

```
In [1]: dataset_path = r"chest_xray"
        print(dataset_path)
```

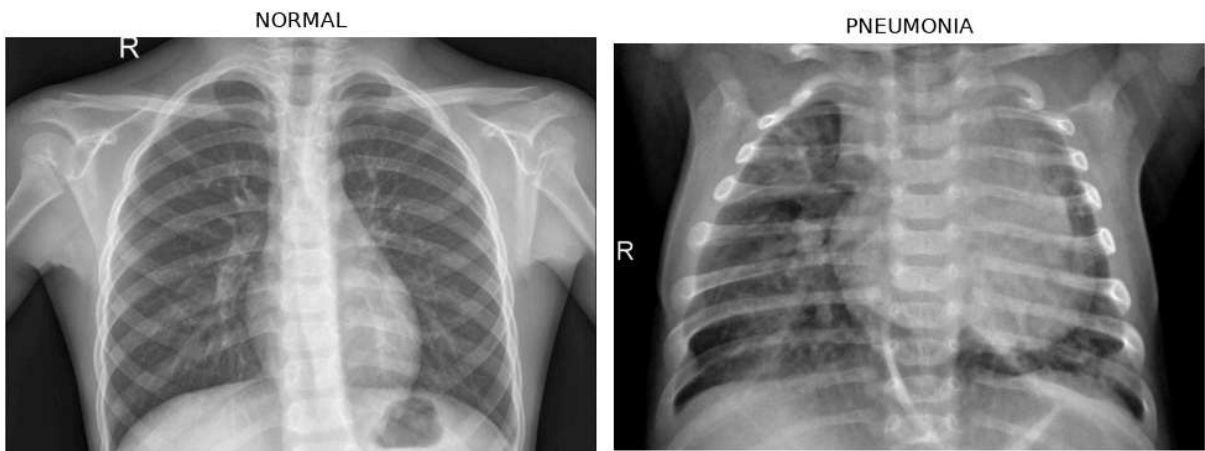
chest_xray

```
In [2]: import os
        import matplotlib.pyplot as plt
        from PIL import Image

        # Find any 'NORMAL' and 'PNEUMONIA' subfolder inside the download
        def find_class_dir(base, class_name):
            for root, dirs, files in os.walk(base):
                if os.path.basename(root) == class_name and len(files) > 0:
                    return root
            return None

        normal_dir = find_class_dir(dataset_path, "NORMAL")
        pneu_dir = find_class_dir(dataset_path, "PNEUMONIA")

        fig, ax = plt.subplots(1, 2, figsize=(10, 5))
        for a, d, t in zip(ax, [normal_dir, pneu_dir], ["NORMAL", "PNEUMONIA"]):
            img_file = next(f for f in os.listdir(d) if f.lower().endswith(('.jpg', '.jpeg')))
            a.imshow(Image.open(os.path.join(d, img_file)).convert("L"), cmap="gray")
            a.set_title(t); a.axis("off")
        plt.tight_layout(); plt.show()
```



```
In [3]: import os, shutil, glob
        from sklearn.model_selection import train_test_split

        base_dir = "dataset" # output folder created next to this not
        train_dir = os.path.join(base_dir, "train")
        validation_dir = os.path.join(base_dir, "validation")
        test_dir = os.path.join(base_dir, "test")

        for d in [train_dir, validation_dir, test_dir]:
            os.makedirs(d, exist_ok=True)

        train_ratio, validation_ratio, test_ratio = 0.70, 0.15, 0.15
        classes = ["NORMAL", "PNEUMONIA"]
```

```

def gather_all_images(base, class_name):
    """Collect every image of one class from the (pre-split) Kaggle download."""
    paths = []
    for root, dirs, files in os.walk(base):
        if os.path.basename(root) == class_name:
            for f in files:
                if f.lower().endswith((".jpg", ".jpeg", ".png")):
                    paths.append(os.path.join(root, f))
    return paths

for class_name in classes:
    all_images = gather_all_images(dataset_path, class_name)

    # 70% train, 30% temp -> then split temp into 15% val / 15% test
    train_images, temp_images = train_test_split(
        all_images, test_size=(1 - train_ratio), random_state=42)
    validation_images, test_images = train_test_split(
        temp_images, test_size=test_ratio / (validation_ratio + test_ratio),
        random_state=42)

    for images, folder in zip([train_images, validation_images, test_images],
                              [train_dir, validation_dir, test_dir]):
        dest_dir = os.path.join(folder, class_name)
        os.makedirs(dest_dir, exist_ok=True)
        for src in images:
            shutil.copy(src, dest_dir)

    print(f"{class_name:10s} | total {len(all_images):5d} | "
          f"train {len(train_images):5d} | val {len(validation_images):4d} | "
          f"test {len(test_images):4d}")

```

NORMAL	total	1575	train	1102	val	236	test	237
PNEUMONIA	total	971	train	679	val	146	test	146

In [4]: `from tensorflow.keras.preprocessing.image import ImageDataGenerator`

```

IMG_SIZE = (224, 224)
BATCH_SIZE = 32

def create_data_generators(train_dir, val_dir, test_dir,
                           img_size=IMG_SIZE, batch_size=BATCH_SIZE):
    # ---- Step 2: data augmentation for the TRAINING set only ----
    train_datagen = ImageDataGenerator(
        rescale=1.0 / 255,
        rotation_range=20,
        width_shift_range=0.10,
        height_shift_range=0.10,
        zoom_range=0.15,
        horizontal_flip=True,
        brightness_range=[0.8, 1.2],
        fill_mode="nearest",
    )

    # validation + test: rescale ONLY (no augmentation)
    val_test_datagen = ImageDataGenerator(rescale=1.0 / 255)

