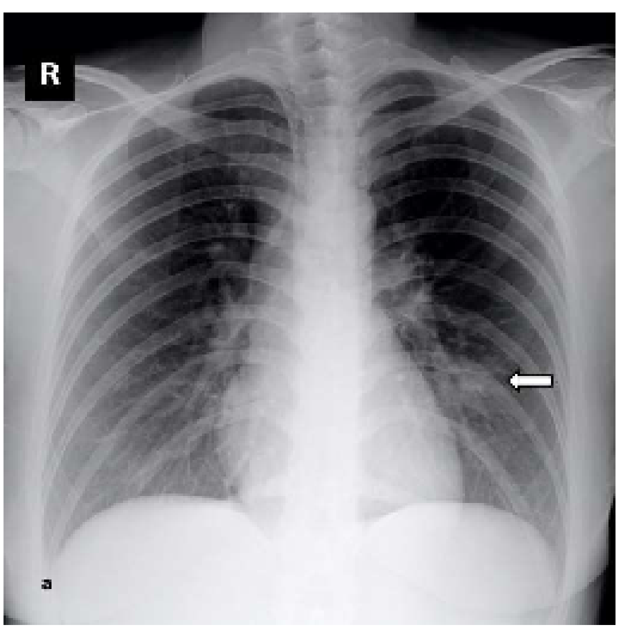
Nicholas Kellogg

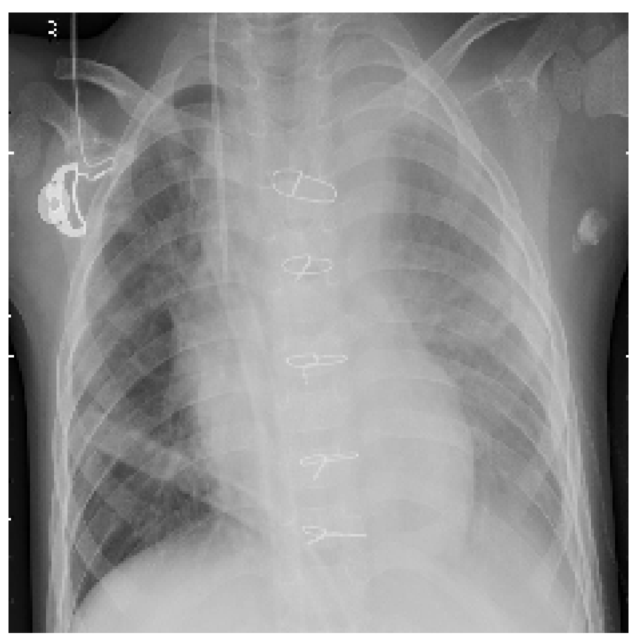
DS4002 - Blog Document

4/28/2025

COVID19: Getting Harder to Diagnose

In 2021, COVID-19 continued to sweep across the globe, overwhelming healthcare systems and leaving doctors racing to keep up. One of the major challenges was diagnosis: COVID-19 pneumonia often looked nearly identical to other forms of viral or bacterial pneumonia on chest x-rays. Subtle differences in opacity, texture, and lung involvement made it extremely difficult even for experienced radiologists to tell infections apart quickly. With hospitals under enormous strain, every missed or delayed diagnosis could mean the difference between life and death. The need for faster, more reliable diagnostic tools became one of the pandemic's most urgent medical frontiers.





DS4002 - Replication Process Summary

Goal: To build a convolutional neural network to successfully distinguish between medical imaging involving COVID-19 cases.

Dataset:https://myuva-my.sharepoint.com/:u:/g/personal/vtf6hv_virginia_edu/EWSXMJozHJ9Kvpso9X93aWoBOzRNgoTCPqoZtmKWjkJmJQ?e=X1QS2K

Simply download the dataset.

Step 1: Preprocessing

Image data was sourced from Mendeley Data, focusing on clear-resolution chest x-ray images, resulting in a dataset of approximately 5,400 images. Preprocessing included standardizing image dimensions, converting categorical labels ('COVID', 'Normal', etc.) into numerical values for analysis, and performing an 80/20 train-test split to prepare for model development.

Step 2: Methodology

The methodology involves constructing a convolutional neural network (CNN) using Keras. Initial steps include defining the model architecture, specifying input image dimensions, and applying dimensionality reduction. The model is then compiled with an optimizer and selected performance metrics. Training is conducted using the training dataset to generate a final CNN capable of classifying the x-ray images.

Quantifiable Goal: Model performance will be evaluated using a basic accuracy metric calculated as (TP + TN) / (TP + TN + FP + FN), where TP, TN, FP, and FN represent true positives, true negatives, false positives, and false negatives, respectively. The ultimate goal is to produce an efficient and reliable tool to assist in the rapid diagnosis of COVID-19 infections based on x-ray imaging (At or above 80% accuracy).

All Code

COVIDModelTrain.py

```
import os

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

import keras

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Sequential

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

from sklearn.metrics import classification_report, confusion_matrix

from sklearn.utils.class_weight import compute_class_weight


data_dir = "./Desktop/Spring 2025/DS Prototyping/Project3/COVID19-ImageDataset"


# kept parameters

img_width, img_height = 150, 150

batch_size = 32

epochs = 15


# 1. data preprocessing / validation

datagen_train = ImageDataGenerator(
```

```
    rescale=1./255,
    validation_split=0.2,  # 80% train, 20% validation
    rotation_range=15,
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)

datagen_val = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_generator = datagen_train.flow_from_directory(
    data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
)

validation_generator = datagen_val.flow_from_directory(
    data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
```

```
    class_mode='categorical',
    subset='validation'
)


# 2. CNN Model build (- had to switch to deeper network. takes ~ 20 minutes to run on Macbook Pro)
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(img_width, img_height, 3)),
    MaxPooling2D(2, 2),

    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),

    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),

    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),

    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
```

```
    Dense(3, activation='softmax')
])

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

# Update validation_generator to not shuffle — important for eval
validation_generator = datagen_val.flow_from_directory(
    data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation',
    shuffle=False
)

# Compute class weights for better balance
y_true = validation_generator.classes
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_true),
    y=y_true
```

```
)
class_weight_dict = dict(enumerate(class_weights))

# 3. Model train (longer, with class weights)
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size,
    epochs=40,
    class_weight=class_weight_dict
)

# 4. Model eval
val_loss, val_accuracy = model.evaluate(validation_generator)
print(f"Validation accuracy: {val_accuracy:.2f}")



def plot_training(history):
    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
```

```python
    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='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.legend()
    plt.title('Loss')

    plt.show()

plot_training(history)



# Continued eval: Confusion Matrix, Classification Report, Misclassified Samples


from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay


Y_pred = model.predict(validation_generator)
y_pred = np.argmax(Y_pred, axis=1)
y_true = validation_generator.classes
class_names = list(validation_generator.class_indices.keys())
```

```
# Classification report
print("\nClassification Report:")
print(classification_report(y_true, y_pred, target_names=class_names))

# Confusion matrix
cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
disp.plot(cmap='Blues')
plt.title("Confusion Matrix")
plt.show()

# misclassed images
misclassified_idxs = np.where(y_pred != y_true)[0]
print(f"\nMisclassified Images: {len(misclassified_idxs)} total")

for idx in misclassified_idxs[:5]:
    batch_index = idx // batch_size
    within_batch = idx % batch_size

    batch = validation_generator[batch_index]
    img_batch, _ = batch
    img_sample = img_batch[within_batch]
```

```
    plt.imshow(img_sample)

    plt.title(f"True: {class_names[y_true[idx]]}, Predicted: {class_names[y_pred[idx]]}")

    plt.axis('off')

    plt.show()
```

