Performance and Resource Trade-Off Analysis of Neural Networks and Support Vector Machines on Pneumonia Detection

Mary Klimasewiski, Leonardo Pinheiro, Sedat Touray

Abstract

Pneumonia remains a significant global health concern, particularly in resource-limited settings where early and accurate diagnosis is critical. This study analyzes the trade-offs between Support Vector Machines (SVMs) and Feed-Forward Neural Networks (FFNNs) for pneumonia detection using the RSNA Pneumonia Detection Challenge dataset. While Convolutional Neural Networks (CNNs) set the benchmark for medical image classification, they demand substantial computational resources. We evaluate SVMs and FFNNs in terms of classification performance, training time, and memory utilization to explore alternatives for resource-constrained environments. Our findings indicate that SVMs offer better recall for pneumonia cases, whereas FFNNs struggle with class imbalance despite higher overall accuracy. CNNs outperform both models but at a significant computational cost. These results highlight the trade-offs between performance and efficiency, guiding model selection based on deployment constraints.

1. Introduction

1.1 Motivation

Pneumonia remains a major global health concern, particularly in resource-limited settings where early and accurate diagnosis can significantly impact patient outcomes. The standard tool in pneumonia is the chest X-ray whose interpretation requires experienced radiologists, making automated detection an attractive alternative. Recent advances in machine learning have enabled automated diagnostic tools that can assist healthcare professionals by improving detection accuracy and treatment efficiency.

1.2 Problem Definition

This study explores the trade-offs between traditional machine learning models, such as Support Vector Machines (SVMs), and more advanced deep learning models, including Feed-Forward Neural Networks (FFNNs), in the context of pneumonia detection from chest X-rays.

While Convolutional Neural Networks (CNNs) are widely regarded as the benchmark for image classification tasks due to their proven performance, our primary focus is on comparing SVMs and FFNNs directly. CNNs will serve as a reference point to understand the upper bound of model performance, but we are specifically interested in evaluating the trade-offs between SVM and FFNN, both of which offer different strengths and computational costs.

SVMs classify data by determining hyperplanes that best separate different classes in a high-dimensional feature space. In the case of our study the two classes are pneumonia and no-pneumonia. Support Vector Machines are particularly effective for smaller datasets and offer lower computational cost but may struggle with high-dimensional, complex data like raw images. Moreover, SVMs typically require manual feature engineering, which can be time-consuming and may limit their ability to automatically capture intricate patterns in the data.

On the other hand, FFNNs are capable of learning hierarchical representations directly from raw data, such as pixel values in images after minimal image preprocessing. They model complex, non-linear relationships within the data and can handle high-dimensional inputs. However, FFNNs still require significant computational resources.

While CNNs remain the best-performing models for image classification, they come with significant trade-offs, particularly in terms of computational costs. Training CNNs requires large datasets, substantial memory, and powerful hardware such as GPUs, which can limit their practical deployment in resource-constrained environments.

1.3 Existing Approaches

Machine learning and deep learning have been widely applied to pneumonia detection using chest X-rays. Among these, Convolutional Neural Networks (CNNs) have demonstrated state-of-the-art performance by automatically learning spatial hierarchies in images. Models such as AlexNet, VGG, and ResNet have been extensively used in medical image classification tasks, achieving high accuracy due to their ability to extract complex features from raw images. However, CNNs require large datasets, significant computational resources, and specialized hardware such as GPUs, making them less practical in resource-constrained environments.

Support Vector Machines (SVMs) have been a long-standing approach for medical image classification. By mapping data into a high-dimensional space and finding an optimal hyperplane for separation, SVMs perform well on structured feature sets. They have been applied to pneumonia detection when combined with feature extraction techniques such as Principal Component Analysis (PCA) or Histogram of Oriented Gradients (HOG). However, SVMs rely on manual feature engineering and may struggle with high-dimensional raw image data.