```

```

train_generator = train_datagen.flow_from_directory(
    train_dir, target_size=img_size, batch_size=batch_size,
    class_mode="categorical", shuffle=True)

val_generator = val_test_datagen.flow_from_directory(
    val_dir, target_size=img_size, batch_size=batch_size,
    class_mode="categorical", shuffle=False)

test_generator = val_test_datagen.flow_from_directory(
    test_dir, target_size=img_size, batch_size=batch_size,
    class_mode="categorical", shuffle=False)

return train_generator, val_generator, test_generator

train_gen, val_gen, test_gen = create_data_generators(train_dir, validation_dir, te

input_shape = IMG_SIZE + (3,)
num_classes = len(train_gen.class_indices)
class_labels = list(train_gen.class_indices.keys())
print("Classes:", train_gen.class_indices)

```

C:\Users\habus\.conda\envs\Pneumonia\Lib\site-packages\requests__init__.py:92: RequestsDependencyWarning: Unable to find acceptable character detection dependency (charset or charset_normalizer).

warnings.warn(

Found 1781 images belonging to 2 classes.

Found 382 images belonging to 2 classes.

Found 383 images belonging to 2 classes.

Classes: {'NORMAL': 0, 'PNEUMONIA': 1}

In [5]: `import numpy as np`

```

batch_imgs, batch_lbls = next(train_gen)
plt.figure(figsize=(14, 4))
for i in range(6):
    plt.subplot(1, 6, i + 1)
    plt.imshow(batch_imgs[i])
    plt.title(class_labels[np.argmax(batch_lbls[i])], fontsize=9)
    plt.axis("off")
plt.suptitle("Augmented training samples")
plt.tight_layout(); plt.show()
train_gen.reset()

```

Augmented training samples



In [6]: `from sklearn.utils.class_weight import compute_class_weight`

```
y_train = train_gen.classes
```

```

class_weights_arr = compute_class_weight(
    class_weight="balanced", classes=np.unique(y_train), y=y_train)
class_weight = {i: w for i, w in enumerate(class_weights_arr)}
print("Class weights:", class_weight)

```

Class weights: {0: np.float64(0.808076225045372), 1: np.float64(1.3114874815905744)}

```

In [7]: from sklearn.metrics import (confusion_matrix, accuracy_score, precision_score,
    recall_score, f1_score, classification_report)

import seaborn as sns

def plot_history(history, title=""):
    fig, ax = plt.subplots(1, 2, figsize=(12, 4))
    ax[0].plot(history.history["accuracy"], label="train")
    ax[0].plot(history.history["val_accuracy"], label="val")
    ax[0].set_title(f"{title} accuracy"); ax[0].set_xlabel("epoch"); ax[0].legend()
    ax[1].plot(history.history["loss"], label="train")
    ax[1].plot(history.history["val_loss"], label="val")
    ax[1].set_title(f"{title} loss"); ax[1].set_xlabel("epoch"); ax[1].legend()
    plt.tight_layout(); plt.show()

def evaluate_model(model_name, model, test_generator):
    test_generator.reset()
    y_pred_prob = model.predict(test_generator, verbose=1)
    y_pred = np.argmax(y_pred_prob, axis=1)
    y_true = test_generator.classes
    labels = list(test_generator.class_indices.keys())

    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average="weighted", zero_division=0)
    recall = recall_score(y_true, y_pred, average="weighted", zero_division=0)
    f1 = f1_score(y_true, y_pred, average="weighted", zero_division=0)

    print(f"==== {model_name} =====")
    print(f"Accuracy : {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall : {recall:.4f}")
    print(f"F1-score : {f1:.4f}\n")
    print("Classification Report:")
    print(classification_report(y_true, y_pred, target_names=labels, zero_division=0))

    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=labels, yticklabels=labels)
    plt.xlabel("Predicted Label"); plt.ylabel("True Label")
    plt.title(f"Confusion Matrix - {model_name}")
    plt.tight_layout(); plt.show()

    return {"model": model_name, "accuracy": accuracy, "precision": precision,
            "recall": recall, "f1": f1}

results = [] # collect every model's metrics for the final comparison

```

```
In [8]: import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
                                     GlobalAveragePooling2D, Input)
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.applications import DenseNet121, ResNet50, EfficientNetB0

EPOCHS = 15

early_stop = EarlyStopping(monitor="val_loss", patience=4,
                           restore_best_weights=True, verbose=1)
```

```
In [9]: def build_cnn_model(input_shape, num_classes):
    model = Sequential([
        Input(shape=input_shape),
        Conv2D(32, (3, 3), activation="relu"), MaxPooling2D(2, 2),
        Conv2D(64, (3, 3), activation="relu"), MaxPooling2D(2, 2),
        Conv2D(128, (3, 3), activation="relu"), MaxPooling2D(2, 2),
        Flatten(),
        Dense(128, activation="relu"),
        Dropout(0.5),
        Dense(num_classes, activation="softmax"),
    ])
    model.compile(optimizer="adam", loss="categorical_crossentropy",
                  metrics=["accuracy"])
    return model

cnn_model = build_cnn_model(input_shape, num_classes)
cnn_model.summary()
```

