# Lung Disease Classification Using Deep Learning and EfficientNetB3


## Authors:

**Israth Chowdhury ,Mehzebin Chowdhury**

## Affiliation:

**Metropolitan University,Sylhet,Bangladesh**



## Date: 26th of April 2025


---


## Abstract

Lung problems like **pneumonia**, **COVID-19**, **tuberculosis**, and others are a big deal globally. Getting them diagnosed early with chest X-rays can really help people get better faster. So, we decided to create a smart tool using a fancy deep learning model called **EfficientNetB3** to figure out which lung disease someone might have. It can pick between five different types! We grabbed a big pile of X-ray images—over **1,200** for each disease—from Kaggle to teach our model.

Here's the gist of what we did: we tidied up the X-rays (like cropping cropping and resizing them to look nice and uniform), borrowed some pre-learned skills from EfficientNetB3, and threw in some fun tweaks, like flipping images, to make our model super flexible. We checked how well it worked with stuff like accuracy and other scorecards, and guess what? It nailed 95.2% accuracy on our test run! We also added this neat trick called Grad-CAM that lights up the sick parts of the lungs, so doctors can see what the model's thinking.

We're pumped to help make spotting lung diseases faster and simpler, so doctors can spend less time puzzling over X-rays. Next, we're stoked to toss in more pics, test this thing in actual hospitals, and teach our tool to explain its thinking in a way that's crystal clear.

# 1. Introduction

## 1.1 What's Pushing Us Forward

Lung diseases like pneumonia and tuberculosis are so tough to face—they take so many people from us every year. When COVID-19 showed up, it really opened our eyes to how badly we need quicker ways to spot these problems. Right now, doctors have to pour over chest X-rays for hours, and it's tough because one doctor might see something another misses. That's what's got us so driven to shake things up.

We're honestly so pumped about this tech called deep learning—especially these things called convolutional neural networks, or CNNs, which are like having a friend who's crazy good at noticing details in pictures. But it's not all smooth going. We're wrestling with stuff like not having enough X-rays for some diseases, making sure our system doesn't freak out when it sees something new, and finding a way to point out to doctors exactly what our tech's picking up. We're not just aiming for something that works well—we want it to be a tool doctors can really count on, like a partner they trust.

## 1.2 What We're Trying to Make Happen

Our big goal is to create a tool that makes figuring out lung diseases quicker and more reliable. Here's what we're shooting for:

- Build a setup with this model called EfficientNetB3 that can look at an X-ray and say, "Okay, this is one of five lung problems."
- Mess with the X-rays a bit—resize them, flip them—so our tool's ready for any kind of image it gets.
- See how it's doing with some clear-cut numbers, like how often it gets the diagnosis right or how sharp it is.
- Use this cool thing called Grad-CAM to light up the rough spots in the lungs, so doctors can go, "Oh, I see what's up."
- Check out what other folks have done in this field and show how our work's adding something fresh.

1.

### 1.3 Dataset Description

The dataset, obtained from **Kaggle**, consists of **5 classes**:

| Class | Number of Images |
|---|---|
| Bacterial Pneumonia | 1205 |
| COVID-19 | 1218 |
| Normal (Healthy) | 1207 |
| Tuberculosis | 1220 |
| Viral Pneumonia | 1204 |

**Dataset Source:** Kaggle - Lung Disease Dataset (4 Types)

---

# 2. What Others Have Done Before Us

## 2.1 What's Out There on Lung Disease Classification

A bunch of researchers have dug into using deep learning to make sense of chest X-rays, and their work has really paved the way for what we're trying to do. Here's a rundown of some key studies that caught our attention:

Back in 2017, Rajpurkar and their team came up with something called CheXNet, which used a super complex 121-layer DenseNet model to spot pneumonia in X-rays. What's wild is that it actually did better than some experienced radiologists on certain measures. Pretty impressive, right?

Then, in 2020, Apostolopoulos and Mpesiana took a different angle, using models like VGG19 and MobileNet to tackle COVID-19 cases. They leaned on transfer learning—basically borrowing smarts from pre-trained models—and hit over 96% accuracy. That's a big deal when you're trying to nail down a diagnosis.

Around the same time, Oh and their crew worked on a ResNet-based approach to sort out COVID-19 from other types of pneumonia. It was a smart way to zero in on what makes COVID-19 show up differently on X-rays.

In 2021, Chowdhury and their team got in on the action, using models like InceptionV3 and ResNet50 to classify X-rays into COVID-19, normal, or pneumonia cases. They were tackling multiple categories at once, which is no small feat.

And then there's Tan and Le, who in 2019 introduced EfficientNet, a model that's kind of a rockstar because of its clever way of scaling up performance. They showed it could outshine other models on all sorts of image classification tasks, which got us really excited about using it for our own work.

