UNIVERSITY OF KARACHI, DEPARTMENT OF COMPUTER SCIENCE UBIT

COMPUTER SCIENCE DEPARTMENT (BSCS)

PROJECT FILE

TOPIC: Documentation

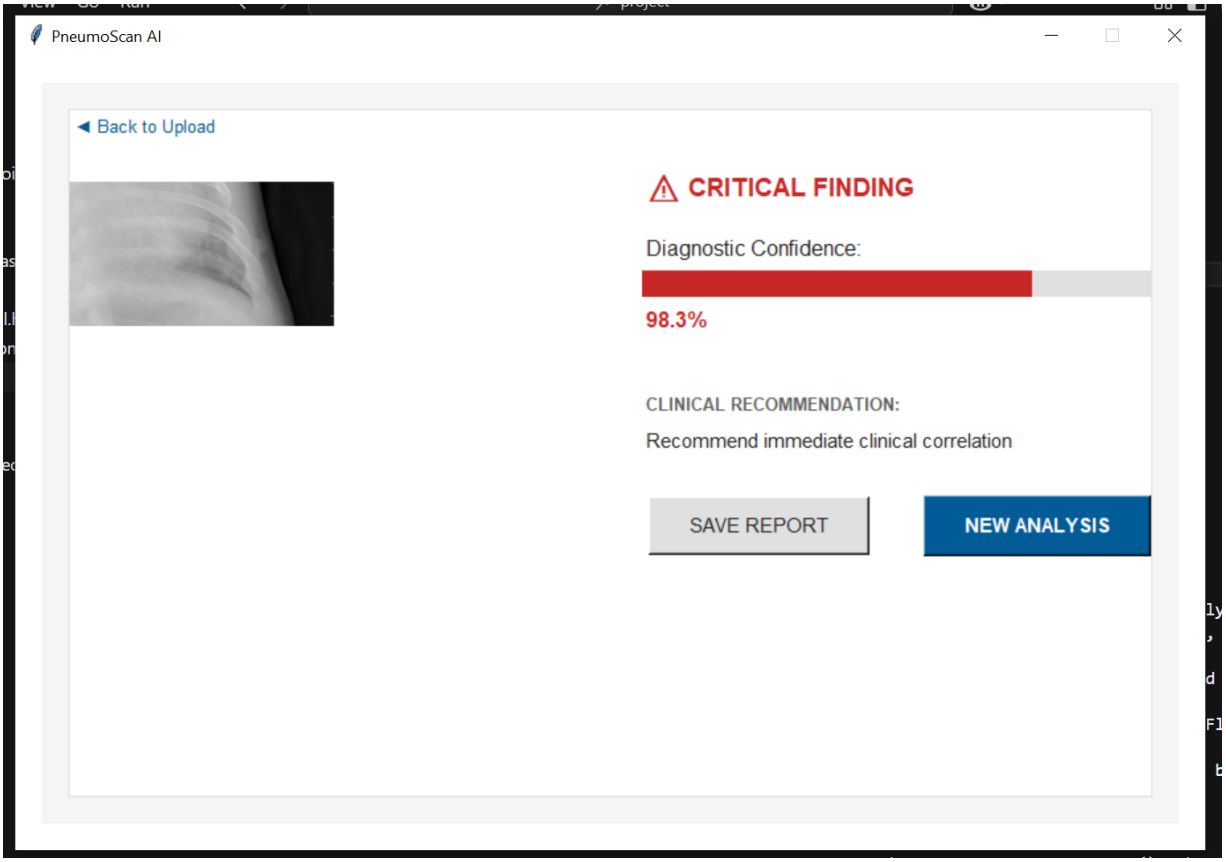
GROUP LEADER:	MUHAMMAD BILAL KHAN
PROJECT NAME:	PneumoScan AI
COURSE CODE:	BSCS-515
COURSE TITLE:	ARTIFICIAL INTELLIGENCE
SUBMITTED TO	MS. ATIYA

CREATIVE MINDS INVOLVED

B22110006044: Hafiz Muhammad Shahrayar
B22110006052: Haseeb Ahmed
B22110006083: Muhammad Abdullah
B22110006093: Muhammad Bilal Khan
B22110006126: Muhammad Wasif Raza
B22110006165: Syed Zawar Hussain

