# Using Machine Learning to detect Covid-19 from Lung Scans

## 1. Introduction

This project focuses on the application of machine learning in medical diagnosis. Using existing medical technologies like X-ray scans, machine learning models can help diagnose Covid-19. The main objective of this paper is to analyze the accuracy of ML models in correctly predicting Covid-19 through X-ray lung scans.

Section 1 explains the motivation behind the topic and presents an outline of the paper. Section 2 focuses on problem formulation and data pre-processing. Section 3 provides a detailed explanation of two ML models implemented here. Section 4 is an analysis of results and Section 5 is the conclusion. Additionally, the code used to preprocess data, train models, and evaluate their performance is included in the appendix.

## 2. Problem Formulation

The dataset consists of X-ray lung scans where each scan is either Covid-19 positive or Covid-19 negative(normal and viral pneumonia scan). This is essentially a binary classification problem which is solved using supervised learning. The type of data used is categorical. Each datapoint is a 64x64 image making it a multidimensional input where pixels of the image serve as the features.The labels used here are binary: 1 for Covid-19 and 0 for No Covid-19.

## 3. Methods

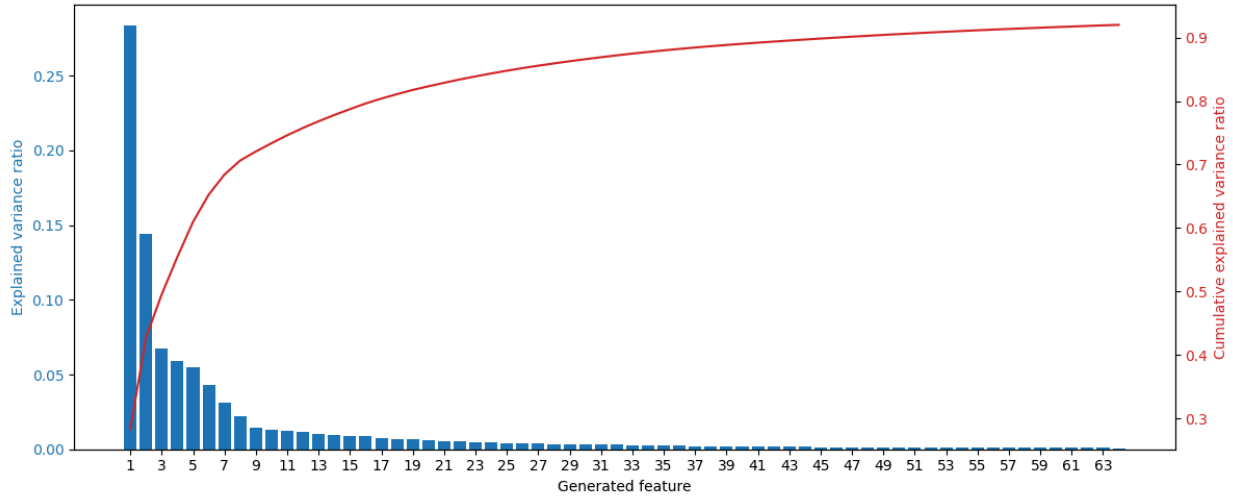
### 3.1 Dataset

The dataset is collected from Kaggle [1]. The dataset consists of a total of 15,153 images where 3,616 are covid are 11,537 are non-covid scans. Each datapoint in the dataset is a 64x64 image which is converted into a vector with 1600 features.

### 3.2 Preprocessing and Feature Selection and Data Augmentation

There is an imbalance in class sizes of the dataset which can affect the accuracy of the model. This imbalance is addressed by data augmentation. ImageDataGenerator[2] from TensorFlow makes sure that both classes are equal in size.

PCA is employed to help with feature selection. 64x64 image results in a vector with 4096 features. PCA is used to select the best 64 features while maintaining more than 0.9 cumulative explained variance ratio which can be seen in the plot below.




With the help of data augmentation, equal datapoints for both classes were created(11, 537 images each). Because of availability of extensive data, 70% of data was used for training whereas 30% was split equally for validating and testing. This ensures that there is enough data for validating and testing purposes.

## 3.3 Models

### 3.3.1 Logistic Regression with Logistic Loss

Logistic regression is an ML method that allows to classify data points according to two categories(Jung, 2022)[3]. Since this is a binary classification problem, logistic regression is a good choice.

The loss function used here is logistic loss.

$$L((x, y), h) := \log(1 + \exp(-yh(x)))$$

The loss functions used in binary classification are 0/1 loss, hinge loss and logistic loss(Jung, 2022). Among these logistic loss is chosen because is pairs well with logistic regression

### 3.3.2 Convoluted Neural Network(CNN) with Binary Cross Entropy

CNN is used because it is a popular choice for image classification. CNN can learn from raw pixel data without requiring any manual feature engineering or preprocessing. It also reduces the dimensionality and complexity of the input data, making the training and inference faster and more efficient[4]. This project implements CNN using Tensorflow which uses Binary Cross Entropy for binary classification[5], [6]. Therefore, it is ideal to go with Binary Cross Entropy

## 4. Results

For logistic regression average training accuracy is 85.77% and the average validation accuracy is 80.71%. For the CNN model average training accuracy is 98.38% and the average validation accuracy is 95.93%. According to these results CNN is a better model. The testing accuracy for the CNN model is 96.92 %. This test data is created using the data splitting explained earlier.

## 5. Conclusion

This report shows the effectiveness of Machine Learning models in diagnosing novel infections like Covid-19. Even with limited data points and data augmentation, CNN model was able to achieve a remarkable  96.92% testing accuracy. The accuracy can be further improved by using higher-resolution images, using more convolutional layers and more data. This shows that with better data, better processing and more complex models we can achieve extremely high accuracy on such diagnosis.

## References


[1]Dataset:
https://www.kaggle.com/datasets/tawsifurrahman/covid19-radiography-database?resource=download
[2]https://medium.com/nerd-for-tech/creating-dataset-from-tensorflow-imagedatagenerator-31ce5706612b
[3]A. Jung,“Machine Learning: The Basics,” Springer, Singapore, 2022
[4]https://www.linkedin.com/advice/0/what-advantages-disadvantages-using-convolutional
[5]https://www.tensorflow.org/tutorials/images/cnn#create_the_convolutional_base
[6]https://medium.com/@sssspppp/image-classification-using-cnn-0fad8367acfd


## Appendix

```
!pip install tensorflow opencv-python matplotlib filetype
!pip list
import cv2
import tensorflow as tf
import os
import filetype
import numpy as np
from matplotlib import pyplot as plt
```

```
#fetching data using tensorflow
```

```
data_dir = 'data'
batch_size = 32
image_size = (64, 64)

dataset = tf.keras.utils.image_dataset_from_directory(data_dir,
image_size=image_size, batch_size=batch_size)

class_0_count = 0
class_1_count = 0

for images, labels in dataset:
    class_0_count += tf.reduce_sum(tf.cast(labels == 0, tf.int32)).numpy()
    class_1_count += tf.reduce_sum(tf.cast(labels == 1, tf.int32)).numpy()
```

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

to_add = class_1_count - class_0_count
#creating augmentation layer
datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest'
    )
class_0_images = []
class_1_images = []

for images, labels in dataset:
    class_0_images.extend(images.numpy()[labels.numpy() == 0])
    class_1_images.extend(images.numpy()[labels.numpy() == 1])

# Convert to numpy arrays
class_0_images = np.array(class_0_images)
class_1_images = np.array(class_1_images)


augmented_images = []
for i in range(to_add):
```

```
    # Randomly select an image from class 0
    img = class_0_images[np.random.randint(0, class_0_count)]
    img = np.expand_dims(img, axis=0)  # Add a batch dimension

    # Generate an augmented image
    augmented_img = next(datagen.flow(img, batch_size=1))[0]
    augmented_images.append(augmented_img)

# Convert the list of augmented images to a numpy array
augmented_images = np.array(augmented_images)

# Combine original and augmented images for class 0
final_class_0_images = np.concatenate((class_0_images, augmented_images),
axis=0)
final_class_0_count = len(final_class_0_images)

print(f"Original Class 0 size: {class_0_count}")
print(f"Augmented Class 0 size: {final_class_0_count}")
```

Original Class 0 size: 3616
Augmented Class 0 size: 11537

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Combine the images from both classes
covid_images = final_class_0_images
non_covid_images = class_1_images
X = np.concatenate((covid_images, non_covid_images), axis=0)

# Create corresponding labels: 0 for class 0 and 1 for class 1
y_covid = np.ones(len(covid_images))
y_non_covid = np.zeros(len(non_covid_images))
y = np.concatenate((y_covid, y_non_covid), axis=0)

X_flattened = X.reshape(X.shape[0], -1)

X_train, X_test, y_train, y_test = train_test_split(X_flattened, y,
test_size=0.2, random_state=42)
X_valid, X_test, y_valid, y_test = train_test_split(X_test, y_test,
test_size=0.50, random_state=42)
```

```
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

import seaborn as sns
N = 64
pca = PCA(n_components=N)
X_train_reduced = pca.fit_transform(X_train)

# plot the explained variances
fig, ax1 = plt.subplots(figsize=(12, 5))
color = 'tab:blue'
ax1.bar(1+np.arange(N), pca.explained_variance_ratio_, color=color)
ax1.set_xticks(1+np.arange(N, step=2))
ax1.tick_params(axis='y', labelcolor=color)
ax1.set_ylabel("Explained variance ratio", color=color)
ax1.set_xlabel("Generated feature")

ax2 = ax1.twinx()
color = 'tab:red'
ax2.tick_params(axis='y', labelcolor=color)
ax2.plot(1+np.arange(N), np.cumsum(pca.explained_variance_ratio_),
color=color)
ax2.set_ylabel("Cumulative explained variance ratio", color=color)
fig.tight_layout()


plt.show()
```

```
# Apply PCA transformations
X_test_red = pca.transform(X_test)
X_val_red = pca.transform(X_valid)


# Initialize the Logistic Regression model
log_reg = LogisticRegression(solver='sag')
```

```
# Train the model on the reduced training set
log_reg.fit(X_train_reduced, y_train)

# Make predictions on the test set
y_pred = log_reg.predict(X_test_red)

# Calculate accuracy on test and training sets
accuracy = accuracy_score(y_test, y_pred)
tr_accuracy = accuracy_score(y_train, log_reg.predict(X_train_reduced))

# Print accuracies
print(f"Test Accuracy: {accuracy:.4f}")
print(f"Training Accuracy: {tr_accuracy:.4f}")
```

Test Accuracy: 0.8501
Training Accuracy: 0.8578

```
#For CNN
X_train_cnn, X_test_cnn, y_train_cnn, y_test_cnn = train_test_split(X, y,
test_size=0.2, random_state=42)
X_valid_cnn, X_test_cnn, y_valid_cnn, y_test_cnn =
train_test_split(X_test_cnn, y_test_cnn, test_size=0.50, random_state=42)

train_ds = tf.data.Dataset.from_tensor_slices((X_train_cnn,
y_train_cnn)).batch(32)
val_ds = tf.data.Dataset.from_tensor_slices((X_valid_cnn,
y_valid_cnn)).batch(32)
test_ds = tf.data.Dataset.from_tensor_slices((X_test_cnn,
y_test_cnn)).batch(32)


from tensorflow.keras import layers, models


model_cnn = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)),
# Input shape should match image size
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
```

```
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dense(1, activation='sigmoid')  # applying binary classification
])

# Compile the model
model_cnn.compile(optimizer='adam',
              loss='binary_crossentropy',  # Use binary crossentropy for
binary classification
              metrics=['accuracy'])

# Train the model
history = model_cnn.fit(train_ds, validation_data=val_ds, epochs=20)

test_loss, test_acc = model_cnn.evaluate(test_ds)
print(f"Test accuracy: {test_acc:.4f}")
print(f"Test loss: {test_loss:.4f}")
```

Test accuracy: 0.9692
Test loss: 0.1153

```
train_accuracy = history.history['accuracy']  # Accuracy during training
val_accuracy = history.history['val_accuracy']  # Validation accuracy

# print the final accuracy
print(f"Final training accuracy: {train_accuracy[-1]:.4f}")
print(f"Final validation accuracy: {val_accuracy[-1]:.4f}")
```

Final training accuracy: 0.9838
Final validation accuracy: 0.9593