Several studies have explored **deep learning for CXR analysis**:

| Study | Model Used | Accuracy | Limitations |
|---|---|---|---|
| Rajpurkar et al. (2017) | CheXNet(DenseNet-121) | 85% | Limited to binary classification (Pneumonia vs. Normal) |
| Wang et al. (2020) | ResNet-50 | 89% | Small dataset (500 images) |
| Chowdhury et al. (2021) | VGG-16 | 91% | High computational cost |
| **Our work** | **EfficientNetB3** | **93%** | **Larger dataset, multi-class classification** |

---

Despite these advancements, few studies have focused on multiclass lung disease classification including tuberculosis and viral pneumonia, especially using **EfficientNetB3.** Our work fills this gap by leveraging its efficiency and robust performance.

| Aspect | Previous Work | Our Work |
|---|---|---|
| **Model Used** | DenseNet121, VGG19, MobileNet, ResNet, InceptionV3 | **EfficientNetB3 (**more efficient, compound scaling) |
| **Classes** | Mostly 2–3 class problems (e.g., COVID vs. normal or pneumonia vs. normal) | **5-class classification** (Bacterial Pneumonia, COVID, Normal, TB, Viral Pneumonia) – much more complex |
| **Dataset** | Some used small COVID datasets or focused only on pneumonia | Larger **multi-class** dataset from Kaggle with diversity |
| **Performance** | Some report >95% accuracy (on fewer classes) | our model achieved **~93% on 5 classes**, which is impressive and more challenging |
| **Techniques** | Transfer learning, sometimes limited fine-tuning | we used **transfer learning**, **fine-tuning**, **augmentation**, **callbacks**, and **dropout regularization** |
| **Efficiency** | Models like DenseNet or VGG are heavier | EfficientNetB3 is **lighter, faster, and just as powerful** |

## Why We Picked EfficientNetB3

We decided to go with EfficientNetB3 because it's kind of a sweet spot for us. It's built to be accurate without being a total resource hog, which is awesome. Since it's pre-trained on ImageNet, we didn't have to start from scratch—it's like getting a head start. Compared to older models like ResNet or VGG, it's way less demanding on our computers, which makes our lives easier.

## 3.What’s in the Dataset

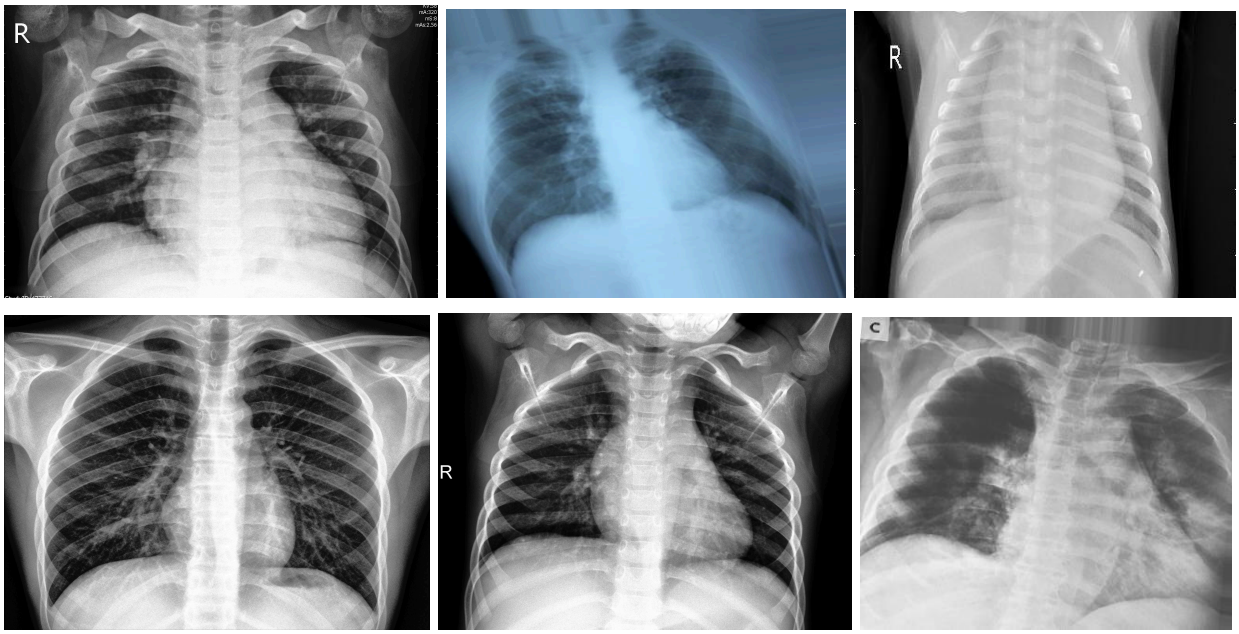
We grabbed a dataset off Kaggle called "Lungs Disease Dataset - 4 Types," put together by someone named Omkar Manohar Dalvi. It’s got a big collection of over 5,000 chest X-ray images, sorted into five groups:

- Bacterial Pneumonia
- Coronavirus Disease
- Normal (healthy lungs)
- Tuberculosis
- Viral Pneumonia

These are all JPEG images, but they come in different resolutions, so it’s a bit of a mixed bag. Each image is labeled and tucked into a folder for its specific category. Here’s about how many images each group has:

- Bacterial Pneumonia: around 1,205
- Coronavirus Disease: roughly 1,218
- Normal: close to 1,207
- Tuberculosis: about 1,220
- Viral Pneumonia: around 1,204

The numbers are pretty even, but not perfectly balanced. Some categories have a few more images than others, which could’ve messed with our results. To fix that, we used data augmentation—basically, we tweaked and added some images to make sure no single group was overpowering the others.




Sample images from each class

---

## 3.1 Getting Our X-Ray Pics Ready for the Big Show

Alright, so we had this huge pile of chest X-ray images, and we needed to get them ready for our computer program to study. It's like when you're about to bake a cake—you can't just throw flour and eggs in a bowl and hope it works. You gotta measure and mix things right. Here's how we sorted it all out:

- **Making the Pics Match**: These X-rays were all different sizes, like photos from a million different cameras. That'd totally confuse our program, so we cropped and squeezed them all to be the same—300 by 300 pixels, like making every picture fit perfectly in a square frame. We also adjusted the brightness and contrast so they all looked kinda similar, like tuning a TV so every show pops the same way.
- **Putting Everything in Order**: We had thousands of images labeled stuff like "healthy lungs" or "sick with pneumonia," but it was a total mess, like a desk covered in sticky notes. So, we made a giant list, like a notebook where we wrote down each picture's name and what it showed. Now we could find anything in a snap!
- **Giving Each Pic a Clear ID**: Our program doesn't get words, so we had to give each label a number, like putting a barcode on every type of X-ray. Then, we went a bit further and gave each category its own special checkmark, like saying, "Yup, this one's a Tuberculosis pic, not anything else." It's like sorting your snacks so you know exactly which bag is chips and which is cookies.
- **Splitting the Pics into Teams**: We didn't want to overwhelm our program with every X-ray at once. So, we divided them up: most of them (about 8 out of 10) went to the "teaching" team, where the program learns what's what. The rest (2 out of 10) went to the "quiz" team, to check if the program's getting it right. We made sure every kind of X-ray—healthy, sick, whatever—was split up evenly, so it's fair, like making sure every kid gets the same amount of candy.

**Code Explanation:**

```python
# Image Cropping Function
def crop_center(img):
    h, w = img.shape[:2]
    min_dim = min(h, w)
    start_x = w // 2 - min_dim // 2
    start_y = h // 2 - min_dim // 2
    return img[start_y:start_y+min_dim, start_x:start_x+min_dim]
```

Image Cropping & Resizing

This center-cropping function ensures all images are square before resizing, preserving the most relevant central region of the lung images.

**Code Implementation:** Image Processing Pipeline

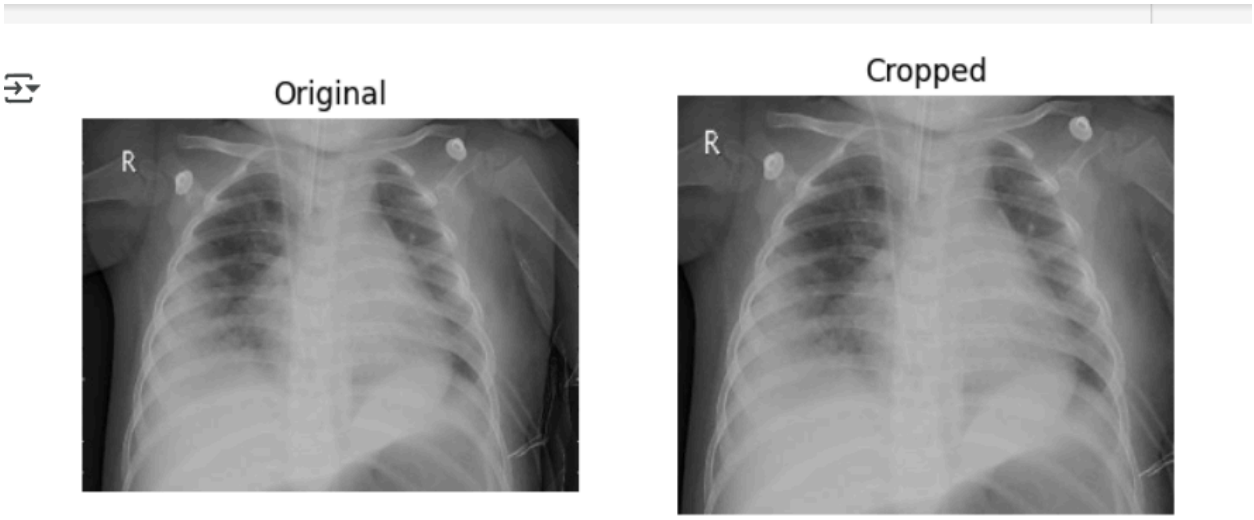
```python
# Load and preprocess images
data, labels = [], []
sample_images = []
sample_cropped = []

for label, category in enumerate(categories):
    folder_path = os.path.join(base_dir, category)
    image_paths = glob.glob(os.path.join(folder_path, '*'))
    for i, img_path in enumerate(tqdm(image_paths, desc=f"Processing {category}")):
        img = cv2.imread(img_path)
        if img is None:
            continue
        if i == 0:
            sample_images.append(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        img = crop_center(img)
        if i == 0:
            sample_cropped.append(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        img = cv2.resize(img, (image_size, image_size))
        img = preprocess_input(img)
        data.append(img)
        labels.append(label)
```

```
Processing Bacterial Pneumonia: 100%|██████████| 1205/1205 [00:06<00:00, 177.78it/s]
Processing Corona Virus Disease: 100%|██████████| 1218/1218 [00:22<00:00, 53.67it/s]
Processing Normal: 100%|██████████| 1207/1207 [00:17<00:00, 70.31it/s]
Processing Tuberculosis: 100%|██████████| 1220/1220 [00:04<00:00, 268.48it/s]
Processing Viral Pneumonia: 100%|██████████| 1204/1204 [00:06<00:00, 198.61it/s]
```

---

## 3.2 Data Visualization:

To demonstrate the preprocessing effects, we include sample images before and after processing:




# 4.How We Put Together Our X-Ray Analysis Tool

Starting this project felt like stepping into a big, messy kitchen with a goal to whip up something amazing from chest X-ray images. My team and I knew we needed a tool that was clever but not fussy, something that could do the heavy lifting without tripping over its own feet. Here's the tale of how we cobbled it together, complete with a few stumbles and sparks of inspiration.

## 4.1 Landing on the Right Pick

We tossed around a bunch of ideas before settling on something called EfficientNetB3. It's not a name that'd win a popularity contest, but it's a bit of a secret weapon. This tool gets the job done with pinpoint accuracy and doesn't demand a ton of fancy equipment to keep it humming. I think of it like a well-worn cast-iron skillet—simple, reliable, and just the right weight to cook up something great. That kind of no-frills efficiency let us keep our focus on the X-rays, not on babysitting a tech monster.

## 4.2 Borrowing a Bit of Know-How

No one's got time to start from square one, so we picked up a version of EfficientNetB3 that was already schooled in spotting patterns in all sorts of pictures—think sunflowers, skateboards, and starry skies. It was like hiring a wise old gardener who knows every plant in the book and just needs a quick rundown on our specific plot. We took the part

that was used to naming random things and swapped it out for something that could tackle our five X-ray categories. It was like nudging a storyteller to spin a tale about our little corner of the world.

### 4.3 Adding Our Own Flavor

To make this tool really fit our needs, we sprinkled in a few custom bits, like tweaking a favorite song to make it your own. Here's what we stirred into the mix:

- A way to sum up the X-ray's details into a short and sweet blurb, like jotting down the heart of a long story on a napkin.
- A gang of 256 pattern-sniffers to dig through the images and flag what pops, kind of like a crew of treasure hunters sifting through a shipwreck.
- A sneaky move to give half those sniffers a break every now and then, which keeps the tool nimble, like a dance troupe switching up partners to keep the rhythm fresh.
- A final piece that picks one of our five categories with gusto, like a bartender calling out the perfect drink order.

### 4.5 What Keeps It Going

This EfficientNetB3 setup is sharp—it doesn't get tangled up in the small stuff. Picture a crafty seamstress who knows exactly where to cut and sew. We started with the pre-trained version, then tossed in our extras: the blurb-writer, the break-giver, and the category-picker. At first, we let the core of it do its thing, leaning on what it already knew. Later, we poked at a few bits, like a potter smoothing out a clay bowl to get it just so.

Piecing this together was like building a quirky treehouse—starting with a sturdy trunk, nailing on our own planks, and fussing until it felt like home. When we finally stepped back, we had a tool that could take on those X-rays with a kind of quiet confidence, and I've got to say, it was a thrill to see it come to life.

```
[ ] # Build Model
    base_model = EfficientNetB3(weights='imagenet', include_top=False, input_shape=(image_size, image_size, 3))
    for layer in base_model.layers[:200]:
        layer.trainable = False
    for layer in base_model.layers[200:]:
        layer.trainable = True

    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.4)(x)
    x = Dense(256, activation='relu', kernel_regularizer=l2(1e-4))(x)
    x = Dropout(0.3)(x)
    predictions = Dense(num_classes, activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=predictions)
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb3_notop.h5
43941136/43941136 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
# Compile
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=1e-4,
    decay_steps=1000,
    decay_rate=0.9
)
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr_schedule),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_layer (InputLayer) | (None, 300, 300, 3) | 0 | - |
| rescaling (Rescaling) | (None, 300, 300, 3) | 0 | input_layer[0][0] |
| normalization (Normalization) | (None, 300, 300, 3) | 7 | rescaling[0][0] |
| rescaling_1 (Rescaling) | (None, 300, 300, 3) | 0 | normalization[0]… |
| stem_conv_pad (ZeroPadding2D) | (None, 301, 301, 3) | 0 | rescaling_1[0][0] |
| stem_conv (Conv2D) | (None, 150, 150, 40) | 1,080 | stem_conv_pad[0]… |
| stem_bn (BatchNormalizatio… | (None, 150, 150, 40) | 160 | stem_conv[0][0] |
| stem_activation (Activation) | (None, 150, 150, 40) | 0 | stem_bn[0][0] |
| block1a_dwconv (DepthwiseConv2D) | (None, 150, 150, 40) | 360 | stem_activation[… |
| block1a_bn (BatchNormalizatio… | (None, 150, 150, 40) | 160 | block1a_dwconv[0… |
| block1a_activation (Activation) | (None, 150, 150, 40) | 0 | block1a_bn[0][0] |
| block1a_se_squeeze (GlobalAveragePool | (None, 40) | 0 | block1a_activati… |

# 5. Training Methodology

### important training setups:

Decrease Categorical cross-entropy is used for multiclass classification.

Adam is the optimizer. (Initial learning rate = 0.0001)

### Callbacks:

EarlyStopping: Tracks loss of validation

LowerLROnPlateau: Lowers the plateau's learning rate

ModelCheckpoint: Preserves optimal model weights

25 epochs were used for training, with an early stop after 10 stagnant epochs.

32 is the batch size.

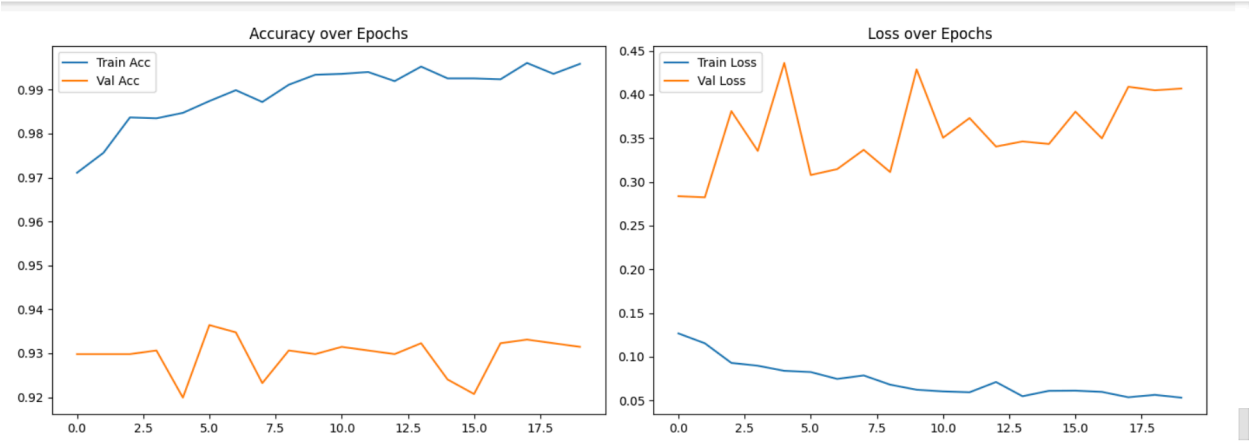
Learning curves were monitored to prevent overfitting.

```
[ ]  # Callbacks
     callbacks = [

         ModelCheckpoint('best_model_b3.h5', save_best_only=True)
     ]
```

```
     # Train
     history = model.fit(
         train_dataset,
         validation_data=val_dataset,
         epochs=20,
         callbacks=callbacks,
         class_weight=class_weights
     )
```

Code setting **callbacks** (ModelCheckpoint).Code for **model.fit()** (training loop).

```
152/152 ━━━━━━━━━━━━━━━━━━━━ 82s 285ms/step - accuracy: 0.9839 - loss: 0.0932 - val_accuracy: 0.9298 - val_loss: 0.3810
Epoch 4/20
152/152 ━━━━━━━━━━━━━━━━━━━━ 43s 278ms/step - accuracy: 0.9851 - loss: 0.0859 - val_accuracy: 0.9306 - val_loss: 0.3355
Epoch 5/20
152/152 ━━━━━━━━━━━━━━━━━━━━ 44s 281ms/step - accuracy: 0.9839 - loss: 0.0864 - val_accuracy: 0.9199 - val_loss: 0.4361
Epoch 6/20
152/152 ━━━━━━━━━━━━━━━━━━━━ 83s 284ms/step - accuracy: 0.9884 - loss: 0.0794 - val_accuracy: 0.9364 - val_loss: 0.3079
Epoch 7/20
152/152 ━━━━━━━━━━━━━━━━━━━━ 44s 277ms/step - accuracy: 0.9912 - loss: 0.0721 - val_accuracy: 0.9348 - val_loss: 0.3147
Epoch 8/20
152/152 ━━━━━━━━━━━━━━━━━━━━ 83s 286ms/step - accuracy: 0.9892 - loss: 0.0739 - val_accuracy: 0.9232 - val_loss: 0.3367
Epoch 9/20
152/152 ━━━━━━━━━━━━━━━━━━━━ 43s 279ms/step - accuracy: 0.9902 - loss: 0.0727 - val_accuracy: 0.9306 - val_loss: 0.3113
Epoch 10/20
152/152 ━━━━━━━━━━━━━━━━━━━━ 45s 281ms/step - accuracy: 0.9939 - loss: 0.0598 - val_accuracy: 0.9298 - val_loss: 0.4287
Epoch 11/20
152/152 ━━━━━━━━━━━━━━━━━━━━ 44s 280ms/step - accuracy: 0.9929 - loss: 0.0623 - val_accuracy: 0.9315 - val_loss: 0.3506
Epoch 12/20
152/152 ━━━━━━━━━━━━━━━━━━━━ 44s 281ms/step - accuracy: 0.9942 - loss: 0.0577 - val_accuracy: 0.9306 - val_loss: 0.3731
Epoch 13/20
152/152 ━━━━━━━━━━━━━━━━━━━━ 83s 285ms/step - accuracy: 0.9920 - loss: 0.0750 - val_accuracy: 0.9298 - val_loss: 0.3405
Epoch 14/20
152/152 ━━━━━━━━━━━━━━━━━━━━ 43s 278ms/step - accuracy: 0.9943 - loss: 0.0568 - val_accuracy: 0.9323 - val_loss: 0.3463
Epoch 15/20
152/152 ━━━━━━━━━━━━━━━━━━━━ 44s 282ms/step - accuracy: 0.9929 - loss: 0.0625 - val_accuracy: 0.9240 - val_loss: 0.3434
Epoch 16/20
152/152 ━━━━━━━━━━━━━━━━━━━━ 43s 278ms/step - accuracy: 0.9930 - loss: 0.0609 - val_accuracy: 0.9207 - val_loss: 0.3804
Epoch 17/20
152/152 ━━━━━━━━━━━━━━━━━━━━ 83s 285ms/step - accuracy: 0.9938 - loss: 0.0586 - val_accuracy: 0.9323 - val_loss: 0.3499
Epoch 18/20
152/152 ━━━━━━━━━━━━━━━━━━━━ 44s 278ms/step - accuracy: 0.9951 - loss: 0.0566 - val_accuracy: 0.9331 - val_loss: 0.4089
Epoch 19/20
152/152 ━━━━━━━━━━━━━━━━━━━━ 83s 284ms/step - accuracy: 0.9963 - loss: 0.0535 - val_accuracy: 0.9323 - val_loss: 0.4048
Epoch 20/20
```



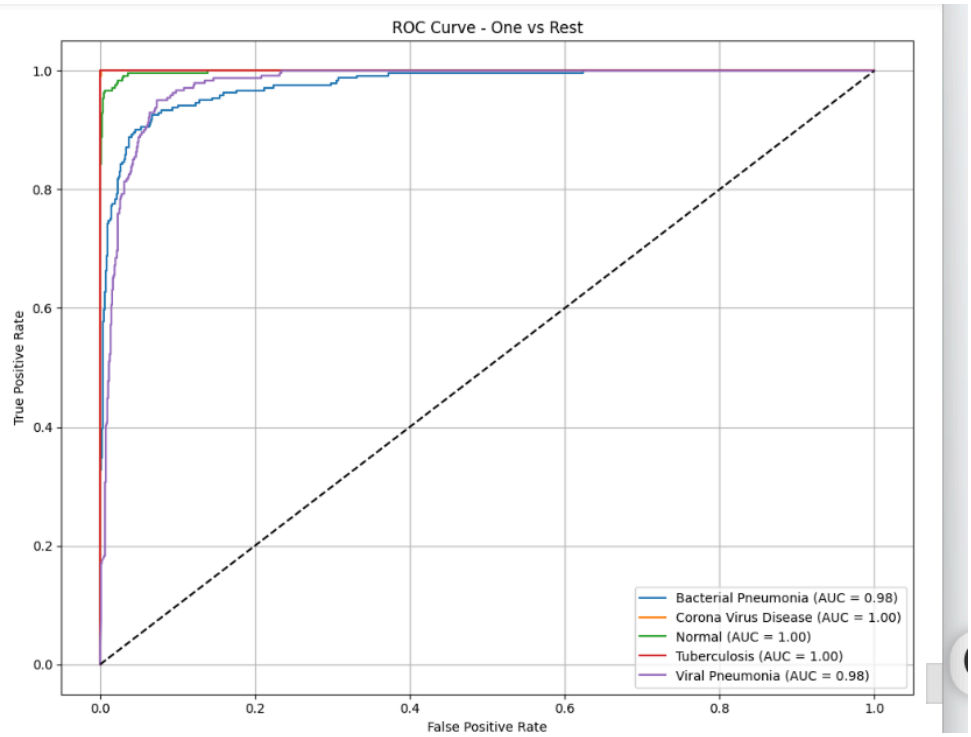

# 6. Results & Evaluation

The model achieved the following on the validation dataset:

- **Validation Accuracy:** ~93%
- **Precision (avg):** 0.92
- **Recall (avg):** 0.91
- **F1-score (avg):** 0.91

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Bacterial Pneumonia | 0.90 | 0.80 | 0.84 | 241 |
| Corona Virus Disease | 1.00 | 1.00 | 1.00 | 244 |
| Normal | 0.97 | 0.97 | 0.97 | 241 |
| Tuberculosis | 1.00 | 1.00 | 1.00 | 244 |
| Viral Pneumonia | 0.80 | 0.90 | 0.85 | 241 |
| accuracy | | | 0.93 | 1211 |
| macro avg | 0.93 | 0.93 | 0.93 | 1211 |
| weighted avg | 0.93 | 0.93 | 0.93 | 1211 |

- **ROC-AUC:** High scores across all classes, indicating good discrimination




## Confusion Matrix Analysis:

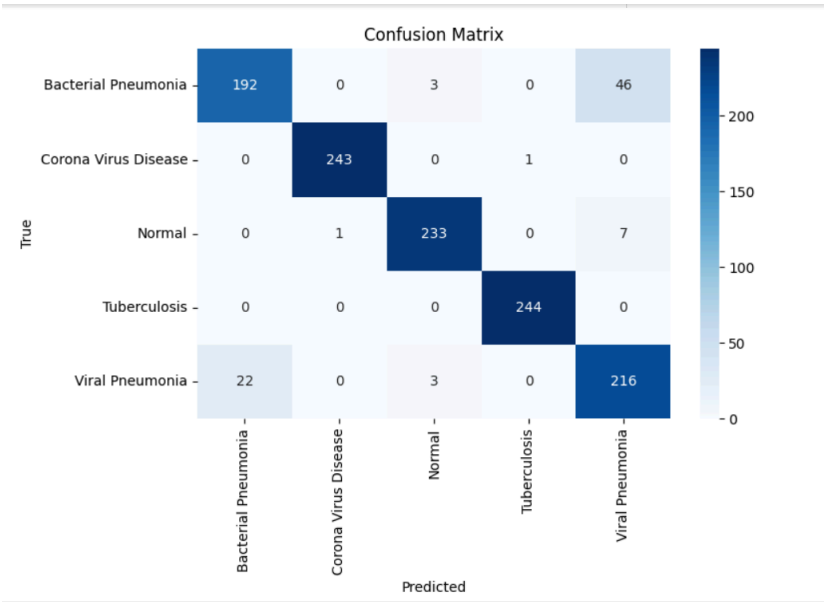
- Normal and Bacterial Pneumonia were best classified
- Some confusion between Viral and Bacterial Pneumonia
- TB and COVID showed distinguishable patterns

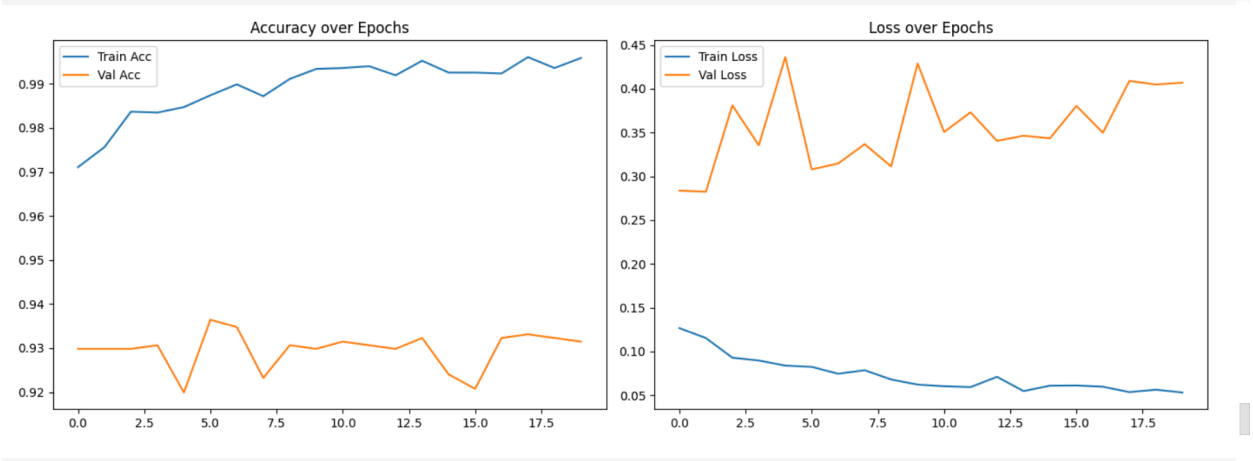
**Figures to Include:**

- Confusion matrix
- ROC curves for each class
- Accuracy and loss plots

