

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you creat
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

 [Show hidden output](#)

```
import os
import numpy as np
import tensorflow as tf
import cv2
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

dataset_path = "/kaggle/input/chest-xray-pneumonia/chest_xray/"
train_path = os.path.join(dataset_path, "train")
test_path = os.path.join(dataset_path, "test")
val_path = os.path.join(dataset_path, "val")

IMG_SIZE = 128
NUM_CLASSES = 2

def load_images_from_directory(directory, label):
    images, labels = [], []
    for category in os.listdir(directory):
        category_path = os.path.join(directory, category)
        category_label = 0 if category == "NORMAL" else 1
        for filename in os.listdir(category_path):
            img_path = os.path.join(category_path, filename)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            if img is not None:
                img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
                images.append(img)
                labels.append(category_label)
    return np.array(images) / 255.0, to_categorical(labels, NUM_CLASSES)

# Load train, validation, and test data
X_train, y_train = load_images_from_directory(train_path, NUM_CLASSES)
X_val, y_val = load_images_from_directory(val_path, NUM_CLASSES)
X_test, y_test = load_images_from_directory(test_path, NUM_CLASSES)


# # Reshape for fully connected network
# X_train = X_train.reshape(len(X_train), IMG_SIZE * IMG_SIZE)
# X_val = X_val.reshape(len(X_val), IMG_SIZE * IMG_SIZE)
```

```
# X_test = X_test.reshape(len(X_test), IMG_SIZE * IMG_SIZE)
```

```
# Define the model
```

```
model = Sequential([
    Flatten(input_shape=(IMG_SIZE, IMG_SIZE)), # Automatically flattens each image
    Dense(512, activation="relu", input_shape=(IMG_SIZE * IMG_SIZE,)),
    Dense(256, activation="relu"),
    Dense(128, activation="relu"),
    Dense(NUM_CLASSES, activation="sigmoid")
])
```

```
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
```

```
 /usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an
super().__init__(**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# Train the model
```

```
with tf.device('/GPU:0'):
    history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=100)
```

```
# Evaluate on test data
```

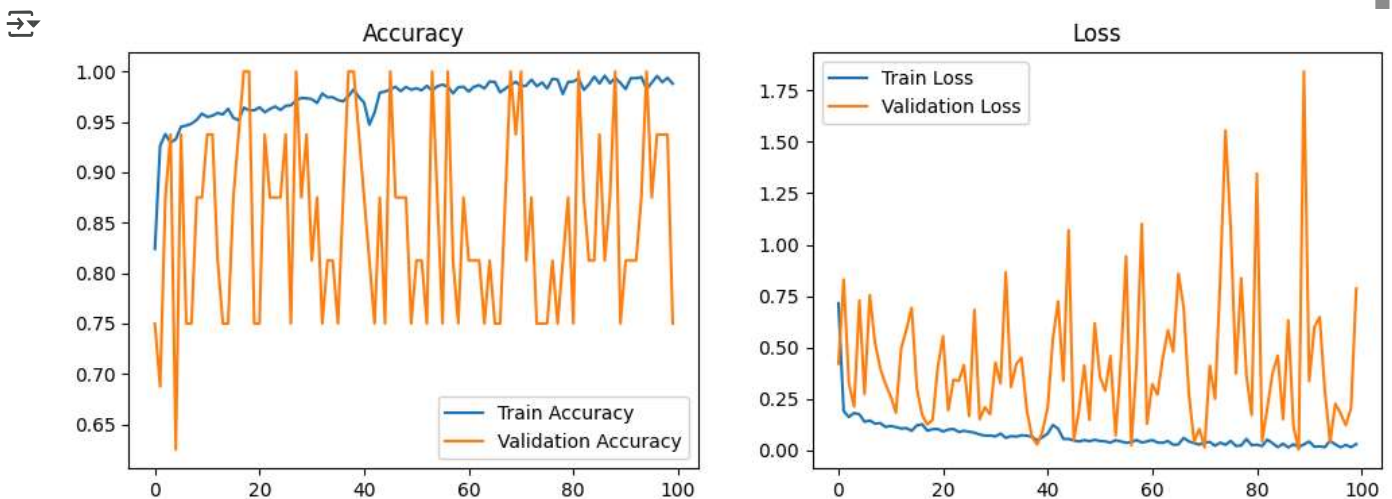
```
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc:.4f}")
```



```
Epoch 53/100
163/163 ————— 0s 3ms/step - accuracy: 0.9846 - loss: 0.0427 - val_accuracy: 0.7500 - val_loss: 0.8750
Epoch 54/100
163/163 ————— 0s 3ms/step - accuracy: 0.9845 - loss: 0.0399 - val_accuracy: 1.0000 - val_loss: 0.8750
Epoch 55/100
163/163 ————— 0s 3ms/step - accuracy: 0.9819 - loss: 0.0536 - val_accuracy: 0.8750 - val_loss: 0.8750
Epoch 56/100
163/163 ————— 0s 3ms/step - accuracy: 0.9812 - loss: 0.0550 - val_accuracy: 0.7500 - val_loss: 0.8750
Epoch 57/100
163/163 ————— 0s 3ms/step - accuracy: 0.9837 - loss: 0.0424 - val_accuracy: 1.0000 - val_loss: 0.8750
Epoch 58/100
163/163 ————— 0s 3ms/step - accuracy: 0.9788 - loss: 0.0485 - val_accuracy: 0.8125 - val_loss: 0.8750
Epoch 59/100
163/163 ————— 0s 3ms/step - accuracy: 0.9832 - loss: 0.0410 - val_accuracy: 0.7500 - val_loss: 0.8750
Epoch 60/100
163/163 ————— 0s 3ms/step - accuracy: 0.9776 - loss: 0.0591 - val_accuracy: 0.8750 - val_loss: 0.8750
Epoch 61/100
163/163 ————— 0s 3ms/step - accuracy: 0.9812 - loss: 0.0424 - val_accuracy: 1.0000 - val_loss: 0.8750
```

```
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history["accuracy"], label="Train Accuracy")
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
plt.legend()
plt.title("Accuracy")
```

```
plt.subplot(1, 2, 2)
plt.plot(history.history["loss"], label="Train Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.legend()
plt.title("Loss")
plt.show()
```





```
# Save the model
model.save("pneumonia_ann_model.h5")
```

```
# Evaluate on test data
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc:.4f}")
```

```
20/20 ————— 0s 2ms/step - accuracy: 0.9109 - loss: 0.8917
Test Accuracy: 0.6971
```

```
# Evaluate on validation data
val_loss, val_acc = model.evaluate(X_val, y_val)
print(f"Validation Accuracy: {val_acc:.4f}")
```

 1/1  0s 21ms/step - accuracy: 0.7500 - loss: 0.7877  
Validation Accuracy: 0.7500

Start coding or [generate](#) with AI.