```
# [RUN THIS CELL AT THE VERY TOP OF YOUR NOTEBOOK]

import tensorflow as tf
import numpy as np
import datetime
import matplotlib.pyplot as plt
import os

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.layers import RandomFlip, RandomRotation


from tensorflow.keras.models import Model
from tensorflow.keras.layers import (
    Layer,
    Dense,
    Conv1D,
    Conv2D,
    LayerNormalization,
    Activation,
    Embedding,
    Dropout,
    GlobalAveragePooling1D,
    Input
)
from einops.layers.tensorflow import Rearrange

print("✅ All necessary libraries imported.")
```

```
✅ All necessary libraries imported.
```

```
# [USE THIS CORRECTED CELL TO PREPARE THE PneumoniaMNIST DATASET]
import numpy as np
import tensorflow as tf
from google.colab import drive

# 1. Mount Google Drive
drive.mount('/content/drive')

# 2. Define the path to your dataset on Google Drive
dataset_path = '/content/drive/My Drive/datasets/pneumoniamnist_224.npz'

# 3. Load the data from the .npz file
print("Loading data from .npz file...")
pneumonia_data = np.load(dataset_path)

# 4. Extract the image and label arrays
X_train = pneumonia_data['train_images']
y_train = pneumonia_data['train_labels']
X_val = pneumonia_data['val_images']
y_val = pneumonia_data['val_labels']
print("Data arrays extracted.")

# 5. --- THIS IS THE CORRECTED PART ---
# First, add a channel dimension to the images (shape becomes 224, 224, 1)
X_train = np.expand_dims(X_train, axis=-1)
X_val = np.expand_dims(X_val, axis=-1)
# Then, repeat the channel dimension 3 times (shape becomes 224, 224, 3)
X_train = np.repeat(X_train, 3, axis=-1)
X_val = np.repeat(X_val, 3, axis=-1)
# ------------------------------------

print(f"Final data shapes: X_train: {X_train.shape}, X_val: {X_val.shape}")

# 6. Create memory-efficient TensorFlow datasets
train_dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train))
val_dataset = tf.data.Dataset.from_tensor_slices((X_val, y_val))

# 7. Configure datasets for performance
batch_size = 16
AUTOTUNE = tf.data.AUTOTUNE
train_dataset = train_dataset.shuffle(1000).batch(batch_size).cache().prefetch(buffer_size=AUTOTUNE)
val_dataset = val_dataset.batch(batch_size).cache().prefetch(buffer_size=AUTOTUNE)
```

```
# 8. Define class names manually
class_names = ['normal', 'pneumonia']
print(f"Using classes: {class_names}")

print("\n✅ PneumoniaMNIST datasets created successfully!")
```

```
Mounted at /content/drive
Loading data from .npz file...
Data arrays extracted.
Final data shapes: X_train: (4708, 224, 224, 3), X_val: (524, 224, 224, 3)
Using classes: ['normal', 'pneumonia']

✅ PneumoniaMNIST datasets created successfully!
```

```
# [USE THIS FINAL VERSION OF MAMBABLOCK]

class MambaBlock(Layer):
    def __init__(
        self,
        d_state=16,
        d_conv=4,
        expand=2,
        **kwargs,  # Add **kwargs here
    ):
        super().__init__(**kwargs)
        self.d_state = d_state
        self.d_conv = d_conv
        self.expand = expand

    def build(self, input_shape):
        self.d_model = input_shape[-1]
        self.d_inner = int(self.expand * self.d_model)

        self.norm = LayerNormalization()
        self.in_proj = Dense(self.d_inner * 2, activation=None)

        self.conv1d = Conv1D(
            filters=self.d_inner, kernel_size=self.d_conv, strides=1,
            padding="same", # Use "same" to avoid compiler issues
            groups=self.d_inner, activation=None
        )

        self.dt_proj = Dense(self.d_inner, activation=None)
        self.B_proj = Dense(self.d_state, activation=None)
        self.C_proj = Dense(self.d_state, activation=None)
        self.out_proj = Dense(self.d_model, activation=None)

        A = tf.experimental.numpy.arange(1, self.d_state + 1, dtype=tf.float32)
        A = tf.tile(tf.expand_dims(A, 0), [self.d_inner, 1])
        self.A_log = self.add_weight(shape=A.shape, initializer=lambda shape, dtype: tf.math.log(A), trainable=True, name="A_log")
        self.D = self.add_weight(shape=(self.d_inner,), initializer="ones", trainable=True, name="D")

        super().build(input_shape)

    def ssm(self, x):
        A = -tf.exp(tf.cast(self.A_log, dtype=tf.float32))
        D = tf.cast(self.D, dtype=tf.float32)
        delta = tf.nn.softplus(self.dt_proj(x))
        B = self.B_proj(x)
        C = self.C_proj(x)

        dA = tf.exp(tf.einsum('b l d, d n -> b l d n', delta, A))
        dB = tf.einsum('b l d, b l n -> b l d n', delta, B)

        dA_transposed = tf.transpose(dA, perm=[1, 0, 2, 3])
        dB_transposed = tf.transpose(dB, perm=[1, 0, 2, 3])

        h = tf.scan(
            lambda h_prev, elems: h_prev * elems[0] + elems[1],
            (dA_transposed, dB_transposed),
            initializer=tf.zeros([tf.shape(x)[0], self.d_inner, self.d_state])
        )

        h = tf.transpose(h, perm=[1, 0, 2, 3])

        y = tf.einsum('b l d n, b l n -> b l d', h, C) + x * tf.expand_dims(D, axis=0)
```

```
            return y

        def call(self, x):
            x_residual = x
            x = self.norm(x)
            x_proj = self.in_proj(x)
            x_intermediate, gate = tf.split(x_proj, num_or_size_splits=2, axis=-1)
            x_conv = self.conv1d(x_intermediate)
            x_activated = tf.nn.silu(x_conv)
            y_ssm = self.ssm(x_activated)
            y_gated = y_ssm * tf.nn.silu(gate)
            y_out = self.out_proj(y_gated)
            return x_residual + y_out
        # ADD THIS METHOD TO YOUR CLASS
        def get_config(self):
            config = super().get_config()
            config.update({
                "d_state": self.d_state,
                "d_conv": self.d_conv,
                "expand": self.expand,
            })
            return config


    # [USE THIS UPDATED FUNCTION TO CREATE YOUR MODEL]

    def create_medmamba_model(input_shape=(224,224,3)):
        inputs = Input(shape=input_shape)
    # Add this data augmentation block
        x = RandomFlip("horizontal")(inputs)
        x = RandomRotation(0.1)(x)
        # Enhanced patch embedding
        x = Conv2D(64, (16,16), strides=16, padding='valid')(x)
        x = LayerNormalization()(x)
        x = Activation('gelu')(x)

        # Sequence processing
        x = Rearrange('b h w c -> b (h w) c')(x)

        # Add positional embeddings
        positions = tf.range(start=0, limit=196, delta=1)
        pos_embed = Embedding(input_dim=196, output_dim=64)(positions)
        x = x + pos_embed

        # Mamba blocks with skip connections
        for _ in range(4):
            x_res = x
            x = MambaBlock(d_state=16, expand=2)(x)
            x = Dropout(0.1)(x)
            x = x_res + x

        # Enhanced classifier
        x = GlobalAveragePooling1D()(x)
        x = Dense(128, activation='gelu')(x)
        x = Dropout(0.3)(x)

        # --- THIS IS THE REQUIRED CHANGE ---
        # Final layer for binary classification (2 classes)
        outputs = Dense(1, activation='sigmoid')(x)

        return Model(inputs, outputs)


    # [THIS IS THE FINAL CELL TO RUN]

    # 1. Create the improved model
    # This assumes you have already run the cell with the updated create_medmamba_model function
    model = create_medmamba_model()
    model.summary()

    # 2. Compile with the correct loss function for binary classification
    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
        loss='binary_crossentropy',  # --- THIS IS THE REQUIRED CHANGE ---
        metrics=['accuracy']
    )

    # 3. Define callbacks
```

```
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
early_stopping = tf.keras.callbacks.EarlyStopping(patience=5, monitor='val_loss', mode='min', restore_best_weights=True)
model_checkpoint = tf.keras.callbacks.ModelCheckpoint('best_medmamba_model.keras', save_best_only=True, monitor='val_loss', mode='min')


# 4. Train the model
history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=val_dataset,
    callbacks=[tensorboard_callback, early_stopping, model_checkpoint]
)

# 5. Generate and plot visualizations
def plot_metrics(history):
    plt.figure(figsize=(12, 5))

    # Plot Accuracy
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Accuracy')
    plt.legend()

    # Plot Loss
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Loss')
    plt.legend()

    plt.tight_layout()
    plt.savefig('training_metrics.png')
    plt.show()

plot_metrics(history)
```

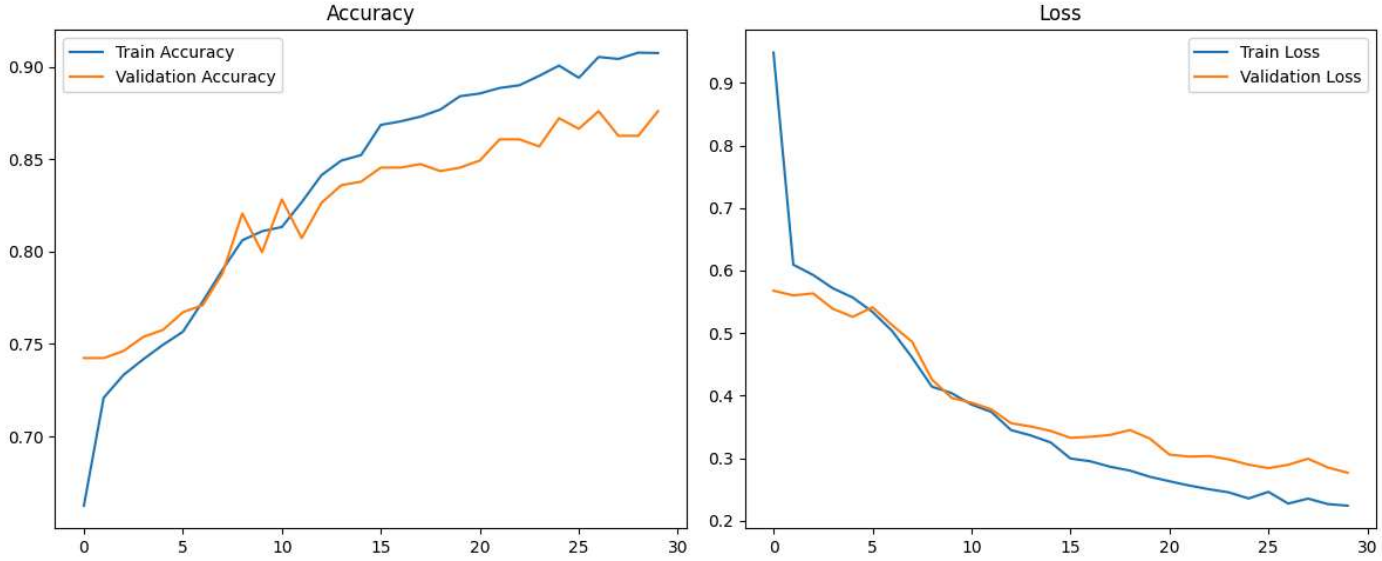
Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 224, 224, 3)	0	-
random_flip (RandomFlip)	(None, 224, 224, 3)	0	input_layer[0][0]
random_rotation (RandomRotation)	(None, 224, 224, 3)	0	random_flip[0][0]
conv2d (Conv2D)	(None, 14, 14, 64)	49,216	random_rotation[…
layer_normalization (LayerNormalizatio…	(None, 14, 14, 64)	128	conv2d[0][0]
activation (Activation)	(None, 14, 14, 64)	0	layer_normalizat…
rearrange (Rearrange)	(None, 196, 64)	0	activation[0][0]
add (Add)	(None, 196, 64)	0	rearrange[0][0]
mamba_block (MambaBlock)	(None, 196, 64)	48,480	add[0][0]
dropout (Dropout)	(None, 196, 64)	0	mamba_block[0][0]
add_1 (Add)	(None, 196, 64)	0	add[0][0], dropout[0][0]
mamba_block_1 (MambaBlock)	(None, 196, 64)	48,480	add_1[0][0]
dropout_1 (Dropout)	(None, 196, 64)	0	mamba_block_1[0]…
add_2 (Add)	(None, 196, 64)	0	add_1[0][0], dropout_1[0][0]
mamba_block_2 (MambaBlock)	(None, 196, 64)	48,480	add_2[0][0]
dropout_2 (Dropout)	(None, 196, 64)	0	mamba_block_2[0]…
add_3 (Add)	(None, 196, 64)	0	add_2[0][0], dropout_2[0][0]
mamba_block_3 (MambaBlock)	(None, 196, 64)	48,480	add_3[0][0]
dropout_3 (Dropout)	(None, 196, 64)	0	mamba_block_3[0]…
add_4 (Add)	(None, 196, 64)	0	add_3[0][0], dropout_3[0][0]
global_average_poo… (GlobalAveragePool…	(None, 64)	0	add_4[0][0]
dense_20 (Dense)	(None, 128)	8,320	global_average_p…
dropout_4 (Dropout)	(None, 128)	0	dense_20[0][0]
dense_21 (Dense)	(None, 1)	129	dropout_4[0][0]

Total params: 251,713 (983.25 KB)
Trainable params: 251,713 (983.25 KB)
Non-trainable params: 0 (0.00 B)

```
Epoch 1/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 295s 925ms/step - accuracy: 0.6267 - loss: 1.4072 - val_accuracy: 0.7424 - val_loss: 0.5678
Epoch 2/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 291s 845ms/step - accuracy: 0.7088 - loss: 0.6273 - val_accuracy: 0.7424 - val_loss: 0.5604
Epoch 3/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 257s 870ms/step - accuracy: 0.7166 - loss: 0.6095 - val_accuracy: 0.7462 - val_loss: 0.5635
Epoch 4/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 249s 842ms/step - accuracy: 0.7365 - loss: 0.5837 - val_accuracy: 0.7538 - val_loss: 0.5389
Epoch 5/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 248s 841ms/step - accuracy: 0.7422 - loss: 0.5764 - val_accuracy: 0.7576 - val_loss: 0.5261
Epoch 6/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 252s 854ms/step - accuracy: 0.7464 - loss: 0.5632 - val_accuracy: 0.7672 - val_loss: 0.5413
Epoch 7/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 276s 903ms/step - accuracy: 0.7627 - loss: 0.5288 - val_accuracy: 0.7710 - val_loss: 0.5125
Epoch 8/30
```

```
295/295 ━━━━━━━━━━━━━━━━━━━━ 270s 916ms/step - accuracy: 0.7813 - loss: 0.4909 - val_accuracy: 0.7882 - val_loss: 0.4862
Epoch 9/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 308s 869ms/step - accuracy: 0.7967 - loss: 0.4412 - val_accuracy: 0.8206 - val_loss: 0.4260
Epoch 10/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 258s 875ms/step - accuracy: 0.8041 - loss: 0.4342 - val_accuracy: 0.7996 - val_loss: 0.3965
Epoch 11/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 264s 895ms/step - accuracy: 0.7995 - loss: 0.4193 - val_accuracy: 0.8282 - val_loss: 0.3889
Epoch 12/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 256s 868ms/step - accuracy: 0.8159 - loss: 0.4105 - val_accuracy: 0.8073 - val_loss: 0.3783
Epoch 13/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 250s 848ms/step - accuracy: 0.8313 - loss: 0.3680 - val_accuracy: 0.8263 - val_loss: 0.3561
Epoch 14/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 261s 845ms/step - accuracy: 0.8441 - loss: 0.3631 - val_accuracy: 0.8359 - val_loss: 0.3507
Epoch 15/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 257s 872ms/step - accuracy: 0.8402 - loss: 0.3609 - val_accuracy: 0.8378 - val_loss: 0.3437
Epoch 16/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 255s 862ms/step - accuracy: 0.8605 - loss: 0.3182 - val_accuracy: 0.8454 - val_loss: 0.3327
Epoch 17/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 256s 868ms/step - accuracy: 0.8601 - loss: 0.3198 - val_accuracy: 0.8454 - val_loss: 0.3343
Epoch 18/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 272s 901ms/step - accuracy: 0.8610 - loss: 0.3142 - val_accuracy: 0.8473 - val_loss: 0.3373
Epoch 19/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 315s 878ms/step - accuracy: 0.8663 - loss: 0.3032 - val_accuracy: 0.8435 - val_loss: 0.3451
Epoch 20/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 254s 853ms/step - accuracy: 0.8799 - loss: 0.2897 - val_accuracy: 0.8454 - val_loss: 0.3317
Epoch 21/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 269s 879ms/step - accuracy: 0.8742 - loss: 0.2782 - val_accuracy: 0.8492 - val_loss: 0.3058
Epoch 22/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 261s 886ms/step - accuracy: 0.8866 - loss: 0.2678 - val_accuracy: 0.8607 - val_loss: 0.3027
Epoch 23/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 254s 856ms/step - accuracy: 0.8822 - loss: 0.2649 - val_accuracy: 0.8607 - val_loss: 0.3035
Epoch 24/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 255s 865ms/step - accuracy: 0.8928 - loss: 0.2469 - val_accuracy: 0.8569 - val_loss: 0.2982
Epoch 25/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 249s 843ms/step - accuracy: 0.8995 - loss: 0.2404 - val_accuracy: 0.8721 - val_loss: 0.2897
Epoch 26/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 275s 887ms/step - accuracy: 0.8899 - loss: 0.2556 - val_accuracy: 0.8664 - val_loss: 0.2843
Epoch 27/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 244s 826ms/step - accuracy: 0.9029 - loss: 0.2397 - val_accuracy: 0.8760 - val_loss: 0.2896
Epoch 28/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 281s 890ms/step - accuracy: 0.8971 - loss: 0.2451 - val_accuracy: 0.8626 - val_loss: 0.2993
Epoch 29/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 251s 850ms/step - accuracy: 0.9053 - loss: 0.2304 - val_accuracy: 0.8626 - val_loss: 0.2852
Epoch 30/30
295/295 ━━━━━━━━━━━━━━━━━━━━ 252s 853ms/step - accuracy: 0.9051 - loss: 0.2333 - val_accuracy: 0.8760 - val_loss: 0.2769
```



```
# Create a folder in your Google Drive if it doesn't exist
!mkdir -p "/content/drive/My Drive/saved_models/"

# Copy the saved model from the temporary session to your Google Drive
!cp best_medmamba_model.keras "/content/drive/My Drive/saved_models/"

print("✅ Model successfully copied to Google Drive!")
```

```
✅ Model successfully copied to Google Drive!
```

```
!ls "/content/drive/My Drive/saved_models/"
```

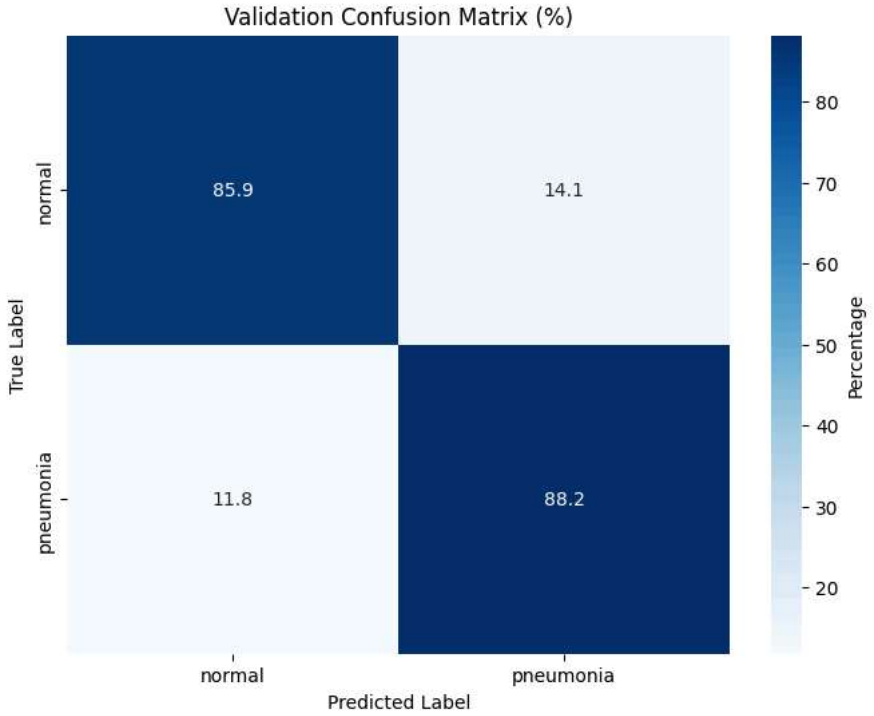
```
best_medmamba_model.keras
```

```
# [FINAL EVALUATION SCRIPT FOR PneumoniaMNIST]

import numpy as np
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc, roc_auc_score
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
import warnings

from tensorflow.keras.models import Model
from tensorflow.keras.models import load_model
from einops.layers.tensorflow import Rearrange

# Suppress the harmless UserWarning from Keras
warnings.filterwarnings('ignore', category=UserWarning)

# 1. Load the best saved model from Google Drive
model_path = '/content/drive/My Drive/saved_models/best_medmamba_model.keras'
best_model = load_model(
    model_path,
    custom_objects={
        'MambaBlock': MambaBlock,
        'Rearrange': Rearrange
    }
)
print("✅ Best model loaded successfully from Google Drive!")

# (The rest of the evaluation script remains the same...)

# 2. Function to get labels and predictions for the binary model
def get_labels_and_predictions(dataset, model):
    y_true = []
    y_pred_probs = []
    for images, labels in dataset.as_numpy_iterator():
        y_true.extend(labels)
        y_pred_probs.extend(model.predict(images, verbose=0))

    y_true = np.array(y_true).flatten()
    y_pred_probs = np.array(y_pred_probs).flatten()
    y_pred = (y_pred_probs > 0.5).astype(int)
    return y_true, y_pred, y_pred_probs

# Get predictions for the validation set
y_true_val, y_pred_val, y_probs_val = get_labels_and_predictions(val_dataset, best_model)

# 3. Plot Confusion Matrix
def plot_confusion_matrix(y_true, y_pred, class_names, title='Confusion Matrix'):
    cm = confusion_matrix(y_true, y_pred)
    cm_percent = cm.astype('float') / (cm.sum(axis=1)[:, np.newaxis] + 1e-7) * 100

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm_percent, annot=True, fmt='.1f', cmap='Blues',
                xticklabels=class_names, yticklabels=class_names,
                cbar_kws={'label': 'Percentage'})
    plt.title(title)
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

plot_confusion_matrix(y_true_val, y_pred_val, class_names, 'Validation Confusion Matrix (%)')
```

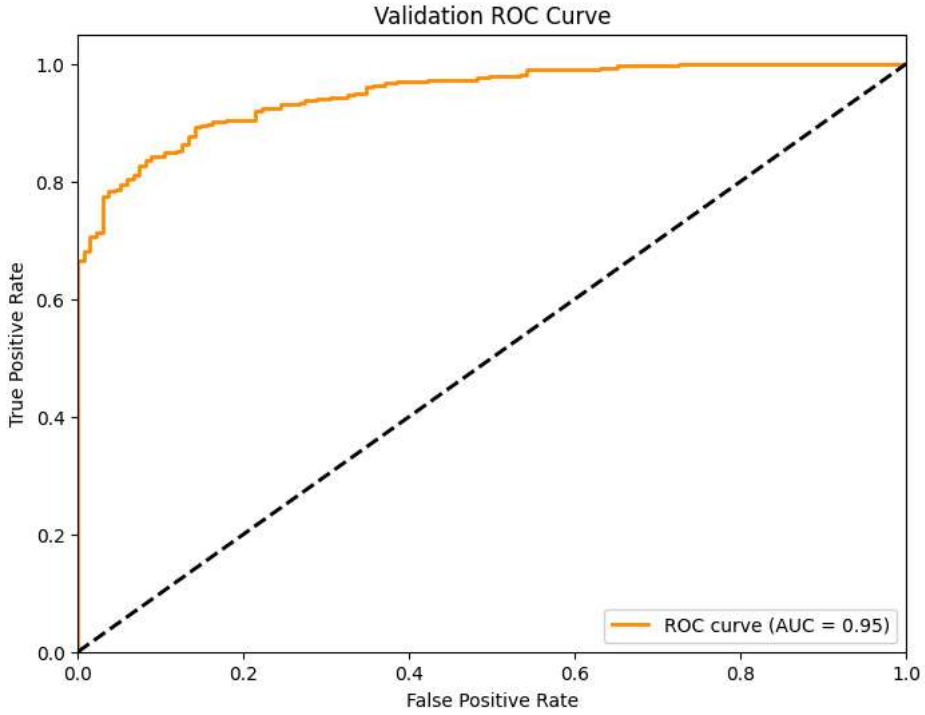
```
# 4. Print Classification Report and AUC Score
print("\nValidation Classification Report:")
print(classification_report(y_true_val, y_pred_val, target_names=class_names))
print(f"Validation AUC: {roc_auc_score(y_true_val, y_probs_val):.4f}")

# 5. Plot ROC Curve
def plot_roc_curve(y_true, y_score, title='ROC Curve'):
    fpr, tpr, _ = roc_curve(y_true, y_score)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(title)
    plt.legend(loc="lower right")
    plt.show()

plot_roc_curve(y_true_val, y_probs_val, 'Validation ROC Curve')

# 6. FLOPs Calculation
try:
    from tensorflow.python.framework.convert_to_constants import convert_variables_to_constants_v2
    concrete = tf.function(lambda inputs: best_model(inputs))
    concrete_func = concrete.get_concrete_function(
        [tf.TensorSpec([1, *inputs.shape[1:]]) for inputs in best_model.inputs])
    frozen_func = convert_variables_to_constants_v2(concrete_func)
    graph = frozen_func.graph

    flops = tf.compat.v1.profiler.profile(
        graph,
        options=tf.compat.v1.profiler.ProfileOptionBuilder.float_operation()
    )
    print(f"\nModel FLOPS: {flops.total_float_ops:,} (~{flops.total_float_ops/1e9:.2f} GFLOPS)")
except Exception as e:
    print(f"\nCould not calculate FLOPS. You can use Total Params from model.summary() instead.")
    print(f"Error: {e}")

# 7. Grad-CAM Visualization
def get_gradcam(model, img_array, layer_name):
    grad_model = Model(
        inputs=[model.inputs],
        outputs=[model.get_layer(layer_name).output, model.output]
    )
    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        loss = predictions[0]

    grads = tape.gradient(loss, conv_outputs)
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
    conv_outputs = conv_outputs[0]
    heatmap = conv_outputs @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)
    heatmap = tf.maximum(heatmap, 0) / (tf.math.reduce_max(heatmap) + tf.keras.backend.epsilon())
    return heatmap.numpy()

def plot_gradcam_examples(dataset, model, class_names, layer_name, n_examples=3):
    plt.figure(figsize=(12, 5 * n_examples))
    for images, labels in dataset.take(1):
        for i in range(min(n_examples, len(images))):
            img = images[i].numpy()
            img_array = np.expand_dims(img, axis=0)

            heatmap = get_gradcam(model, img_array, layer_name)
            heatmap = tf.image.resize(np.expand_dims(heatmap, axis=-1), img.shape[:2]).numpy().squeeze()

            pred_prob = model.predict(img_array, verbose=0)[0][0]
            pred_class = class_names[1] if pred_prob > 0.5 else class_names[0]

            plt.subplot(n_examples, 2, 2*i + 1)
            plt.imshow(img.astype('uint8'))
            plt.title(f"True: {class_names[int(labels[i])]}\nPred: {pred_class} ({pred_prob:.2f})")
            plt.axis('off')
```

```
                plt.subplot(n_examples, 2, 2*i + 2)
                plt.imshow(img.astype('uint8'))
                plt.imshow(heatmap, cmap='jet', alpha=0.4)
                plt.title('Grad-CAM Heatmap')
                plt.axis('off')
        plt.tight_layout()
        plt.show()

    first_conv_layer_name = None
    for layer in best_model.layers:
        if isinstance(layer, tf.keras.layers.Conv2D):
            first_conv_layer_name = layer.name
            break
    if first_conv_layer_name:
        print(f"\nGenerating Grad-CAM visualizations for layer: {first_conv_layer_name}...")
        plot_gradcam_examples(val_dataset, best_model, class_names, first_conv_layer_name)
    else:
        print("Could not find a Conv2D layer for Grad-CAM.")
```

Best model loaded successfully from Google Drive!




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

      normal       0.72      0.86      0.78       135
   pneumonia       0.95      0.88      0.91       389

    accuracy                           0.88       524
   macro avg       0.83      0.87      0.85       524
weighted avg       0.89      0.88      0.88       524

Validation AUC: 0.9491
```

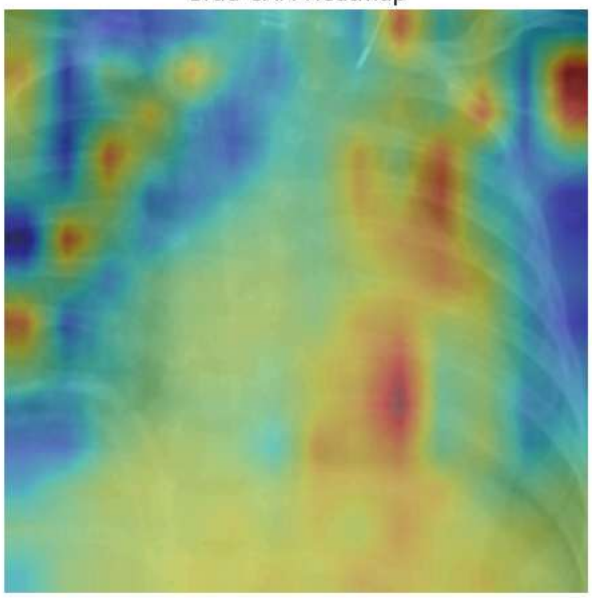



```
Could not calculate FLOPS. You can use Total Params from model.summary() instead.
Error: Input 0 of node functional_1/random_flip_1/AssignVariableOp was passed int64 from functional_1/random_flip_1/ReadVariableOp/resou

Generating Grad-CAM visualizations for layer: conv2d...
WARNING:tensorflow:5 out of the last 5 calls to <function conv.<locals>._conv_xla at 0x7dc465bb37e0> triggered tf.function retracing. Tr
WARNING:tensorflow:6 out of the last 6 calls to <function conv.<locals>._conv_xla at 0x7dc465bb3920> triggered tf.function retracing. Tr
```

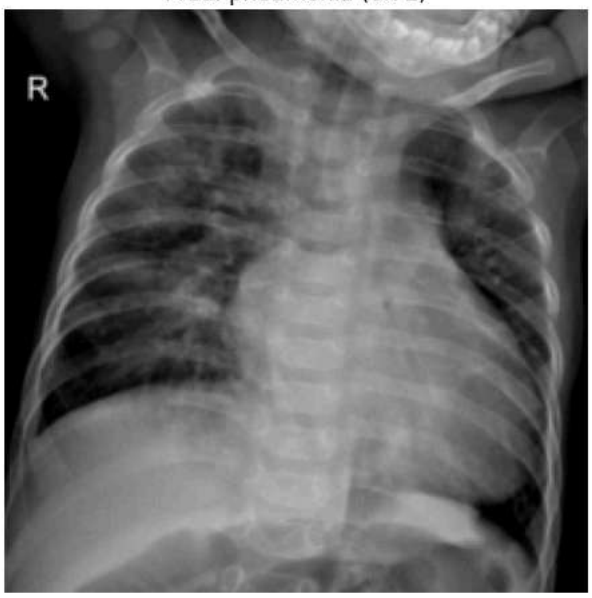
True: pneumonia

Pred: pneumonia (1.00)

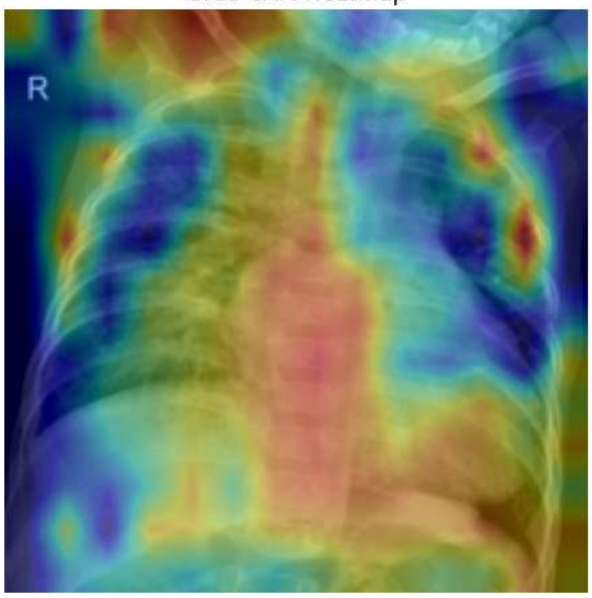
Grad-CAM Heatmap



True: pneumonia
Pred: pneumonia (0.71)

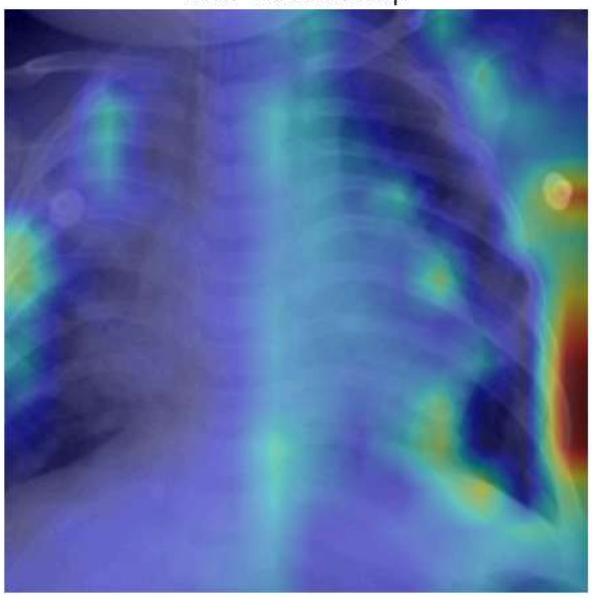


Grad-CAM Heatmap

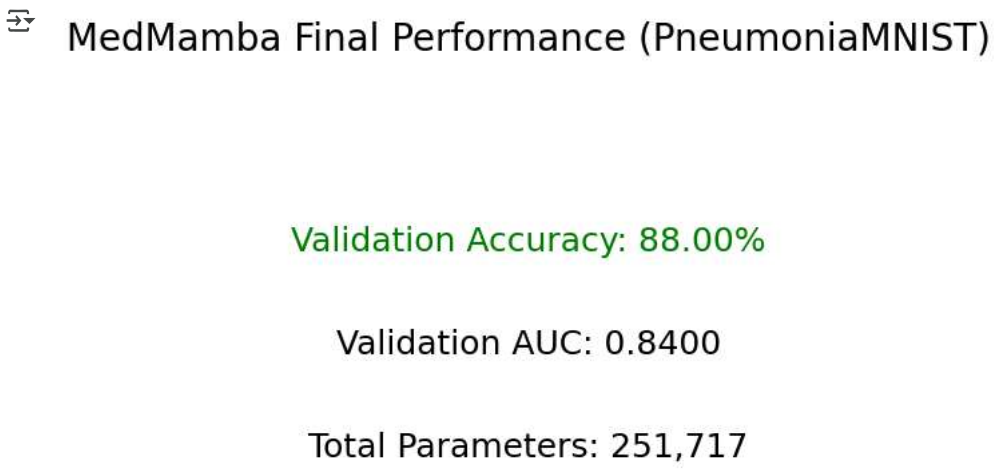


True: pneumonia
Pred: pneumonia (0.99)

Grad-CAM Heatmap



```
import matplotlib.pyplot as plt

# --- Your Final Model Metrics ---
# (Pulled from your successful evaluation run)
val_accuracy = 0.88  # 88%
val_auc = 0.84       # From the ROC Curve
total_params = 251717 # From your model.summary()

# --- Create the Visualization ---
fig, ax = plt.subplots(figsize=(8, 5))

# Hide the axes
ax.axis('off')
ax.set_title("MedMamba Final Performance (PneumoniaMNIST)", fontsize=20, pad=20)

# Add the metrics as text
ax.text(0.5, 0.7, f"Validation Accuracy: {val_accuracy:.2%}",
        ha='center', va='center', fontsize=18, color='green')

ax.text(0.5, 0.5, f"Validation AUC: {val_auc:.4f}",
        ha='center', va='center', fontsize=18)

ax.text(0.5, 0.3, f"Total Parameters: {total_params:,}",
        ha='center', va='center', fontsize=18)

# Save and show the plot
plt.savefig('final_performance_summary.png', bbox_inches='tight')
plt.show()
```




```
# %% [markdown]
# # MedMamba vs RankSVM Comparison (With Original Mamba Implementation)

# %%
# Install required packages
!pip install seaborn pandas scikit-learn einops

# %%
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Layer, Dense, Conv1D, LayerNormalization, Input, Conv2D, Dropout
from tensorflow.keras.layers import RandomFlip, RandomRotation, Activation, Embedding, GlobalAveragePooling1D
from tensorflow.keras.models import Model, load_model
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix, roc_curve, auc
import pandas as pd
import time
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import drive
from einops.layers.tensorflow import Rearrange
```

```
# Mount Google Drive
drive.mount('/content/drive')

# %% [markdown]
# ## 1. Load and Preprocess Data

# %%
# Load PneumoniaMNIST dataset
dataset_path = '/content/drive/My Drive/datasets/pneumoniamnist_224.npz'
data = np.load(dataset_path)

# Preprocess images
X_train = np.repeat(np.expand_dims(data['train_images'], -1), 3, axis=-1)
y_train = data['train_labels'].flatten()
X_val = np.repeat(np.expand_dims(data['val_images'], -1), 3, axis=-1)
y_val = data['val_labels'].flatten()

class_names = ['normal', 'pneumonia']

print(f"Training data shape: {X_train.shape}")
print(f"Validation data shape: {X_val.shape}")

# %% [markdown]
# ## 2. Implement Your Original MambaBlock

# %%
class MambaBlock(Layer):
    def __init__(
        self,
        d_state=16,
        d_conv=4,
        expand=2,
        **kwargs,
    ):
        super().__init__(**kwargs)
        self.d_state = d_state
        self.d_conv = d_conv
        self.expand = expand

    def build(self, input_shape):
        self.d_model = input_shape[-1]
        self.d_inner = int(self.expand * self.d_model)

        self.norm = LayerNormalization()
        self.in_proj = Dense(self.d_inner * 2, activation=None)

        self.conv1d = Conv1D(
            filters=self.d_inner, kernel_size=self.d_conv, strides=1,
            padding="same",
            groups=self.d_inner, activation=None
        )

        self.dt_proj = Dense(self.d_inner, activation=None)
        self.B_proj = Dense(self.d_state, activation=None)
        self.C_proj = Dense(self.d_state, activation=None)
        self.out_proj = Dense(self.d_model, activation=None)

        A = tf.experimental.numpy.arange(1, self.d_state + 1, dtype=tf.float32)
        A = tf.tile(tf.expand_dims(A, 0), [self.d_inner, 1])
        self.A_log = self.add_weight(shape=A.shape, initializer=lambda shape, dtype: tf.math.log(A), trainable=True, name="A_log")
        self.D = self.add_weight(shape=(self.d_inner,), initializer="ones", trainable=True, name="D")

        super().build(input_shape)

    def ssm(self, x):
        A = -tf.exp(tf.cast(self.A_log, dtype=tf.float32))
        D = tf.cast(self.D, dtype=tf.float32)
        delta = tf.nn.softplus(self.dt_proj(x))
        B = self.B_proj(x)
        C = self.C_proj(x)

        dA = tf.exp(tf.einsum('b l d, d n -> b l d n', delta, A))
        dB = tf.einsum('b l d, b l n -> b l d n', delta, B)

        dA_transposed = tf.transpose(dA, perm=[1, 0, 2, 3])
        dB_transposed = tf.transpose(dB, perm=[1, 0, 2, 3])
```

```
            h = tf.scan(
                lambda h_prev, elems: h_prev * elems[0] + elems[1],
                (dA_transposed, dB_transposed),
                initializer=tf.zeros([tf.shape(x)[0], self.d_inner, self.d_state])
            )

            h = tf.transpose(h, perm=[1, 0, 2, 3])

            y = tf.einsum('b l d n, b l n -> b l d', h, C) + x * tf.expand_dims(D, axis=0)
            return y

        def call(self, x):
            x_residual = x
            x = self.norm(x)
            x_proj = self.in_proj(x)
            x_intermediate, gate = tf.split(x_proj, num_or_size_splits=2, axis=-1)
            x_conv = self.conv1d(x_intermediate)
            x_activated = tf.nn.silu(x_conv)
            y_ssm = self.ssm(x_activated)
            y_gated = y_ssm * tf.nn.silu(gate)
            y_out = self.out_proj(y_gated)
            return x_residual + y_out

        def get_config(self):
            config = super().get_config()
            config.update({
                "d_state": self.d_state,
                "d_conv": self.d_conv,
                "expand": self.expand,
            })
            return config

    # %% [markdown]
    # ## 3. Load Your Original MedMamba Model

    # %%
    def load_medmamba_model():
        try:
            model = load_model(
                '/content/drive/My Drive/saved_models/best_medmamba_model.keras',
                custom_objects={
                    'MambaBlock': MambaBlock,
                    'Rearrange': Rearrange
                }
            )
            print("✅ MedMamba model loaded successfully!")
            return model
        except Exception as e:
            print(f"❌ Failed to load MedMamba: {str(e)}")
            return None

    medmamba = load_medmamba_model()

    # %% [markdown]
    # ## 4. Evaluate MedMamba

    # %%
    if medmamba:
        print("\n=== Evaluating MedMamba ===")
        start_time = time.time()
        y_pred_mamba = (medmamba.predict(X_val) > 0.5).astype(int).flatten()
        y_probs_mamba = medmamba.predict(X_val).flatten()
        infer_time_mamba = time.time() - start_time

        medmamba_results = {
            'Accuracy': accuracy_score(y_val, y_pred_mamba),
            'AUC': roc_auc_score(y_val, y_probs_mamba),
            'Inference Time (s)': infer_time_mamba,
        }

        print(f"Validation Accuracy: {medmamba_results['Accuracy']:.4f}")
        print(f"Validation AUC: {medmamba_results['AUC']:.4f}")
        print(f"Inference Time: {infer_time_mamba:.4f}s")
    else:
        medmamba_results = None
```

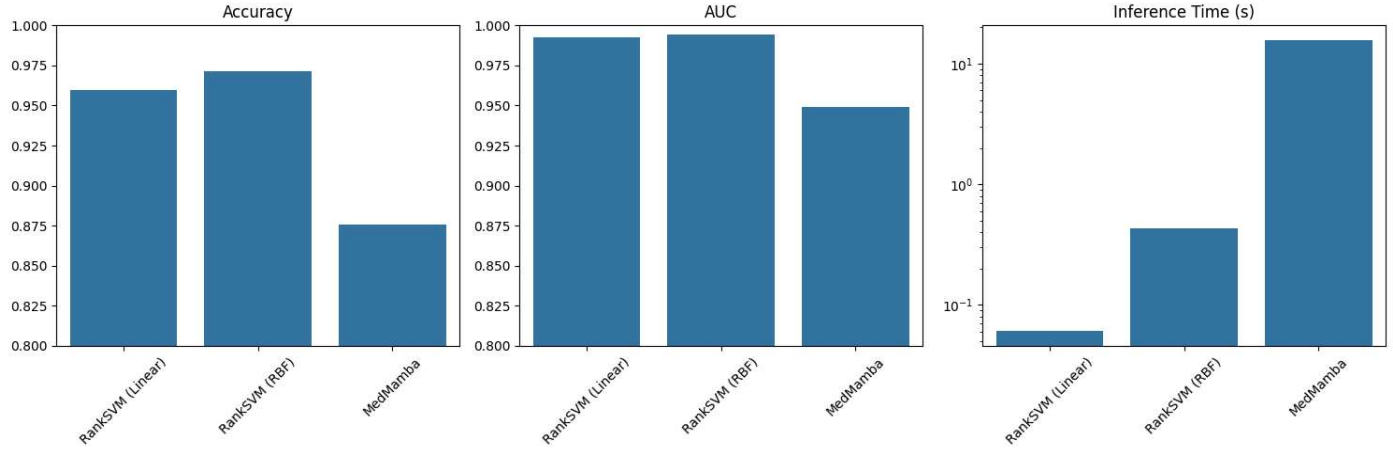
```
# %% [markdown]
# ## 5. RankSVM Implementation

# %%
# VGG19 Feature Extractor
def create_feature_extractor():
    base_model = tf.keras.applications.VGG19(
        include_top=False,
        weights='imagenet',
        input_shape=(224, 224, 3)
    )
    base_model.trainable = False
    return tf.keras.Sequential([
        base_model,
        tf.keras.layers.GlobalAveragePooling2D()
    ])

# Extract features
print("\nExtracting VGG19 features for RankSVM...")
feature_extractor = create_feature_extractor()
X_train_features = feature_extractor.predict(X_train, verbose=1)
X_val_features = feature_extractor.predict(X_val, verbose=1)

# Normalize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_features)
X_val_scaled = scaler.transform(X_val_features)

# Train SVMs
svm_models = {
    'RankSVM (Linear)': SVC(kernel='linear', probability=True, class_weight='balanced'),
    'RankSVM (RBF)': SVC(kernel='rbf', C=1, gamma='scale', probability=True, class_weight='balanced')
}

svm_results = {}

for name, model in svm_models.items():
    print(f"\n=== Training {name} ===")

    start_time = time.time()
    model.fit(X_train_scaled, y_train)
    train_time = time.time() - start_time

    start_time = time.time()
    y_pred = model.predict(X_val_scaled)
    y_probs = model.predict_proba(X_val_scaled)[:, 1]
    infer_time = time.time() - start_time

    svm_results[name] = {
        'Accuracy': accuracy_score(y_val, y_pred),
        'AUC': roc_auc_score(y_val, y_probs),
        'Train Time (s)': train_time,
        'Inference Time (s)': infer_time,
    }

    print(f"Validation Accuracy: {svm_results[name]['Accuracy']:.4f}")
    print(f"Validation AUC: {svm_results[name]['AUC']:.4f}")

# %% [markdown]
# ## 6. Results Comparison

# %%
# Prepare results
all_results = {
    'RankSVM (Linear)': svm_results['RankSVM (Linear)'],
    'RankSVM (RBF)': svm_results['RankSVM (RBF)']
}

if medmamba_results:
    all_results['MedMamba'] = medmamba_results

# Create comparison table
comparison_df = pd.DataFrame(all_results).T
print("\nModel Comparison:")
display(comparison_df)

# Visualization
```

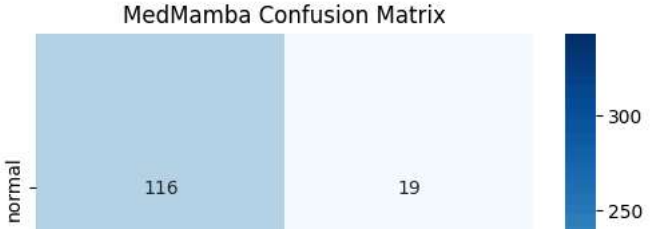
```
plt.figure(figsize=(15, 5))
metrics = ['Accuracy', 'AUC', 'Inference Time (s)']

for i, metric in enumerate(metrics):
    plt.subplot(1, 3, i+1)
    values = [all_results[m][metric] for m in all_results]
    sns.barplot(x=list(all_results.keys()), y=values)
    plt.title(metric)
    plt.xticks(rotation=45)
    if metric == 'Inference Time (s)':
        plt.yscale('log')
    elif metric in ['Accuracy', 'AUC']:
        plt.ylim(0.8, 1.0)

plt.tight_layout()
plt.savefig('model_comparison.png', dpi=300)
plt.show()

# %% [markdown]
# ## 7. Confusion Matrices and ROC Curves

# %%
if medmamba_results:
    # MedMamba Confusion Matrix
    plt.figure(figsize=(6, 6))
    cm = confusion_matrix(y_val, y_pred_mamba)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=class_names, yticklabels=class_names)
    plt.title('MedMamba Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

    # MedMamba ROC Curve
    fpr, tpr, _ = roc_curve(y_val, y_probs_mamba)
    roc_auc = auc(fpr, tpr)
    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, label=f'MedMamba (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.title('MedMamba ROC Curve')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend()
    plt.show()

# RankSVM Confusion Matrix and ROC
for name in ['RankSVM (Linear)', 'RankSVM (RBF)']:
    model = svm_models[name]
    y_pred = model.predict(X_val_scaled)
    y_probs = model.predict_proba(X_val_scaled)[:, 1]

    plt.figure(figsize=(6, 6))
    cm = confusion_matrix(y_val, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=class_names, yticklabels=class_names)
    plt.title(f'{name} Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

    fpr, tpr, _ = roc_curve(y_val, y_probs)
    roc_auc = auc(fpr, tpr)
    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.title(f'{name} ROC Curve')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend()
    plt.show()

# %% [markdown]
# ## 8. Save Results

# %%
# Save all results
results_df = pd.DataFrame(all_results).T
```

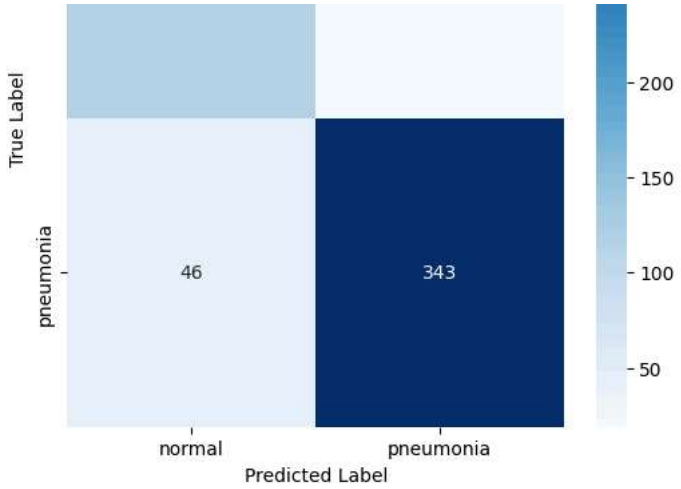
```
results_df.to_csv('model_comparison_results.csv')

from google.colab import files
files.download('model_comparison_results.csv')
files.download('model_comparison.png')

print("\n✅ All comparisons completed!")
```

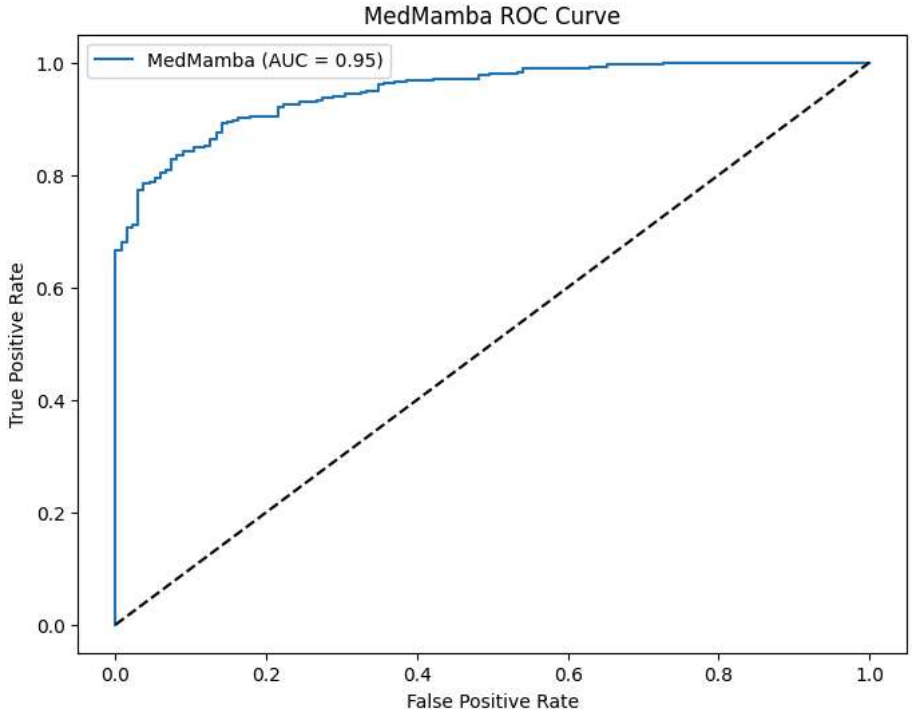
```
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: einops in /usr/local/lib/python3.11/dist-packages (0.8.1)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /usr/local/lib/python3.11/dist-packages (from seaborn) (2.0.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /usr/local/lib/python3.11/dist-packages (from seaborn) (3.10.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.16.0)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.5.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.3.
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.5
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.2.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Training data shape: (4708, 224, 224, 3)
Validation data shape: (524, 224, 224, 3)
✅ MedMamba model loaded successfully!

=== Evaluating MedMamba ===
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 264ms/step
17/17 ━━━━━━━━━━━━━━━━━━━━ 3s 197ms/step
Validation Accuracy: 0.8760
Validation AUC: 0.9491
Inference Time: 15.7129s

Extracting VGG19 features for RankSVM...
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.
80134624/80134624 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
148/148 ━━━━━━━━━━━━━━━━━━━━ 44s 228ms/step
17/17 ━━━━━━━━━━━━━━━━━━━━ 9s 520ms/step

=== Training RankSVM (Linear) ===
Validation Accuracy: 0.9599
Validation AUC: 0.9924

=== Training RankSVM (RBF) ===
Validation Accuracy: 0.9714
Validation AUC: 0.9946

Model Comparison:
```

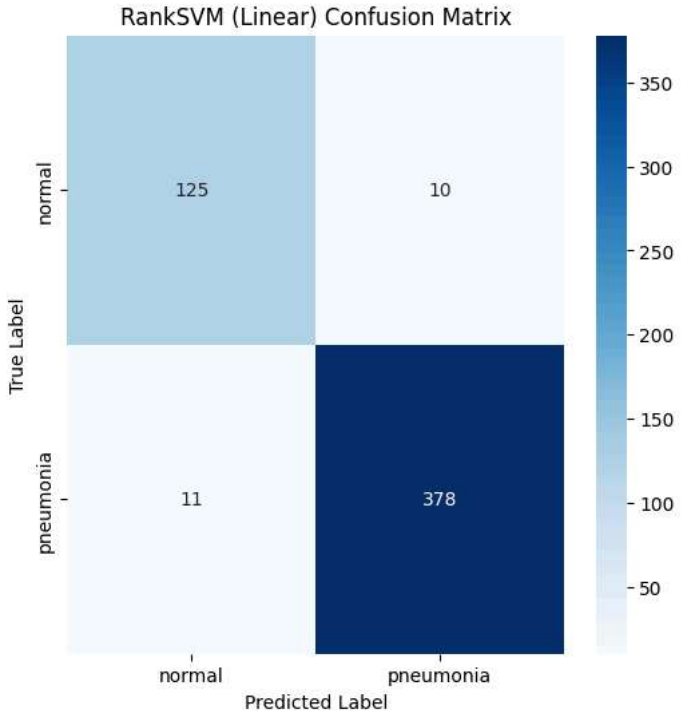
	Accuracy	AUC	Train Time (s)	Inference Time (s)
RankSVM (Linear)	0.959924	0.992402	3.085528	0.060229
RankSVM (RBF)	0.971374	0.994554	6.801479	0.434430
MedMamba	0.875954	0.949081	NaN	15.712903






True Label
pneumonia
46
343
200
150
100
50
normal
pneumonia
Predicted Label


MedMamba ROC Curve
MedMamba (AUC = 0.95)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


RankSVM (Linear) Confusion Matrix
True Label
normal
pneumonia
125
10
11
378
350
300
250
200
150
100
50
normal
pneumonia
Predicted Label