Deep Learning with Tensor Flow: Image Classification

INDEPENDENT STUDY REPORT

SUBMITTED TOWARDS THE

FULFILLMENT OF THE REQUIREMENTS OF

INDEPENDENT STUDY (MASTER'S OF COMPUTER SCIENCE) BY

Prerana Mukherjee

(B01020606)

Under the guidance of

Prof. Leslie C. Lander

DEPARTMENT OF COMPUTER SCIENCE WATSON SCHOOL OF ENGINEERING
CLASS OF 2025

Overview

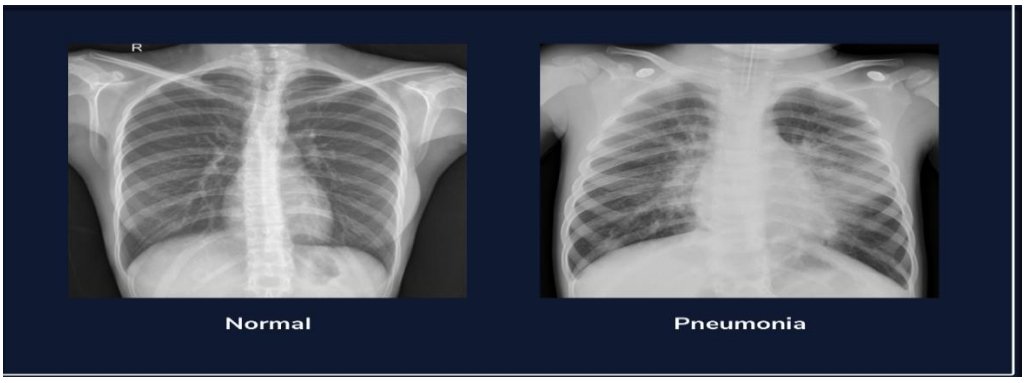
Image classification, a fundamental task in computer vision, involves assigning a label to an input image from a predefined set of categories. Leveraging the power of deep learning, TensorFlow has emerged as a leading framework for implementing and optimizing complex neural networks.

Neural networks are perfectly suited for image classification: the task of finding the complex patterns in pixels necessary to map an image to its label. As a result, image classification is a common application of deep learning.

Here, we see how neural networks can be applied to image classification. To do this, we will be using convolutional layers: layers designed to process image data by focusing on local relationships between features.

Objectives

- Understand Pneumonia and Its Impact: Gain insights into pneumonia, its symptoms, and its global health implications.
- Explore Medical Imaging: Learn about the role of chest X-rays in diagnosing pneumonia.
- Data Preparation: Acquire skills in loading, preprocessing, and augmenting medical image datasets.
- Build a CNN: Develop a Convolutional Neural Network for classifying chest X-ray images.
- Model Training: Train the CNN using appropriate techniques to ensure robust performance.
- Model Evaluation: Assess the model's accuracy, precision, recall, and F1-score.
- Deploy the Model: Understand the potential for deploying deep learning models in real-world medical applications.



Approach

Image Classification

Image classification involves finding the complex patterns in pixels necessary to map an image to its label and is a common application of deep learning.

To preprocess image data, we can use an ImageDataGenerator() from the TensorFlow library. We can augment our image data using parameters such as zoom_range and width_shift_range, among others.

To load in image data, we can use the flow_from_directory() method from the TensorFlow library to import our image directory and specify parameters, such as class_mode and color_mode.

Convolutional Neural Network (CNN)

CNNs are designed for image data and capture local relationships between nearby features in an image. Convolutional layers use filters to generate feature maps by applying filters across the input image.

Benefits of Convolutional Layers:

- Convolution can reduce the size of an input image using only a few parameters.
- Filters compute new features by only combining features that are near each other in the image. This operation encourages the model to look for local patterns (e.g., edges and objects).
- Convolutional layers will produce similar outputs even when the objects in an image are translated (for example, if there were a giraffe in the bottom or top of the frame). This is because the same filters are applied across the entire image.

Key Concepts in CNNs

- Stride Hyperparameter: Determines the step size for moving the filter across the image. A stride of 2 moves the filter two columns each time.
- Padding Hyperparameter: Defines how to handle the edges of the input image:
- Valid Padding: Stops when the kernel moves off the image.