Eda_project3.py

```
import os

import random

import matplotlib.pyplot as plt

import seaborn as sns

import cv2

import numpy as np

from collections import Counter


DATASET_PATH = "./Desktop/Spring 2025/DS Prototyping/Project3/COVID19-ImageDataset"

# <- update if needed


classes = ['COVID', 'PNEUMONIA', 'NORMAL']


# Count Images

def count_images(dataset_path, classes):
```

```
    counts = {}
    for label in classes:
        path = os.path.join(dataset_path, label)
        counts[label] = len(os.listdir(path))
    return counts


# Image Shape Dimensions
def get_image_shapes(dataset_path, classes, num_samples=100):
    shapes = []
    for label in classes:
        path = os.path.join(dataset_path, label)
        images = os.listdir(path)
        sampled_images = random.sample(images, min(num_samples, len(images)))
        for img_name in sampled_images:
            img_path = os.path.join(path, img_name)
            img = cv2.imread(img_path)
            if img is not None:
                shapes.append(img.shape)
    return shapes


# Visualizing Random Images (not rly needed)
def show_random_images(dataset_path, classes, num_images=5):
    plt.figure(figsize=(15, 5))
```

```
    for idx, label in enumerate(classes):
        path = os.path.join(dataset_path, label)
        img_name = random.choice(os.listdir(path))
        img_path = os.path.join(path, img_name)
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        plt.subplot(1, len(classes), idx + 1)
        plt.imshow(img)
        plt.title(label)
        plt.axis('off')
    plt.show()

# Class distribution
def plot_class_distribution(counts):
    sns.barplot(x=list(counts.keys()), y=list(counts.values()))
    plt.title('Class Distribution')
    plt.ylabel('Number of Images')
    plt.show()




def plot_image_size_distribution(shapes):
```

```
    pixel_counts = [w * h for (h, w, _) in shapes]
    plt.figure(figsize=(8, 5))
    sns.histplot(pixel_counts, bins=30, kde=True)
    plt.title('Image Size (Pixel Count) Distribution')
    plt.xlabel('Number of Pixels (width x height)')
    plt.ylabel('Frequency')
    plt.show()

def plot_aspect_ratio_distribution(shapes):
    aspect_ratios = [w / h for (h, w, _) in shapes if h != 0]
    plt.figure(figsize=(8, 5))
    sns.histplot(aspect_ratios, bins=30, kde=True)
    plt.title('Aspect Ratio Distribution')
    plt.xlabel('Aspect Ratio (width / height)')
    plt.ylabel('Frequency')
    plt.show()


def plot_average_brightness(dataset_path, classes, num_samples=100):
    brightness = {label: [] for label in classes}

    for label in classes:
        path = os.path.join(dataset_path, label)
```

```
        images = os.listdir(path)
        sampled_images = random.sample(images, min(num_samples, len(images)))

        for img_name in sampled_images:
            img_path = os.path.join(path, img_name)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)  # Grayscale for brightness
            if img is not None:
                brightness[label].append(np.mean(img))

    plt.figure(figsize=(10, 6))
    for label, values in brightness.items():
        sns.kdeplot(values, label=label)
    plt.title('Average Brightness Distribution by Class')
    plt.xlabel('Average Pixel Intensity')
    plt.ylabel('Density')
    plt.legend()
    plt.show()

# Hypothetical split
def sketch_split(counts, train_ratio=0.7, val_ratio=0.15, test_ratio=0.15):
    print("Proposed dataset split:")
    for label, total in counts.items():
        train = int(total * train_ratio)
```

```
        val = int(total * val_ratio)

        test = total - train - val

        print(f"{label}: Train={train}, Validation={val}, Test={test}")




if __name__ == "__main__":

    counts = count_images(DATASET_PATH, classes)

    print("Image counts per class:", counts)


    shapes = get_image_shapes(DATASET_PATH, classes)

    print("\nUnique image shapes found:", Counter(shapes))


    plot_class_distribution(counts)

    show_random_images(DATASET_PATH, classes)


    plot_image_size_distribution(shapes)

    plot_aspect_ratio_distribution(shapes)

    plot_average_brightness(DATASET_PATH, classes)


    sketch_split(counts)
```