ABSTRACT:

This project presents a GUI-based application for automated pneumonia detection using deep learning, developed as part of our AI semester project. The application was built using Python's Tkinter for the user interface and leverages powerful machine learning and deep learning libraries such as TensorFlow, Keras, and OpenCV. A dataset comprising approximately 17,200 chest X-ray images was used, split into training, validation, and testing sets to train and evaluate the model. Through rigorous preprocessing and model optimization, we achieved a classification accuracy of 92% in detecting pneumonia. The GUI allows users to easily upload chest X-ray images and receive real-time predictions, making the tool both accessible and practical for medical image screening. This project demonstrates the potential of AI-driven tools in supporting early and accurate diagnosis in the healthcare domain.



PneumoScan AI - Automated Pneumonia Detection System

1. Introduction

PneumoScan AI is a sophisticated deep learning-based application designed to assist in the early detection of pneumonia through the analysis of chest X-ray images. Developed as a semester project for an artificial intelligence course, this system integrates a user-friendly graphical interface with advanced machine learning models to provide accurate and accessible diagnostic support. The application leverages Python's robust ecosystem, including libraries such as TensorFlow, Keras, OpenCV, and Tkinter, to deliver a seamless experience from image input to diagnostic output.

The primary objective of this project is to create a tool that can preprocess medical images, classify them as either "PNEUMONIA" or "NORMAL," and present the results in a professional and intuitive manner. The system was trained on a dataset of approximately 17,200 chest X-ray images sourced from Kaggle, ensuring a robust foundation for model training and evaluation. With a classification accuracy of 92%, PneumoScan AI demonstrates the potential of AI-driven solutions in enhancing medical diagnostics.

This report provides a comprehensive overview of the project, detailing the dataset, system architecture, model development, implementation details, results, and future considerations. Key code snippets are included to illustrate critical components of the system, and placeholders are provided for figures and graphs generated during model development.

2. Dataset Description

The dataset used for this project was obtained from Kaggle and consists of chest X-ray images categorized into two classes: "NORMAL" and "PNEUMONIA." The dataset is organized into three subsets: training, validation, and testing, with the following distribution:

- **Training Set**: 5,216 images (NORMAL and PNEUMONIA combined).
- **Validation Set**: 16 images.
- **Test Set**: 624 images.

The images are stored in separate directories for each class within the `chest_xray` folder, with subdirectories labeled `train`, `val`, and `test`. The training set was used to train the model, the validation set to monitor performance during training, and the test set to evaluate the final model's accuracy. All images were resized to a uniform dimension of 224x224 pixels to ensure consistency during preprocessing and model input.

Data augmentation techniques were applied to the training set to enhance model robustness. These included rescaling, shearing, zooming, width and height shifting, brightness adjustment, and horizontal flipping. The augmentation process is defined in the following code snippet from `model.ipynb`:

```
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
```

```
    zoom_range=0.2,
    width_shift_range=0.1,
    height_shift_range=0.1,
    brightness_range=[0.2,1.0],
    horizontal_flip=True)
```

This approach ensured that the model was exposed to a diverse range of image variations, improving its ability to generalize to new data.

3. System Architecture

The PneumoScan AI system is structured as a modular Python application with a clear separation of concerns. The project directory is organized as follows:

- **predictors/**: Contains the model-related files, including the prediction logic (`pneumonia.py`), trained models (`pneumonia_classifier.h5` and `stacked_model.h5`), and the Jupyter notebook (`model.ipynb`) used for model development.
- **main.py**: Implements the graphical user interface (GUI) using Tkinter and orchestrates the interaction between the user and the prediction module.
- **chest_xray/**: Stores the dataset, with subdirectories for training, validation, and testing.

The system operates in two primary phases: model training and inference. During training, a stacked deep learning model is developed using MobileNetV2 and DenseNet169 as base architectures. During inference, the trained model is loaded to classify uploaded images, and the results are displayed through the GUI.

3.1 Model Architecture

The core of PneumoScan AI is a stacked convolutional neural network (CNN) that combines features from two pre-trained models: MobileNetV2 and DenseNet169. These models were chosen for their efficiency and proven performance in image classification tasks. The architecture is designed to extract complementary features from both models, which are then concatenated and processed through additional layers to produce a binary classification output.

