset, test_dataset = torch.utils.data.random_split(dataset, [train_size, test_size])

    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

    return train_loader, test_loader
```

```python
class ChannelAttention(nn.Module):
    def __init__(self, in_planes, reduction=16):
        super(ChannelAttention, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.max_pool = nn.AdaptiveMaxPool2d(1)
        self.fc = nn.Sequential(
            nn.Conv2d(in_planes, in_planes // reduction, 1, bias=False),
            nn.ReLU(),
            nn.Conv2d(in_planes // reduction, in_planes, 1, bias=False)
        )
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        avg_out = self.fc(self.avg_pool(x))
        max_out = self.fc(self.max_pool(x))
        return self.sigmoid(avg_out + max_out)

class SpatialAttention(nn.Module):
    def __init__(self, kernel_size=7):
        super(SpatialAttention, self).__init__()
        padding = kernel_size // 2
        self.conv = nn.Conv2d(2, 1, kernel_size, padding=padding, bias=False)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        avg = torch.mean(x, dim=1, keepdim=True)
        max_, _ = torch.max(x, dim=1, keepdim=True)
        x = torch.cat([avg, max_], dim=1)
```

```python
def train_and_evaluate():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    train_loader, test_loader = get_dataloaders("/content/filtered_dataset", batch_size=32)
    model = CBAMWDnet(num_classes=2).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=1e-4)

    train_losses = []

    for epoch in range(10):
        model.train()
        total_loss = 0
        for imgs, labels in train_loader:
            imgs, labels = imgs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(imgs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        print(f"Epoch {epoch+1}, Loss: {total_loss / len(train_loader):.4f}")

    # Evaluation
    model.eval()
    all_preds, all_labels = [], []
    with torch.no_grad():
        for imgs, labels in test_loader:
            imgs = imgs.to(device)
            outputs = model(imgs)
            preds = torch.argmax(outputs, dim=1).cpu()
```

```python
if __name__ == "__main__":
    print("Preparing dataset...")
    prepare_dataset()
    print("Training and evaluating model...")
    train_and_evaluate()
```

```
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprec
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum c
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/densenet121-a639ec97.pth" to /root/.cache/torch/hub/checkpoints/densenet1
100%|██████████| 30.8M/30.8M [00:00<00:00, 163MB/s]
Epoch 1, Loss: 0.5895
Epoch 2, Loss: 0.3711
Epoch 3, Loss: 0.2457
Epoch 4, Loss: 0.1332
Epoch 5, Loss: 0.1292
Epoch 6, Loss: 0.0953
Epoch 7, Loss: 0.0763
Epoch 8, Loss: 0.0589
Epoch 9, Loss: 0.0459
Epoch 10, Loss: 0.0167
              precision    recall  f1-score   support

      Normal       0.83      0.90      0.86        84
          TB       0.88      0.79      0.83        76

    accuracy                           0.85       160
   macro avg       0.85      0.85      0.85       160
weighted avg       0.85      0.85      0.85       160
```