Pneumonia Detection Using Deep learning

Project Overview

This project aims to develop a deep learning model for pneumonia detection using the TensorFlow and Keras frameworks. Pneumonia is a serious respiratory infection, and early detection is crucial for effective treatment. The deep learning model will be trained on a dataset of chest X-ray images to classify whether a patient has pneumonia or not.

Table of Contents

Code Snippets

Step: 1 Import the necessary libraries.

```
!pip install tensorflow
import numpy as np
import pandas as pd
```

```
import PIL #python image library used for image operations
import tensorflow as tf  # deeplearning for neural networks
import matplotlib.pyplot as plt
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
#ImageDataGenerator is used to split the data into 3 types
#training, validation, and test sets
```

Step: 2 Rescaling is applied to normalize pixel values (Train data).

```
train_dir=r"C:\Users\Ravikrishna J\OneDrive\Desktop\Documents\sign language\data\chest_xray\chest_xray\train"
train_generator=ImageDataGenerator(rescale=1/225)
```

This rescaling is applied to normalize pixel values.

Neural networks often work better with input data that is in a small, normalized range.

Dividing by 255 scales the pixel values to be between 0 and 1.

```
data_train=train_generator.flow_from_directory(train_dir,target_size=(120,120),batch_size=8,class_mode="binary")
```

is a convenient way to organize and process your image dataset for training a neural network.

It takes care of loading images, resizing them, creating batches, and even applying data augmentation,

making it easier to train a model on your image data.

```
Found 5216 images belonging to 2 classes.
```

Step: 3 Rescaling is applied to normalize pixel values (Validation data).

```
val_dir=r"C:\Users\Ravikrishna J\OneDrive\Desktop\Documents\sign language\data\chest_xray\val"
val_generator=ImageDataGenerator(rescale=1/225)

data_valid=val_generator.flow_from_directory(val_dir,target_size=(120,120),batch_size=8,class_mode="binary")
```

```
Found 16 images belonging to 2 classes.
```

Step: 4 Rescaling is applied to normalize pixel values (Test data).

```
test_dir=r"C:\Users\Ravikrishna J\OneDrive\Desktop\Documents\sign language\data\chest_xray\test"
test_generator=ImageDataGenerator(rescale=1/225)

data_test=test_generator.flow_from_directory(test_dir,target_size=(120,120),batch_size=8,class_mode="binary")
```

```
Found 624 images belonging to 2 classes.
```

Step: 5 CNN for Image classification(Building the model architecture).

```
#CNN
model = tf.keras.Sequential([tf.keras.layers.Conv2D(32,(3,3),input_shape=(120,120,3),activation="relu"),
                             tf.keras.layers.MaxPooling2D(2,2),
                             tf.keras.layers.Conv2D(64,(3,3),activation="relu"),
                             tf.keras.layers.MaxPooling2D(2,2),
                             tf.keras.layers.Conv2D(128,(3,3),activation="relu"),
                             tf.keras.layers.MaxPooling2D(2,2),
                             tf.keras.layers.Conv2D(256,(3,3),activation="relu"),
                             tf.keras.layers.MaxPooling2D(2,2),
                              tf.keras.layers.Conv2D(512,(3,3),activation="relu"),
                             tf.keras.layers.MaxPooling2D(2,2),
                              tf.keras.layers.Flatten(),
                              tf.keras.layers.Dense(1,activation="sigmoid")
                             ])
```

Explanation:

1. **Sequential Model:**
    - **tf.keras.Sequential**: This creates a linear stack of layers where you can add layers one by one.
2. **Convolutional Layers:**
    - **tf.keras.layers.Conv2D(32, (3, 3), input_shape=(120, 120, 3), activation="relu")**: This is a convolutional layer with 32 filters, each of size 3x3. The input shape is set to (120, 120, 3) indicating images with a size of 120x120 pixels and 3 color channels (RGB). ReLU (Rectified Linear Unit) is used as the activation function.
3. **MaxPooling Layers:**
    - **tf.keras.layers.MaxPooling2D(2, 2)**: This layer performs max pooling with a pool size of 2x2. It reduces the spatial dimensions of the representation, helping to reduce computation in the network.
4. **Flatten Layer:**
    - **tf.keras.layers.Flatten()**: This layer flattens the input, transforming it into a one-dimensional array. It prepares the data for the fully connected layers.