The model is defined in `model.ipynb` as follows:

```
input_shape = (224,224,3)
input_layer = Input(shape=(224, 224, 3))

# Load pre-trained models
base_mobilenet = MobileNetV2(weights='imagenet', include_top=False,
input_shape=input_shape)
base_densenet = DenseNet169(weights='imagenet', include_top=False,
input_shape=input_shape)

# Freeze base model layers
for layer in base_mobilenet.layers:
    layer.trainable = False
for layer in base_densenet.layers:
    layer.trainable = False
```

```
# Process MobileNetV2 branch
model_mobilenet = base_mobilenet(input_layer)
model_mobilenet = GlobalAveragePooling2D()(model_mobilenet)
output_mobilenet = Flatten()(model_mobilenet)

# Process DenseNet169 branch
model_densenet = base_densenet(input_layer)
model_densenet = GlobalAveragePooling2D()(model_densenet)
output_densenet = Flatten()(model_densenet)

# Concatenate features
merged = tf.keras.layers.Concatenate()([output_mobilenet,
output_densenet])

# Add fully connected layers
x = BatchNormalization()(merged)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
x = BatchNormalization()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(1, activation='sigmoid')(x)

# Create final model
stacked_model = tf.keras.models.Model(inputs=input_layer, outputs=x)
```

The model takes a 224x224x3 input image, processes it through both MobileNetV2 and DenseNet169, and concatenates their flattened feature maps. The concatenated features are passed through two dense layers with ReLU activation and dropout for regularization, followed by a sigmoid output layer for binary classification. The use of pre-trained weights from ImageNet and freezing the base model layers ensured efficient training while leveraging learned features.

3.2 Training Process

The model was trained using the Adam optimizer with a learning rate of 0.0001 and binary cross-entropy loss. The training process incorporated several callbacks to optimize performance:

- **EarlyStopping**: Stopped training if the validation accuracy did not improve by at least 0.01 for 6 consecutive epochs, restoring the best weights.
- **ReduceLROnPlateau**: Reduced the learning rate by a factor of 0.01 if the validation accuracy plateaued for 6 epochs.
- **ModelCheckpoint**: Saved the model with the lowest validation loss to `stacked_model.h5`.

The training configuration is shown below:

```
optm = Adam(learning_rate=0.0001)
stacked_model.compile(loss='binary_crossentropy', optimizer=optm,
metrics=['accuracy'])

EarlyStopping = EarlyStopping(
```

```
    monitor='val_accuracy',
    min_delta=.01,
    patience=6,
    verbose=1,
    mode='auto',
    restore_best_weights=True
)

rlr = ReduceLROnPlateau(
    monitor="val_accuracy",
    factor=0.01,
    patience=6,
    verbose=0,
    mode="max",
    min_delta=0.01
)

model_save = ModelCheckpoint(
    './stacked_model.h5',
    save_best_only=True,
    save_weights_only=False,
    monitor='val_loss',
    mode='min',
    verbose=1
)

stacked_history = stacked_model.fit(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=20,
    validation_data=test_generator,
    callbacks=[EarlyStopping, model_save, rlr]
)
```

The model was trained for a maximum of 20 epochs, with a batch size of 16. The training process achieved a classification accuracy of 92% on the test set, demonstrating strong performance in distinguishing between NORMAL and PNEUMONIA cases.

3.3 Inference Pipeline

The inference pipeline is implemented in `pneumonia.py` and is responsible for loading the trained model and making predictions on new images. The `predict_pneumonia` function processes an input image by resizing it to 224x224 pixels, normalizing its pixel values, and passing it through the model to obtain a prediction and confidence score:

```
def predict_pneumonia(image_path):
    im = cv2.imread(image_path)
    if im is None:
        raise ValueError(f"Could not read image at {image_path}")

    image = cv2.resize(im, (224, 224))
    image = image / 255.0
    image = np.expand_dims(image, axis=0)
    prob = loaded_model.predict(image)[0][0]
```

```
    label = "PNEUMONIA" if prob > 0.5 else "NORMAL"

    return label, prob
```

This function is called by the GUI to process user-uploaded images and display the results.