WARNING:tensorflow:TensorFlow GPU support is not available on native Windows for TensorFlow >= 2.11. Even if CUDA/cuDNN are installed, GPU will not be used. Please use WSL2 or the TensorFlow-DirectML plugin.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_2 (Conv2D)	(None, 52, 52, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 26, 26, 128)	0
flatten (Flatten)	(None, 86528)	0
dense (Dense)	(None, 128)	11,075,712
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 2)	258

Total params: 11,169,218 (42.61 MB)

Trainable params: 11,169,218 (42.61 MB)

Non-trainable params: 0 (0.00 B)

```
In [11]: history_cnn = cnn_model.fit(
        train_gen, epochs=EPOCHS, validation_data=val_gen,
        class_weight=class_weight, callbacks=[early_stop])
        plot_history(history_cnn, "Baseline CNN")
```

Epoch 1/15

56/56  **137s** 2s/step - accuracy: 0.8332 - loss: 0.4375 - val_accuracy: 0.8743 - val_loss: 0.3371

Epoch 2/15

56/56  **125s** 2s/step - accuracy: 0.8579 - loss: 0.3977 - val_accuracy: 0.8010 - val_loss: 0.4087

Epoch 3/15

56/56  **955s** 17s/step - accuracy: 0.8675 - loss: 0.3596 - val_accuracy: 0.8927 - val_loss: 0.3588

Epoch 4/15

56/56  **105s** 2s/step - accuracy: 0.8563 - loss: 0.3744 - val_accuracy: 0.9136 - val_loss: 0.2460

Epoch 5/15

56/56  **144s** 2s/step - accuracy: 0.8843 - loss: 0.3193 - val_accuracy: 0.9162 - val_loss: 0.2182

Epoch 6/15

56/56  **106s** 2s/step - accuracy: 0.8877 - loss: 0.3132 - val_accuracy: 0.9188 - val_loss: 0.2119

Epoch 7/15

56/56  **142s** 2s/step - accuracy: 0.8737 - loss: 0.3112 - val_accuracy: 0.9188 - val_loss: 0.2059

Epoch 8/15

56/56  **117s** 2s/step - accuracy: 0.8832 - loss: 0.3134 - val_accuracy: 0.9031 - val_loss: 0.2562

Epoch 9/15

56/56  **107s** 2s/step - accuracy: 0.8843 - loss: 0.3066 - val_accuracy: 0.8743 - val_loss: 0.2788

Epoch 10/15

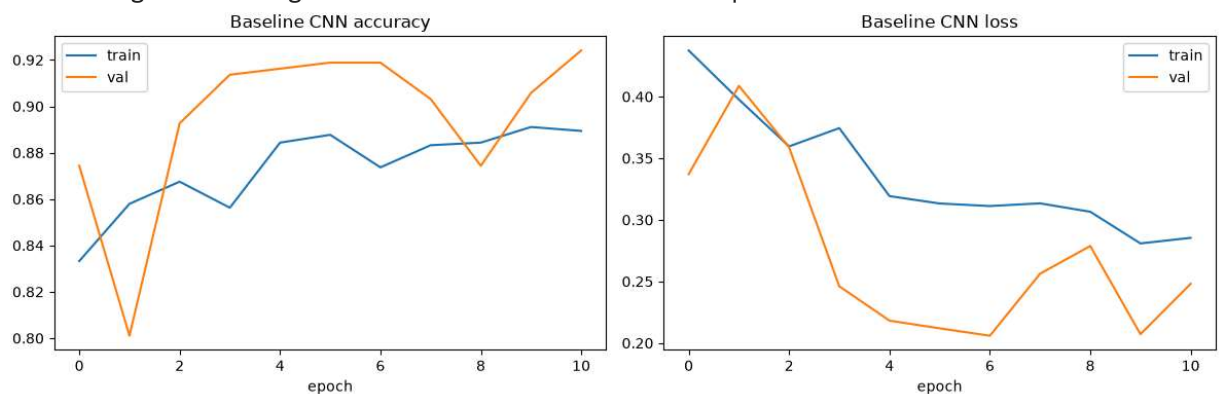
56/56  **116s** 2s/step - accuracy: 0.8911 - loss: 0.2808 - val_accuracy: 0.9058 - val_loss: 0.2072

Epoch 11/15

56/56  **110s** 2s/step - accuracy: 0.8894 - loss: 0.2854 - val_accuracy: 0.9241 - val_loss: 0.2481

Epoch 11: early stopping

Restoring model weights from the end of the best epoch: 7.



```
In [12]: results.append(evaluate_model("Baseline CNN", cnn_model, test_gen))
```

