Chest X-Ray Classification using GLCM, Wavelet Transforms, and CLAHE

(a) Normal (b) Pneumonia (c) COVD-19

```
import numpy as np
import pandas as pd
import os

import warnings
warnings.filterwarnings('ignore')

root_path = '/kaggle/input/chest-x-ray-images-normal-and-
pneumonia/chest_xray/train/'

image_paths = []
labels = []

for label in os.listdir(root_path):
    label_path = os.path.join(root_path, label)
    if os.path.isdir(label_path):
        for img_file in os.listdir(label_path):
            if img_file.lower().endswith(('.png', '.jpg', '.jpeg')):
                image_paths.append(os.path.join(label_path, img_file))
                labels.append(label)

df = pd.DataFrame({'image_path': image_paths, 'label': labels})

df.head()
```

	image_path	label
0	/kaggle/input/chest-x-ray-images-normal-and-pn...	PNEUMONIA
1	/kaggle/input/chest-x-ray-images-normal-and-pn...	PNEUMONIA
2	/kaggle/input/chest-x-ray-images-normal-and-pn...	PNEUMONIA

```
3  /kaggle/input/chest-x-ray-images-normal-and-pn...  PNEUMONIA
4  /kaggle/input/chest-x-ray-images-normal-and-pn...  PNEUMONIA
```

```
df.tail()
```

```
                                            image_path   label
5211  /kaggle/input/chest-x-ray-images-normal-and-pn...  NORMAL
5212  /kaggle/input/chest-x-ray-images-normal-and-pn...  NORMAL
5213  /kaggle/input/chest-x-ray-images-normal-and-pn...  NORMAL
5214  /kaggle/input/chest-x-ray-images-normal-and-pn...  NORMAL
5215  /kaggle/input/chest-x-ray-images-normal-and-pn...  NORMAL
```

```
df.shape
```

```
(5216, 2)
```

```
df.columns
```

```
Index(['image_path', 'label'], dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5216 entries, 0 to 5215
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   image_path  5216 non-null   object
 1   label       5216 non-null   object
dtypes: object(2)
memory usage: 81.6+ KB
```

```
df['label'].unique()
```

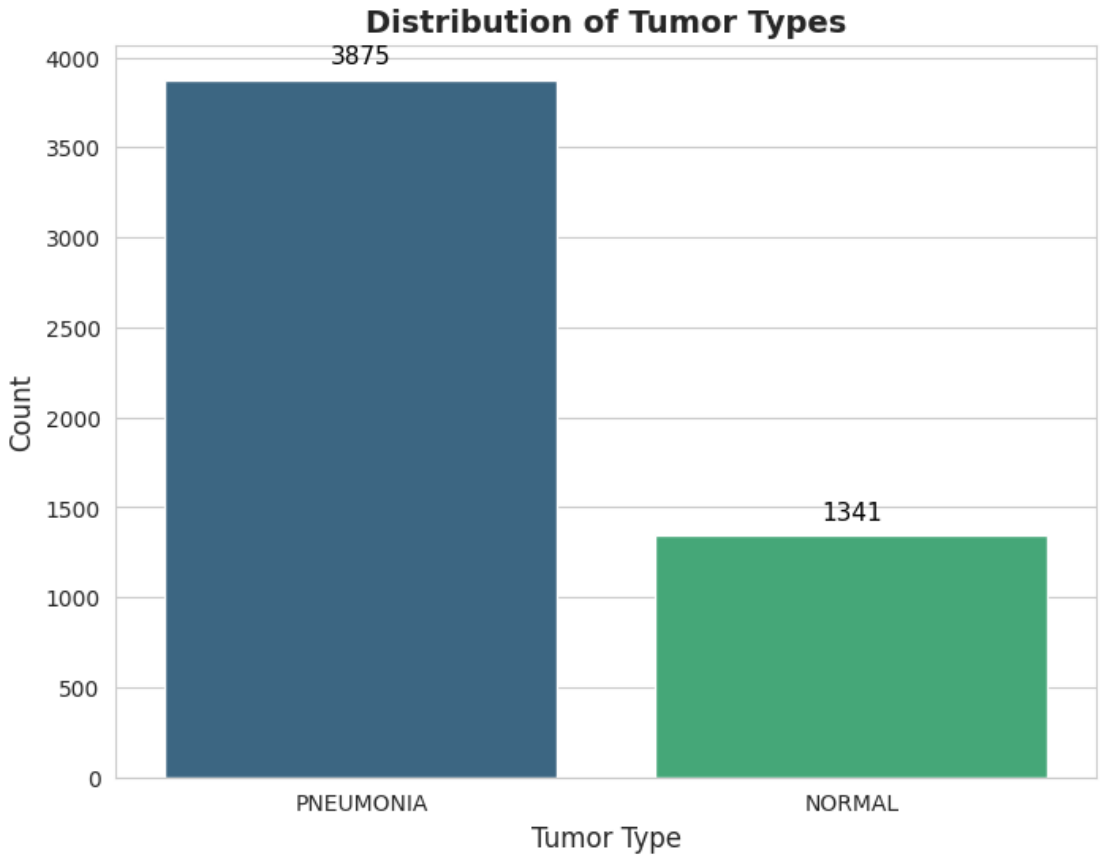
```
array(['PNEUMONIA', 'NORMAL'], dtype=object)
```

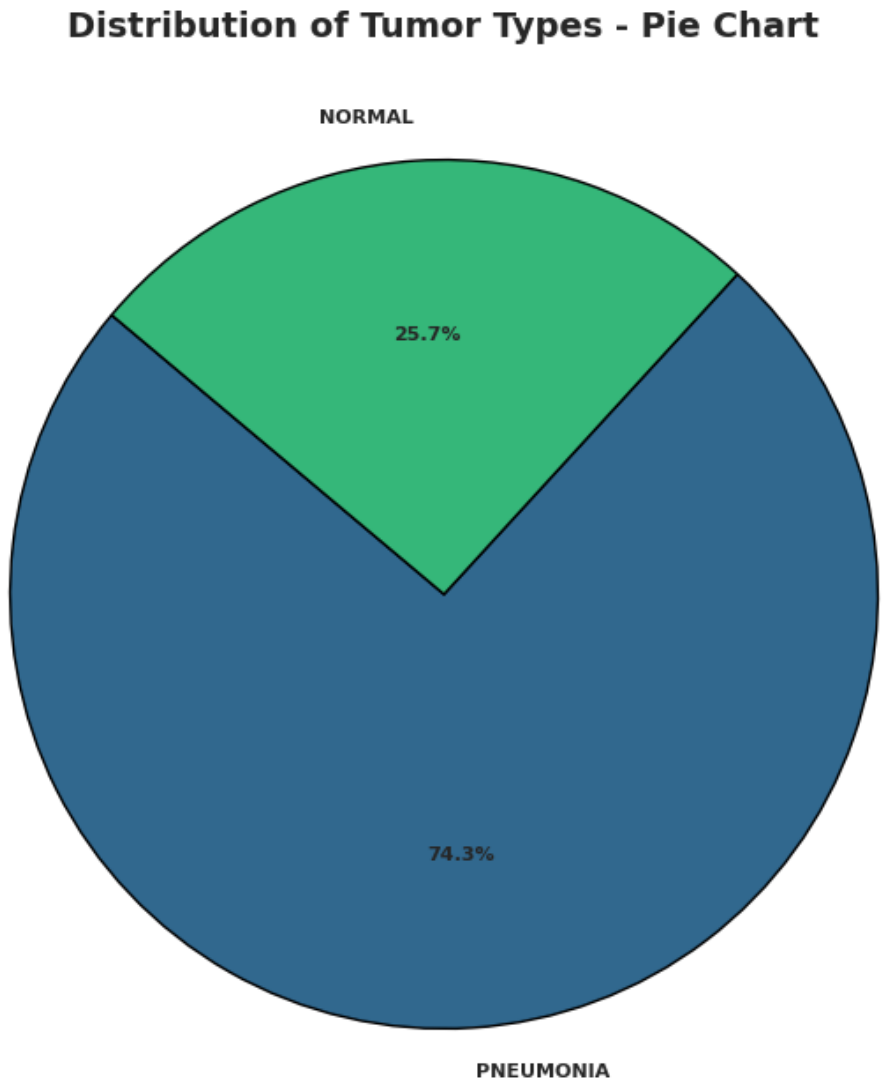
```
df['label'].value_counts()
```

```
label
PNEUMONIA    3875
NORMAL       1341
Name: count, dtype: int64
```

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_style("whitegrid")

fig, ax = plt.subplots(figsize=(8, 6))
sns.countplot(data=df, x="label", palette="viridis", ax=ax)

ax.set_title("Distribution of Tumor Types", fontsize=14, fontweight='bold')
```

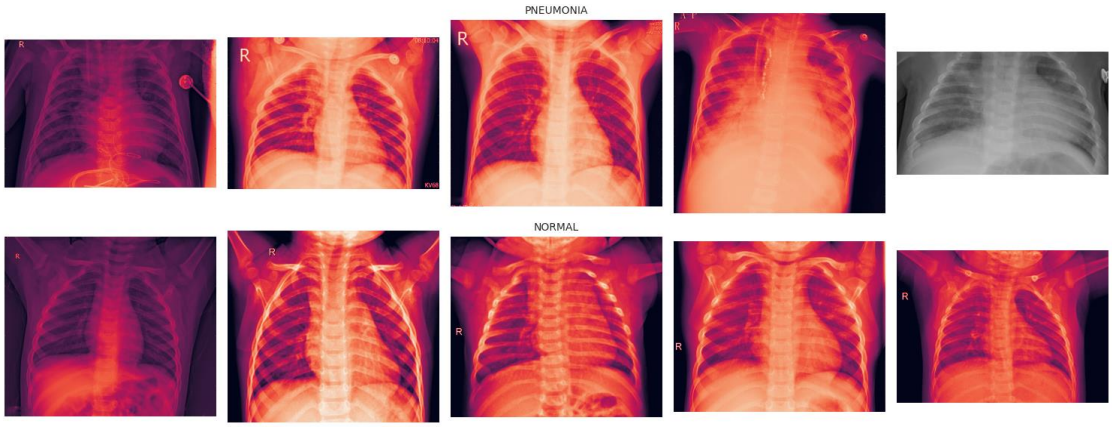
```
ax.set_xlabel("Tumor Type", fontsize=12)
ax.set_ylabel("Count", fontsize=12)