5. **Dense Layer:**
    - **tf.keras.layers.Dense(1, activation="sigmoid")**: This is the output layer with a single neuron using the sigmoid activation function. For binary classification tasks, like pneumonia detection (1 or 0), a sigmoid activation is common.

Why This Structure?

- **Convolutional Layers:** These layers learn hierarchical features in the input images, capturing patterns of increasing complexity.
- **MaxPooling Layers:** They reduce the spatial dimensions, focusing on the most important information while decreasing computational requirements.
- **Flatten Layer:** Converts the multi-dimensional data into a one-dimensional array to feed into the fully connected layers.
- **Dense Layer:** Produces the final classification output using a sigmoid activation for binary classification.

Step 6: Summary of the model.

```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_15 (Conv2D)          (None, 118, 118, 32)      34592

 max_pooling2d_14 (MaxPooli  (None, 59, 59, 32)        0
 ng2D)

 conv2d_16 (Conv2D)          (None, 57, 57, 64)        18496

 max_pooling2d_15 (MaxPooli  (None, 28, 28, 64)        0
 ng2D)

 conv2d_17 (Conv2D)          (None, 26, 26, 128)       73856

 max_pooling2d_16 (MaxPooli  (None, 13, 13, 128)       0
 ng2D)

 conv2d_18 (Conv2D)          (None, 11, 11, 256)       295168

 max_pooling2d_17 (MaxPooli  (None, 5, 5, 256)         0
 ng2D)

 flatten_3 (Flatten)         (None, 6400)              0

 dense_2 (Dense)             (None, 1)                 6401
```

```
=================================================================
Total params: 428513 (1.63 MB)
Trainable params: 428513 (1.63 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

- **Model:** "sequential_3"
    - The name assigned to the model.
- **Layers:**
    - Conv2D Layers: Four convolutional layers with 32, 64, 128, and 256 filters, respectively. Each convolutional layer applies a 3x3 filter.
    - MaxPooling2D Layers: Four max-pooling layers, each with a 2x2 pool size, reducing the spatial dimensions.
    - Flatten Layer: Flattens the output from the previous layer into a 1D array.
    - Dense Layer: Final dense layer with 1 neuron using a sigmoid activation for binary classification.
- **Output Shape:**
    - The shape of the output after each layer. For example, the first convolutional layer outputs (None, 118, 118, 32), indicating 32 feature maps of size 118x118.
- **Param #:**
    - The number of parameters in each layer. Parameters are the weights and biases that the model learns during training.
- **Total params:**
    - The total number of trainable parameters in the model.
- **Trainable params:**
    - The number of parameters that will be updated during training.
- **Non-trainable params:**
    - Parameters that are not updated during training (e.g., fixed weights in a pre-trained model).

Step 7 : Compiling The Model

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), loss="binary_crossentropy", metrics=["acc"])
```

The compile method in Keras is used to configure the learning process of a model. It requires specifying three key components: the optimizer, the loss function, and the metrics used for evaluation.

1. **Optimizer:**
    - **optimizer=tf.keras.optimizers.Adam(learning_rate=0.00)**: Adam is an optimization algorithm commonly used for training deep learning models. It adapts the learning rates of each parameter individually, which can be beneficial for training. The **learning_rate** parameter sets the step size at each iteration during optimization.
2. **Loss Function:**
    - **loss="binary_crossentropy"**: Binary Cross entropy is a loss function suitable for binary classification problems. In the context of pneumonia detection (1 or 0), binary crossentropy measures the difference between the true labels and the predicted probabilities. It is a common choice for binary classification tasks.
3. **Metrics:**
    - **metrics=["acc"]**: This specifies the metric(s) used to monitor the training and evaluation of the model. In this case, "acc" stands for accuracy. Accuracy is a commonly used metric for classification problems and represents the percentage of correctly classified samples.