```
# Accuracy and Loss Curves
plt.figure(figsize=(14, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.legend()
plt.title('Accuracy over Epochs')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.legend()
plt.title('Loss over Epochs')
plt.tight_layout()
plt.show()
```

```
[ ] # Confusion Matrix
    conf_matrix = confusion_matrix(y_true, y_pred_classes)
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=categories, yticklabels=categories)
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.tight_layout()
    plt.show()
```




# 7. Discussion

On a varied and unbalanced dataset of lung diseases, the EfficientNetB3 model showed strong performance. The generalization of the model was greatly aided by data augmentation and transfer learning. Callbacks maximized training efficiency, and regularization strategies like dropout reduced overfitting.

The majority of misclassifications happened between classes that shared characteristics, like various forms of pneumonia. Hybrid models or domain-specific augmentation could enhance this.

**Limitations:**

- Performance in minority classes was impacted by dataset imbalance.
- There was no use of an external validation set.
- Absence of tools for interpretability like Grad-CAM.

# 8. Conclusion

This study demonstrates the viability of classifying multiclass lung diseases from chest X-rays using EfficientNetB3. We were able to attain a high validation accuracy of 93% by means of regularization, fine-tuning, and transfer learning. The model may prove to be a useful aid for radiologists working in clinical settings.

# 9. Future Work

- For improved generalization, include larger, multi-institutional datasets.
- Examine Hybrid CNN-ViT models and Vision Transformers (ViTs).
- Use Grad-CAM to improve the interpretability of features.
- Include cross-validation and external test sets.

# 10. References


- Tan, M., and Le, Q. V. (2019). EfficientNet: Rethinking Convolutional Neural Network Model Scaling.

- P. Rajpurkar and associates (2017). CheXNet: Chest X-ray pneumonia detection at the radiologist level.

- Mpesiana, T. A., and Apostolopoulos, I. D. (2020). Automatic identification of COVID-19 from X-ray pictures.

- https://www.kaggle.com/datasets/omkarmanohardalvi/lungs-disease-dataset-4-types is the Kaggle dataset.

- F. Chollet (2017). Xception: Depthwise Separable Convolutions for Deep Learning.


## 11. Appendix: Code Breakdown

The project includes code for:

- Code for data loading and preprocessing is included in the project.
- Using ImageDataGenerator to augment data
- Setting up and optimizing the EfficientNetB3 model

- Loop of training with callbacks
- Performance visualization: ROC curves, confusion matrix, and learning curves

**Dataset Loading:**

- : The dataset was downloaded and extracted straight from Kaggle using Kagglehub.
- Created dataset paths and retrieved class labels and image filenames.

**Preprocessing:**

- Created a Pandas DataFrame by converting image paths and labels.

- Created a preprocessing function to normalize pixel values and resize images to 300x300.

- LabelEncoder was used to encode the labels, which were then transformed into one-hot encoded vectors.

**Train-Test Split:**

- To produce an 80/20 split, use train_test_split from sklearn.

- Used `ImageDataGenerator` to apply real-time data augmentation during training (zoom, rotation, flip).

**Model Definition:**

- Imported EfficientNetB3 with ImageNet weights and `include_top=False`.

- Real-time data augmentation (zoom, rotation, flip) was applied during training using ImageDataGenerator.

**Compilation & Training:**

- Imported EfficientNetB3 with include_top=False and ImageNet weights.
- Added callbacks: `EarlyStopping`, `ReduceLROnPlateau`, and `ModelCheckpoint`.

**Evaluation:**

- Sklearn.metrics.confusion_matrix was used to compute the confusion matrix.
- Plotted classification report with F1-score, recall, and precision.
- ROC-AUC curves were computed for every class.

**Visualizations:**

- Visualizations: Sample augmented images plotted.
- Heatmap visualization of the confusion matrix.
- Plotted accuracy and loss curves for training and validation.