for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='bottom', fontsize=11, color='black',
                xytext=(0, 5), textcoords='offset points')

plt.show()

label_counts = df["label"].value_counts()

fig, ax = plt.subplots(figsize=(10, 8))
colors = sns.color_palette("viridis", len(label_counts))

ax.pie(label_counts, labels=label_counts.index, autopct='%1.1f%%',
       startangle=140, colors=colors, textprops={'fontsize': 8, 'weight':
'bold'},
       wedgeprops={'edgecolor': 'black', 'linewidth': 1})

ax.set_title("Distribution of Tumor Types - Pie Chart", fontsize=14,
fontweight='bold')

plt.show()
```


Distribution of Tumor Types
3875
1341
4000
3500
3000
2500
2000
1500
1000
500
0
Count
PNEUMONIA
NORMAL
Tumor Type




```
from PIL import Image

num_images = 5

unique_labels = df['label'].unique()

plt.figure(figsize=(15, len(unique_labels) * 3))

for row_idx, label in enumerate(unique_labels):

    label_images = df[df['label'] ==
label].head(num_images)['image_path'].tolist()

    for col_idx, img_path in enumerate(label_images):
        plt_idx = row_idx * num_images + col_idx + 1
```

```
        plt.subplot(len(unique_labels), num_images, plt_idx)
        img = Image.open(img_path)
        plt.imshow(img)
        plt.axis('off')
        if col_idx == 2:
            plt.title(label, fontsize=10)

plt.tight_layout()
plt.show()
```




```
max_samples = df['label'].value_counts().max()

balanced_df = df.groupby('label', group_keys=False).apply(
    lambda x: x.sample(n=max_samples, replace=True, random_state=42)
).reset_index(drop=True)

balanced_df = balanced_df[['image_path', 'label']]

df = balanced_df

df
```

	image_path	label
0	/kaggle/input/chest-x-ray-images-normal-and-pn...	NORMAL
1	/kaggle/input/chest-x-ray-images-normal-and-pn...	NORMAL
2	/kaggle/input/chest-x-ray-images-normal-and-pn...	NORMAL
3	/kaggle/input/chest-x-ray-images-normal-and-pn...	NORMAL
4	/kaggle/input/chest-x-ray-images-normal-and-pn...	NORMAL
...	...	...
7745	/kaggle/input/chest-x-ray-images-normal-and-pn...	PNEUMONIA
7746	/kaggle/input/chest-x-ray-images-normal-and-pn...	PNEUMONIA
7747	/kaggle/input/chest-x-ray-images-normal-and-pn...	PNEUMONIA
7748	/kaggle/input/chest-x-ray-images-normal-and-pn...	PNEUMONIA
7749	/kaggle/input/chest-x-ray-images-normal-and-pn...	PNEUMONIA

[7750 rows x 2 columns]

```
from PIL import Image
import cv2
from skimage.feature import graycomatrix, graycoprops
import pywt
import matplotlib.pyplot as plt

def get_rotation_invariant_glcm_features(image_array, distance=[1],
angles=[0, np.pi/4, np.pi/2, 3*np.pi/4], levels=32):
    image_array = image_array.astype(np.uint8)

    max_val = image_array.max()
    if max_val >= levels:
        quantized_array = np.floor(image_array * ((levels - 1) /
max_val)).astype(np.uint8)
    else:
        quantized_array = image_array

    gcom = graycomatrix(
        quantized_array,
        distances=distance,
        angles=angles,
        levels=levels,
        symmetric=True,
        normed=True
    )

    dissimilarity = np.mean(graycoprops(gcom, 'dissimilarity'))
    correlation = np.mean(graycoprops(gcom, 'correlation'))
    homogeneity = np.mean(graycoprops(gcom, 'homogeneity'))
    energy = np.mean(graycoprops(gcom, 'energy'))

    return [dissimilarity, correlation, homogeneity, energy]

def get_wavelet_features(image_array, wavelet='haar', level=1):
    coeffs = pywt.wavedec2(image_array.astype(float), wavelet, level=level)

    LH, HL, HH = coeffs[1]

    lh_energy = np.sum(LH**2)
    hl_energy = np.sum(HL**2)
    hh_energy = np.sum(HH**2)

    return [lh_energy, hl_energy, hh_energy]


def get_multi_feature_embedding(image_path, resize_to=(128, 128), n_blocks=4,
bit_depth=16):
    if resize_to[0] % n_blocks != 0 or resize_to[1] % n_blocks != 0:
        raise ValueError("resize_to dimensions must be divisible by
```

```
n_blocks.")

    block_w = resize_to[0] // n_blocks
    block_h = resize_to[1] // n_blocks

    try:
        img = Image.open(image_path).convert('L')
        img = img.resize(resize_to)
        img_array = np.array(img).astype(np.uint8)

        clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
        clahe_img = clahe.apply(img_array)

        embedding = []

        MAX_DISSIMILARITY = 16.0
        MAX_INTENSITY = 255.0
        MAX_WAVELET_ENERGY = 500000.0

        for i in range(n_blocks):
            for j in range(n_blocks):
                block = clahe_img[i*block_h:(i+1)*block_h,
j*block_w:(j+1)*block_w]

                glcm_features = get_rotation_invariant_glcm_features(block)
                wavelet_features = get_wavelet_features(block)
                mean_intensity = np.mean(block)
                std_intensity = np.std(block)

                features = [
                    min(glcm_features[0] / MAX_DISSIMILARITY, 1.0),
                    (glcm_features[1] + 1.0) / 2.0,
                    glcm_features[2],
                    glcm_features[3],
                    min(wavelet_features[0] / MAX_WAVELET_ENERGY, 1.0),
                    min(wavelet_features[1] / MAX_WAVELET_ENERGY, 1.0),
                    min(wavelet_features[2] / MAX_WAVELET_ENERGY, 1.0),
                    mean_intensity / MAX_INTENSITY,
                    std_intensity / MAX_INTENSITY
                ]

                for feat in features:
                    scaled_value = int(feat * (2**bit_depth - 1))
                    bin_str = bin(scaled_value)[2:].zfill(bit_depth)
                    bin_list = [int(bit) for bit in bin_str]
                    embedding.extend(bin_list)

        return embedding
    except Exception as e:
```

```
        print(f"Error processing {image_path}: {e}")
        return None

def embedding_to_analog(embedding, bit_depth=16):
    if not embedding: return []
    analog = []
    for i in range(0, len(embedding), bit_depth):
        bin_str = ''.join(str(bit) for bit in embedding[i:i+bit_depth])
        value = int(bin_str, 2)
        analog.append(value)
    return analog

def generate_em_wave(analog_values, feature_weights=None, carrier_freq=5,
sampling_rate=50, duration_per_value=0.1, max_value=2**16 - 1):
    if not analog_values: return np.array([0]), np.array([0])

    total_duration = len(analog_values) * duration_per_value
    t = np.linspace(0, total_duration, int(total_duration * sampling_rate),
endpoint=False)
    wave = np.zeros_like(t)
    samples_per_value = int(duration_per_value * sampling_rate)

    if feature_weights is None:
        feature_weights = np.ones(len(analog_values))

    for i, value in enumerate(analog_values):
        amp = value / max_value
        weighted_amp = amp * feature_weights[i]

        start = i * samples_per_value
        end = min(start + samples_per_value, len(t))

        if start < len(t):
            wave[start:end] = weighted_amp * np.sin(2 * np.pi * carrier_freq
* t[start:end])

    return t, wave

df['multi_feature_embedding'] = df['image_path'].apply(lambda path:
get_multi_feature_embedding(path))

df
```

```
                                          image_path      label  \
0     /kaggle/input/chest-x-ray-images-normal-and-pn...     NORMAL
1     /kaggle/input/chest-x-ray-images-normal-and-pn...     NORMAL
2     /kaggle/input/chest-x-ray-images-normal-and-pn...     NORMAL
3     /kaggle/input/chest-x-ray-images-normal-and-pn...     NORMAL
4     /kaggle/input/chest-x-ray-images-normal-and-pn...     NORMAL
```

```
...                                                    ...        ...
7745  /kaggle/input/chest-x-ray-images-normal-and-pn...  PNEUMONIA
7746  /kaggle/input/chest-x-ray-images-normal-and-pn...  PNEUMONIA
7747  /kaggle/input/chest-x-ray-images-normal-and-pn...  PNEUMONIA
7748  /kaggle/input/chest-x-ray-images-normal-and-pn...  PNEUMONIA
7749  /kaggle/input/chest-x-ray-images-normal-and-pn...  PNEUMONIA

                               multi_feature_embedding
0     [0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, ...
1     [0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, ...
2     [0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, ...
3     [0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, ...
4     [0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, ...
...                                                   ...
7745  [0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, ...
7746  [0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, ...
7747  [0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, ...
7748  [0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, ...
7749  [0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, ...

[7750 rows x 3 columns]
```

```python
def embedding_to_analog(embedding, bit_depth=16):
    if not embedding: return []
    analog = []
    for i in range(0, len(embedding), bit_depth):
        bin_str = ''.join(str(bit) for bit in embedding[i:i+bit_depth])
        value = int(bin_str, 2)
        analog.append(value)
    return analog

def generate_em_wave(analog_values, feature_weights=None, carrier_freq=5,
sampling_rate=50, duration_per_value=0.1, max_value=2**16 - 1):
    if not analog_values: return np.array([0]), np.array([0])

    total_duration = len(analog_values) * duration_per_value
    t = np.linspace(0, total_duration, int(total_duration * sampling_rate),
endpoint=False)
    wave = np.zeros_like(t)
    samples_per_value = int(duration_per_value * sampling_rate)

    if feature_weights is None:
        feature_weights = np.ones(len(analog_values))

    for i, value in enumerate(analog_values):
        amp = value / max_value
        weighted_amp = amp * feature_weights[i]

        start = i * samples_per_value
```

```
        end = min(start + samples_per_value, len(t))

        if start < len(t):
            wave[start:end] = weighted_amp * np.sin(2 * np.pi * carrier_freq
* t[start:end])

    return t, wave

df['multi_analog_values'] =
df['multi_feature_embedding'].apply(embedding_to_analog)

N_PLOTS = 5
fig, axes = plt.subplots(N_PLOTS, 1, figsize=(14, 2 * N_PLOTS))
plt.suptitle('Simulated EM Wave from Multi-Feature Embedding (First 5
Images)', fontsize=14, y=1.02)

FEATURES_TO_PLOT = 50

for i in range(N_PLOTS):
    analog_values = df['multi_analog_values'].iloc[i]
    label = df['label'].iloc[i]

    analog_subsample = analog_values[:FEATURES_TO_PLOT]

    t, wave = generate_em_wave(analog_subsample, carrier_freq=10,
sampling_rate=100)

    ax = axes[i]
    ax.plot(t, wave, color='purple')
    ax.set_title(f"Image {i+1} - Label: {label}", fontsize=10, loc='left')
    ax.set_ylabel('Amplitude')
    ax.set_ylim(-1.1, 1.1)
    ax.grid(axis='y', linestyle='--')

    if i < N_PLOTS - 1:
        ax.set_xticks([])

axes[-1].set_xlabel(f'Time (s) - Showing first {FEATURES_TO_PLOT} features.')
plt.tight_layout()
plt.show()
```


```python
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

def embedding_to_analog(embedding, bit_depth=16):
    """Converts the binary list back to a list of analog integer values."""
    if not embedding: return []
    analog = []
    for i in range(0, len(embedding), bit_depth):
        bin_str = ''.join(str(bit) for bit in embedding[i:i+bit_depth])
        value = int(bin_str, 2)
        analog.append(value)
    return analog

FEATURE_NAMES = [
    'GLCM: Dissimilarity (Avg)',
    'GLCM: Correlation (Avg)',
    'GLCM: Homogeneity (Avg)',
    'GLCM: Energy (Avg)',
    'Wavelet: LH Energy',
    'Wavelet: HL Energy',
    'Wavelet: HH Energy',
    'Intensity: Mean',
    'Intensity: Std Dev'
]
N_FEATURES_PER_BLOCK = 9
```

```
N_BLOCKS = 4

if 'multi_analog_values' not in df.columns:
    df['multi_analog_values'] =
df['multi_feature_embedding'].apply(embedding_to_analog)

analog_values = df['multi_analog_values'].iloc[0]
image_label = df['label'].iloc[0]

analog_matrix = np.array(analog_values).reshape(N_BLOCKS * N_BLOCKS,
N_FEATURES_PER_BLOCK)

fig = plt.figure(figsize=(12, 12))
gs = gridspec.GridSpec(3, 3, figure=fig, hspace=0.3, wspace=0.3)
plt.suptitle(f'Feature Maps for First Image (Label: {image_label})',
fontsize=16, y=0.95)

for f_idx, feature_name in enumerate(FEATURE_NAMES):
    feature_magnitudes = analog_matrix[:, f_idx]

    feature_map = feature_magnitudes.reshape(N_BLOCKS, N_BLOCKS)

    ax = fig.add_subplot(gs[f_idx // 3, f_idx % 3])

    im = ax.imshow(feature_map, cmap='magma', interpolation='nearest')

    ax.set_title(feature_name, fontsize=10)
    ax.set_xticks([])
    ax.set_yticks([])

    cbar = fig.colorbar(im, ax=ax, fraction=0.045, pad=0.04)
    cbar.ax.tick_params(labelsize=8)

plt.tight_layout(rect=[0, 0, 1, 0.9])
plt.show()
```

df

	image_path	label	\
0	/kaggle/input/chest-x-ray-images-normal-and-pn...	NORMAL	
1	/kaggle/input/chest-x-ray-images-normal-and-pn...	NORMAL	
2	/kaggle/input/chest-x-ray-images-normal-and-pn...	NORMAL	
3	/kaggle/input/chest-x-ray-images-normal-and-pn...	NORMAL	
4	/kaggle/input/chest-x-ray-images-normal-and-pn...	NORMAL	
...	...	...	
7745	/kaggle/input/chest-x-ray-images-normal-and-pn...	PNEUMONIA	
7746	/kaggle/input/chest-x-ray-images-normal-and-pn...	PNEUMONIA	
7747	/kaggle/input/chest-x-ray-images-normal-and-pn...	PNEUMONIA	
7748	/kaggle/input/chest-x-ray-images-normal-and-pn...	PNEUMONIA	
7749	/kaggle/input/chest-x-ray-images-normal-and-pn...	PNEUMONIA	

multi_feature_embedding	\

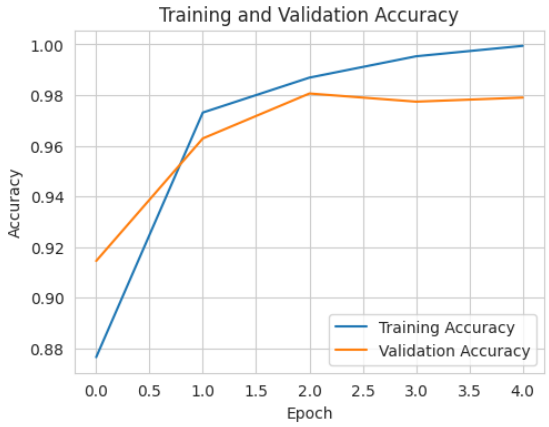
```
0     [0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, ...
1     [0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, ...
2     [0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, ...
3     [0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, ...
4     [0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, ...
...                                                   ...
7745  [0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, ...
7746  [0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, ...
7747  [0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, ...
7748  [0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, ...
7749  [0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, ...

                                      multi_analog_values
0     [5696, 64320, 36841, 11664, 3428, 8345, 1094, ...
1     [6244, 63693, 33233, 5876, 8908, 5479, 2024, 3...
2     [5905, 64272, 34866, 9217, 3188, 10463, 1082, ...
3     [7223, 62548, 29962, 6183, 4374, 14396, 1542, ...
4     [7347, 61600, 32665, 6383, 9102, 13798, 4268, ...
...                                                   ...
7745  [5653, 63222, 35038, 6911, 4154, 6083, 1326, 4...
7746  [6828, 57273, 35833, 7872, 8856, 21497, 4005, ...
7747  [7018, 61047, 32561, 7962, 4034, 13563, 1470, ...
7748  [7639, 61642, 31029, 5852, 5725, 11356, 2023, ...
7749  [5496, 63608, 40484, 11421, 2583, 28977, 2680,...

[7750 rows x 4 columns]
```

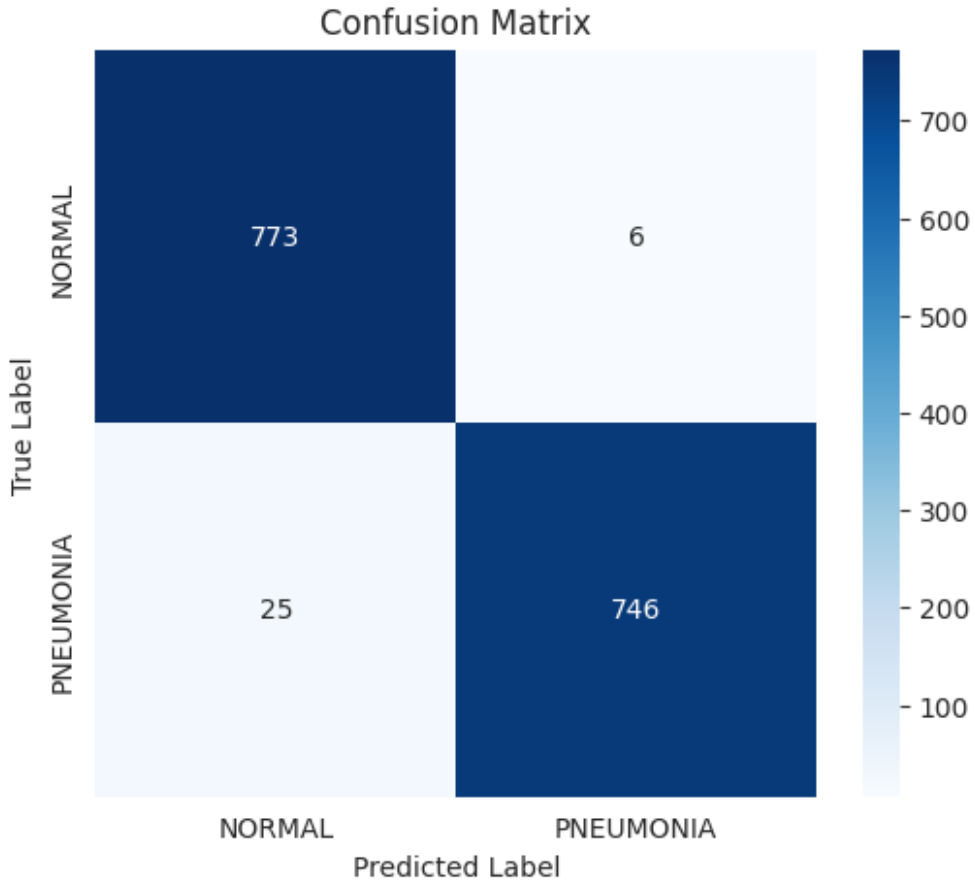
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import confusion_matrix, classification_report

X = np.array(df['multi_feature_embedding'].tolist())
le = LabelEncoder()
y = le.fit_transform(df['label'])
target_names = le.classes_

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer=Adam(learning_rate=0.001),
```

```
              loss='binary_crossentropy',
              metrics=['accuracy'])

EPOCHS = 5
history = model.fit(
    X_train, y_train,
    epochs=EPOCHS,
    batch_size=32,
    validation_split=0.1,
    verbose=1
)

y_pred_proba = model.predict(X_test)
y_pred = (y_pred_proba > 0.5).astype(int)

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=target_names, yticklabels=target_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

report = classification_report(y_test, y_pred, target_names=target_names)
print("\n--- Classification Report ---")
print(report)
```

```
2025-10-21 07:25:54.852301: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to
register cuFFT factory: Attempting to register factory for plugin cuFFT when
one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are written
```

```
to STDERR
E0000 00:00:1761031555.257599      37 cuda_dnn.cc:8310] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
E0000 00:00:1761031555.392853      37 cuda_blas.cc:1418] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has
already been registered
I0000 00:00:1761031574.373998      37 gpu_device.cc:2022] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 13942 MB memory:  ->
device: 0, name: Tesla T4, pci bus id: 0000:00:04.0, compute capability: 7.5
I0000 00:00:1761031574.374819      37 gpu_device.cc:2022] Created device
/job:localhost/replica:0/task:0/device:GPU:1 with 13942 MB memory:  ->
device: 1, name: Tesla T4, pci bus id: 0000:00:05.0, compute capability: 7.5

Epoch 1/5

WARNING: All log messages before absl::InitializeLog() is called are written
to STDERR
I0000 00:00:1761031578.216555     119 service.cc:148] XLA service
0x7bbd3000d430 initialized for platform CUDA (this does not guarantee that
XLA will be used). Devices:
I0000 00:00:1761031578.217756     119 service.cc:156]   StreamExecutor device
(0): Tesla T4, Compute Capability 7.5
I0000 00:00:1761031578.217776     119 service.cc:156]   StreamExecutor device
(1): Tesla T4, Compute Capability 7.5
I0000 00:00:1761031578.544507     119 cuda_dnn.cc:529] Loaded cuDNN version
90300

 69/175 ━━━━━━━━━━━━━━━━━━━━0s 2ms/step - accuracy: 0.6833 - loss: 0.6473

I0000 00:00:1761031579.531396     119 device_compiler.h:188] Compiled cluster
using XLA!  This line is logged at most once for the lifetime of the process.

175/175 ━━━━━━━━━━━━━━━━━━━━5s 10ms/step - accuracy: 0.7839 - loss: 0.4671 -
val_accuracy: 0.9145 - val_loss: 0.1869
Epoch 2/5
175/175 ━━━━━━━━━━━━━━━━━━━━1s 3ms/step - accuracy: 0.9700 - loss: 0.0995 -
val_accuracy: 0.9629 - val_loss: 0.1069
Epoch 3/5
175/175 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.9877 - loss: 0.0438 -
val_accuracy: 0.9806 - val_loss: 0.0814
Epoch 4/5
175/175 ━━━━━━━━━━━━━━━━━━━━0s 3ms/step - accuracy: 0.9955 - loss: 0.0191 -
val_accuracy: 0.9774 - val_loss: 0.0917
Epoch 5/5
175/175 ━━━━━━━━━━━━━━━━━━━━1s 3ms/step - accuracy: 0.9997 - loss: 0.0049 -
val_accuracy: 0.9790 - val_loss: 0.0934
49/49 ━━━━━━━━━━━━━━━━━━━━1s 7ms/step
```





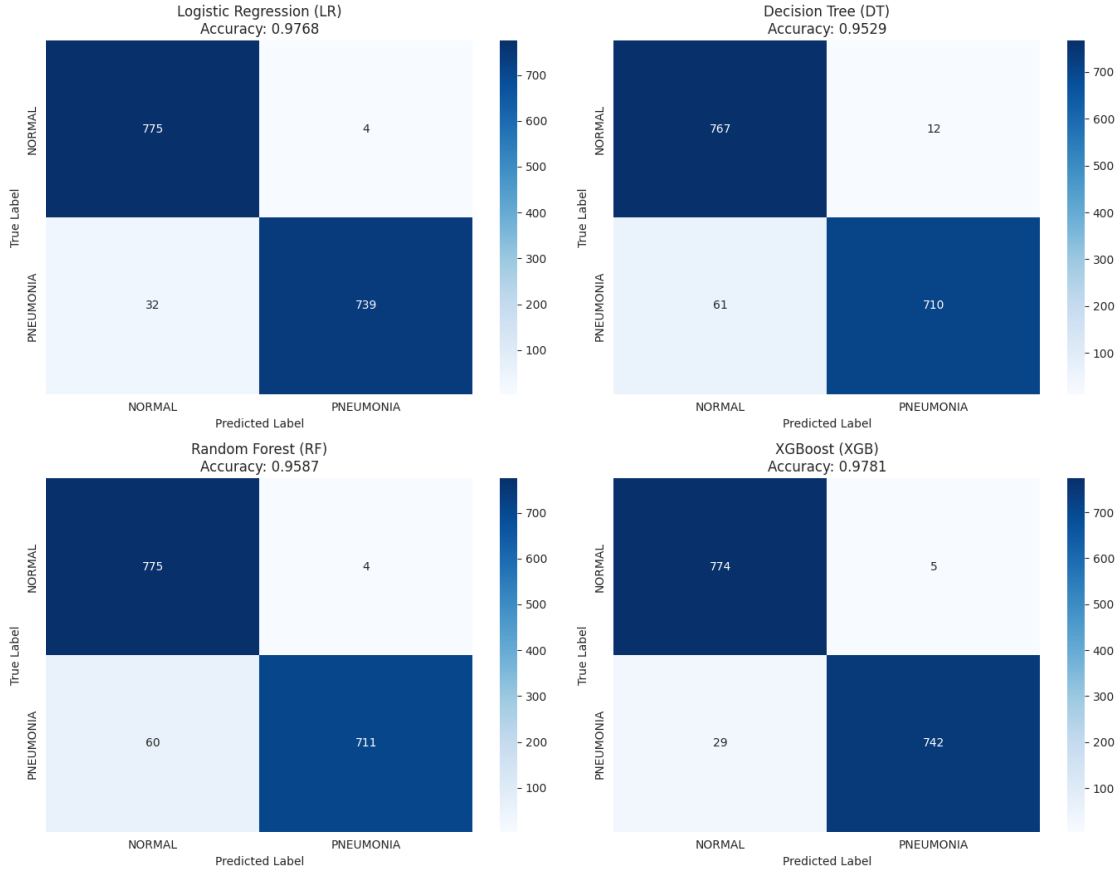

```
--- Classification Report ---
              precision    recall  f1-score   support

      NORMAL       0.97      0.99      0.98       779
   PNEUMONIA       0.99      0.97      0.98       771

    accuracy                           0.98      1550
   macro avg       0.98      0.98      0.98      1550
```

```
weighted avg       0.98      0.98      0.98      1550
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
import time

models = {
    "Logistic Regression (LR)": LogisticRegression(max_iter=1000,
solver='liblinear', random_state=42),
    "Decision Tree (DT)": DecisionTreeClassifier(max_depth=10,
random_state=42),
    "Random Forest (RF)": RandomForestClassifier(n_estimators=100,
max_depth=10, random_state=42, n_jobs=-1),
    "XGBoost (XGB)": XGBClassifier(n_estimators=100, use_label_encoder=False,
eval_metric='logloss', random_state=42, n_jobs=-1)
}

results = {}

print("--- Starting Model Training and Evaluation ---")
for name, model in models.items():
    start_time = time.time()

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    end_time = time.time()

    results[name] = {
        'model': model,
        'predictions': y_pred,
        'report': classification_report(y_test, y_pred,
target_names=target_names, output_dict=True),
        'time': end_time - start_time
    }

    print(f"\n{name} Trained in: {results[name]['time']:.2f} seconds")

fig, axes = plt.subplots(2, 2, figsize=(14, 12))
axes = axes.flatten()
plt.suptitle('Model Comparison: Confusion Matrices', fontsize=16)

summary_data = []
for i, (name, res) in enumerate(results.items()):
    y_pred = res['predictions']
```

```
    cm = confusion_matrix(y_test, y_pred)
    acc = res['report']['accuracy']

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=target_names, yticklabels=target_names,
ax=axes[i])
    axes[i].set_title(f"{name}\nAccuracy: {acc:.4f}", fontsize=12)
    axes[i].set_xlabel('Predicted Label')
    axes[i].set_ylabel('True Label')

    summary_data.append([
        name,
        f"{acc:.4f}",
        f"{res['report']['PNEUMONIA']['f1-score']:.4f}",
        f"{res['report']['NORMAL']['recall']:.4f}",
        f"{res['report']['PNEUMONIA']['recall']:.4f}",
        f"{res['time']:.2f}s"
    ])

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()

print("\n" + "="*80)
print("             DETAILED CLASSIFICATION REPORTS (Macro Avg F1-Score)")
print("="*80)
for name, res in results.items():
    print(f"\n--- {name} ---")
    print(classification_report(y_test, res['predictions'],
target_names=target_names))

summary_df = pd.DataFrame(summary_data, columns=['Model', 'Accuracy',
'Pneumonia F1', 'Normal Recall', 'Pneumonia Recall', 'Train Time'])
print("\n" + "="*80)
print("                    SUMMARY PERFORMANCE TABLE")
print("="*80)
print(summary_df.to_string(index=False))
```

```
--- Starting Model Training and Evaluation ---

Logistic Regression (LR) Trained in: 1.85 seconds

Decision Tree (DT) Trained in: 1.37 seconds

Random Forest (RF) Trained in: 1.09 seconds

XGBoost (XGB) Trained in: 2.31 seconds
```




```
================================================================================
                DETAILED CLASSIFICATION REPORTS (Macro Avg F1-Score)
================================================================================

--- Logistic Regression (LR) ---
              precision    recall  f1-score   support

      NORMAL       0.96      0.99      0.98       779
   PNEUMONIA       0.99      0.96      0.98       771

    accuracy                           0.98      1550
   macro avg       0.98      0.98      0.98      1550
weighted avg       0.98      0.98      0.98      1550


--- Decision Tree (DT) ---
              precision    recall  f1-score   support
```

```
      NORMAL       0.93      0.98      0.95       779
   PNEUMONIA       0.98      0.92      0.95       771

    accuracy                           0.95      1550
   macro avg       0.95      0.95      0.95      1550
weighted avg       0.95      0.95      0.95      1550


--- Random Forest (RF) ---
              precision    recall  f1-score   support

      NORMAL       0.93      0.99      0.96       779
   PNEUMONIA       0.99      0.92      0.96       771

    accuracy                           0.96      1550
   macro avg       0.96      0.96      0.96      1550
weighted avg       0.96      0.96      0.96      1550


--- XGBoost (XGB) ---
              precision    recall  f1-score   support

      NORMAL       0.96      0.99      0.98       779
   PNEUMONIA       0.99      0.96      0.98       771

    accuracy                           0.98      1550
   macro avg       0.98      0.98      0.98      1550
weighted avg       0.98      0.98      0.98      1550


================================================================================
                    SUMMARY PERFORMANCE TABLE
================================================================================
                    Model Accuracy Pneumonia F1 Normal Recall Pneumonia Recall
Train Time
Logistic Regression (LR)   0.9768       0.9762        0.9949           0.9585
1.85s
      Decision Tree (DT)   0.9529       0.9511        0.9846           0.9209
1.37s
      Random Forest (RF)   0.9587       0.9569        0.9949           0.9222
1.09s
           XGBoost (XGB)   0.9781       0.9776        0.9936           0.9624
2.31s
```