Feed-Forward Neural Networks (FFNNs) provide an alternative approach, offering some advantages of deep learning without the full computational burden of CNNs. FFNNs can learn from raw pixel values and model complex non-linear relationships, but they lack the spatial awareness of CNNs, which can limit their effectiveness for image classification tasks.

This study builds on these existing approaches by directly comparing SVMs and FFNNs for pneumonia detection, assessing their trade-offs in terms of classification performance and computational efficiency, with CNNs serving as a benchmark.

2. Methodology

2.1 The RSNA Pneumonia Detection Dataset

The RSNA Pneumonia Detection Challenge 2018, hosted by the Radiological Society of North America (RSNA) in collaboration with the National Institutes of Health (NIH) and Stanford University, provides a comprehensive dataset for pneumonia detection from chest X-ray images.

The dataset was sourced from the National Institutes of Health (NIH)'s publicly available Chest X-ray 8 (CXR8) collection. This dataset comprises over 112,000 frontal-view chest radiographs, each labeled with 14 different disease labels, including pneumonia.

For the challenge, a subset of 30,000 images was selected, consisting of 16,248 posteroanterior views and 13,752 anteroposterior views. These images were converted from Portable Network Graphics (PNG) format to Digital Imaging and Communications in Medicine (DICOM) format to standardize the dataset for medical imaging applications.

The dataset was meticulously annotated by multiple expert reviewers, including specialists from the Society of Thoracic Radiology. These professionals identified abnormal areas in the lung images and assessed the probability of pneumonia, providing the ground truth for participants to train and evaluate their algorithms.

This comprehensive dataset has been instrumental in advancing research in automated pneumonia detection, offering a valuable resource for developing and evaluating machine learning models in the medical imaging domain.

2.2. Dataset Description

The RSNA dataset consists of 30,000 frontal-view chest X-ray images sourced from patients with varying levels of pneumonia severity. The dataset includes:

- Normal Class (6,012 images): Chest X-rays from healthy individuals without any signs of pneumonia.

- Pneumonia Class (20,672): Chest X-rays indicating the presence of pneumonia, with varying levels of severity.

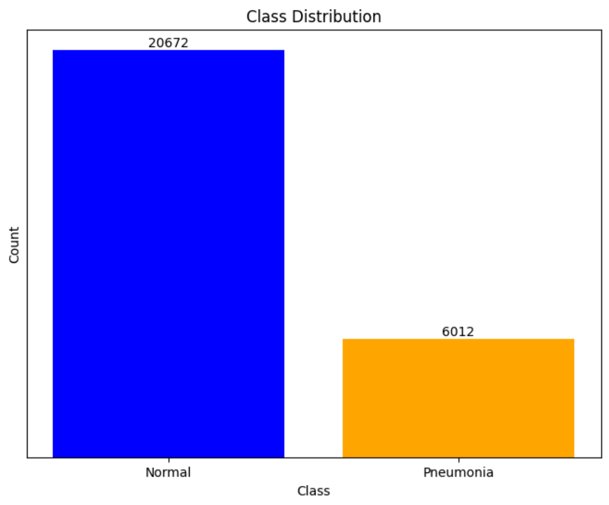


Figure 1. Class Distributions

Each image is annotated with labels from radiologists to indicate the presence or absence of pneumonia. The images are in DICOM format (Digital Imaging and Communications in Medicine), commonly used in medical imaging.

