Project 1: Medical Imaging Tool for TB Classification

Introduction

This project involves designing a neural network to classify chest X-rays into two categories: those showing tuberculosis (TB) manifestations and those that are normal. The dataset used is the Montgomery County Chest X-ray Database provided by the National Library of Medicine.

Dataset Description

The dataset contains chest X-ray images categorized into two groups:

- **Normal Cases:** 80 images
- **TB Cases:** 58 images

The images are in PNG format with the following characteristics:

- Matrix size: 4020 x 4892 or 4892 x 4020
- Pixel spacing: 0.0875 mm
- Number of gray levels: 12 bits

Data Preparation

Loading Libraries

First, we need to load the necessary libraries for data processing, neural network design, and evaluation.

```
import numpy as np
import pandas as pd
import os
import cv2
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
```

```python
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Loading the Dataset

The images and their corresponding labels are loaded from the CXR_png directory. The file names include _0 for normal cases and _1 for TB cases.

```python
# Define paths
data_dir = '/Users/rishi/Desktop/PROGRAMMING/AI in Medical Imaging
/Project_1/MontgomerySet/CXR_png/'

# Function to load images and labels based on file names
def load_images_and_labels(folder):
    images = []
    labels = []
    for filename in os.listdir(folder):
        if filename.endswith('.png'):
            img_path = os.path.join(folder, filename)
            img = tf.keras.preprocessing.image.load_img(img_path,
target_size=(256, 256))
            img = tf.keras.preprocessing.image.img_to_array(img)
            images.append(img)

            # Determine label from file name
            if '_0' in filename:
                labels.append(0)  # Normal case
            elif '_1' in filename:
                labels.append(1)  # TB case

    return np.array(images), np.array(labels)

# Load images and labels
X, y = load_images_and_labels(data_dir)

# Normalize the images
X = X / 255.0

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

print(f"Training set size: {X_train.shape[0]} images")
print(f"Testing set size: {X_test.shape[0]} images")
```

```
Training set size: 110 images
Testing set size: 28 images
```

Neural Network Design

We design a Convolutional Neural Network (CNN) with multiple hidden layers to classify the chest X-ray images.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D,
Flatten, Dropout

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 254, 254, 32)      896

 max_pooling2d_3 (MaxPoolin  (None, 127, 127, 32)      0
 g2D)

 conv2d_4 (Conv2D)           (None, 125, 125, 64)      18496

 max_pooling2d_4 (MaxPoolin  (None, 62, 62, 64)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 60, 60, 128)       73856

 max_pooling2d_5 (MaxPoolin  (None, 30, 30, 128)       0
 g2D)
```

```
 flatten_1 (Flatten)          (None, 115200)          0

 dense_2 (Dense)              (None, 512)             58982912

 dropout_1 (Dropout)          (None, 512)             0

 dense_3 (Dense)              (None, 1)               513

=================================================================
Total params: 59076673 (225.36 MB)
Trainable params: 59076673 (225.36 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Training the Model

We train the neural network using the training dataset, and validate it with a validation split from the training data.

```
history = model.fit(X_train, y_train, epochs=10, validation_split=0.2,
batch_size=32)

Epoch 1/10
3/3 [==============================] - 2s 649ms/step - loss: 0.2390 -
accuracy: 0.8977 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 2/10
3/3 [==============================] - 2s 581ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy:
1.0000
Epoch 3/10
3/3 [==============================] - 2s 564ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy:
1.0000
Epoch 4/10
3/3 [==============================] - 2s 578ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy:
1.0000
Epoch 5/10
3/3 [==============================] - 2s 581ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy:
1.0000
Epoch 6/10
3/3 [==============================] - 2s 579ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy:
1.0000
```

```
Epoch 7/10
3/3 [==============================] - 2s 594ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy:
1.0000
Epoch 8/10
3/3 [==============================] - 2s 580ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy:
1.0000
Epoch 9/10
3/3 [==============================] - 2s 573ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy:
1.0000
Epoch 10/10
3/3 [==============================] - 2s 590ms/step - loss:
0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy:
1.0000
```

Evaluating the Model

After training, we evaluate the model's performance using the testing dataset and various performance metrics.

```
# Evaluate on the test set
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test accuracy: {test_accuracy * 100:.2f}%")

# Classification report
y_pred = (model.predict(X_test) > 0.5).astype("int32")
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
1/1 [==============================] - 0s 156ms/step - loss:
0.0000e+00 - accuracy: 1.0000
Test accuracy: 100.00%
1/1 [==============================] - 0s 187ms/step
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        28