12/12 ————— 12s 1s/step

===== Baseline CNN =====

Accuracy : 0.9295

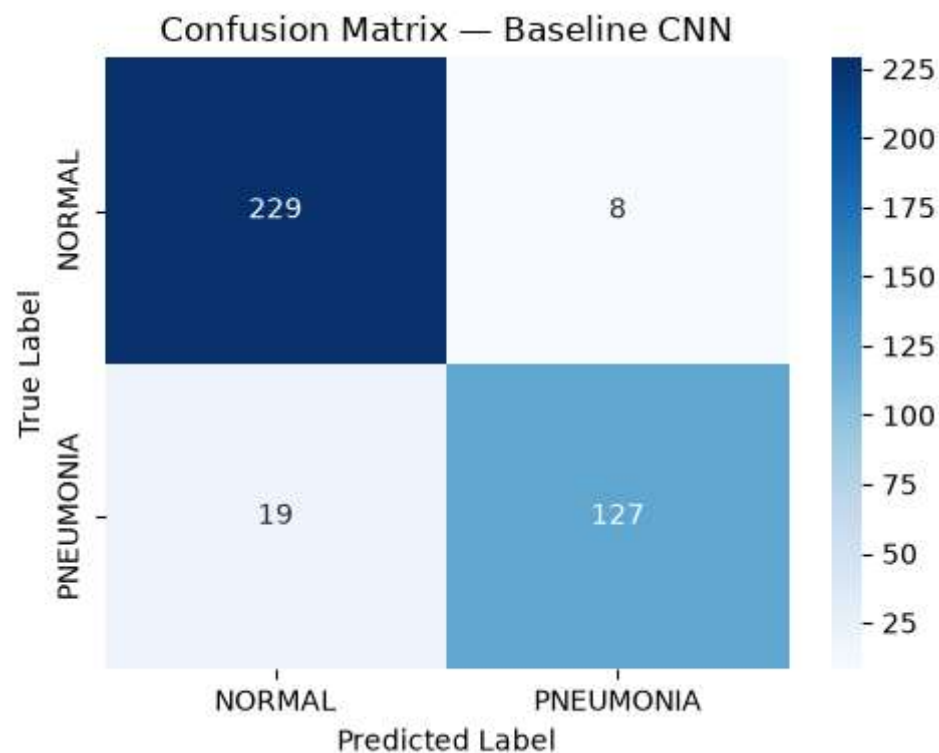
Precision: 0.9300

Recall : 0.9295

F1-score : 0.9289

Classification Report:

	precision	recall	f1-score	support
NORMAL	0.92	0.97	0.94	237
PNEUMONIA	0.94	0.87	0.90	146
accuracy			0.93	383
macro avg	0.93	0.92	0.92	383
weighted avg	0.93	0.93	0.93	383



```
In [13]: def build_transfer_model(base_model_class, input_shape, num_classes):
          base_model = base_model_class(weights="imagenet", include_top=False,
                                         input_shape=input_shape)
          base_model.trainable = False # freeze the pretrained feature extractor
          model = Sequential([
              base_model,
              GlobalAveragePooling2D(),
              Dense(128, activation="relu"),
              Dropout(0.5),
              Dense(num_classes, activation="softmax"),
          ])
          model.compile(optimizer="adam", loss="categorical_crossentropy",
                      metrics=["accuracy"])
          return model
```

```
densenet_model = build_transfer_model(DenseNet121, input_shape, num_classes)
history_dn = densenet_model.fit(
    train_gen, epochs=EPOCHS, validation_data=val_gen,
    class_weight=class_weight, callbacks=[early_stop])
plot_history(history_dn, "DenseNet121")
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf_dim_ordering_tf_kernels_notop.h5

29084464/29084464 ————— **2s** 0us/step

Epoch 1/15

56/56 ————— **234s** 4s/step - accuracy: 0.8069 - loss: 0.4455 - val_accuracy: 0.9162 - val_loss: 0.2562

Epoch 2/15

56/56 ————— **251s** 4s/step - accuracy: 0.8821 - loss: 0.3123 - val_accuracy: 0.9110 - val_loss: 0.2320

Epoch 3/15

56/56 ————— **262s** 4s/step - accuracy: 0.8905 - loss: 0.2860 - val_accuracy: 0.9293 - val_loss: 0.2055

Epoch 4/15

56/56 ————— **267s** 4s/step - accuracy: 0.8989 - loss: 0.2534 - val_accuracy: 0.9372 - val_loss: 0.1945

Epoch 5/15

56/56 ————— **256s** 4s/step - accuracy: 0.9085 - loss: 0.2560 - val_accuracy: 0.9319 - val_loss: 0.1893

Epoch 6/15

56/56 ————— **213s** 4s/step - accuracy: 0.9034 - loss: 0.2429 - val_accuracy: 0.9346 - val_loss: 0.1651

Epoch 7/15

56/56 ————— **236s** 3s/step - accuracy: 0.9085 - loss: 0.2417 - val_accuracy: 0.9450 - val_loss: 0.1548

Epoch 8/15

56/56 ————— **219s** 4s/step - accuracy: 0.9107 - loss: 0.2328 - val_accuracy: 0.9372 - val_loss: 0.1681

Epoch 9/15

56/56 ————— **201s** 4s/step - accuracy: 0.9203 - loss: 0.2112 - val_accuracy: 0.9450 - val_loss: 0.1509

Epoch 10/15

56/56 ————— **207s** 4s/step - accuracy: 0.9248 - loss: 0.2102 - val_accuracy: 0.9293 - val_loss: 0.1795

Epoch 11/15

56/56 ————— **207s** 4s/step - accuracy: 0.9147 - loss: 0.2143 - val_accuracy: 0.9476 - val_loss: 0.1502

Epoch 12/15

56/56 ————— **259s** 4s/step - accuracy: 0.9276 - loss: 0.2001 - val_accuracy: 0.9529 - val_loss: 0.1486

Epoch 13/15

56/56 ————— **208s** 4s/step - accuracy: 0.9293 - loss: 0.1979 - val_accuracy: 0.9503 - val_loss: 0.1414

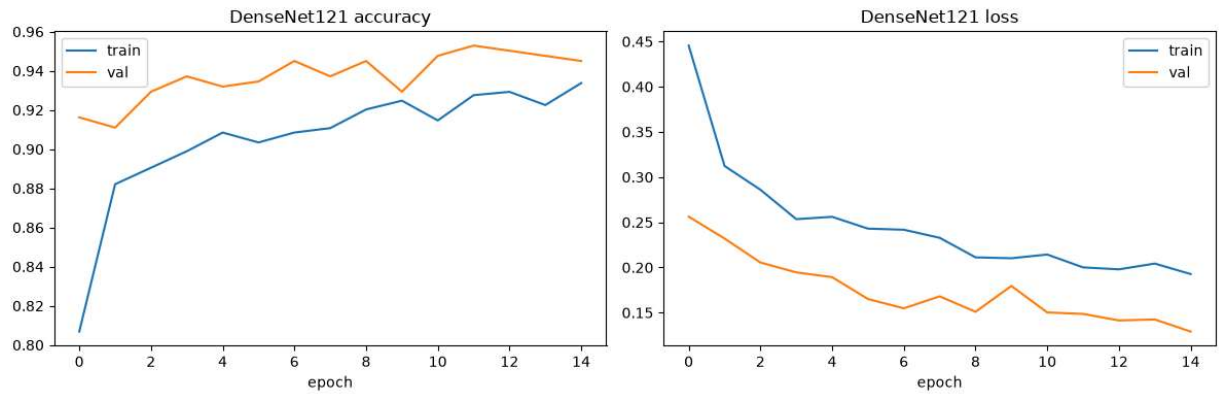
Epoch 14/15

56/56 ————— **209s** 4s/step - accuracy: 0.9225 - loss: 0.2043 - val_accuracy: 0.9476 - val_loss: 0.1425

Epoch 15/15

56/56 ————— **206s** 4s/step - accuracy: 0.9337 - loss: 0.1927 - val_accuracy: 0.9450 - val_loss: 0.1291

Restoring model weights from the end of the best epoch: 15.