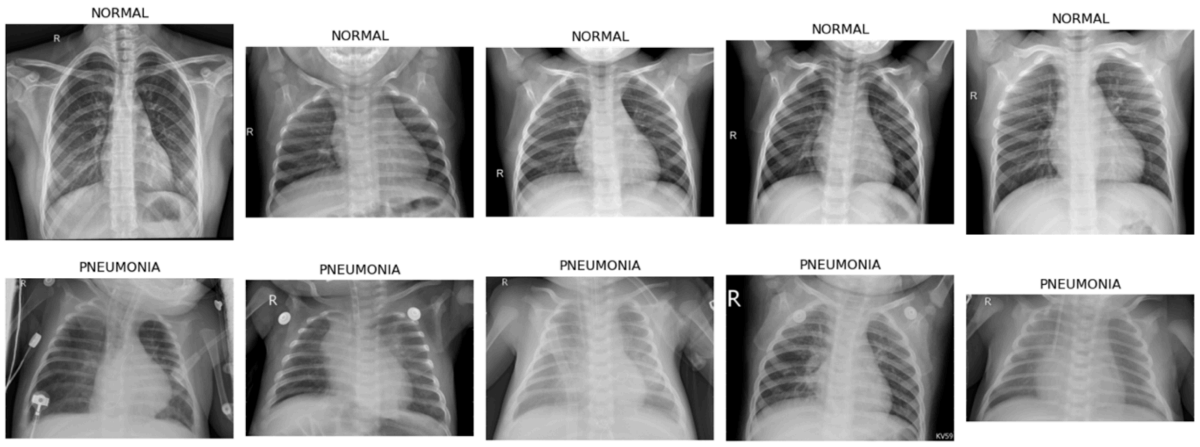


Figure 2 Sample Images from Dataset

2.3 Preprocessing Steps

The raw chest X-ray images require several preprocessing steps before being fed into machine learning models to ensure optimal performance and compatibility. DICOM images were processed using the `pydicom` library, which allows easy manipulation of DICOM files. These images were converted into `NumPy` arrays for further processing. All images were resized to a uniform size of 128x128 pixels to maintain consistency in dimensions, making them suitable for input into neural networks. This resizing was done using cv2 to avoid potential distortions and loss of crucial image features.

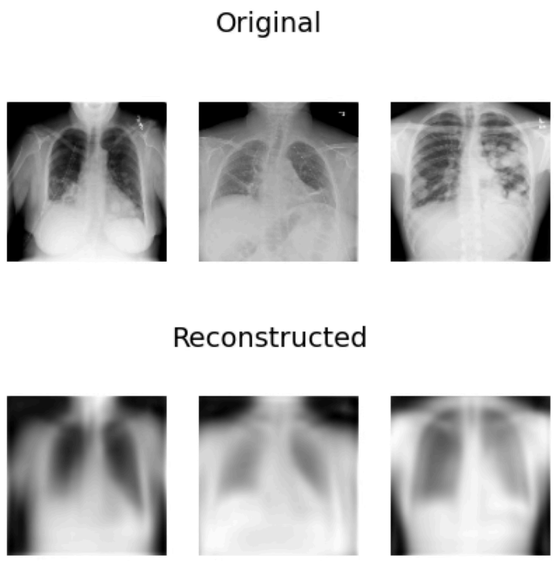


Figure 3. Orginal and PCA reconstructed images

For the SVM model, we applied Principal Component Analysis (PCA) to reduce the dimensionality of the data. After flattening the images into vectors, 100 principal components were retained, explaining approximately 91.5% of the variance in the data. This helped reduce the computational cost for SVM without significant loss of performance. For SVM and FFNN models, feature scaling was applied. The pixel values were standardized using zero-mean and unit-variance transformation, which normalizes the feature distribution to improve model convergence.

The dataset was divided in train and test splits with 80% of the images used for training and 20% reserved for testing. This split ensures that the model is evaluated on unseen data and helps prevent overfitting. Furthermore, to handle class imbalance (as pneumonia images are more abundant), a stratified shuffle split was employed to maintain the same proportion of classes in both training and validation sets.

Completing these preprocessing steps prepared the data for the application of our machine learning models, ensuring that they could learn effectively from the images while maintaining generalizability and performance.

2.4 Machine Learning Models and Theoretical Foundations

2.4.1 Support Vector Machine (SVM)

Developed in the 1990s by Vapnik and Cortes, SVMs are among the most widely used classification algorithms due to their strong theoretical foundation in statistical learning theory (Cortes & Vapnik, 1995).