- Same Padding: Pads the input with zeros to maintain the output size.
- Max Pooling: Reduces dimensionality by replacing patches with their maximum value, preserving important features.
- Feature Maps: Visual representations of how filters interact with the input image, highlighting areas of interest.

Implementation Details

1. Load the Dataset

   Code Snippet to load the data set:

```
# Paths to the training, validation, and test datasets
train = 'C:/Users/prera/Documents/Spring 2024/Independent Study/archive (2)/chest_xray/train'
test = 'C:/Users/prera/Documents/Spring 2024/Independent Study/archive (2)/chest_xray/test'
validation = 'C:/Users/prera/Documents/Spring 2024/Independent Study/archive (2)/chest_xray/val'
```

2. Preprocessing Image Data

   - We need to pass these X-ray images into the network, and to classify them according to their respective labels. At a high-level, this is very similar to how we would approach classifying non-image data.
   - Whereas, for our image data, our features are going to come from image pixels.
   - Each image will be 256 pixels tall and 256 pixels wide, and each pixel has a value between 0 (black) - 255 (white).

```
# Creating an ImageDataGenerator for training data with rescaling and data generators for training, validation, and test datasets
batch_size = 32  # or higher
training_generator = ImageDataGenerator(rescale=1/255)
data_train = training_generator.flow_from_directory(train, target_size=(256, 256), batch_size=batch_size)
validation_generator = ImageDataGenerator(rescale=1/255)
data_valid = validation_generator.flow_from_directory(validation, target_size=(256, 256), batch_size=batch_size)
test_generator = ImageDataGenerator(rescale=1/255)
data_test = test_generator.flow_from_directory(test, target_size=(256, 256), batch_size=batch_size)
```

```
Found 5216 images belonging to 2 classes.
Found 16 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
```

3. Data Generators
   ImageDataGenerator: This class generates batches of tensor image data with real-time data augmentation. Here, it rescales the pixel values of the images by 1/255 to normalize them.
   flow_from_directory(): This method takes the path to a directory and generates batches of augmented data. It supports sub-directory classification and labels the images accordingly.

4. Building the Convolutional Neural Network (CNN)
   tf.keras.Sequential(): A linear stack of layers where you can build a model layer by layer.
   Conv2D layers: Convolutional layers with ReLU activation functions. These layers apply filters to the input image to detect features like edges, textures, etc. The kernel_regularizer with L2 regularization helps prevent overfitting.
   MaxPooling2D: Downsamples the input along its spatial dimensions (height and width), reducing the number of parameters and computation in the network.
   Flatten(): Converts the 2D matrix data to a vector to connect it to the fully connected layers.
   Dense layers: Fully connected layers where each neuron is connected to every neuron in the previous layer. The final Dense layer with a sigmoid activation function is used for binary classification.

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_14 (Conv2D)	(None, 254, 254, 16)	448
max_pooling2d_7 (MaxPooling2D)	(None, 127, 127, 16)	0
conv2d_15 (Conv2D)	(None, 125, 125, 32)	4,640
max_pooling2d_8 (MaxPooling2D)	(None, 62, 62, 32)	0
conv2d_16 (Conv2D)	(None, 60, 60, 64)	18,496
max_pooling2d_9 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_17 (Conv2D)	(None, 28, 28, 128)	73,856
flatten_3 (Flatten)	(None, 100352)	0
dense_6 (Dense)	(None, 64)	6,422,592
dense_7 (Dense)	(None, 2)	130

5. Compiling the Model
   Adam optimizer: An adaptive learning rate optimization algorithm that's designed to handle sparse gradients on noisy problems.
   binary_crossentropy: A loss function for binary classification problems.
   Metrics: acc for accuracy and f1_score for evaluating the balance between precision and recall.

```
# Compiling the model with Adam optimizer, binary crossentropy loss, and accuracy & F1 score metrics
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),loss='binary_crossentropy',metrics=['acc','f1_score'])
model.summary()
```