Summary:

- **Optimizer:** Adam optimizer with a learning rate of 0.001.
- **Loss Function:** Binary Crossentropy, suitable for binary classification.
- **Metrics:** Accuracy, which is the percentage of correctly classified samples.

Step 8: Fit the model

```
history = model.fit_generator(data_train, epochs=2, validation_data=data_valid)
```

```
C:\Users\Ravikrishna J\AppData\Local\Temp\ipykernel_7244\3227875826.py:1: UserWarning: `Model.fit_generator` is deprecated and
will be removed in a future version. Please use `Model.fit`, which supports generators.
  history = model.fit_generator(data_train, epochs=2, validation_data=data_valid)
```

```
Epoch 1/2
652/652 [==============================] - 327s 498ms/step - loss: 0.4654 - acc: 0.8014 - val_loss: 1.8723 - val_acc: 0.6250
Epoch 2/2
652/652 [==============================] - 138s 212ms/step - loss: 0.1841 - acc: 0.9283 - val_loss: 0.5237 - val_acc: 0.7500
```

The fit_generator method is used in Keras to train a model using Python data generators. It's often employed when dealing with large datasets that cannot fit into memory and require on-the-fly data augmentation or preprocessing.

1. **data_train:**
    - This is the training data generator. It generates batches of training data on-the-fly during training. The generator is responsible for providing the model with the input data and corresponding labels.
2. **epochs=2:**
    - The number of epochs defines how many times the model will iterate over the entire training dataset. In this case, the model will be trained for 2 epochs.
3. **validation_data=data_valid:**
    - **data_valid** is the validation data generator. Similar to the training data generator, it provides batches of validation data during training. The model's performance on the validation set is monitored to assess generalization.
4. **history:**
    - The **fit_generator** method returns a **History** object, which contains information about the training process. It includes training and validation loss and metrics at each epoch.

Output:

Epoch 1/2

- **652/652**: This indicates that there are 652 batches of training data in total for this epoch.
- **[==============================]**: The progress bar showing the progress through the batches.
- **327s 498ms/step**: The time taken for each training step (batch). In this case, it took approximately 327 seconds and 498 milliseconds for each step.
- **loss: 0.4654**: The training loss for this epoch is 0.4654.
- **acc: 0.8014**: The training accuracy for this epoch is 80.14%.
- **val_loss: 1.8723**: The validation loss for this epoch is 1.8723.
- **val_acc: 0.6250**: The validation accuracy for this epoch is 62.50%.

Epoch 2/2

- Similar to the first epoch, this section provides information about the second epoch.
- **652/652**: There are again 652 batches of training data.
- **138s 212ms/step**: The time taken for each training step is now 138 seconds and 212 milliseconds.
- **loss: 0.1841**: The training loss for this epoch is 0.1841.
- **acc: 0.9283**: The training accuracy for this epoch is 92.83%.
- **val_loss: 0.5237**: The validation loss for this epoch is 0.5237.
- **val_acc: 0.7500**: The validation accuracy for this epoch is 75.00%.

Training Loss and Accuracy:

- The training loss decreases from 0.4654 to 0.1841 over the two epochs, indicating that the model is improving its ability to fit the training data.
- The training accuracy increases from 80.14% to 92.83%, indicating that the model is getting better at correctly classifying the training data.

- **Validation Loss and Accuracy:**
  - The validation loss decreases from 1.8723 to 0.5237, suggesting that the model is generalizing well to unseen data.
  - The validation accuracy increases from 62.50% to 75.00%, which also indicates improvement in the model's ability to generalize.
- **Time per Step:**
  - The time taken per training step decreases from approximately 327 seconds to 138 seconds, likely due to optimizations or the model becoming more efficient as training progresses.

This training log provides insights into how well the model is learning from the training data and generalizing to the validation set over the specified epochs. It's important to monitor both training and validation metrics to understand the model's performance.

Step 9: Evaluate

```
model.evaluate(data_test)
```

```
78/78 [==============================] - 29s 371ms/step - loss: 0.5386 - acc: 0.7772
[0.5386249423027039, 0.7772436141967773]
```

Explanation:

- **78/78:**
  - This indicates that there are 78 batches in total for the test dataset.