SVMs classify data by finding an optimal hyperplane that maximizes the margin between different classes. The decision function is given by

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i K\left(x_i, x\right) + b$$

Where α_i are Lagrange multipliers, y_i are class labels, $K(x\_i,\ x)$ is a kernel function, b is the bias term. For this study, we used a linear kernel with C = 1.0.

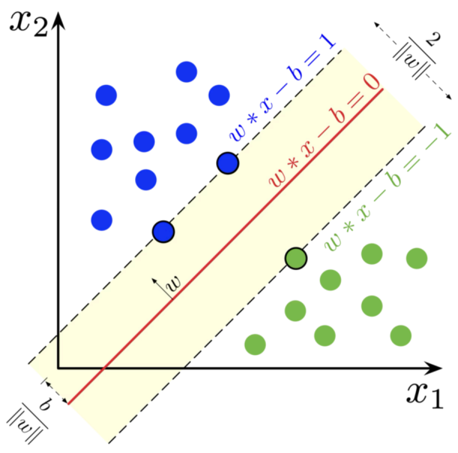


Figure 4. Separating Hyperplane

2.4.2 Feed-Forward Neural Network (FFNN)

Inspired by biological neurons, FFNNs were introduced in the 1940s by McCulloch and Pitts and later formalized by Rosenblatt's perceptron model (1958). However, it was not until the development of backpropagation (Rumelhart, Hinton, & Williams, 1986) that neural networks became practical for real-world applications.

A fully connected feed-forward network processes inputs through multiple layers of neurons, where each neuron computes

$$hi = \sigma\left(W_i x + b_i\right)$$

where W_i are weight matrices, b_i are bias terms, and σ is an activation function (ReLU in our case).

We used the following FFNN architecture, an input Layer taking in flattened pixel values from the X-ray images, two hidden layers with 64 and 32 neurons, using ReLU activation and an output layer with softmax activation for binary classification

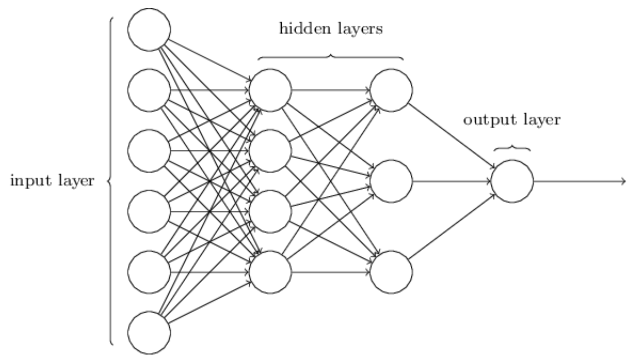


Figure 5. Feed-forward neural network

2.4.3 Convolutional Neural Network (CNN)

Introduced by LeCun et al. (1998), CNNs revolutionized computer vision by learning hierarchical feature representations directly from images. Modern CNN architectures such as AlexNet, VGG, and ResNet have demonstrated state-of-the-art performance in image classification.

CNNs use convolutional layers to detect spatial patterns in images. The core operation is:

$$z_{i,j}^{(l)} = \sum_{m=0}^{M} \sum_{n=0}^{N} W_{m,n}^{(l)} x_{i+m,j+n}^{(l-1)} + b^{(l)}$$

Where W is the filter matrix, x is the input feature map, and b is the bias term.

We used the following CNN Architecture. One convolutional Layer with 32 filters, 3×3 kernel, ReLU activation, one pooling Layer with 2×2 max pooling, a fully connected Layer with128 neurons with dropout regularization and one output layer for softmax activation

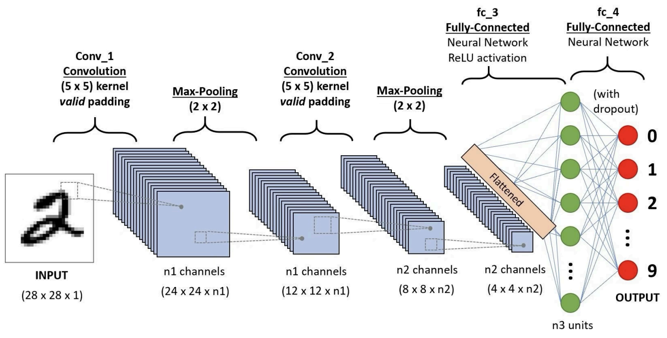
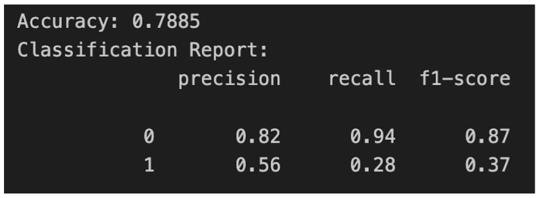


Figure 6. Convolution Neural Network

3 Outcomes

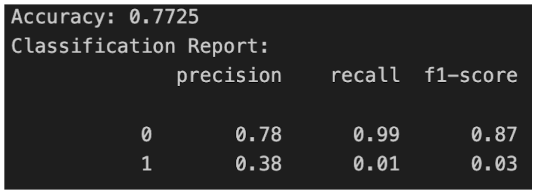
3.1 Model Performance and Computational Trade-Offs

Each of the three models was trained and evaluated using the standard implementations of SVM, FFNN, and CNN in `scikitlearn.` The metrics we have elected to explore are *accuracy, f1-score, precision,* and *recall.* We also recorded training runtime for each model. We attempted to get good measurements for resource utilization, but Python's memory management system made this seemingly simple task somewhat difficult to accomplish. We did not get a reliable measurement of resource usage for CNN.



```
Accuracy: 0.7885
Classification Report:
              precision    recall  f1-score

           0       0.82      0.94      0.87
           1       0.56      0.28      0.37
```



```
Accuracy: 0.7725
Classification Report:
              precision    recall  f1-score

           0       0.78      0.99      0.87
           1       0.38      0.01      0.03
```

Figure 7. Performance Metrics for SVM (left) and FFNN (right).

The first model, Support Vector Machine (SVM), has higher recall (0.28 vs. 0.01) and f1-score (0.37 vs. 0.03) for pneumonia, meaning it does a better job detecting cases of pneumonia. The second model, a Feed-Forward Neural Network (FFNN), has higher recall for normal cases (0.99 vs. 0.94) but almost completely fails to detect pneumonia, with a recall of only 0.01. While both models have similar accuracy (0.7885 for SVM vs. 0.7725 for FFNN), the FFNN is heavily biased toward predicting normal cases, essentially ignoring pneumonia.

This suggests that the FFNN might be overfitting to the majority class if the dataset is imbalanced or that it requires better tuning of hyperparameters or training data augmentation. Since accuracy alone is not a reliable metric in such cases, the SVM should be the preferable model as it provides a better balance between precision and recall for both normal and pneumonia cases.

3.2 Resource utilization

We recorded training time and memory utilization for each model. Python memory management is not consistent across different machine learning models, so we were not able to produce a reliable memory utilization measurement for CNN.

	Training Runtime (s)	Memory Utilization (MB)
SVM	4.20	4272
FFNN	8.16	3181
CNN	1887.74	n/a

FFNN took, as expected, longer to train than SVM. We were surprised to see that FFNN used less memory than SVM. We believe this is related to how Python manages memory while training neural networks. It does so more efficiently than when training more traditional machine learning algorithms.

4 Conclusion

This study indicates, as expected, that CNNs outperform both SVMs and FFNNs in terms of accuracy for pneumonia detection. CNNs' ability to automatically extract hierarchical features from images gives them a clear advantage in image classification tasks. However, this superior performance comes at a cost: CNNs require significantly more computational resources, including high memory usage, longer training times, and often the need for specialized hardware like GPUs.