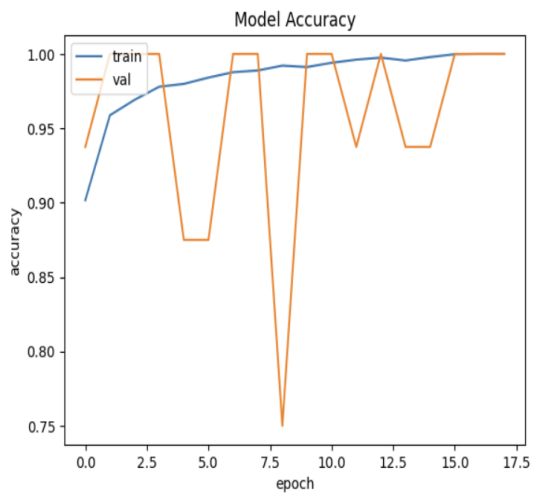
6. Training the Model
   EarlyStopping: A callback that stops training when a monitored metric (validation loss here) has stopped improving to prevent overfitting.
   model.fit(): Trains the model for a fixed number of epochs on the provided data.

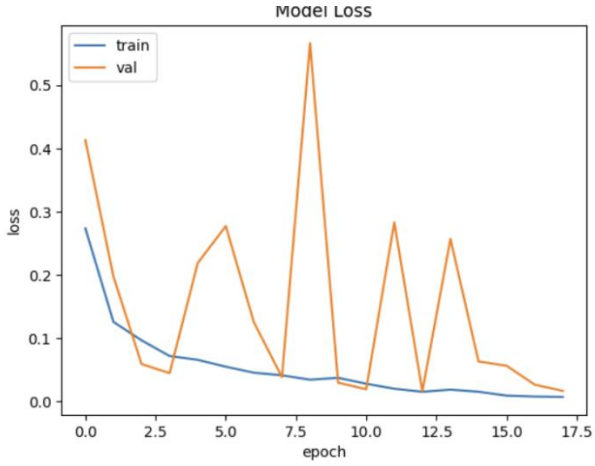
```
# Defining early stopping callback to prevent overfitting
callbacks = tf.keras.callbacks.EarlyStopping(monitor='val_loss',patience=5,verbose=1)
```

```
# Training the model
history = model.fit(
    data_train,
    epochs=20,
    validation_data=data_valid,
    callbacks=[callbacks]
)
```

```
Epoch 1/20
C:\Users\prera\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning: Your `
yDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`.
o not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
163/163 ━━━━━━━━━━━━━━━━━━━━ 144s 837ms/step - acc: 0.8182 - f1_score: 0.7091 - loss: 0.4569 - val_acc: 0.9375 - val_f1_score: 0.9373 - val_loss: 0.4129
Epoch 2/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 77s 457ms/step - acc: 0.9638 - f1_score: 0.9518 - loss: 0.1070 - val_acc: 1.0000 - val_f1_score: 1.0000 - val_loss: 0.1974
Epoch 3/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 77s 456ms/step - acc: 0.9652 - f1_score: 0.9547 - loss: 0.1091 - val_acc: 1.0000 - val_f1_score: 1.0000 - val_loss: 0.0593
Epoch 4/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 76s 454ms/step - acc: 0.9760 - f1_score: 0.9687 - loss: 0.0768 - val_acc: 1.0000 - val_f1_score: 1.0000 - val_loss: 0.0448
Epoch 5/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 73s 436ms/step - acc: 0.9840 - f1_score: 0.9786 - loss: 0.0573 - val_acc: 0.8750 - val_f1_score: 0.8730 - val_loss: 0.2188
Epoch 6/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 71s 419ms/step - acc: 0.9801 - f1_score: 0.9740 - loss: 0.0625 - val_acc: 0.8750 - val_f1_score: 0.8730 - val_loss: 0.2772
Epoch 7/20
163/163 ━━━━━━━━━━━━━━━━━━━━ 75s 446ms/step - acc: 0.9910 - f1_score: 0.9882 - loss: 0.0378 - val_acc: 1.0000 - val_f1_score: 1.0000 - val_loss: 0.1252
Epoch 8/20
```

7. Plotting Training History
   The training history, including accuracy, loss, and F1 score for both training and validation sets, is plotted using matplotlib to visualize the performance of the model over epochs.





	precision	recall	f1-score	support
NORMAL	0.38	0.15	0.21	234
PNEUMONIA	0.63	0.86	0.72	390
accuracy			0.59	624
macro avg	0.50	0.50	0.47	624
weighted avg	0.53	0.59	0.53	624