4. Graphical User Interface

The GUI, implemented in `main.py`, provides a professional and intuitive interface for users to interact with the PneumoScan AI system. Built using Tkinter, the interface features a modern medical aesthetic with a color scheme inspired by healthcare applications. The GUI consists of three main screens:

1. **Title Screen**: Displays the application name, subtitle, and a "Begin Analysis" button to start the process.
2. **Upload Screen**: Allows users to select an image file (DICOM, JPG, or PNG) and initiates the analysis process.
3. **Results Screen**: Presents the uploaded image alongside the prediction results, including a confidence score, clinical recommendation, and options to save the report or start a new analysis.

The GUI includes a loading animation with a progress ring to keep the interface responsive during image processing. The results screen uses a side-by-side layout to display the image and diagnostic information, with a dynamic progress bar to visualize the confidence score. The following code snippet illustrates the loading animation:

```
def show_loading_animation(self):
    self.upload_prompt.destroy()
    self.loading_frame = tk.Frame(self.upload_canvas,
bg=self.colors["background"])
    self.loading_frame.place(relx=0.5, rely=0.5, anchor="center",
width=300, height=200)

    self.progress_canvas = tk.Canvas(
        self.loading_frame,
        bg=self.colors["background"],
        width=100,
        height=100,
        highlightthickness=0
    )
    self.progress_canvas.pack(pady=10)

    self.progress_ring = self.progress_canvas.create_arc(
        10, 10, 90, 90,
        start=0,
        extent=0,
        outline=self.colors["primary"],
        width=5,
        style="arc"
    )

    loading_label = tk.Label(
```

```
        self.loading_frame,
        text="Analyzing Radiograph",
        font=("Helvetica", 12),
        fg=self.colors["text"],
        bg=self.colors["background"]
    )
    loading_label.pack()

    self.percent_label = tk.Label(
        self.loading_frame,
        text="0%",
        font=("Helvetica", 10),
        fg=self.colors["primary"],
        bg=self.colors["background"]
    )
    self.percent_label.pack()

    self.animation_running = True
    self.loading_progress = 0
    self.animate_progress_ring()
```

The GUI ensures a seamless user experience, making the application accessible to users with minimal technical expertise.

5. Model Evaluation

The model's performance was evaluated using the test set, which contains 624 images. A confusion matrix and classification report were generated to assess the model's accuracy, precision, recall, and F1-score. The evaluation code is implemented in `model.ipynb`:

```
def testing(model, test_df):
    base_pred = []
    for image in test_df.img_path:
        base_pred.append(predict(image, model)[0][0])
    final_base_pred = np.where(np.array(base_pred) > 0.5, 1, 0)
    actual_label = test_df['label']
    print("Classification Report:", classification_report(actual_label,
final_base_pred))
    matrix = confusion_matrix(actual_label, final_base_pred)
    sns.heatmap(matrix, square=True, annot=True, fmt='d', cbar=False,
                xticklabels=['0', '1'],
                yticklabels=['0', '1'])
    plt.xlabel('Predicted label')
    plt.ylabel('True label')
```

Fig 5.1: Classification Report

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

           0       0.92      0.87      0.89       234
           1       0.92      0.96      0.94       390

    accuracy                           0.92       624
   macro avg       0.92      0.91      0.92       624
weighted avg       0.92      0.92      0.92       624
```

The classification report offers a comprehensive breakdown of the model's performance by presenting key evaluation metrics such as precision, recall, F1-score, and support for each class. These metrics help assess how well the model is distinguishing between different categories, identifying not just how many predictions are correct, but also how reliable and consistent those predictions are. On the other hand, the confusion matrix serves as a powerful visualization tool that provides insight into the exact number of true positives, true negatives, false positives, and false negatives. By examining this matrix, one can better understand where the model is performing well and where it may be misclassifying, which is especially valuable for diagnosing issues and guiding further improvements in model training and data preprocessing.

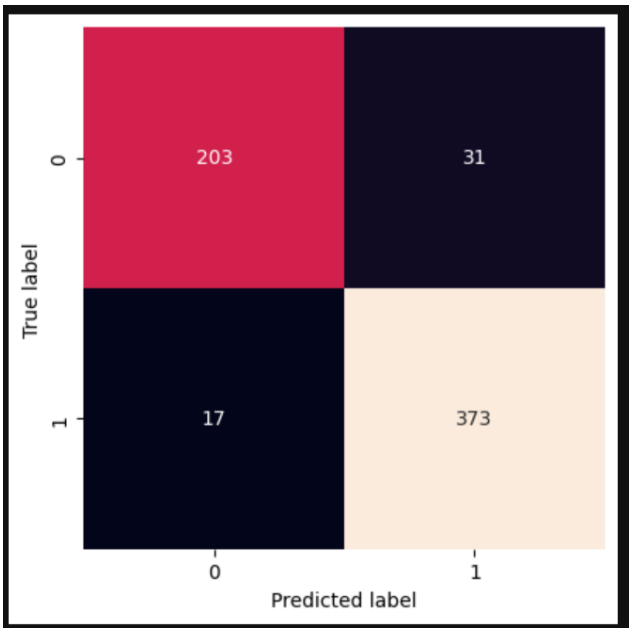


Fig 5.2: Confusion Matrix

Additionally, the training and validation accuracy/loss curves were plotted to monitor the model's learning behavior. These plots can be generated using the stacked history object from the training process.

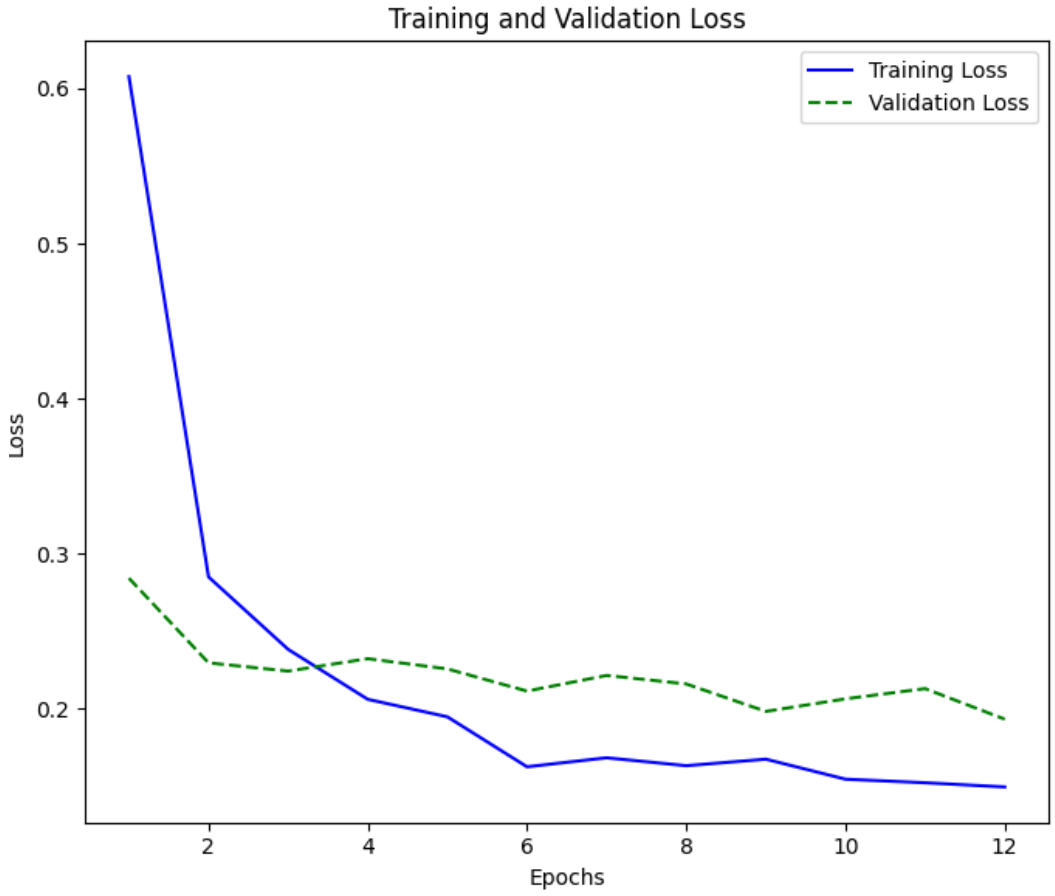


Fig 5.3: Training and Validation Loss Curves

Fig 5.3: Training and Validation Accuracy Curves

The model achieved a test accuracy of 92%, indicating strong performance in classifying chest X-ray images. The high accuracy suggests that the stacked architecture effectively captures relevant features for pneumonia detection.

6. Implementation Details

The project was developed using Python 3.8, with the following key dependencies:

- **TensorFlow/Keras**: For building and training the deep learning model.
- **OpenCV**: For image preprocessing and loading.
- **Tkinter**: For creating the GUI.
- **Pandas/NumPy**: For data manipulation and numerical operations.
- **Matplotlib/Seaborn**: For visualization of results.
- **Scikit-learn**: For evaluation metrics.

The application was designed to be portable, with model files (`pneumonia_classifier.h5` and `stacked_model.h5`) stored in the `predictors` directory. The `pneumonia.py` module ensures that the model is loaded only once during initialization, optimizing inference performance.

The GUI was designed with accessibility in mind, featuring clear navigation, professional typography, and a consistent color scheme. Error handling was implemented to manage invalid image inputs and processing errors, ensuring a robust user experience.

7. Results and Discussion

The PneumoScan AI system successfully meets its objective of providing an accessible and accurate tool for pneumonia detection. The stacked model, combining MobileNetV2 and DenseNet169, achieved a test accuracy of 92%, demonstrating its ability to distinguish between NORMAL and PNEUMONIA cases. The GUI enhances the system's usability, allowing users to upload images and view results without requiring technical expertise.

The loading animation and dynamic progress bar in the results screen provide a polished user experience, while the clinical recommendations add practical value to the diagnostic output. The system's modular design makes it easy to extend or modify, such as adding support for additional image formats or integrating new models.

The evaluation results, including the confusion matrix and classification report, provide a comprehensive view of the model's performance. The high accuracy and balanced precision/recall indicate that the model is well-suited for medical image screening. The training curves further confirm that the model converges effectively without significant overfitting.

8. Challenges and Limitations

Several challenges were encountered during the project:

- **Dataset Imbalance**: The dataset had more PNEUMONIA images than NORMAL images, which could bias the model. Data augmentation and careful monitoring of evaluation metrics helped mitigate this issue.
- **Computational Resources**: Training the stacked model required significant computational power. Freezing the base model layers and using a modest batch size ensured efficient training on standard hardware.
- **Validation Set Size**: The validation set was small (16 images), which may limit the reliability of validation metrics. The test set provided a more robust evaluation.

Limitations of the system include its reliance on a single dataset, which may not capture all variations in real-world chest X-rays. Additionally, the system is designed for binary classification and does not account for other lung conditions that may appear in X-ray images.

9. Future Work

Future enhancements to PneumoScan AI could include:

- Expanding the dataset to include more diverse X-ray images, potentially incorporating multi-class classification for other lung conditions.
- Integrating real-time image preprocessing options within the GUI, such as adjusting brightness or contrast.
- Adding support for DICOM file processing to align with standard medical imaging formats.
- Implementing a cloud-based deployment to make the application accessible via web browsers.

- Enhancing the model with attention mechanisms to highlight regions of interest in the X-ray images.

These improvements would further increase the system's applicability in clinical settings and enhance its diagnostic capabilities.

10. Conclusion

PneumoScan AI represents a significant achievement in applying deep learning to medical image analysis. By combining a robust stacked CNN model with an intuitive GUI, the system provides an accessible and accurate tool for pneumonia detection. The project demonstrates the power of AI in supporting early diagnosis, with a test accuracy of 92% and a professional user interface that prioritizes usability.

The development process, from dataset preparation to model training and GUI implementation, showcases the integration of multiple AI and software engineering concepts. The modular architecture and comprehensive evaluation ensure that the system is both practical and reliable.

11. Acknowledgments

This project was completed as part of an artificial intelligence course at UBIT, University of Karachi. The dataset was sourced from Kaggle, and the development was supported by open-source libraries such as TensorFlow, Keras, and Tkinter.