For environments with limited hardware resources, such as mobile devices, small clinics, or real-time applications where computational efficiency is crucial, deploying CNNs may not be practical. In such cases, SVMs or smaller FFNNs present viable alternatives. While they may exhibit a slight trade-off in accuracy, they are considerably more lightweight, requiring less memory and processing power. SVMs, in particular, perform well on smaller datasets and structured features, while FFNNs can still leverage deep learning's advantages without the full computational burden of CNNs.

Ultimately, the choice between these models depends on the specific constraints of the deployment setting. If maximizing accuracy is the primary goal and computational resources are available, CNNs remain the best option. However, if real-time performance, lower memory consumption, or deployment on resource-limited hardware is a priority, SVMs or FFNNs provide practical alternatives that balance performance and feasibility.

For future work, extending the project to include a deeper dive into convolutional neural network (CNN) would provide a more direct comparison of deep learning techniques. Since CNNs are designed for image-based tasks, their performance could offer insights into whether more complex architectures justify the increased computational cost.

We would also like to integrate model interpretability techniques, such as Grad-CAM for CNNs or SHAP values for SVMs which could improve our understanding of feature importance in pneumonia detection. One area for improvement is optimizing the FNN's architecture. Experimenting with different activation functions, dropout rates, or deeper architectures could enhance performance while mitigating overfitting.

Throughout the project, group members found several aspects particularly interesting and challenging. One of the most engaging elements was understanding and comparing the performance of the support vector machine (SVM) and the feed-forward neural network (FNN) particularly when it came to the image preprocessing.   The trade-offs between model complexity, computational efficiency, and predictive accuracy provided valuable insights into real-world machine learning applications. It was particularly interesting to see how the FNN leveraged feature extraction differently from the SVM, leading to variations in performance.

References

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning, 20*(3), 273-297.

Scholkopf, B., & Smola, A. J. (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*, 533-536.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.* MIT Press.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278-2324.
European Respiratory Society. (2023, January 12). *Urgent need for increased global access to effective prevention and treatment of pneumonia*. https://www.ersnet.org/news-and-features/news/urgent-need-for-increased-global-access-to-effective-prevention-and-treatment-of-pneumonia/

Mannan, M. M. N., & Rahman, M. M. (2023). Pneumonia disease detection using chest X-rays and machine learning. *Algorithms*, 18(2), 82. https://doi.org/10.3390/a18020082
Sartorius, B., & Parker, R. (2023). Implementing the severe community-acquired pneumonia guidelines in low-resource settings. *Intensive Care Medicine*, 49, 1–3. https://doi.org/10.1007/s00134-023-07220-7

Code

This notebook contains the implementation of SVM, FFNN, and CNN to Pneumonia data set using SVM with and without PCA

In [ ]:

```
import os
import numpy as np
import pydicom
import cv2
import cv2
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
from tqdm import tqdm
```

In [ ]:

```
import cv2
```

In [ ]:

```
# setting paths
data_dir = '/Users/Lpinheiro/Desktop/MS Data Science/DSP
577/rsna-pneumonia-detection-challenge/pneumonia_data'
```

In [ ]:

```
train_images_dir = os.path.join(data_dir, 'stage_2_train_images')
```

In [ ]:

```
train_labels_file = os.path.join(data_dir, 'stage_2_train_labels.csv')
```

In [ ]:

```
# loading labels
import pandas as pd
labels = pd.read_csv(train_labels_file)
```

In [ ]:

```
# labels: 1 = pneumonia, 0 = no pneumonia
labels['Target'] = labels['Target'].apply(lambda x: 1 if x == 1 else 0)
labels = labels[['patientId', 'Target']].drop_duplicates()
```