29s 371ms/step:

- The time taken for each evaluation step (batch). In this case, it took approximately 29 seconds and 371 milliseconds for each step.

- loss: 0.5386:
  - The overall loss on the test dataset is 0.5386.
- acc: 0.7772:
  - The overall accuracy on the test dataset is 77.72%.
- [0.5386249423027039, 0.7772436141967773]:
  - The evaluate method returns a list containing the test loss and test accuracy. In this case, the list is [0.5386249423027039, 0.7772436141967773].

Interpretation:

- The test loss of 0.5386 indicates how well the model generalizes to unseen data. Lower values are generally better.
- The test accuracy of 77.72% shows the percentage of correctly classified samples on the test dataset.

These results give an indication of how well your trained model performs on data it has never seen before. It's common practice to evaluate a model on a separate test set to assess its generalization performance and avoid overfitting to the training data. If the test metrics are consistent with or close to the validation metrics, it suggests that the model has generalized well.

Step 10: Predict

```
predictions=model.predict(data_test)
```

```
78/78 [==============================] - 11s 141ms/step
```

predictions = model.predict(data_test) is used to generate predictions on the test dataset using your trained model.

- model.predict:
    - This method is used to obtain predictions from the model. Given input data, the model outputs predictions for each sample.
- data_test:
    - This presumably refers to the test dataset or test data generator. It contains a set of input samples that the model will make predictions on.
- predictions:
    - This variable will store the model's predictions on the test dataset.

Usage:

Once you run this line of code, the variable predictions will contain the output of the model on the test dataset. Each element in predictions corresponds to the model's prediction for a particular sample in the test set.

Step 11: Testing

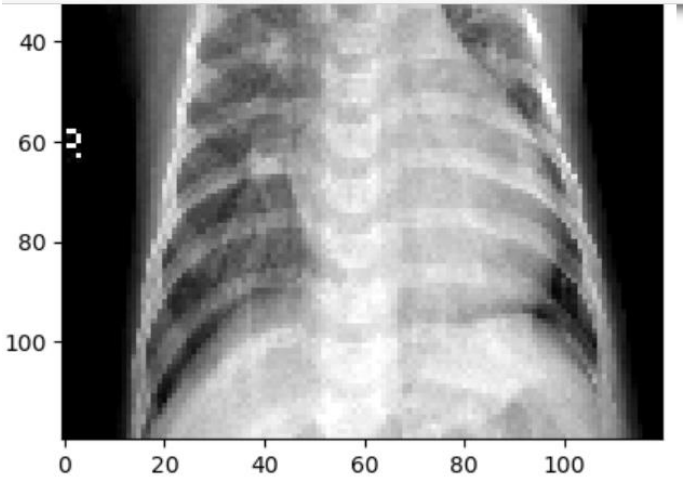
```
x=data_test.next()
for i in range(0,1):
    image=x[i]
    for j in range(0,8):
        plt.imshow(image[j])
        plt.show()
        print("The probability of pneumonia is:",predictions[j])
```

1. x = data_test.next():
    - This line retrieves the next batch of data from your test dataset generator (data_test). The variable x now contains a batch of input samples along with their corresponding labels.
2. for i in range(0, 1):
    - This loop iterates over the batches in x. Since you've specified range(0, 1), it will only execute once.
3. image = x[i]:
    - This line extracts the input images from the batch. image is now a set of images from the test dataset.
4. for j in range(0, 8):
    - This loop iterates over a subset of the images (8 images) in the current batch.
5. plt.imshow(image[j]):
    - This line uses Matplotlib to display the j-th image in the subset.
6. plt.show():
    - This displays the current image.
7. print("The probability of pneumonia is:", predictions[j]):
    - This prints the predicted probability of pneumonia for the j-th image in the subset. It assumes that predictions is an

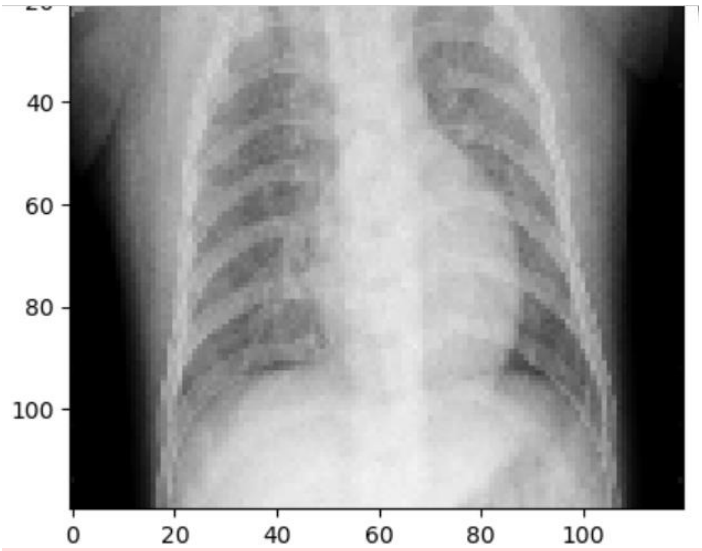
array containing the model's predictions for the entire test set.

Interpretation:

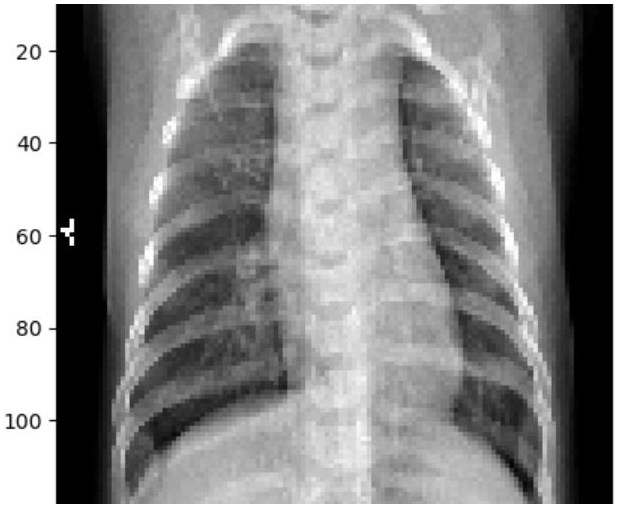
The code essentially shows a set of images from the test dataset along with their predicted probabilities of being pneumonia. This can be useful for visually inspecting how well the model is performing. The predicted probabilities give an indication of the model's confidence in its predictions for each image.




```
The probability of pneumonia is: [0.97993]
```

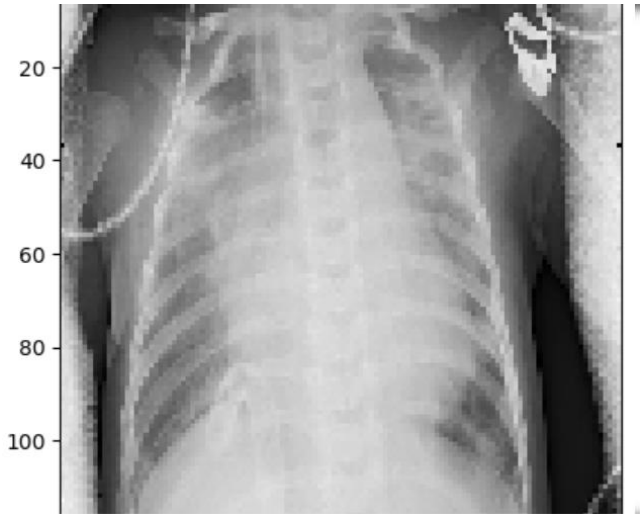



```
The probability of pneumonia is: [0.23940499]
```

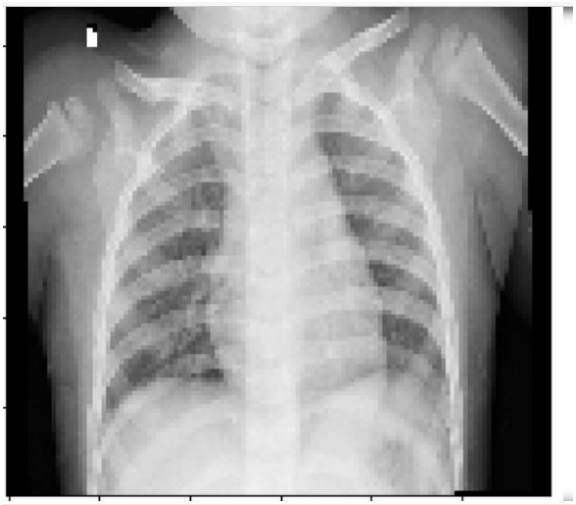



```
The probability of pneumonia is: [0.99938256]
```

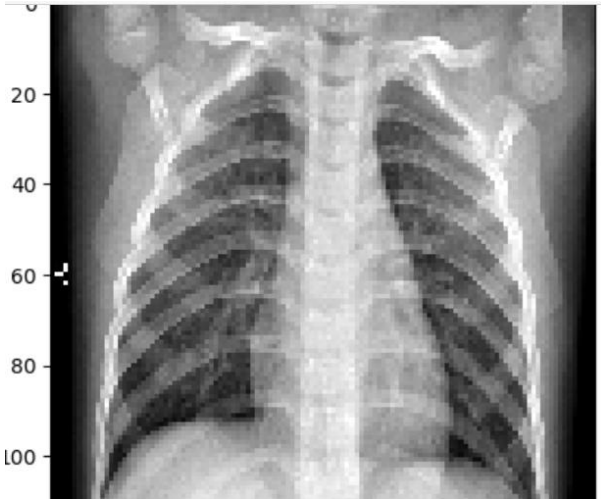

```
The probability of pneumonia is: [0.6490302]
```




```
The probability of pneumonia is: [0.52377707]
```



```
The probability of pneumonia is: [0.9999328]
```




```
The probability of pneumonia is: [0.9967267]
```

5. **Results**
    - Training and validation curves
    - Confusion matrix
    - Accuracy, precision, recall, and F1-score
6. **Conclusion**
    - Summary of findings
    - Future improvements
7. **References**

- **Datasets:**

Chest X-Ray Images (Pneumonia) on Kaggle: A popular dataset for pneumonia detection.

- **Tutorials and Documentation:**TensorFlow Tutorials: Official TensorFlow tutorials covering a wide range of topics, including image classification.
- **Keras Documentation:** Official Keras documentation provides detailed information on various layers, models, and utilities.
- **Transfer Learning with TensorFlow Hub:** Learn how to perform transfer learning using pre-trained models.
- **Medical Imaging in Deep Learning:**Deep Learning in Medical Imaging: A comprehensive review of deep learning applications in medical imaging.
- **Research Papers: CheXNet:** Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning: A research paper on pneumonia detection using deep learning.

- **Books:** “Deep Learning with Python" by François Chollet: A book by the creator of Keras, covering deep learning fundamentals with practical examples.

"Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" by Aurélien Géron: A comprehensive guide to machine learning and deep learning.

- **Online Courses:**

1. **Coursera** - TensorFlow for Deep Learning: A specialization by TensorFlow on Coursera covering various aspects of deep learning.
2. **Udacity -** Deep Learning with TensorFlow: A deep learning course using TensorFlow on Udacity.

- **GitHub Repositories:**

1. **GitHub - TensorFlow Models:** Official TensorFlow models repository containing implementations of various deep learning models.
2. **GitHub - Keras Applications:** Pre-trained models for Keras, including popular architectures like VGG16, ResNet, etc.

- **Community Forums:**TensorFlow Community: A place to ask questions, share knowledge, and connect with other TensorFlow users.

- **Kaggle Discussions:** Kaggle forums often have discussions on various machine learning and deep learning topics.

Conclusion

This documentation provides a comprehensive guide to implementing pneumonia detection sing deep learning with TensorFlow and Keras. Feel free to customize the code and parameters based on your specific requirements and dataset characteristics. Regularly update the model with new data to enhance its performance and stay informed about the latest advancements in deep learning for medical image analysis.