```
In [14]: results.append(evaluate_model("DenseNet121", densenet_model, test_gen))
```

12/12 ————— 44s 3s/step

===== DenseNet121 =====

Accuracy : 0.9504

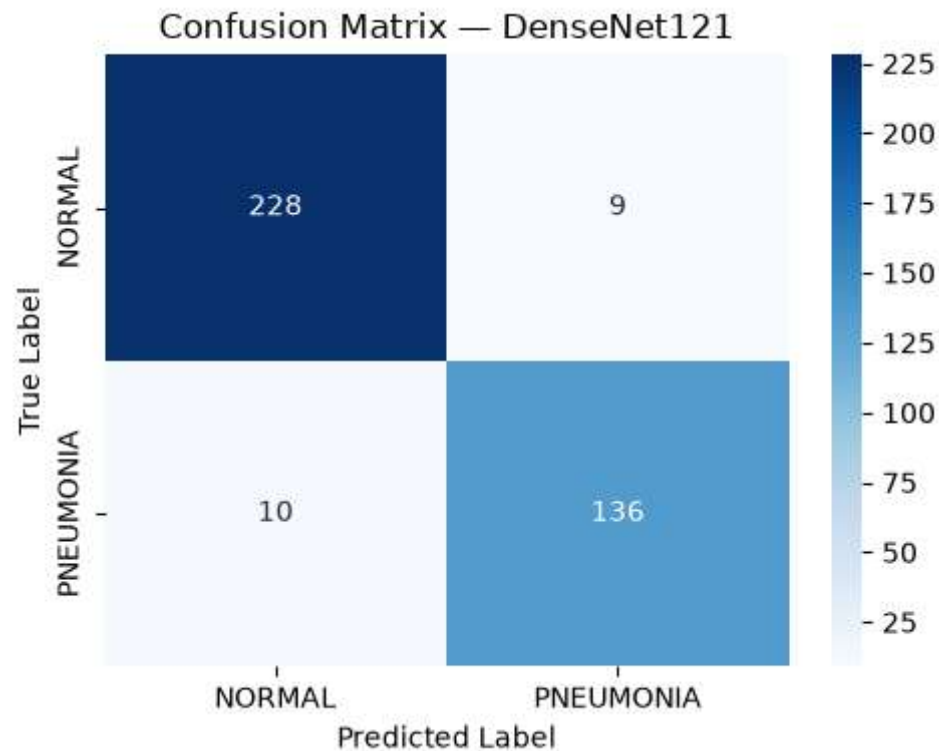
Precision: 0.9503

Recall : 0.9504

F1-score : 0.9504

Classification Report:

	precision	recall	f1-score	support
NORMAL	0.96	0.96	0.96	237
PNEUMONIA	0.94	0.93	0.93	146
accuracy			0.95	383
macro avg	0.95	0.95	0.95	383
weighted avg	0.95	0.95	0.95	383



```
In [15]: resnet_model = build_transfer_model(ResNet50, input_shape, num_classes)
history_rn = resnet_model.fit(
    train_gen, epochs=EPOCHS, validation_data=val_gen,
    class_weight=class_weight, callbacks=[early_stop])
plot_history(history_rn, "ResNet50")
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5

94765736/94765736 ————— **14s** 0us/step

Epoch 1/15

56/56 ————— **197s** 3s/step - accuracy: 0.5177 - loss: 0.8651 - val_accuracy: 0.6230 - val_loss: 0.6531

Epoch 2/15

56/56 ————— **198s** 3s/step - accuracy: 0.5862 - loss: 0.6770 - val_accuracy: 0.8063 - val_loss: 0.6371

Epoch 3/15

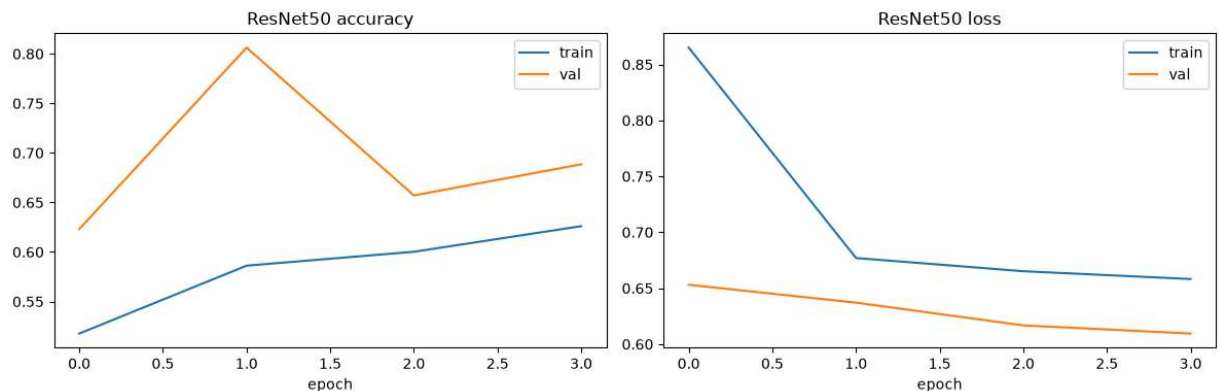
56/56 ————— **186s** 3s/step - accuracy: 0.6002 - loss: 0.6652 - val_accuracy: 0.6571 - val_loss: 0.6167

Epoch 4/15

56/56 ————— **190s** 3s/step - accuracy: 0.6261 - loss: 0.6582 - val_accuracy: 0.6885 - val_loss: 0.6094

Epoch 4: early stopping

Restoring model weights from the end of the best epoch: 1.



```
In [16]: results.append(evaluate_model("ResNet50", resnet_model, test_gen))
```

12/12 ————— **35s** 2s/step

==== ResNet50 =====

Accuracy : 0.6240

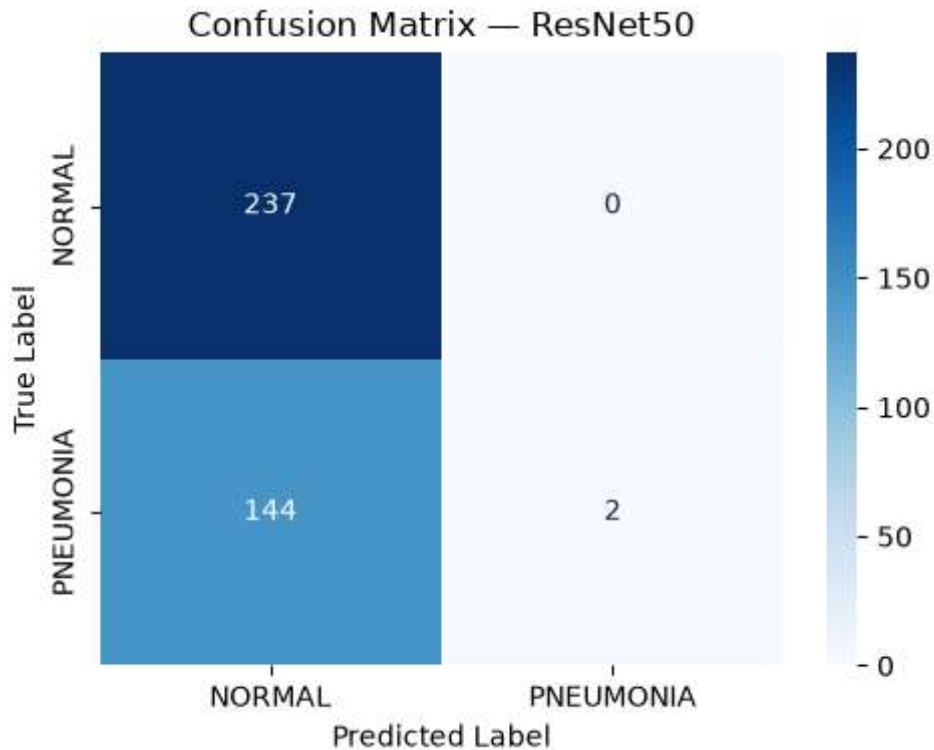
Precision: 0.7661

Recall : 0.6240

F1-score : 0.4849

Classification Report:

	precision	recall	f1-score	support
NORMAL	0.62	1.00	0.77	237
PNEUMONIA	1.00	0.01	0.03	146
accuracy			0.62	383
macro avg	0.81	0.51	0.40	383
weighted avg	0.77	0.62	0.48	383



```
In [17]: efficientnet_model = build_transfer_model(EfficientNetB0, input_shape, num_classes)
history_en = efficientnet_model.fit(
    train_gen, epochs=EPOCHS, validation_data=val_gen,
    class_weight=class_weight, callbacks=[early_stop])
plot_history(history_en, "EfficientNetB0")
```

Downloading data from https://storage.googleapis.com/keras-applications/efficientnet_b0_notop.h5

16705208/16705208 ————— 2s 0us/step

Epoch 1/15

56/56 ————— 136s 2s/step - accuracy: 0.4964 - loss: 0.7120 - val_accuracy: 0.3822 - val_loss: 0.6997

Epoch 2/15

56/56 ————— 136s 2s/step - accuracy: 0.4868 - loss: 0.6945 - val_accuracy: 0.3822 - val_loss: 0.6957

Epoch 3/15

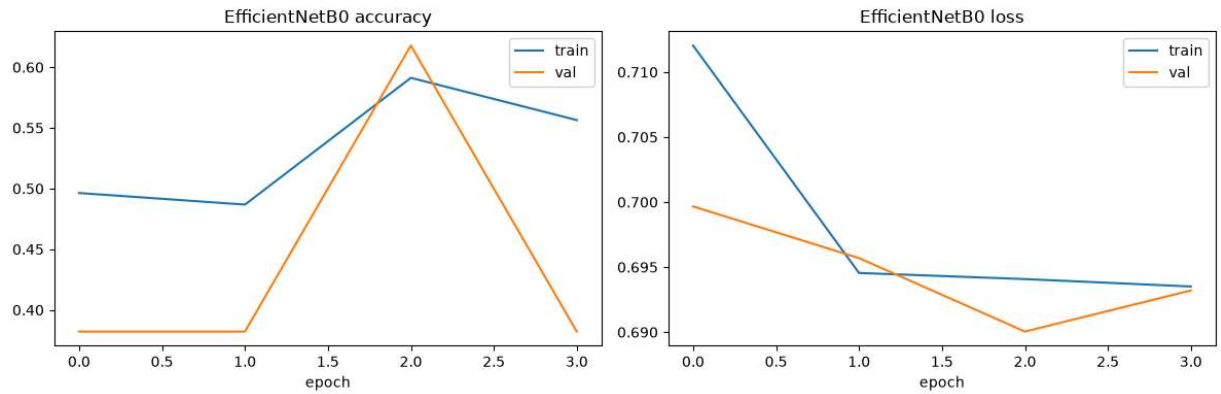
56/56 ————— 123s 2s/step - accuracy: 0.5912 - loss: 0.6941 - val_accuracy: 0.6178 - val_loss: 0.6900

Epoch 4/15

56/56 ————— 138s 2s/step - accuracy: 0.5564 - loss: 0.6935 - val_accuracy: 0.3822 - val_loss: 0.6932

Epoch 4: early stopping

Restoring model weights from the end of the best epoch: 1.



```
In [18]: results.append(evaluate_model("EfficientNetB0", efficientnet_model, test_gen))
```

12/12 ————— 23s 2s/step

==== EfficientNetB0 =====

Accuracy : 0.3812

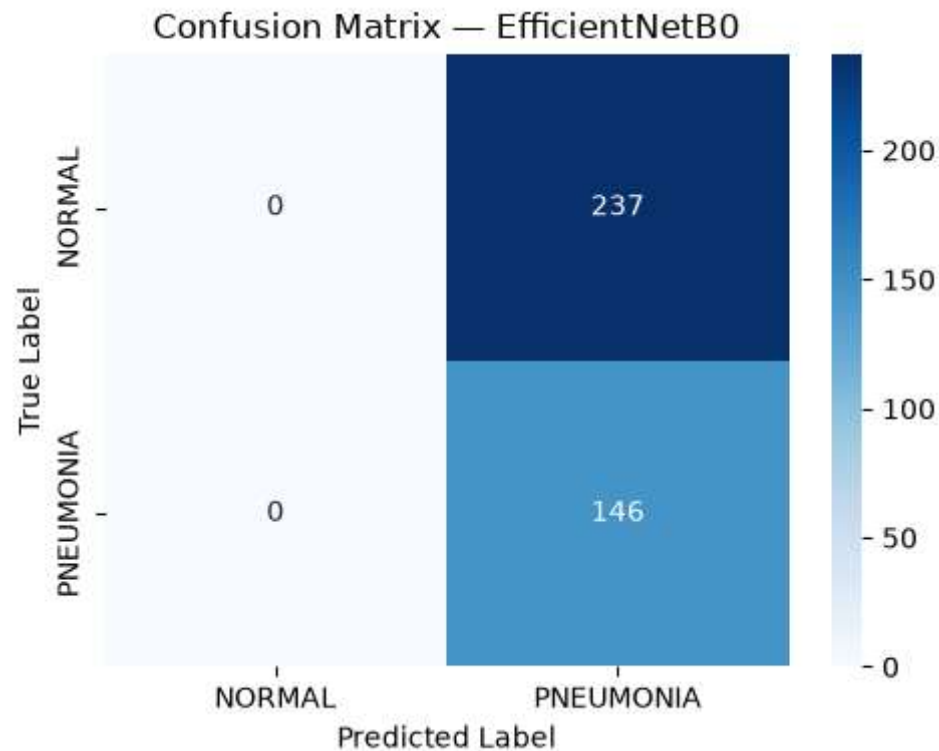
Precision: 0.1453

Recall : 0.3812

F1-score : 0.2104

Classification Report:

	precision	recall	f1-score	support
NORMAL	0.00	0.00	0.00	237
PNEUMONIA	0.38	1.00	0.55	146
accuracy			0.38	383
macro avg	0.19	0.50	0.28	383
weighted avg	0.15	0.38	0.21	383



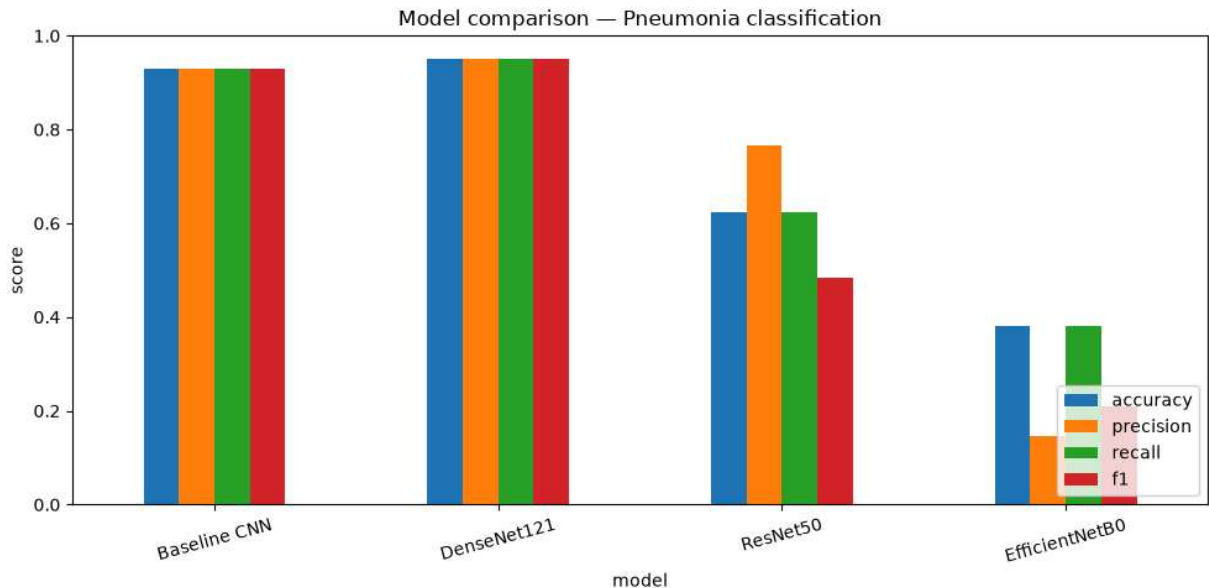
```
In [19]: import pandas as pd

df = pd.DataFrame(results).set_index("model")
df = df[["accuracy", "precision", "recall", "f1"]]
print(df.round(4))

df.plot(kind="bar", figsize=(10, 5))
plt.title("Model comparison - Pneumonia classification")
plt.ylabel("score"); plt.ylim(0, 1.0)
plt.xticks(rotation=15); plt.legend(loc="lower right")
plt.tight_layout(); plt.show()

best = df["f1"].idxmax()
print(f"\nBest model by F1-score: {best} (F1 = {df.loc[best, 'f1']:.4f})")
```

	accuracy	precision	recall	f1
model				
Baseline CNN	0.9295	0.9300	0.9295	0.9289
DenseNet121	0.9504	0.9503	0.9504	0.9504
ResNet50	0.6240	0.7661	0.6240	0.4849
EfficientNetB0	0.3812	0.1453	0.3812	0.2104



Best model by F1-score: DenseNet121 (F1 = 0.9504)

```
In [20]: from tensorflow.keras.preprocessing import image as keras_image

# pick the best-performing model object
model_lookup = {
    "Baseline CNN": cnn_model,
    "DenseNet121": densenet_model,
    "ResNet50": resnet_model,
    "EfficientNetB0": efficientnet_model,
}
best_model = model_lookup[best]

def predict_xray(img_path, model=best_model, img_size=IMG_SIZE):
    img = keras_image.load_img(img_path, target_size=img_size)
    arr = keras_image.img_to_array(img) / 255.0
    arr = np.expand_dims(arr, axis=0)
```

```
prob = model.predict(arr, verbose=0)[0]
idx = int(np.argmax(prob))
label = class_labels[idx]

plt.imshow(img); plt.axis("off")
plt.title(f"Prediction: {label} ({prob[idx]*100:.1f}% confidence)")
plt.show()
return label, float(prob[idx])

# Example: predict on the first test image
sample = glob.glob(os.path.join(test_dir, "PNEUMONIA", "*"))[0]
print(predict_xray(sample))
```

Prediction: PNEUMONIA (99.9% confidence)



('PNEUMONIA', 0.9990498423576355)

In []: