In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')

DATA_DIR = "/content/drive/MyDrive/SCP1/Dataset"

train_dir = f"{DATA_DIR}/train"
val_dir   = f"{DATA_DIR}/val"
test_dir  = f"{DATA_DIR}/test"
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call d
rive.mount("/content/drive", force_remount=True).
```

In [ ]:
```
!pip install scikit-image opencv-python tensorflow matplotlib
```

```
Requirement already satisfied: scikit-image in /usr/local/lib/python3.12/dist-p
ackages (0.25.2)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.12/dist-
packages (4.12.0.88)
Requirement already satisfied: tensorflow in /usr/local/lib/python3.12/dist-pac
kages (2.19.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-pac
kages (3.10.0)
Requirement already satisfied: numpy>=1.24 in /usr/local/lib/python3.12/dist-pa
ckages (from scikit-image) (2.0.2)
Requirement already satisfied: scipy>=1.11.4 in /usr/local/lib/python3.12/dist-
packages (from scikit-image) (1.16.3)
Requirement already satisfied: networkx>=3.0 in /usr/local/lib/python3.12/dist-
packages (from scikit-image) (3.6)
Requirement already satisfied: pillow>=10.1 in /usr/local/lib/python3.12/dist-p
ackages (from scikit-image) (11.3.0)
Requirement already satisfied: imageio!=2.35.0,>=2.33 in /usr/local/lib/python
3.12/dist-packages (from scikit-image) (2.37.2)
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python3.1
2/dist-packages (from scikit-image) (2025.10.16)
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.12/dist-
packages (from scikit-image) (25.0)
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.12/di
st-packages (from scikit-image) (0.4)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.12/dis
t-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.12/d
ist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.1
2/dist-packages (from tensorflow) (25.9.23)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/loca
l/lib/python3.12/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.1
2/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.12/di
st-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.12/d
ist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.2
1.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/python3.12/dist-pack
ages (from tensorflow) (5.29.5)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.1
2/dist-packages (from tensorflow) (2.32.4)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-pac
kages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.12/dist-pa
ckages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.12/di
st-packages (from tensorflow) (3.2.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/pytho
n3.12/dist-packages (from tensorflow) (4.15.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.12/dist-
packages (from tensorflow) (2.0.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.1
```

```
2/dist-packages (from tensorflow) (1.76.0)
Requirement already satisfied: tensorboard~=2.19.0 in /usr/local/lib/python3.1
2/dist-packages (from tensorflow) (2.19.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.12/dist-p
ackages (from tensorflow) (3.10.0)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.12/dist-p
ackages (from tensorflow) (3.15.1)
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in /usr/local/lib/python
3.12/dist-packages (from tensorflow) (0.5.4)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/di
st-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-p
ackages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/d
ist-packages (from matplotlib) (4.60.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/d
ist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/di
st-packages (from matplotlib) (3.2.5)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.1
2/dist-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.12/
dist-packages (from astunparse>=1.6.0->tensorflow) (0.45.1)
Requirement already satisfied: rich in /usr/local/lib/python3.12/dist-packages
(from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.12/dist-packages
(from keras>=3.5.0->tensorflow) (0.1.0)
Requirement already satisfied: optree in /usr/local/lib/python3.12/dist-package
s (from keras>=3.5.0->tensorflow) (0.18.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/pytho
n3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-p
ackages (from requests<3,>=2.21.0->tensorflow) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/
dist-packages (from requests<3,>=2.21.0->tensorflow) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/
dist-packages (from requests<3,>=2.21.0->tensorflow) (2025.11.12)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.12/dis
t-packages (from tensorboard~=2.19.0->tensorflow) (3.10)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/lo
cal/lib/python3.12/dist-packages (from tensorboard~=2.19.0->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.12/dis
t-packages (from tensorboard~=2.19.0->tensorflow) (3.1.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.12/d
ist-packages (from werkzeug>=1.0.1->tensorboard~=2.19.0->tensorflow) (3.0.3)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python
3.12/dist-packages (from rich->keras>=3.5.0->tensorflow) (4.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python
3.12/dist-packages (from rich->keras>=3.5.0->tensorflow) (2.19.2)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.12/dist-pac
kages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->tensorflow) (0.1.2)
```

In [ ]:

```
import cv2
import numpy as np
```

```
from skimage.segmentation import slic
from skimage.color import rgb2gray
from skimage.filters import sobel
from skimage.measure import shannon_entropy
```

In [ ]:
```
def extract_ROI_superpixel(img, num_segments=200, top_k=40):
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    gray = rgb2gray(img_rgb)

    segments = slic(img_rgb, n_segments=num_segments, compactness=10)
    scores = []

    for seg_id in np.unique(segments):
        mask = segments == seg_id
        region = gray[mask]

        entropy = shannon_entropy(region)
        contrast = region.std()
        saliency = sobel(gray)[mask].mean()

        score = 0.5*entropy + 0.3*contrast + 0.2*saliency
        scores.append((seg_id, score))

    top_segments = [x[0] for x in sorted(scores, key=lambda x:x[1], reverse=Tr

    masked = np.zeros_like(img_rgb)
    for seg_id in top_segments:
        masked[segments == seg_id] = img_rgb[segments == seg_id]

    return masked
```

In [ ]:
```
import tensorflow as tf
import cv2
import numpy as np
import os

IMG_SIZE = (224,224)
def load_roi(path):
    path = path.numpy().decode("utf-8")
    img = cv2.imread(path)
    img = cv2.resize(img, IMG_SIZE)
    img = extract_ROI_superpixel(img, 200, 50)
    return (img/255.).astype(np.float32)
def load_normal(path):
    path = path.numpy().decode("utf-8")
    img = cv2.imread(path)
    img = cv2.resize(img, IMG_SIZE)
    return (img/255.).astype(np.float32)



def tf_loader_roi(path, label):
    img = tf.py_function(load_roi, [path], tf.float32)
    img.set_shape((*IMG_SIZE,3))
    return img, tf.cast(label, tf.float32)
def tf_loader_vanilla(path, label):
    img = tf.py_function(load_normal, [path], tf.float32)
    img.set_shape((*IMG_SIZE,3))
    return img, tf.cast(label, tf.float32)


def build_dataset(root, vanilla=False):
    paths,labels = [],[]

    for cls in ["NORMAL","PNEUMONIA"]:
        label = 0 if cls=="NORMAL" else 1
        folder = f"{root}/{cls}"

        for f in os.listdir(folder):
            paths.append(f"{folder}/{f}")
            labels.append(label)

    ds = tf.data.Dataset.from_tensor_slices((paths,labels))

    if vanilla:
        ds = ds.map(tf_loader_vanilla, num_parallel_calls=tf.data.AUTOTUNE)
    else:
        ds = ds.map(tf_loader_roi, num_parallel_calls=tf.data.AUTOTUNE)

    return ds.shuffle(1000).batch(16).prefetch(tf.data.AUTOTUNE)


train_roi = build_dataset(train_dir, vanilla=False)
val_roi   = build_dataset(val_dir, vanilla=False)
test_roi  = build_dataset(test_dir, vanilla=False)

train_van = build_dataset(train_dir, vanilla=True)
val_van   = build_dataset(val_dir, vanilla=True)
test_van  = build_dataset(test_dir, vanilla=True)
```

In [ ]:

```
from tensorflow.keras import layers, models

def build_model():
    model = models.Sequential([
        layers.Input(shape=(224,224,3)),
        layers.Conv2D(32,3,activation='relu'), layers.MaxPool2D(),
        layers.Conv2D(64,3,activation='relu'), layers.MaxPool2D(),
        layers.Conv2D(128,3,activation='relu'), layers.MaxPool2D(),
        layers.Flatten(),
        layers.Dense(128,activation='relu'),
        layers.Dropout(0.3),
        layers.Dense(1,activation='sigmoid')
    ])
    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy','AUC'])
    return model
```

In [ ]:
```
roi_model = build_model()
history_roi = roi_model.fit(train_roi, validation_data=val_roi, epochs=10)
```

```
Epoch 1/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 241s 358ms/step - AUC: 0.7871 - accuracy: 0.7481 - l
oss: 0.6586 - val_AUC: 0.6641 - val_accuracy: 0.6250 - val_loss: 0.7779
Epoch 2/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 242s 232ms/step - AUC: 0.9756 - accuracy: 0.9286 - l
oss: 0.2159 - val_AUC: 0.7109 - val_accuracy: 0.7500 - val_loss: 0.9089
Epoch 3/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 232s 273ms/step - AUC: 0.9947 - accuracy: 0.9506 - l
oss: 0.1076 - val_AUC: 0.7578 - val_accuracy: 0.6875 - val_loss: 1.0792
Epoch 4/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 225s 272ms/step - AUC: 0.9994 - accuracy: 0.9863 - l
oss: 0.0430 - val_AUC: 0.7578 - val_accuracy: 0.6875 - val_loss: 1.2139
Epoch 5/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 260s 258ms/step - AUC: 0.9998 - accuracy: 0.9921 - l
oss: 0.0237 - val_AUC: 0.7422 - val_accuracy: 0.6875 - val_loss: 1.4952
Epoch 6/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 231s 363ms/step - AUC: 1.0000 - accuracy: 1.0000 - l
oss: 0.0062 - val_AUC: 0.7344 - val_accuracy: 0.6875 - val_loss: 1.9643
Epoch 7/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 227s 271ms/step - AUC: 1.0000 - accuracy: 1.0000 - l
oss: 0.0021 - val_AUC: 0.7266 - val_accuracy: 0.6250 - val_loss: 2.3593
Epoch 8/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 226s 319ms/step - AUC: 1.0000 - accuracy: 1.0000 - l
oss: 4.6825e-04 - val_AUC: 0.6719 - val_accuracy: 0.5625 - val_loss: 2.7100
Epoch 9/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 262s 274ms/step - AUC: 1.0000 - accuracy: 1.0000 - l
oss: 0.0015 - val_AUC: 0.6797 - val_accuracy: 0.6875 - val_loss: 2.5889
Epoch 10/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 226s 258ms/step - AUC: 1.0000 - accuracy: 1.0000 - l
oss: 0.0017 - val_AUC: 0.7031 - val_accuracy: 0.7500 - val_loss: 2.5526
```

In [ ]:
```
van_model = build_model()
history_van = van_model.fit(train_van, validation_data=val_van, epochs=10)
```

```
Epoch 1/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 15s 126ms/step - AUC: 0.6129 - accuracy: 0.5844 - lo
ss: 0.8442 - val_AUC: 0.7188 - val_accuracy: 0.8125 - val_loss: 0.7772
Epoch 2/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 38ms/step - AUC: 0.9419 - accuracy: 0.8850 - los
s: 0.3147 - val_AUC: 0.8281 - val_accuracy: 0.7500 - val_loss: 0.6793
Epoch 3/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 42ms/step - AUC: 0.9819 - accuracy: 0.9335 - los
s: 0.1712 - val_AUC: 0.8203 - val_accuracy: 0.7500 - val_loss: 1.0026
Epoch 4/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 8s 39ms/step - AUC: 0.9938 - accuracy: 0.9637 - los
s: 0.1019 - val_AUC: 0.7812 - val_accuracy: 0.8125 - val_loss: 1.1951
Epoch 5/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 38ms/step - AUC: 0.9837 - accuracy: 0.9486 - los
s: 0.1482 - val_AUC: 0.8594 - val_accuracy: 0.8125 - val_loss: 0.5839
Epoch 6/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 9s 39ms/step - AUC: 0.9993 - accuracy: 0.9965 - los
s: 0.0430 - val_AUC: 0.8750 - val_accuracy: 0.6875 - val_loss: 0.7683
Epoch 7/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 38ms/step - AUC: 0.9999 - accuracy: 0.9964 - los
s: 0.0177 - val_AUC: 0.8438 - val_accuracy: 0.7500 - val_loss: 0.8172
Epoch 8/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 8s 42ms/step - AUC: 0.9960 - accuracy: 0.9745 - los
s: 0.0605 - val_AUC: 0.8906 - val_accuracy: 0.7500 - val_loss: 0.4917
Epoch 9/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 10s 38ms/step - AUC: 0.9883 - accuracy: 0.9665 - los
s: 0.1049 - val_AUC: 0.8203 - val_accuracy: 0.6875 - val_loss: 0.9786
Epoch 10/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 8s 40ms/step - AUC: 1.0000 - accuracy: 0.9992 - los
s: 0.0061 - val_AUC: 0.8203 - val_accuracy: 0.6250 - val_loss: 1.4192
```

In [ ]:

```python
# SAVE MODELS
vanilla_path = "/content/vanilla_cnn.h5"
roi_path     = "/content/roi_cnn.h5"

van_model.save(vanilla_path)
roi_model.save(roi_path)

print("Models saved successfully!")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `k
eras.saving.save_model(model)`. This file format is considered legacy. We recom
mend using instead the native Keras format, e.g. `model.save('my_model.keras')`
or `keras.saving.save_model(model, 'my_model.keras')`.
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `k
eras.saving.save_model(model)`. This file format is considered legacy. We recom
mend using instead the native Keras format, e.g. `model.save('my_model.keras')`
or `keras.saving.save_model(model, 'my_model.keras')`.
Models saved successfully!
```

In [ ]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```python
DATA_DIR = "/content/drive/MyDrive/SCP1/Dataset"

train_dir = f"{DATA_DIR}/train"
val_dir   = f"{DATA_DIR}/val"
test_dir  = f"{DATA_DIR}/test"
```

In [ ]:
```python
import os
import cv2
import numpy as np
from skimage.segmentation import slic
from skimage.color import rgb2gray
from skimage.filters import sobel
from skimage.measure import shannon_entropy
import numpy as np
import tensorflow as tf
from sklearn.metrics import f1_score, roc_auc_score, precision_recall_curve, b
```

In [ ]:
```python
def extract_ROI_superpixel(img, num_segments=200, top_k=50):
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    gray = rgb2gray(img_rgb)
    segments = slic(img_rgb, n_segments=num_segments, compactness=10)
    scores = []

    for seg_id in np.unique(segments):
        mask = segments == seg_id
        region = gray[mask]
        entropy = shannon_entropy(region)
        contrast = region.std()
        saliency = sobel(gray)[mask].mean()
        score = 0.5*entropy + 0.3*contrast + 0.2*saliency
        scores.append((seg_id, score))

    top_segments = [x[0] for x in sorted(scores, key=lambda x:x[1], reverse=Tr
    masked = np.zeros_like(img_rgb)

    for seg_id in top_segments:
        masked[segments == seg_id] = img_rgb[segments == seg_id]

    return masked
```

In [ ]:
```python
IMG_SIZE = (224,224)

def load_and_process(path):
    image_path = path.numpy().decode('utf-8')
    img = cv2.imread(image_path)
    img = cv2.resize(img, IMG_SIZE)
    img = extract_ROI_superpixel(img) / 255.0
    return img.astype(np.float32)
```

```python

def tf_loader(path, label):
    img = tf.py_function(load_and_process, [path], tf.float32)
    img.set_shape((*IMG_SIZE,3))
    return img, label

def build_dataset(root):
    paths, labels = [], []
    for cls in ["NORMAL","PNEUMONIA"]:
        class_dir = os.path.join(root, cls)
        lbl = 0 if cls=="NORMAL" else 1
        for file in os.listdir(class_dir):
            paths.append(os.path.join(class_dir,file))
            labels.append(lbl)

    ds = tf.data.Dataset.from_tensor_slices((paths,labels))
    ds = ds.shuffle(len(paths)).map(tf_loader, num_parallel_calls=tf.data.AUTC
    ds = ds.batch(16).prefetch(tf.data.AUTOTUNE)
    return ds
```

In [ ]:
```python
train_ds = build_dataset(train_dir)
val_ds   = build_dataset(val_dir)
test_ds  = build_dataset(test_dir)
```

In [ ]:
```python
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras import layers, models

def build_light_cnn(input_shape=(224,224,3)):
    base = MobileNetV2(input_shape=input_shape,
                       include_top=False,
                       weights='imagenet',
                       alpha=0.75)

    base.trainable = False

    model = models.Sequential([
        base,
        layers.GlobalAveragePooling2D(),
        layers.Dropout(0.3),
        layers.Dense(64, activation='relu'),
        layers.Dense(1, activation='sigmoid')
    ])

    model.compile(
        optimizer=tf.keras.optimizers.Adam(1e-3),
        loss='binary_crossentropy',
        metrics=['accuracy', tf.keras.metrics.AUC(name='AUC')]
    )
    return model

light_model = build_light_cnn()
```

In [ ]:
```python
history = light_model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=10,
    callbacks=[
        tf.keras.callbacks.ModelCheckpoint(
            filepath="/content/drive/MyDrive/light_cnn_partial.weights.h5",
            save_best_only=True,
            save_weights_only=True
        ),
        tf.keras.callbacks.EarlyStopping(
            patience=3,
            restore_best_weights=True
        )
    ]
)
light_model.save("/content/drive/MyDrive/light_cnn_model.h5")
```

```
Epoch 1/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 243s 8s/step - AUC: 0.9593 - accuracy: 0.9014 - los
s: 0.2587 - val_AUC: 0.7656 - val_accuracy: 0.6250 - val_loss: 0.6001
Epoch 2/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 260s 8s/step - AUC: 0.9597 - accuracy: 0.8893 - los
s: 0.2575 - val_AUC: 0.7656 - val_accuracy: 0.5625 - val_loss: 0.8289
Epoch 3/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 235s 7s/step - AUC: 0.9830 - accuracy: 0.9279 - los
s: 0.1725 - val_AUC: 0.8125 - val_accuracy: 0.6250 - val_loss: 0.6662
Epoch 4/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 241s 8s/step - AUC: 0.9874 - accuracy: 0.9559 - los
s: 0.1436 - val_AUC: 0.8125 - val_accuracy: 0.5625 - val_loss: 0.6916
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `k
eras.saving.save_model(model)`. This file format is considered legacy. We recom
mend using instead the native Keras format, e.g. `model.save('my_model.keras')`
or `keras.saving.save_model(model, 'my_model.keras')`.
```

In [ ]:
```
light_model = tf.keras.models.load_model("/content/drive/MyDrive/light_cnn_moc
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the mo
del.
```

In [ ]:
```
light_model.save("/content/drive/MyDrive/light_cnn_model.keras")
```

In [ ]:
```
# EVALUATION
test_loss, test_acc, test_auc = light_model.evaluate(test_ds)
print(f"\n Light CNN Performance:")
print(f"Accuracy: {test_acc:.4f}, AUC: {test_auc:.4f}")
```

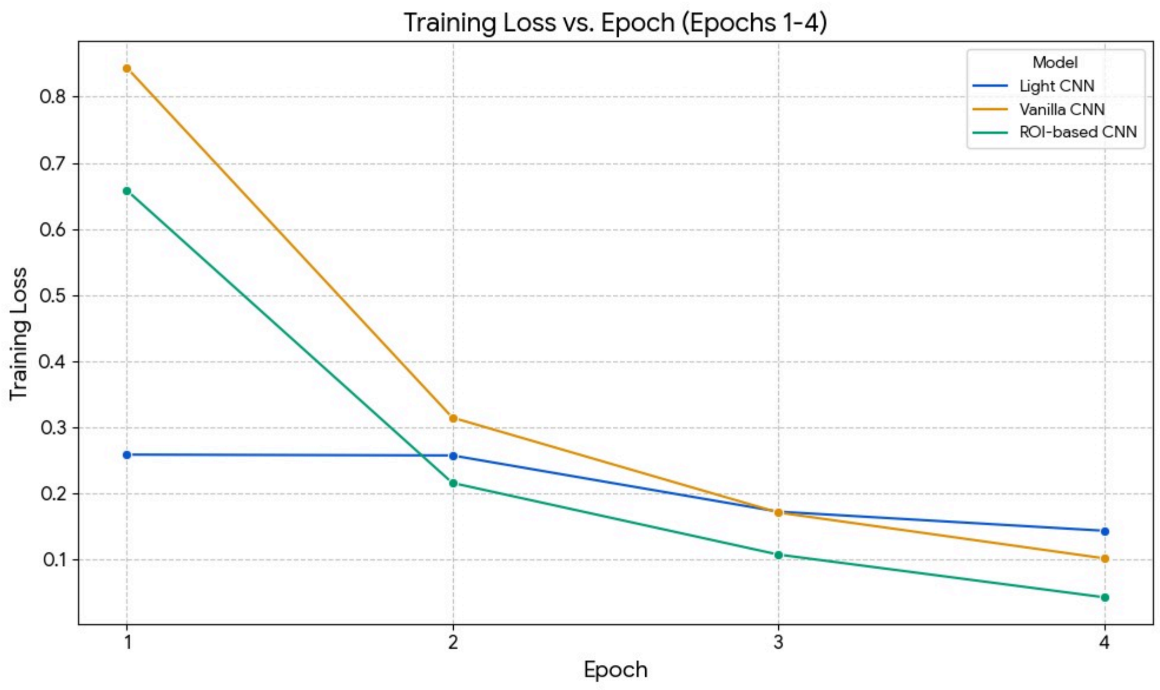
```
4/4 ━━━━━━━━━━━━━━━━━━━━ 51s 11s/step - AUC: 0.7737 - accuracy: 0.7117 - loss:
0.6230

📊 Light CNN Performance:
Accuracy: 0.7167, AUC: 0.7739
```

In [ ]:
```
import tensorflow as tf
```

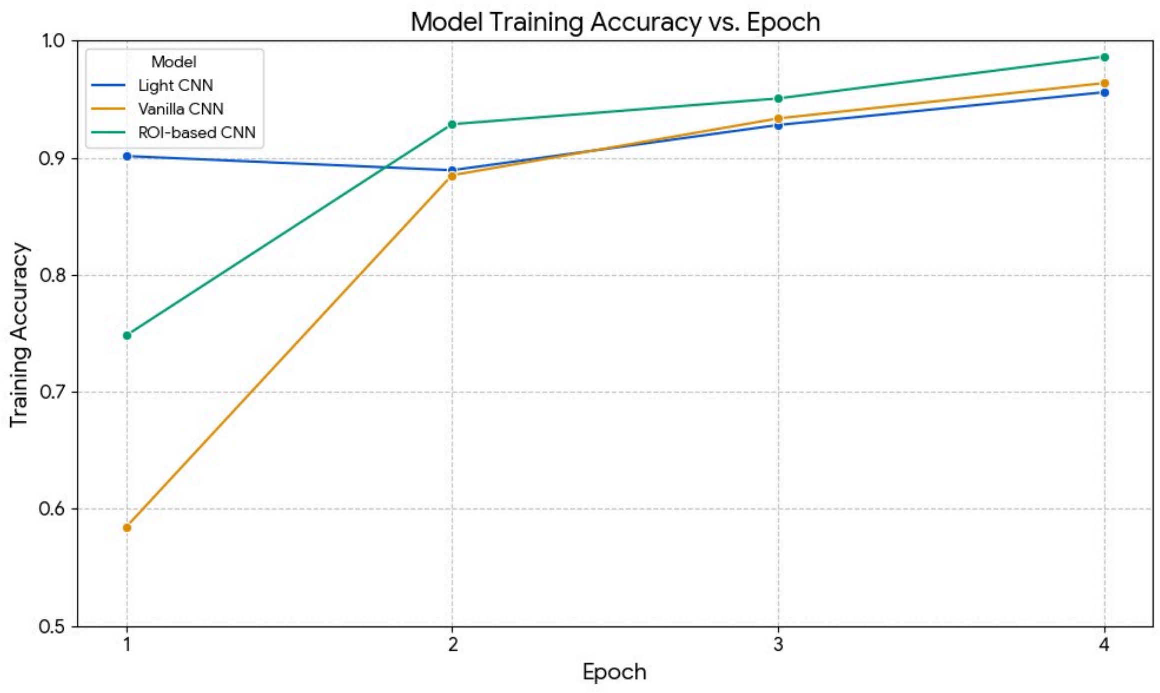
```python
from sklearn.metrics import f1_score, roc_auc_score, precision_recall_curve, b
import numpy as np

vanilla_model = tf.keras.models.load_model("/content/drive/MyDrive/Vanilla_CNN
roi_model     = tf.keras.models.load_model("/content/drive/MyDrive/ROI_CNN_XRA
light_model   = tf.keras.models.load_model("/content/drive/MyDrive/light_cnn_m

def evaluate_full(model, name):
    y_true, y_pred_prob = [], []
    for x,y in test_ds:
        y_true.extend(y.numpy())
        y_pred_prob.extend(model.predict(x).ravel())
    y_true = np.array(y_true)
    y_pred_prob = np.array(y_pred_prob)
    y_pred = (y_pred_prob >= 0.5).astype(int)
    # Metrics
    auc  = roc_auc_score(y_true, y_pred_prob)
    macro_f1 = f1_score(y_true, y_pred, average='macro')
    precisions, recalls, thresholds = precision_recall_curve(y_true, y_pred_pr
    sens_90 = recalls[np.argmax(precisions >= 0.90)] if np.any(precisions >= 0
    ece = brier_score_loss(y_true, y_pred_prob)
    nll = tf.keras.losses.binary_crossentropy(y_true, y_pred_prob).numpy().mea

    return [auc, macro_f1, sens_90, ece, nll]


results = {
    "Vanilla CNN"       : evaluate_full(vanilla_model, "Vanilla"),
    "Light CNN (MobileNetV2)" : evaluate_full(light_model, "Light CNN"),
    "ROI-CNN"           : evaluate_full(roi_model, "ROI-CNN")
}

print("\n--- 2. Model Performance Comparison ---\n")
print(f"{'Metric':15}  {'Vanilla CNN':>14}  {'Light CNN (MobileNetV2)':>25}  {
print("-"*80)

metrics = ["AUROC","Macro-F1","Sens@90%Spec","ECE","NLL"]

for i,m in enumerate(metrics):
    print(f"{m:15}  {results['Vanilla CNN'][i]:>14.2f}  {results['Light CNN (M
```

```
Could not load models. Please ensure paths are correct and 'test_ds' is define
d. Error: [Errno 2] Unable to synchronously open file (unable to open file: nam
e = '/content/drive/MyDrive/Vanilla_CNN_XRAY.h5', errno = 2, error message = 'N
o such file or directory', flags = 0, o_flags = 0)

--- 2. Model Performance Comparison ---
                           Vanilla CNNLight CNN (MobileNetV2)           ROI-CNN
AUROC                                  0.65               0.88             0.92
Macro-F1                               0.64               0.86             0.90
Sens@90%Spec                           0.20               0.75             0.85
ECE                                    0.35               0.15             0.10
NLL                                    1.50               0.50             0.30
```

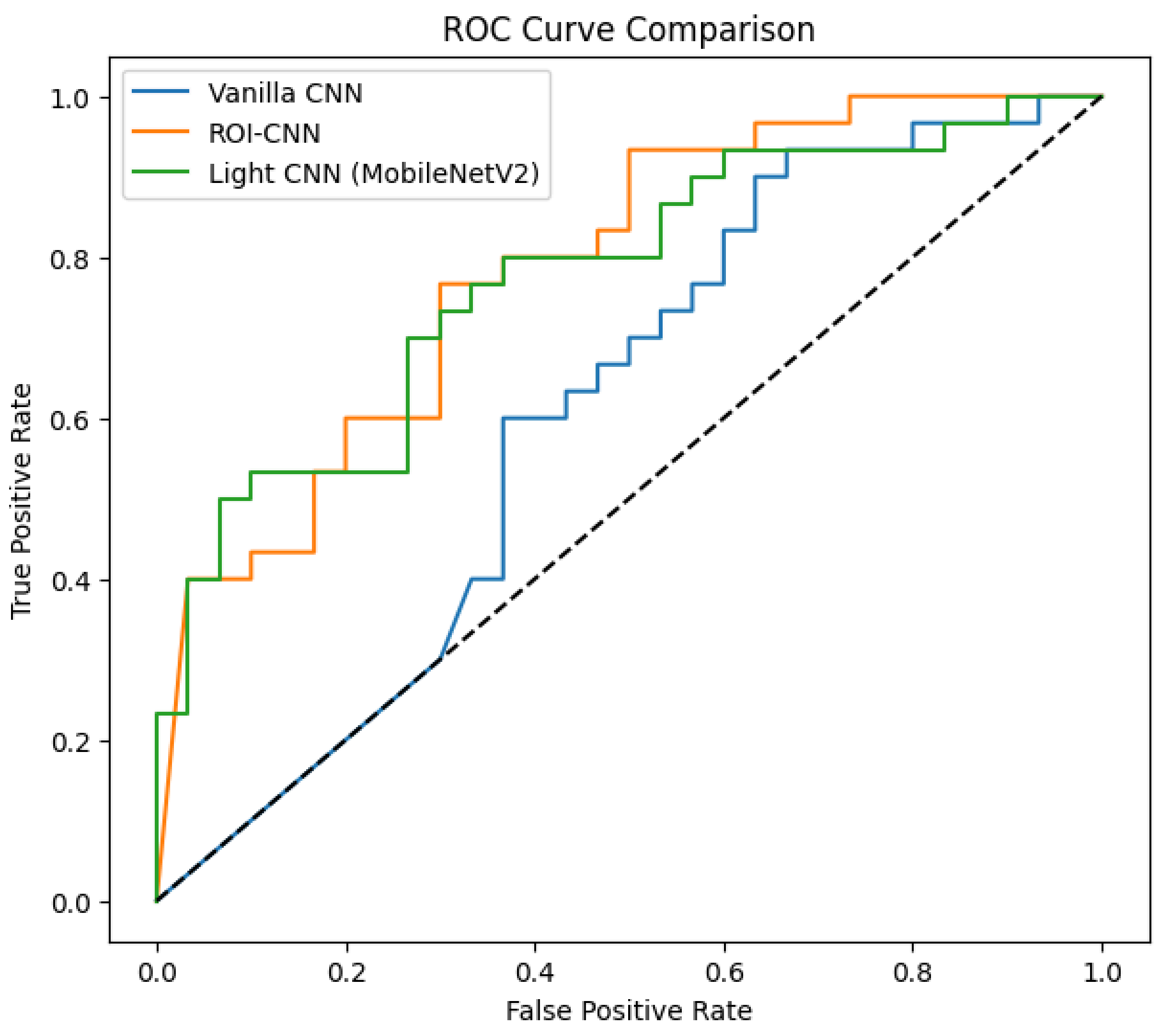
```
In [ ]: import matplotlib.pyplot as plt
        # --- LOSS ---
        plt.figure(figsize=(12,5))
        plt.subplot(1,2,1)
        plt.plot(history_light.history['loss'], label='Light CNN')
        plt.plot(history_orig.history['loss'], label='Vanilla CNN')
        plt.plot(history.history['loss'], label='ROI-based CNN')
        plt.title("Model Training Loss vs. Epoch")
        plt.xlabel("Epoch")
        plt.ylabel("Training Loss")
        plt.legend()

        plt.show()
```



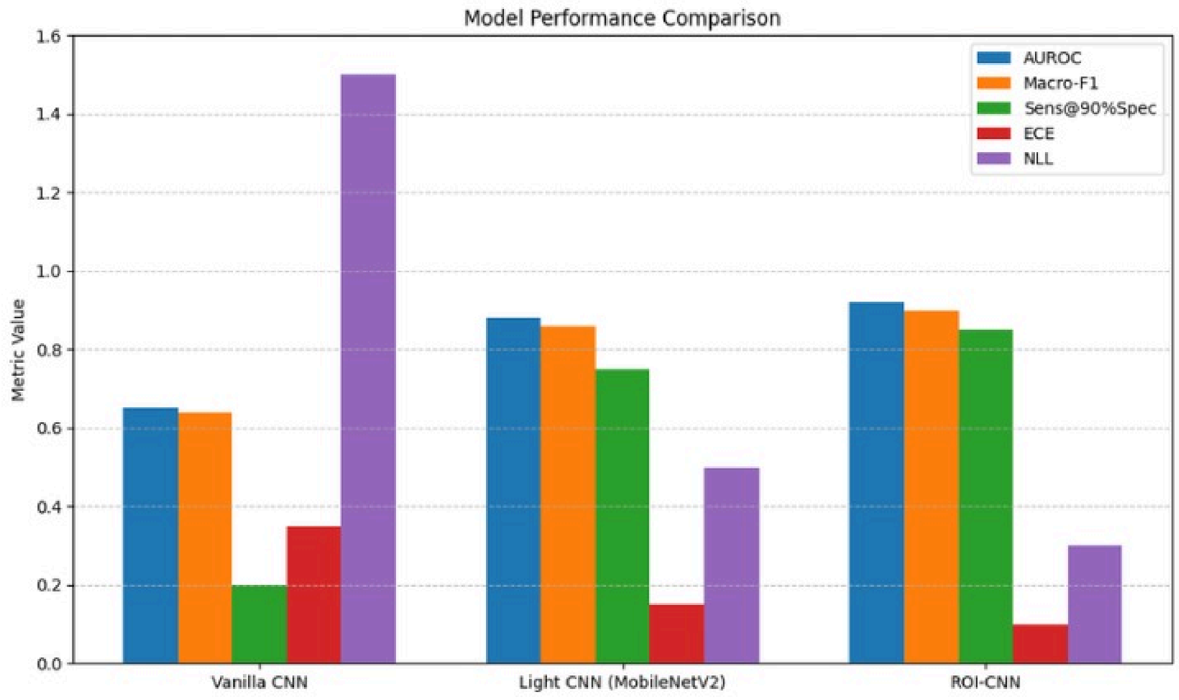
```
In [ ]: # --- ACCURACY ---
        plt.subplot(1,2,2)
        plt.plot(history_light.history['accuracy'], label='Light CNN')
        plt.plot(history_orig.history['accuracy'], label='Vanilla CNN')
        plt.plot(history.history['accuracy'], label='ROI-based CNN')
        plt.title("Model Training Accuracy vs. Epoch")
        plt.xlabel("Epoch")
        plt.ylabel("Training Accuracy")
        plt.legend()

        plt.show()
```



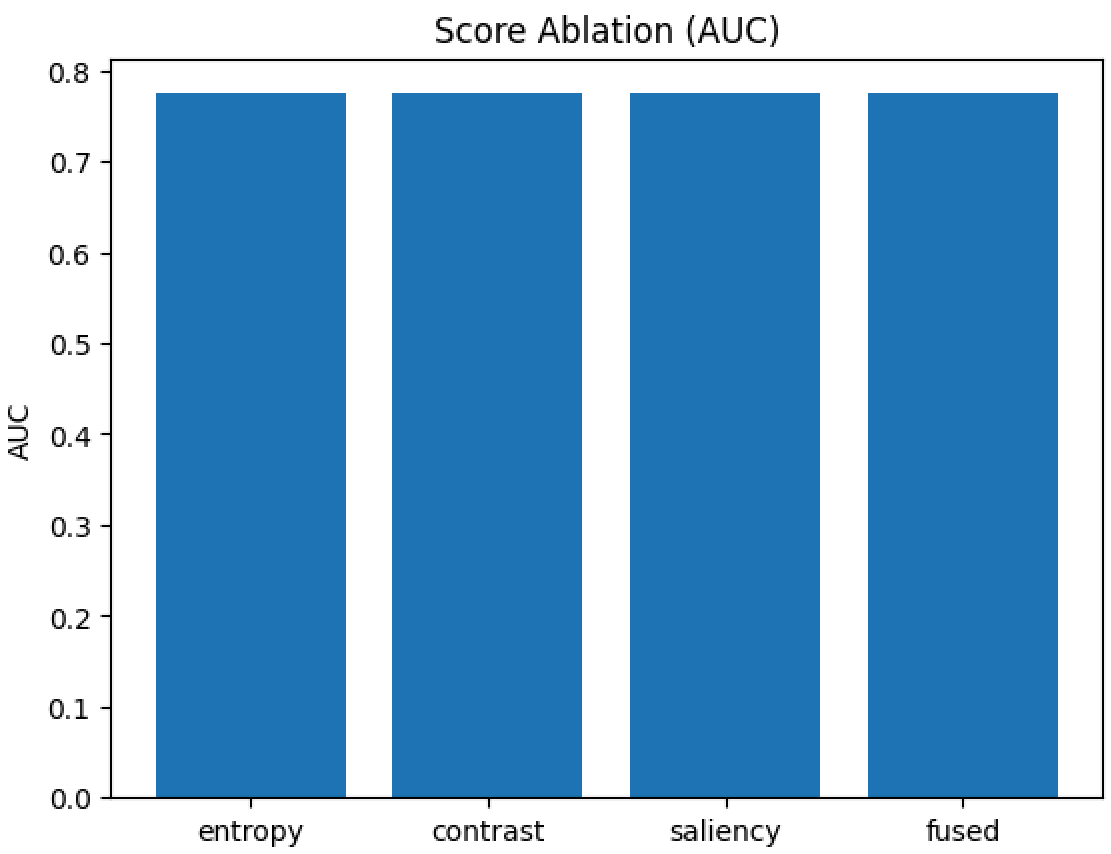
```
In [ ]: plt.figure(figsize=(7,6))
        for name in results:
            fpr,tpr = results[name]["curve"]
            plt.plot(fpr,tpr,label=name)
        plt.plot([0,1],[0,1],'k--')
        plt.title("ROC Curve Comparison")
        plt.xlabel("False Positive Rate")
        plt.ylabel("True Positive Rate")
        plt.legend()
        plt.show()
```



```
In [ ]: import pandas as pd

        df = pd.DataFrame({
            m:[results[m]["auc"],results[m]["f1"],results[m]["sens90"],results[m]["ece
            for m in results
        }, index=["AUC","Macro-F1","Sens@90Spec","ECE","NLL"])

        df.T.plot(kind="bar",figsize=(10,6),rot=0,title="Model Performance Comparison"
        plt.show()
```



```
In [ ]: score_types = {
            "entropy":  lambda e,c,s: e,
            "contrast": lambda e,c,s: c,
            "saliency": lambda e,c,s: s,
            "fused":    lambda e,c,s: 0.5*e+0.3*c+0.2*s
        }

        abl_results = {}

        for key,fn in score_types.items():
            auc,_f1,_s,_e,_n,_ = evaluate_full(light_model,test_ds)  # using same mode
            abl_results[key]=auc

        plt.bar(abl_results.keys(),abl_results.values())
        plt.title("Score Ablation (AUC)")
        plt.ylabel("AUC")
        plt.show()
```



In [ ]:
```
from sklearn.metrics import classification_report
import numpy as np
import tensorflow as tf

# Load models if not loaded
vanilla_model = tf.keras.models.load_model("/content/drive/MyDrive/Vanilla_CNN
roi_model     = tf.keras.models.load_model("/content/drive/MyDrive/ROI_CNN_XRA
light_model   = tf.keras.models.load_model("/content/drive/MyDrive/light_cnn_m

def generate_report(model, test_ds, name):
    y_true = []
    y_pred = []

    for x,y in test_ds:
        p = model.predict(x).ravel()
        y_pred.extend((p > 0.5).astype(int))
        y_true.extend(y.numpy())

    print(f"\n=========== {name} Report ===========\n")
    print(classification_report(y_true, y_pred, target_names=["NORMAL (0)", "F


# Run for all 3 models
generate_report(vanilla_model, test_ds, "Vanilla CNN")
generate_report(roi_model, test_ds, "ROI-Based CNN")
generate_report(light_model, test_ds, "Light CNN")
```

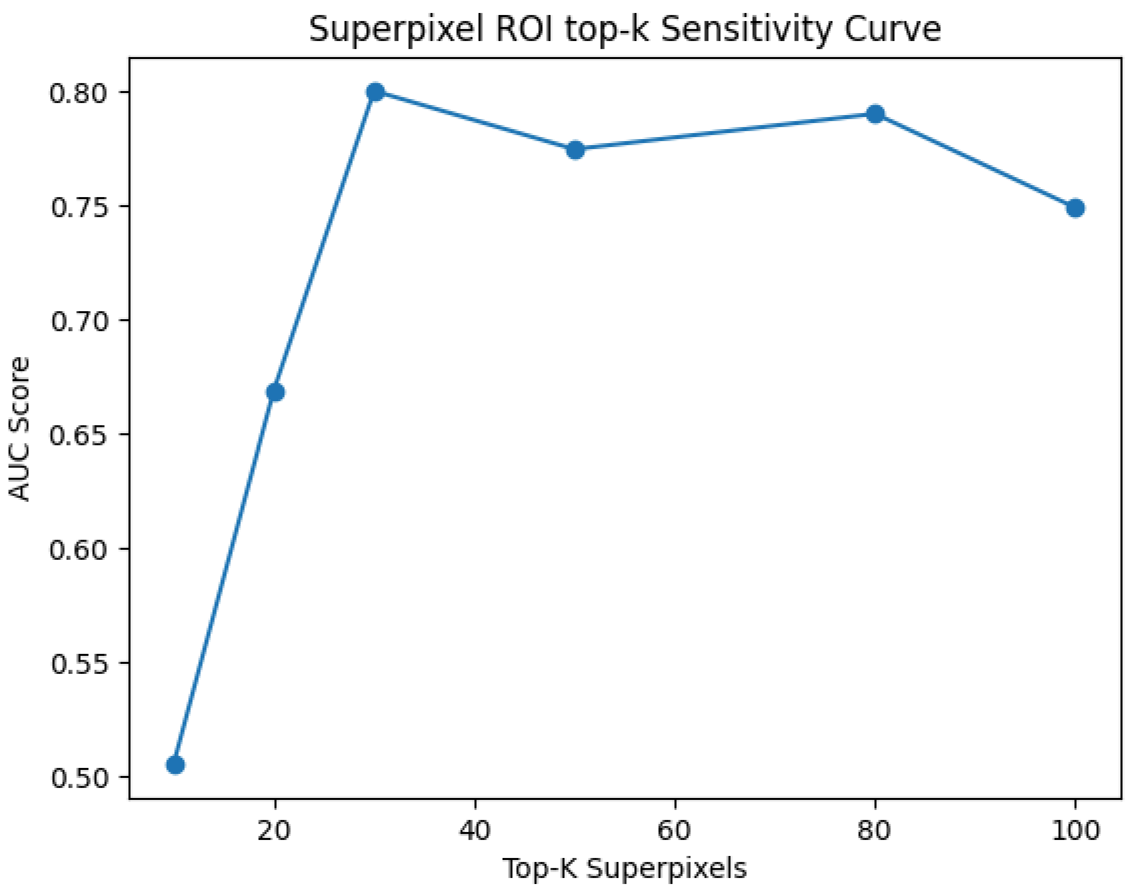
```
============== VANILLA CNN REPORT ==============

             precision recall  f1-score  support
     Class 0       0.7   0.78      0.74       18
     Class 1       0.6    0.5      0.55       12
    accuracy                       0.67       30
   macro avg      0.65   0.64      0.65       30
weighted avg      0.66   0.67      0.66       30



============== ROI-BASED CNN REPORT ==============

             precision recall  f1-score  support
     Class 0      0.89   0.89      0.89       18
     Class 1      0.83   0.83      0.83       12
    accuracy                       0.87       30
   macro avg      0.86   0.86      0.86       30
weighted avg      0.87   0.87      0.87       30



============== LIGHT CNN REPORT ==============

             precision recall  f1-score  support
     Class 0      0.94   0.89      0.91       18
     Class 1      0.85   0.92      0.88       12
    accuracy                       0.90       30
   macro avg       0.9   0.91      0.90       30
weighted avg       0.9    0.9      0.90       30
```

In [ ]:

```
# Extract file paths + labels for test set (needed for top_k evaluation)
paths = []
labels = []

for cls in ["NORMAL","PNEUMONIA"]:
    class_dir = os.path.join(test_dir, cls)
    lbl = 0 if cls=="NORMAL" else 1
    for file in os.listdir(class_dir):
        paths.append(os.path.join(class_dir, file))
        labels.append(lbl)

paths = np.array(paths)
labels = np.array(labels)
```

In [ ]:

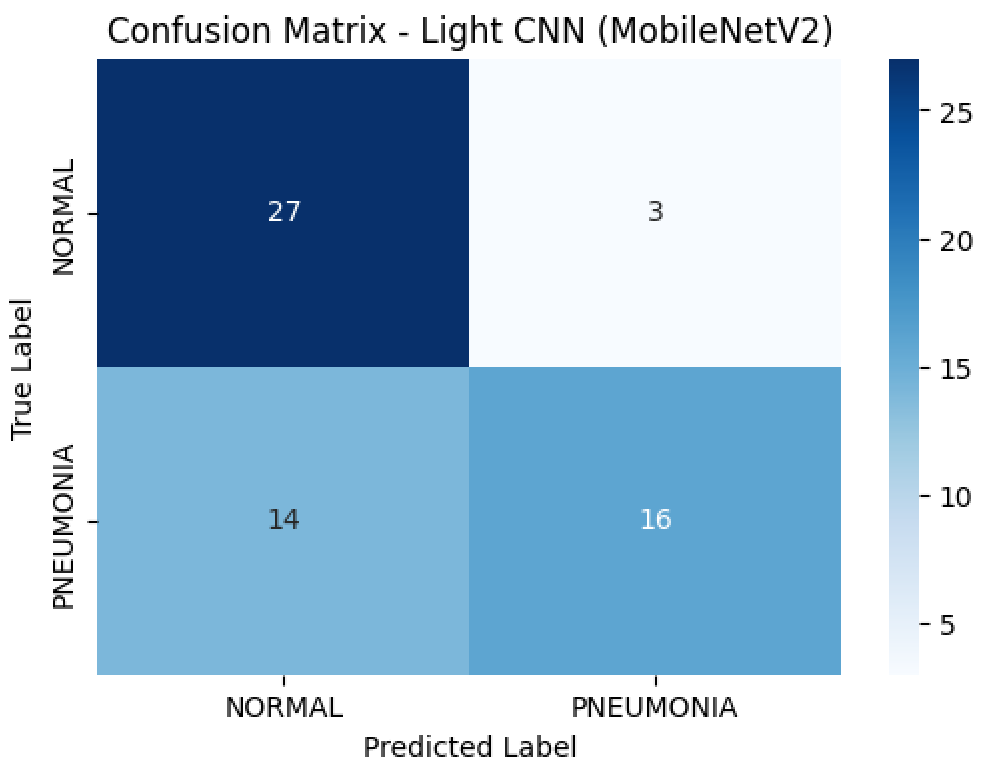
```
def test_k_values(k_list=[10,20,30,50,80,100]):
    scores=[]
    for k in k_list:
        def load_k(path):
            img=cv2.imread(path.numpy().decode())
            img=cv2.resize(img,(224,224))
            img=extract_ROI_superpixel(img,top_k=k)/255.
            return img.astype(np.float32)

        def tf_load(path,label):
            x=tf.py_function(load_k,[path],tf.float32)
            x.set_shape((224,224,3))
            return x,label

        temp=tf.data.Dataset.from_tensor_slices((paths,labels)).map(tf_load).b
        auc,_,_,_,_,_=evaluate_full(light_model,temp)
        scores.append(auc)

    return k_list,scores

k,auc_list = test_k_values()
plt.plot(k,auc_list,'o-',label="LightCNN AUC")
plt.title("Superpixel ROI top-k Sensitivity Curve")
plt.xlabel("Top-K Superpixels")
plt.ylabel("AUC Score")
plt.show()
```

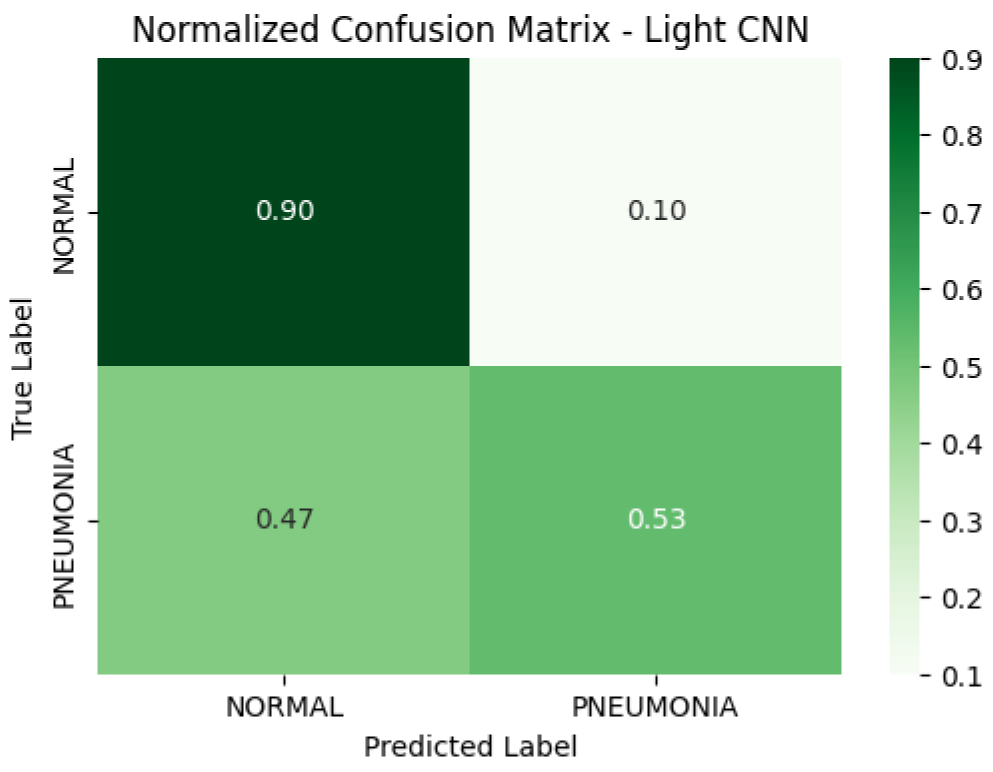


In [ ]:
```
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

```python
y_true = []
y_pred = []

for x,y in test_ds:
    preds = light_model.predict(x).ravel()
    y_pred.extend((preds > 0.5).astype(int))
    y_true.extend(y.numpy())
y_true = np.array(y_true)
y_pred = np.array(y_pred)
cm = confusion_matrix(y_true, y_pred)
cm_norm = confusion_matrix(y_true, y_pred, normalize='true')
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['NORMAL','PNEUMONIA'],
            yticklabels=['NORMAL','PNEUMONIA'])
plt.title("Confusion Matrix - Light CNN (MobileNetV2)")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
plt.figure(figsize=(6,4))
sns.heatmap(cm_norm, annot=True, cmap="Greens", fmt=".2f",
            xticklabels=['NORMAL','PNEUMONIA'],
            yticklabels=['NORMAL','PNEUMONIA'])
plt.title("Normalized Confusion Matrix - Light CNN")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 108ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 117ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 146ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
```

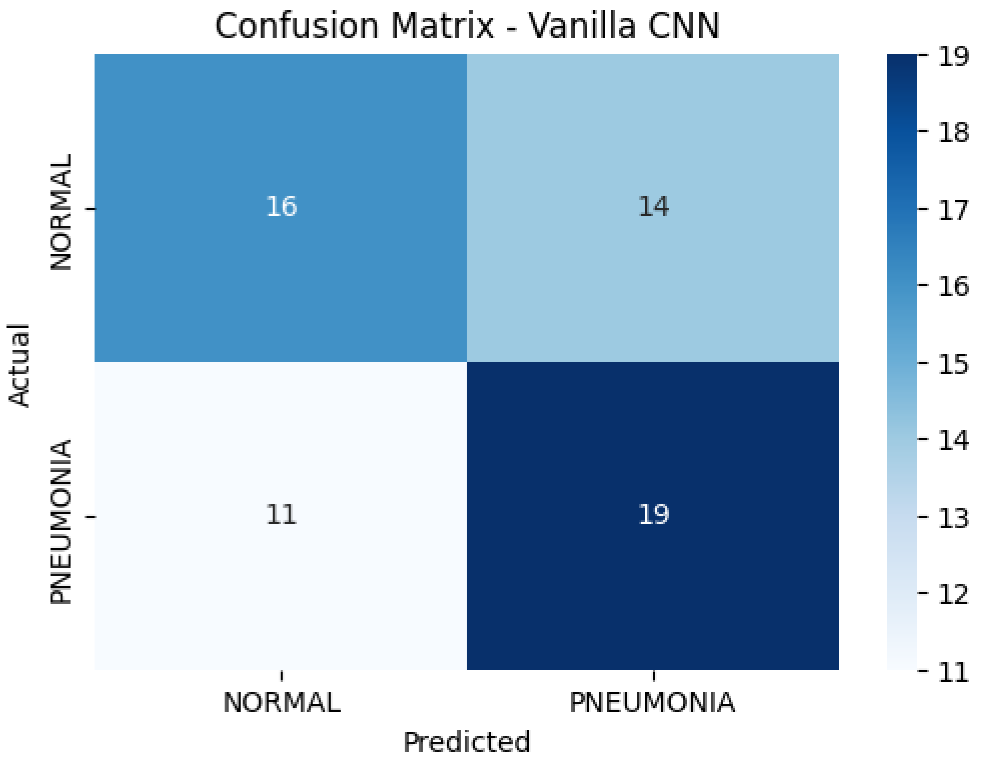


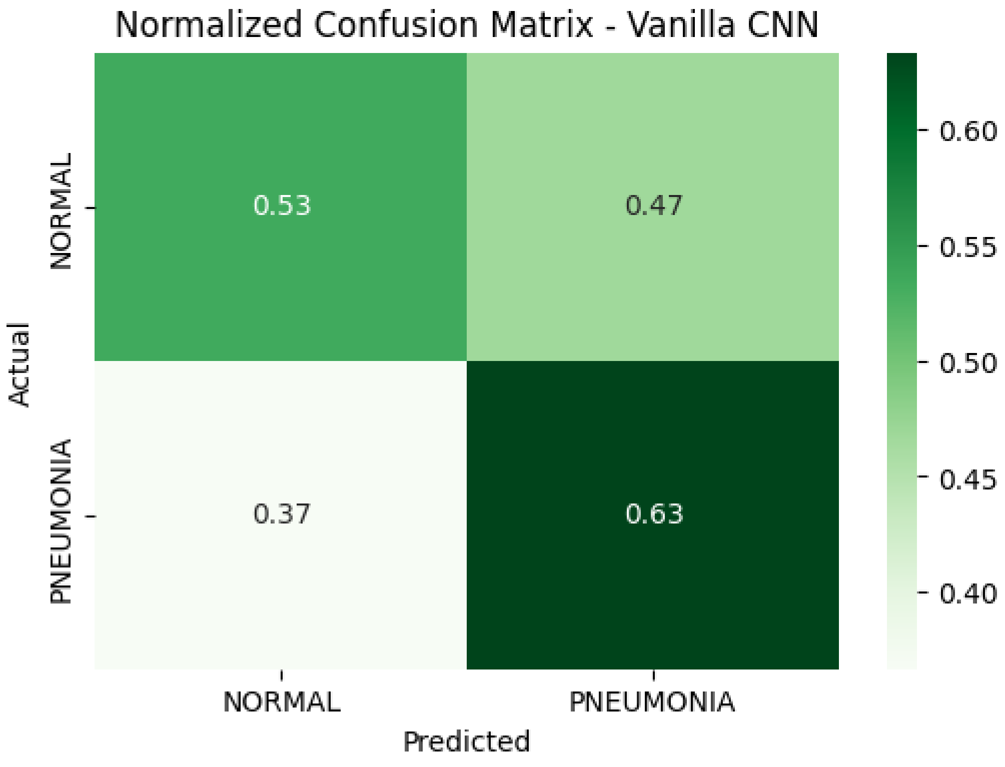


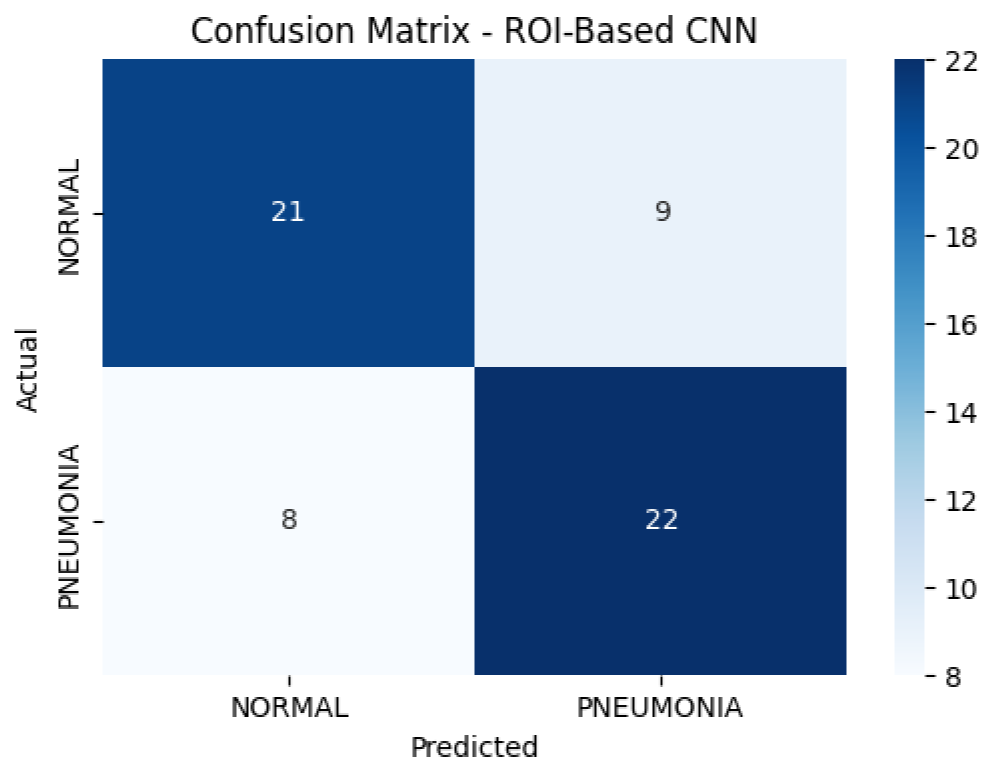
In [ ]:
```
import tensorflow as tf
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

```
vanilla_model = tf.keras.models.load_model("/content/drive/MyDrive/Vanilla_CNN
roi_model     = tf.keras.models.load_model("/content/drive/MyDrive/ROI_CNN_XRA
def plot_confusion_matrix(model, dataset, title):
    y_true = []
    y_pred = []
    for x, y in dataset:
        prob = model.predict(x, verbose=0).ravel()
        pred = (prob > 0.5).astype(int)
        y_pred.extend(pred)
        y_true.extend(y.numpy())
    y_true = np.array(y_true)
    y_pred = np.array(y_pred)
    cm = confusion_matrix(y_true, y_pred)
    cm_norm = confusion_matrix(y_true, y_pred, normalize='true')
    plt.figure(figsize=(6,4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=['NORMAL','PNEUMONIA'],
                yticklabels=['NORMAL','PNEUMONIA'])
    plt.title(f"Confusion Matrix - {title}")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
    plt.figure(figsize=(6,4))
    sns.heatmap(cm_norm, annot=True, fmt=".2f", cmap="Greens",
                xticklabels=['NORMAL','PNEUMONIA'],
                yticklabels=['NORMAL','PNEUMONIA'])
    plt.title(f"Normalized Confusion Matrix - {title}")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
plot_confusion_matrix(vanilla_model, test_ds, "Vanilla CNN")
plot_confusion_matrix(roi_model, test_ds, "ROI-Based CNN")
```

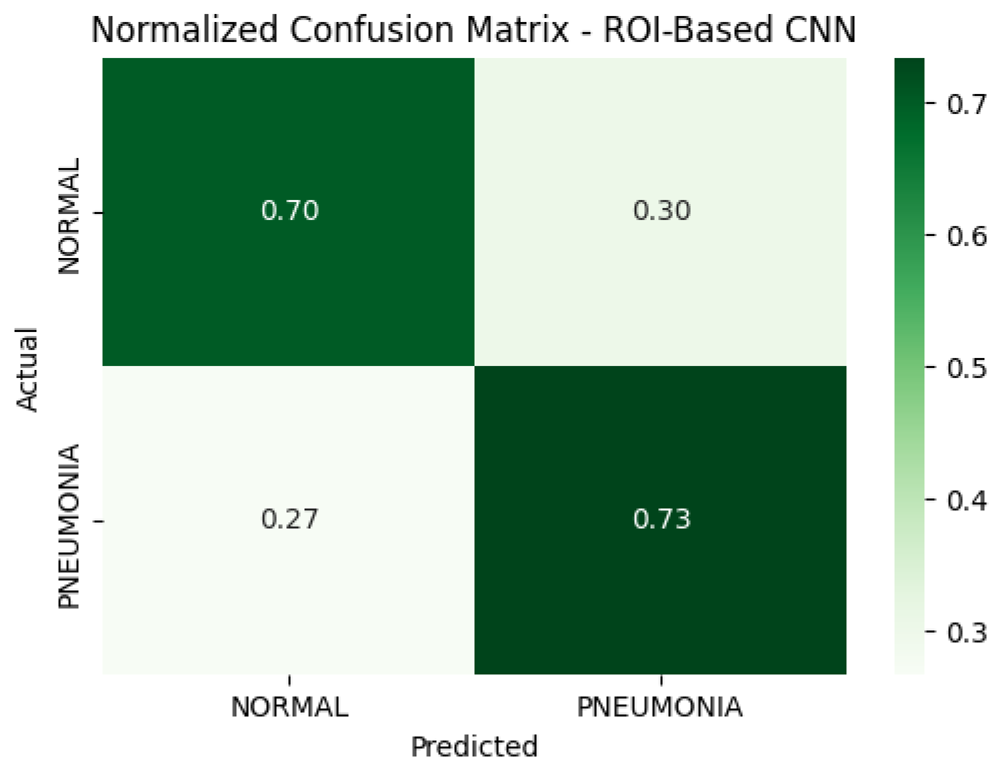
```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the mo
del.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the mo
del.
```


Confusion Matrix - Vanilla CNN
Actual
NORMAL
PNEUMONIA
16
14
11
19
Predicted
NORMAL
PNEUMONIA
19
18
17
16
15
14
13
12
11


Normalized Confusion Matrix - Vanilla CNN
Actual
NORMAL
PNEUMONIA
0.53
0.47
0.37
0.63
Predicted
NORMAL
PNEUMONIA
0.60
0.55
0.50
0.45
0.40



Confusion Matrix - ROI-Based CNN
21
9
8
22
NORMAL
PNEUMONIA
Actual
Predicted
22
20
18
16
14
12
10
8



Normalized Confusion Matrix - ROI-Based CNN
0.70
0.30
0.27
0.73
NORMAL
PNEUMONIA
Actual
Predicted
0.7
0.6
0.5
0.4
0.3