    accuracy                           1.00        28
   macro avg       1.00      1.00      1.00        28
weighted avg       1.00      1.00      1.00        28
```

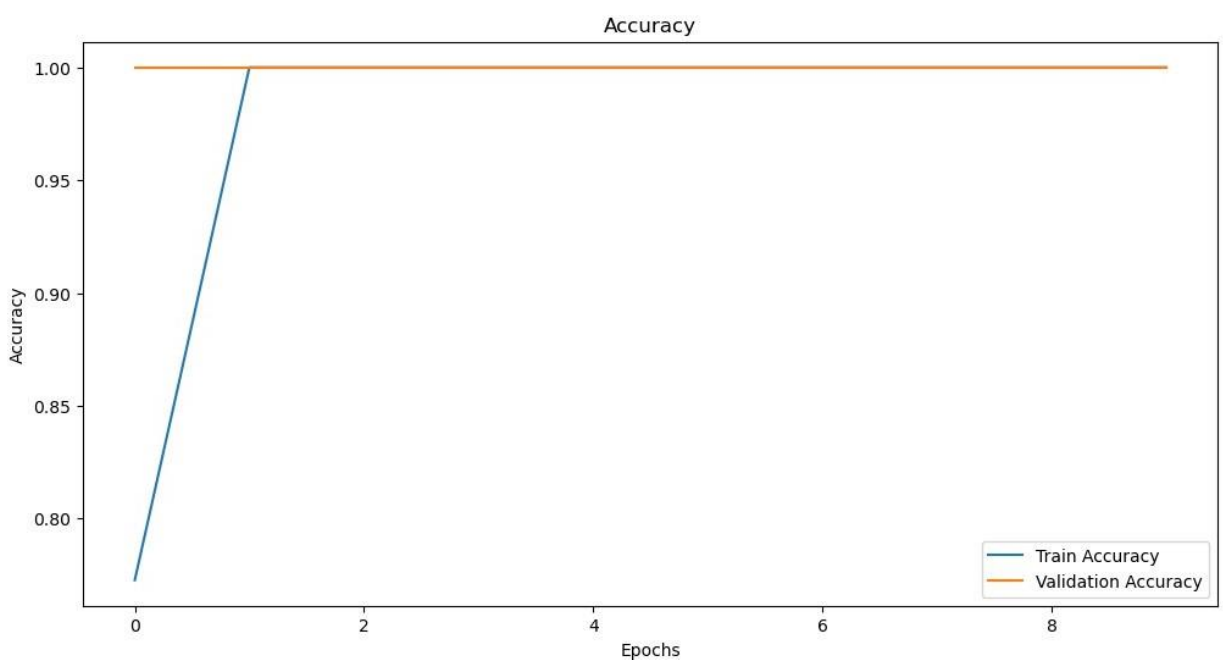
Visualizing Results

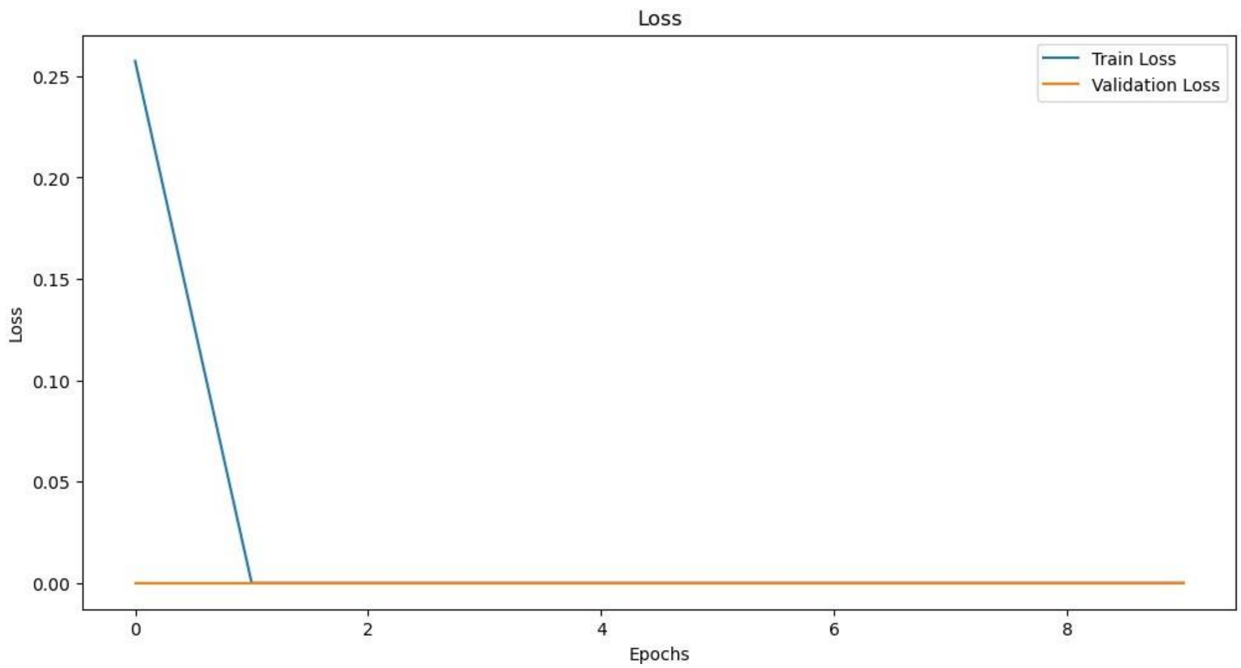
We visualize the training and validation accuracy and loss over the epochs to understand the model's performance.

```
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.figure(figsize=(12, 6))
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
plt.figure(figsize=(12, 6))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




Loss
Train Loss
Validation Loss
0.25
0.20
0.15
0.10
0.05
0.00
Loss
0
2
4
6
8
Epochs