8. Model Evaluation
   model.evaluate(): Evaluates the model on the test data.
   model.predict(): Generates output predictions for the input samples.

```
# Generating predictions on the test data
pred = model.predict(data_test)
pred = np.argmax(pred, axis=1)

# Mapping class indices to labels
labels = data_test.class_indices
labels = {v: k for k, v in labels.items()}

# Assigning predicted labels to the predictions
pred = [labels[k] for k in pred]
```

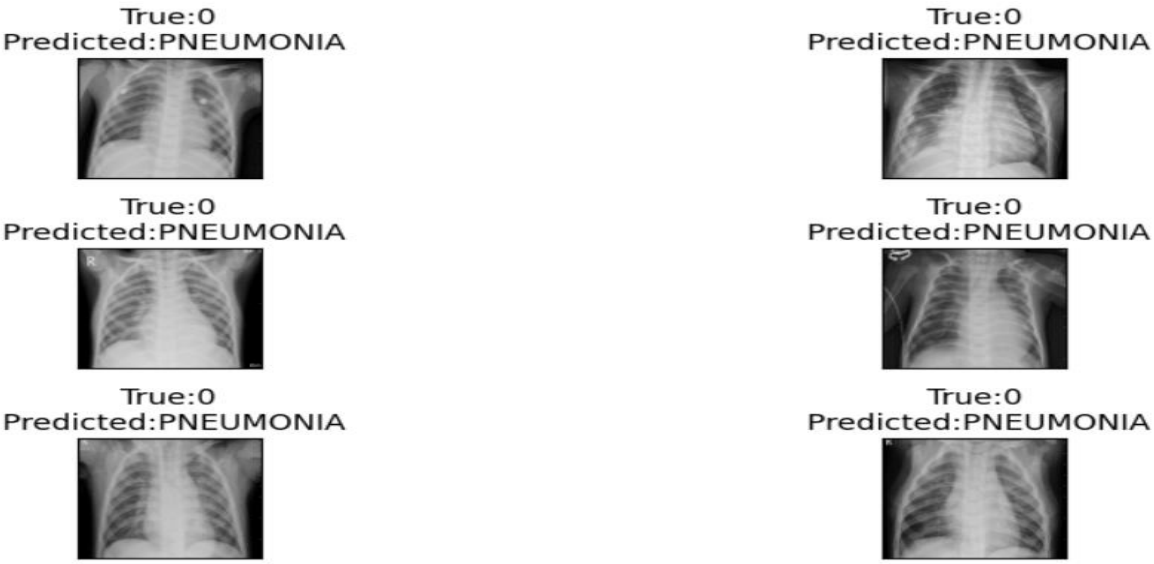
```
20/20 ━━━━━━━━━━━━━━━━━━━━ 6s 286ms/step
```

```
# Printing predictions
print(pred)
```

```
['PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'NORMAL', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA'
'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'NORMAL', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA',
'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'NORMAL', 'PNEUMONIA', 'PNEUMONIA', 'NORMAL', 'PNEUMONIA', 'NORMAL', 'NORMAL', 'NORMAL'
'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONI
A', 'NORMAL', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'NORMAL', 'PNEUMONIA', 'PNEUMONIA', 'NORMAL', 'PNEUMONIA', '
NEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'NORMAL', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', '
NEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'NORMAL', '
NEUMONIA', 'PNEUMONIA', 'NORMAL', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'NORMAL', 'PNE
MONIA', 'NORMAL', 'NORMAL', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'NORMAL', 'NORMAL', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'NORMAL', 'NO
MAL', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'NORMAL', 'PNEUM
NIA', 'NORMAL', 'PNEUMONIA', 'PNEUMONIA', 'NORMAL', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONI
A', 'PNEUMONIA', 'NORMAL', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'NORMAL', 'PNEUMONIA', 'PNEUMONIA'
```

9. Classification Report
   classification_report: Generates a report showing the main classification metrics: precision, recall, and F1-score.

10. Visualization
    The code visualizes a batch of test images along with their true and predicted labels using matplotlib.



References

1. Dataset: https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia/data
2. https://www.codecademy.com/enrolled/courses/deep-learning-with-tensor-flow-image-classification

Certificate