In [ ]:

```
# image preprocessing
# this reduces the images to the desired size
# it does not flatten the images
def preprocess_images(image_dir, labels_df, img_size):
    images = []
    targets = []
    for _, row in tqdm(labels_df.iterrows(), total=len(labels_df)):
        patient_id = row['patientId']
        target = row['Target']
        image_path = os.path.join(image_dir, f'{patient_id}.dcm')

        # Load and preprocess DICOM image
        try:
            dicom = pydicom.dcmread(image_path)
            img = dicom.pixel_array
            img_resized = cv2.resize(img, (img_size, img_size))
            images.append(img_resized)
            targets.append(target)
        except Exception as e:
            print(f'Error processing {patient_id}: {e}')

    return np.array(images), np.array(targets)
```

In [ ]:

```
# PCA
# the function flattens images to a vector as required by PCA
from sklearn.decomposition import PCA


def apply_pca(images, n_components):
    # Flatten each image to 1D (assuming images have shape (samples, height, 
width, channels))
    n_samples = images.shape[0]
    flattened_images = images.reshape(n_samples, -1)  # Flatten each image to a 
1D vector

    # Apply PCA
    pca = PCA(n_components=n_components)
    reduced_images = pca.fit_transform(flattened_images)

    print(f'Dimensionality reduced to {n_components} components.')
    return reduced_images, pca
```

In [ ]:

```
# Preprocess images
img_size=128
```

In [ ]:

```
X, y = preprocess_images(image_dir=train_images_dir, labels_df=labels, 
img_size=img_size)
```

In [ ]:

```
# Apply PCA
X_pca, pca_model = apply_pca(X, n_components=100)
```

In [ ]:

```
from sklearn.model_selection import train_test_split
```

In [ ]:

```
X_subset, _, y_subset, _ = train_test_split(X_pca, y,
                                            random_state=42,
                                            stratify=y)
```

In [ ]:

```
# First, split the data into training+validation and test sets
X_train_val, X_test, y_train_val, y_test = train_test_split(
    X_subset, y_subset, test_size=0.2, random_state=31415, stratify=y_subset
)

# Now, split the training+validation set into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(
    X_train_val, y_train_val, test_size=0.2, random_state=31415,
stratify=y_train_val
)
# training, testing, and validation  split

#X_train, X_test, y_train, y_test = train_test_split(X_subset, y_subset,
test_size=0.2, random_state=31415, stratify=y_subset)
```

In [ ]:

```
# scale data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [ ]:

```
import time
import psutil
import torch

start_time = time.time()
cpu_start = psutil.cpu_percent(interval=None)
mem_start = psutil.virtual_memory().used

# Original code

# train SVM
svm_model = SVC(kernel='linear', C=1.0, random_state=31415)
svm_model.fit(X_train_scaled, y_train)


# Resource tracking
end_time = time.time()
cpu_end = psutil.cpu_percent(interval=None)
mem_end = psutil.virtual_memory().used
```

```
# GPU tracking (if available)
if torch.cuda.is_available():
    gpu_mem = torch.cuda.memory_allocated() / 1024**2
else:
    gpu_mem = "N/A"

print(f"Runtime: {end_time - start_time:.2f} seconds")
print(f"CPU Usage: {cpu_end - cpu_start:.2f}%")
print(f"Memory Used: {(mem_end - mem_start) / 1024**2:.2f} MB")
print(f"GPU Memory Used: {gpu_mem} MB")
```

In [ ]:

```
import matplotlib.pyplot as plt

# Count the occurrences of each class in the labels
class_counts = labels['Target'].value_counts()

# Plot the bar graph
plt.figure(figsize=(8, 6))
bars = plt.bar(class_counts.index, class_counts, color=['blue', 'orange'])

# Add counts on top of the bars
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, yval + 0.5, int(yval),
ha='center', va='bottom')

# Set the title and labels
plt.title('Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')

# Remove x axis scale
plt.xticks([0, 1], ['Normal', 'Pneumonia'])

# Grid only on y axis
#plt.grid(axis='y')
plt.yticks([])



# Show the plot
plt.show()
```

In [ ]:

```
# evaluate SVM
y_pred = svm_model.predict(X_test_scaled)
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Classification Report:')
print(classification_report(y_test, y_pred))
```

In [ ]:

```
# Feed Forward Neural Network (FFNN) Implementation
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
```

In [ ]:

```
# FFNN model
ffnn_model = Sequential([
    #Flatten(input_shape=(img_size, img_size)),  # Flatten image data
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')  # Binary classification output
])
```

In [ ]:

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

In [ ]:

```
import time
import psutil
import torch

start_time = time.time()
cpu_start = psutil.cpu_percent(interval=None)
mem_start = psutil.virtual_memory().used

# Original code
# fit model

history= ffnn_model.fit(X_train, y_train, epochs=10, batch_size=8,
validation_data=(X_val, y_val))


# Resource tracking
end_time = time.time()
cpu_end = psutil.cpu_percent(interval=None)
mem_end = psutil.virtual_memory().used

# GPU tracking (if available)
if torch.cuda.is_available():
    gpu_mem = torch.cuda.memory_allocated() / 1024**2

else:
    gpu_mem = "N/A"
```

```
print(f"Runtime: {end_time - start_time:.2f} seconds")
print(f"CPU Usage: {cpu_end - cpu_start:.2f}%")
print(f"Memory Used: {(mem_end - mem_start) / 1024**2:.2f} MB")
print(f"GPU Memory Used: {gpu_mem} MB")
```

In [ ]:

```
mem_end/1024**2
```

In [ ]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss Function by Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```

In [ ]:

```
# evaluate FFNN model
ffnn_loss, ffnn_acc = ffnn_model.evaluate(X_test, y_test)
print(f"FFNN Test Accuracy: {ffnn_acc:.2f}")
```

In [ ]:

```
# CNN model using keras

def build_cnn(img_size):
    model = keras.Sequential([
        layers.Conv2D(32, (3,3), activation='relu', input_shape=(img_size,
img_size, 1)),
        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.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dense(1, activation='sigmoid')  # Binary classification
    ])

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

In [ ]:

```
#Example usage:
import tensorflow as tf
```

```
from tensorflow import keras
from tensorflow.keras import layers
CNN = build_cnn(128)
```

In [ ]:

```
# First, split the data into training+validation and test sets
X_train_val, X_test, y_train_val, y_test = train_test_split(
    X, y, test_size=0.2, random_state=31415, stratify=y
)
```

In [ ]:

```
# Now, split the training+validation set into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(
    X_train_val, y_train_val, test_size=0.2, random_state=31415,
stratify=y_train_val
)
```

In [ ]:

```
import time
import psutil
import torch

start_time = time.time()
cpu_start = psutil.cpu_percent(interval=None)
mem_start = psutil.virtual_memory().used

# Original code
# This takes 17 minutes or so with 1000 images
CNN.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val,
y_val))


# Resource tracking
end_time = time.time()
cpu_end = psutil.cpu_percent(interval=None)
mem_end = psutil.virtual_memory().used

# GPU tracking (if available)
if torch.cuda.is_available():
    gpu_mem = torch.cuda.memory_allocated() / 1024**2
else:
    gpu_mem = "N/A"

print(f"Runtime: {end_time - start_time:.2f} seconds")
print(f"CPU Usage: {cpu_end - cpu_start:.2f}%")
print(f"Memory Used: {(mem_end - mem_start) / 1024**2:.2f} MB")
print(f"GPU Memory Used: {gpu_mem} MB")
```

In [ ]:

```
from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

# Get model predictions
y_pred_probs =CNN.predict(X_test)  # Probabilities
y_pred = (y_pred_probs > 0.5).astype(int)  # Convert to binary labels

# Compute confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(6,5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['No
Pneumonia', 'Pneumonia'], yticklabels=['No Pneumonia', 'Pneumonia'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()

# Print performance metrics
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Classification Report:\n', classification_report(y_test, y_pred,
target_names=['No Pneumonia', 'Pneumonia']))
```

In [4]: