Diagnostic AI: Pediatric Pneumonia Detection Research

Pediatric Pneumonia Detection using Deep Learning Objective: The primary goal of this research project is to develop a high-precision Convolutional Neural Network (CNN) model capable of identifying Pneumonia from pediatric chest X-ray images. By leveraging a balanced dataset and advanced image preprocessing, we aim to minimize diagnostic errors and assist medical professionals in early detection.

📂 Step 1 — Dataset Download using KaggleHub

In this step, we download the Pediatric Chest X-ray Pneumonia dataset directly from Kaggle using KaggleHub. This method avoids manual upload and automatically stores the dataset in the Colab environment for further processing.

```
!pip install kagglehub

import kagglehub

path = kagglehub.dataset_download(
    "yusufmurtaza01/pediatric-chest-xray-pneumonia-balanced-dataset"
)

print("Dataset Path:", path)
```

```
Requirement already satisfied: kagglehub in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: pyyaml in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/d
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/d
Warning: Looks like you're using an outdated `kagglehub` version (installed: 0.3
Downloading from https://www.kaggle.com/api/v1/datasets/download/yusufmurtaza01/
100%|██████████| 1.66G/1.66G [00:17<00:00, 105MB/s]Extracting files...

Dataset Path: /root/.cache/kagglehub/datasets/yusufmurtaza01/pediatric-chest-xra
```

Step 2 — Understanding Dataset Structure

Here, we explore the dataset directory to verify its structure. We check whether it contains Train, Validation, and Test folders along with their respective class subfolders (NORMAL and PNEUMONIA).

```
import os

os.listdir(path)
```

```
['NORMAL', 'PNEUMONIA']
```

Step 2.1 — Exploring Complete Dataset Directory

In this step, we explore the full directory tree of the downloaded dataset to understand its internal folder hierarchy. This helps us identify whether the dataset is already split into training, validation, and testing sets or if we need to perform the split manually.

```
for root, dirs, files in os.walk(path):
    print("Root:", root)
    print("Folders:", dirs)
    print("Files count:", len(files))
    print("-"*40)
```

```
Root: /root/.cache/kagglehub/datasets/yusufmurtaza01/pediatric-chest-xray-pneumc
Folders: ['NORMAL', 'PNEUMONIA']
Files count: 0
----------------------------------------
Root: /root/.cache/kagglehub/datasets/yusufmurtaza01/pediatric-chest-xray-pneumc
Folders: []
Files count: 4265
----------------------------------------
Root: /root/.cache/kagglehub/datasets/yusufmurtaza01/pediatric-chest-xray-pneumc
Folders: []
Files count: 4265
----------------------------------------
```

Step 3 — Dataset Splitting (Train / Validation / Test)

In this step, we split the dataset into three subsets: Training, Validation, and Testing.

Since the dataset is not pre-split, we manually divide the images into:

- 70% Training set

- 15% Validation set
- 15% Testing set

This ensures that the model is trained on one portion of the data and evaluated on unseen data for fair performance assessment.

```
import os
import shutil
import random
```

📂 Creating Directory Structure for Split Data

Here, we create a new directory structure to store the split datasets. Each subset (Train, Validation, Test) will contain two class folders: NORMAL and PNEUMONIA.

```
base_dir = "/content/chest_xray_split"

folders = [
    "train/NORMAL",
    "train/PNEUMONIA",
    "val/NORMAL",
    "val/PNEUMONIA",
    "test/NORMAL",
    "test/PNEUMONIA"
]

for folder in folders:
    os.makedirs(os.path.join(base_dir, folder), exist_ok=True)
```

🔀 Splitting Images into Train, Validation, and Test Sets

In this step, we randomly shuffle the images and divide them into training, validation, and testing subsets according to the defined ratios.

```
def split_data(source, train, val, test, split_ratio=(0.7, 0.15, 0.15)):

    images = os.listdir(source)
    random.shuffle(images)

    total = len(images)
    train_end = int(split_ratio[0] * total)
    val_end = int((split_ratio[0] + split_ratio[1]) * total)
```

```
    train_imgs = images[:train_end]
    val_imgs = images[train_end:val_end]
    test_imgs = images[val_end:]

    for img in train_imgs:
        shutil.copy(os.path.join(source, img), train)

    for img in val_imgs:
        shutil.copy(os.path.join(source, img), val)

    for img in test_imgs:
        shutil.copy(os.path.join(source, img), test)
```

🚀 Applying Dataset Splitting

Here, we apply the splitting function to both classes (Normal and Pneumonia) to generate the final dataset structure for model training and evaluation.

```
original_path = path

split_data(
    original_path + "/NORMAL",
    base_dir + "/train/NORMAL",
    base_dir + "/val/NORMAL",
    base_dir + "/test/NORMAL"
)

split_data(
    original_path + "/PNEUMONIA",
    base_dir + "/train/PNEUMONIA",
    base_dir + "/val/PNEUMONIA",
    base_dir + "/test/PNEUMONIA"
)

print("Dataset splitting completed ✅")
```

```
Dataset splitting completed ✅
```

📊 Verifying Dataset Split

In this step, we verify the number of images in each subset to ensure the dataset has been split correctly.

```
for root, dirs, files in os.walk(base_dir):
    print(root, "→", len(files))
```

```
/content/chest_xray_split → 0
/content/chest_xray_split/val → 0
/content/chest_xray_split/val/NORMAL → 640
/content/chest_xray_split/val/PNEUMONIA → 640
/content/chest_xray_split/train → 0
/content/chest_xray_split/train/NORMAL → 2985
/content/chest_xray_split/train/PNEUMONIA → 2985
/content/chest_xray_split/test → 0
/content/chest_xray_split/test/NORMAL → 640
/content/chest_xray_split/test/PNEUMONIA → 640
```

🖼 Step 4 — Visualizing Sample Chest X-ray Images

In this step, we display sample images from both classes (Normal and Pneumonia) to visually understand the dataset.

This helps in identifying visible differences between healthy and infected lungs and ensures that images are loaded correctly before training the model.

```
import matplotlib.pyplot as plt
import cv2
import os
import random
```

🔍 Displaying Random Samples from Each Class

Here, we randomly select images from both categories and display them for visual comparison.

```
def show_samples(class_path, title):

    images = os.listdir(class_path)
    sample_images = random.sample(images, 4)

    plt.figure(figsize=(10,5))

    for i, img_name in enumerate(sample_images):
        img_path = os.path.join(class_path, img_name)
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        plt.subplot(1,4,i+1)
```

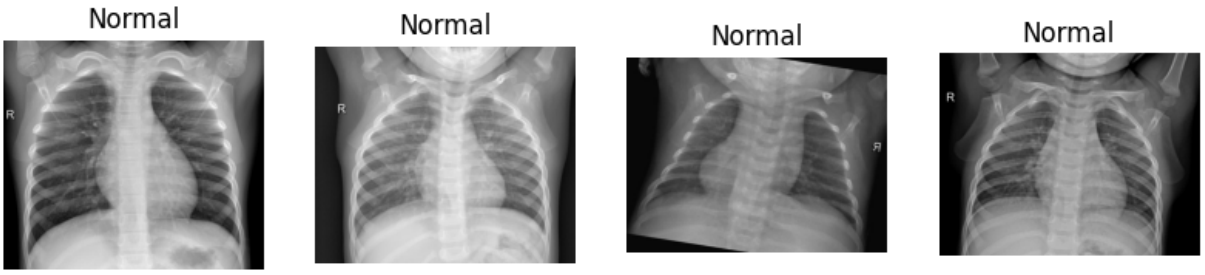
```
            plt.imshow(img)
            plt.title(title)
            plt.axis("off")

        plt.show()
```

Normal Chest X-ray Samples

These images represent healthy lungs without pneumonia infection.

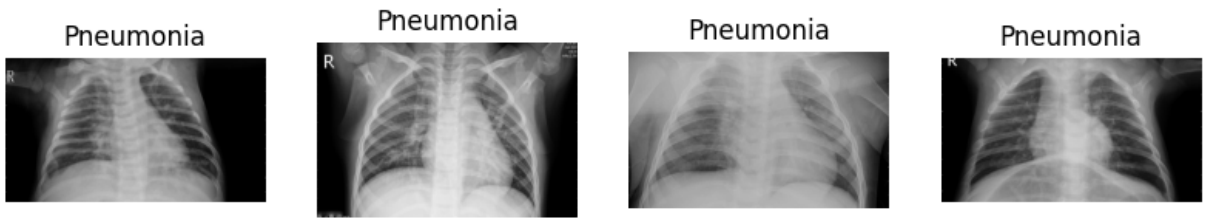
```
show_samples(
    base_dir + "/train/NORMAL",
    "Normal"
)
```



Pneumonia Chest X-ray Samples

These images represent lungs affected by pneumonia infection, showing visible opacities or abnormalities.

```
show_samples(
    base_dir + "/train/PNEUMONIA",
    "Pneumonia"
)
```



🧹 Step 5 — Image Preprocessing and Data Augmentation

In this step, we prepare the dataset for model training by applying preprocessing techniques.

The preprocessing includes:

- Rescaling pixel values (normalization)
- Resizing images to a fixed shape
- Applying augmentation techniques to the training set

Data augmentation helps improve model generalization and reduces overfitting by creating slightly modified variations of existing images.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

🔄 Training Data Generator with Augmentation

Here, we apply augmentation techniques such as rotation, zoom, and horizontal flipping to artificially expand the training dataset and improve model robustness.

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,
    zoom_range=0.1,
    horizontal_flip=True
)
```

📏 Validation and Test Data Preprocessing

For validation and testing datasets, we only apply rescaling.

No augmentation is used here to ensure fair evaluation on real, unmodified images.

```
val_test_datagen = ImageDataGenerator(
    rescale=1./255
)
```

📥 Step 6 — Loading Images Using Data Generators

In this step, we load images from the split dataset directories using Keras data generators.

The images are resized to 224×224 pixels and converted into batches for efficient model training.

```
img_size = (224, 224)
batch_size = 32
```

```
train_data = train_datagen.flow_from_directory(
    base_dir + "/train",
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary'
)
```

```
Found 5970 images belonging to 2 classes.
```

```
val_data = val_test_datagen.flow_from_directory(
    base_dir + "/val",
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary'
)
```

```
Found 1280 images belonging to 2 classes.
```

```
test_data = val_test_datagen.flow_from_directory(
    base_dir + "/test",
    target_size=img_size,
    batch_size=batch_size,
    class_mode='binary',
    shuffle=False
)
```

```
Found 1280 images belonging to 2 classes.
```

🔎 Checking Class Indices

This step verifies how class labels are encoded by the generator.

Typically:

- Normal → 0
- Pneumonia → 1

```
train_data.class_indices
```

```
{'NORMAL': 0, 'PNEUMONIA': 1}
```

🧠 Step 7 — Building Transfer Learning Model (DenseNet121)

In this step, we build a deep learning model using the DenseNet121 architecture with pretrained ImageNet weights.

Transfer learning allows us to leverage previously learned features from large datasets, improving performance on medical imaging tasks such as pneumonia detection.

```
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras import layers, models
```

📦 Loading Pretrained DenseNet121

Here, we load DenseNet121 without its top classification layer and use ImageNet pretrained weights for feature extraction.

```
base_model = DenseNet121(
    weights='imagenet',
    include_top=False,
    input_shape=(224,224,3)
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatic
29084464/29084464 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

❄ Freezing Base Model Layers

We freeze the convolutional base to prevent pretrained weights from updating during initial training. This helps retain learned low-level image features.

```
base_model.trainable = False
```

🏗 Adding Custom Classification Layers

We add new fully connected layers on top of the pretrained base model to adapt it for binary pneumonia classification.

```
x = layers.GlobalAveragePooling2D()(base_model.output)
x = layers.Dense(128, activation='relu')(x)
x = layers.Dropout(0.5)(x)
output = layers.Dense(1, activation='sigmoid')(x)

model = models.Model(
    inputs=base_model.input,
    outputs=output
)
```

⚙ Compiling the Model

Here, we configure the model for training using:

- Adam optimizer
- Binary crossentropy loss
- Accuracy as evaluation metric

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

```
model.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 224, 224, 3)	0	-
zero_padding2d (ZeroPadding2D)	(None, 230, 230, 3)	0	input_layer[0][0]
conv1_conv (Conv2D)	(None, 112, 112, 64)	9,408	zero_padding2d[0…
conv1_bn (BatchNormalizatio…	(None, 112, 112, 64)	256	conv1_conv[0][0]
conv1_relu (Activation)	(None, 112, 112, 64)	0	conv1_bn[0][0]
zero_padding2d_1 (ZeroPadding2D)	(None, 114, 114, 64)	0	conv1_relu[0][0]
pool1 (MaxPooling2D)	(None, 56, 56, 64)	0	zero_padding2d_1…
conv2_block1_0_bn (BatchNormalizatio…	(None, 56, 56, 64)	256	pool1[0][0]
conv2_block1_0_relu (Activation)	(None, 56, 56, 64)	0	conv2_block1_0_b…
conv2_block1_1_conv (Conv2D)	(None, 56, 56, 128)	8,192	conv2_block1_0_r…
conv2_block1_1_bn (BatchNormalizatio…	(None, 56, 56, 128)	512	conv2_block1_1_c…
conv2_block1_1_relu (Activation)	(None, 56, 56, 128)	0	conv2_block1_1_b…
conv2_block1_2_conv (Conv2D)	(None, 56, 56, 32)	36,864	conv2_block1_1_r…
conv2_block1_concat (Concatenate)	(None, 56, 56, 96)	0	pool1[0][0], conv2_block1_2_c…
conv2_block2_0_bn (BatchNormalizatio…	(None, 56, 56, 96)	384	conv2_block1_con…
conv2_block2_0_relu (Activation)	(None, 56, 56, 96)	0	conv2_block2_0_b…
conv2_block2_1_conv (Conv2D)	(None, 56, 56, 128)	12,288	conv2_block2_0_r…
conv2_block2_1_bn (BatchNormalizatio…	(None, 56, 56, 128)	512	conv2_block2_1_c…

conv2_block2_1_relu (Activation)	(None, 56, 56, 128)	0	conv2_block2_1_b…
conv2_block2_2_conv (Conv2D)	(None, 56, 56, 32)	36,864	conv2_block2_1_r…
conv2_block2_concat (Concatenate)	(None, 56, 56, 128)	0	conv2_block1_con… conv2_block2_2_c…
conv2_block3_0_bn (BatchNormalizatio…	(None, 56, 56, 128)	512	conv2_block2_con…
conv2_block3_0_relu (Activation)	(None, 56, 56, 128)	0	conv2_block3_0_b…
conv2_block3_1_conv (Conv2D)	(None, 56, 56, 128)	16,384	conv2_block3_0_r…
conv2_block3_1_bn (BatchNormalizatio…	(None, 56, 56, 128)	512	conv2_block3_1_c…
conv2_block3_1_relu (Activation)	(None, 56, 56, 128)	0	conv2_block3_1_b…
conv2_block3_2_conv (Conv2D)	(None, 56, 56, 32)	36,864	conv2_block3_1_r…
conv2_block3_concat (Concatenate)	(None, 56, 56, 160)	0	conv2_block2_con… conv2_block3_2_c…
conv2_block4_0_bn (BatchNormalizatio…	(None, 56, 56, 160)	640	conv2_block3_con…
conv2_block4_0_relu (Activation)	(None, 56, 56, 160)	0	conv2_block4_0_b…
conv2_block4_1_conv (Conv2D)	(None, 56, 56, 128)	20,480	conv2_block4_0_r…
conv2_block4_1_bn (BatchNormalizatio…	(None, 56, 56, 128)	512	conv2_block4_1_c…
conv2_block4_1_relu (Activation)	(None, 56, 56, 128)	0	conv2_block4_1_b…
conv2_block4_2_conv (Conv2D)	(None, 56, 56, 32)	36,864	conv2_block4_1_r…
conv2_block4_concat (Concatenate)	(None, 56, 56, 192)	0	conv2_block3_con… conv2_block4_2_c…
conv2_block5_0_bn (BatchNormalizatio…	(None, 56, 56, 192)	768	conv2_block4_con…
conv2_block5_0_relu (Activation)	(None, 56, 56, 192)	0	conv2_block5_0_b…

conv2_block5_1_conv (Conv2D)	(None, 56, 56, 128)	24,576	conv2_block5_0_r…
conv2_block5_1_bn (BatchNormalizatio…	(None, 56, 56, 128)	512	conv2_block5_1_c…
conv2_block5_1_relu (Activation)	(None, 56, 56, 128)	0	conv2_block5_1_b…
conv2_block5_2_conv (Conv2D)	(None, 56, 56, 32)	36,864	conv2_block5_1_r…
conv2_block5_concat (Concatenate)	(None, 56, 56, 224)	0	conv2_block4_con… conv2_block5_2_c…
conv2_block6_0_bn (BatchNormalizatio…	(None, 56, 56, 224)	896	conv2_block5_con…
conv2_block6_0_relu (Activation)	(None, 56, 56, 224)	0	conv2_block6_0_b…
conv2_block6_1_conv (Conv2D)	(None, 56, 56, 128)	28,672	conv2_block6_0_r…
conv2_block6_1_bn (BatchNormalizatio…	(None, 56, 56, 128)	512	conv2_block6_1_c…
conv2_block6_1_relu (Activation)	(None, 56, 56, 128)	0	conv2_block6_1_b…
conv2_block6_2_conv (Conv2D)	(None, 56, 56, 32)	36,864	conv2_block6_1_r…
conv2_block6_concat (Concatenate)	(None, 56, 56, 256)	0	conv2_block5_con… conv2_block6_2_c…
pool2_bn (BatchNormalizatio…	(None, 56, 56, 256)	1,024	conv2_block6_con…
pool2_relu (Activation)	(None, 56, 56, 256)	0	pool2_bn[0][0]
pool2_conv (Conv2D)	(None, 56, 56, 128)	32,768	pool2_relu[0][0]
pool2_pool (AveragePooling2D)	(None, 28, 28, 128)	0	pool2_conv[0][0]
conv3_block1_0_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	pool2_pool[0][0]
conv3_block1_0_relu (Activation)	(None, 28, 28, 128)	0	conv3_block1_0_b…
conv3_block1_1_conv (Conv2D)	(None, 28, 28, 128)	16,384	conv3_block1_0_r…
conv3_block1_1_bn	(None, 28, 28,	512	conv3_block1_1_c

(BatchNormalizatio…	128)		
conv3_block1_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_block1_1_b…
conv3_block1_2_conv (Conv2D)	(None, 28, 28, 32)	36,864	conv3_block1_1_r…
conv3_block1_concat (Concatenate)	(None, 28, 28, 160)	0	pool2_pool[0][0], conv3_block1_2_c…
conv3_block2_0_bn (BatchNormalizatio…	(None, 28, 28, 160)	640	conv3_block1_con…
conv3_block2_0_relu (Activation)	(None, 28, 28, 160)	0	conv3_block2_0_b…
conv3_block2_1_conv (Conv2D)	(None, 28, 28, 128)	20,480	conv3_block2_0_r…
conv3_block2_1_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_block2_1_c…
conv3_block2_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_block2_1_b…
conv3_block2_2_conv (Conv2D)	(None, 28, 28, 32)	36,864	conv3_block2_1_r…
conv3_block2_concat (Concatenate)	(None, 28, 28, 192)	0	conv3_block1_con… conv3_block2_2_c…
conv3_block3_0_bn (BatchNormalizatio…	(None, 28, 28, 192)	768	conv3_block2_con…
conv3_block3_0_relu (Activation)	(None, 28, 28, 192)	0	conv3_block3_0_b…
conv3_block3_1_conv (Conv2D)	(None, 28, 28, 128)	24,576	conv3_block3_0_r…
conv3_block3_1_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_block3_1_c…
conv3_block3_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_block3_1_b…
conv3_block3_2_conv (Conv2D)	(None, 28, 28, 32)	36,864	conv3_block3_1_r…
conv3_block3_concat (Concatenate)	(None, 28, 28, 224)	0	conv3_block2_con… conv3_block3_2_c…
conv3_block4_0_bn (BatchNormalizatio…	(None, 28, 28, 224)	896	conv3_block3_con…
conv3_block4_0_relu	(None, 28, 28,	0	conv3_block4_0_b…

(Activation)	224)		
conv3_block4_1_conv (Conv2D)	(None, 28, 28, 128)	28,672	conv3_block4_0_r…
conv3_block4_1_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_block4_1_c…
conv3_block4_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_block4_1_b…
conv3_block4_2_conv (Conv2D)	(None, 28, 28, 32)	36,864	conv3_block4_1_r…
conv3_block4_concat (Concatenate)	(None, 28, 28, 256)	0	conv3_block3_con… conv3_block4_2_c…
conv3_block5_0_bn (BatchNormalizatio…	(None, 28, 28, 256)	1,024	conv3_block4_con…
conv3_block5_0_relu (Activation)	(None, 28, 28, 256)	0	conv3_block5_0_b…
conv3_block5_1_conv (Conv2D)	(None, 28, 28, 128)	32,768	conv3_block5_0_r…
conv3_block5_1_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_block5_1_c…
conv3_block5_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_block5_1_b…
conv3_block5_2_conv (Conv2D)	(None, 28, 28, 32)	36,864	conv3_block5_1_r…
conv3_block5_concat (Concatenate)	(None, 28, 28, 288)	0	conv3_block4_con… conv3_block5_2_c…
conv3_block6_0_bn (BatchNormalizatio…	(None, 28, 28, 288)	1,152	conv3_block5_con…
conv3_block6_0_relu (Activation)	(None, 28, 28, 288)	0	conv3_block6_0_b…
conv3_block6_1_conv (Conv2D)	(None, 28, 28, 128)	36,864	conv3_block6_0_r…
conv3_block6_1_bn (BatchNormalizatio…	(None, 28, 28, 128)	512	conv3_block6_1_c…
conv3_block6_1_relu (Activation)	(None, 28, 28, 128)	0	conv3_block6_1_b…
conv3_block6_2_conv (Conv2D)	(None, 28, 28, 32)	36,864	conv3_block6_1_r…
conv3_block6_concat (Concatenate)	(None, 28, 28, 320)	0	conv3_block5_con… conv3_block6_2_c…