In [1]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import numpy as np
```

In [5]:
```
# Paths to the directories
train_dir = 'train'
val_dir = 'val'
test_dir ='test'
```

In [6]:
```
# 1. Data Preprocessing and Augmentation
train_datagen = ImageDataGenerator(rescale=1./255, zoom_range=0.2, horizontal_flip=
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
```

In [7]:
```
# Loading images from 'train' directory
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),  # Resize images to 150x150
    batch_size=32,
    class_mode='binary'  # Binary classification (pneumonia vs normal)
)
```

```
Found 5216 images belonging to 2 classes.
```

In [8]:
```
# Loading images from 'val' directory
val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary'
)
```

```
Found 16 images belonging to 2 classes.
```

In [9]:
```
# Loading images from 'test' directory
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary'
)
```

```
Found 624 images belonging to 2 classes.
```

In [10]:
```
# 2. Build the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(pool_size=(2, 2)),

    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
```

```
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),

    Flatten(),
    Dense(128, activation='relu'),
    Dense(1, activation='sigmoid')  # Binary classification (pneumonia vs normal)
])
```

```
C:\Users\bonde\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.
py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. W
hen using Sequential models, prefer using an `Input(shape)` object as the first laye
r in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [11]:
```
# 3. Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [12]:
```
# 4. Train the model with validation
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=val_generator  # Use the validation set for evaluation during t
)
```

```
Epoch 1/10
```

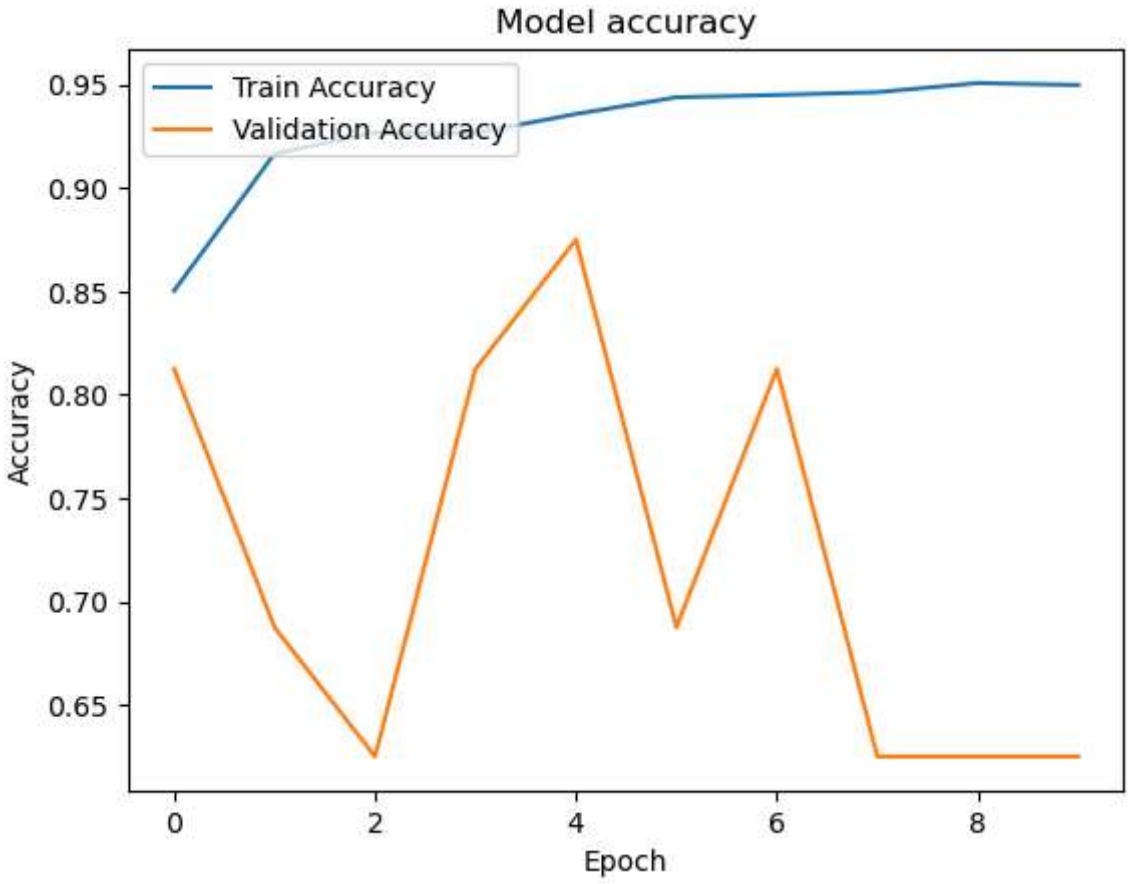
```
C:\Users\bonde\anaconda3\Lib\site-packages\keras\src\trainers\data_adapters\py_datas
et_adapter.py:122: UserWarning: Your `PyDataset` class should call `super().__init__
(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessi
ng`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignor
ed.
  self._warn_if_super_not_called()
```

```
163/163 ━━━━━━━━━━━━━━━━━━━━ 106s 612ms/step - accuracy: 0.7878 - loss: 0.4909 - val
_accuracy: 0.8125 - val_loss: 0.4107
Epoch 2/10
163/163 ━━━━━━━━━━━━━━━━━━━━ 97s 574ms/step - accuracy: 0.9130 - loss: 0.2123 - val_
accuracy: 0.6875 - val_loss: 0.7915
Epoch 3/10
163/163 ━━━━━━━━━━━━━━━━━━━━ 97s 574ms/step - accuracy: 0.9304 - loss: 0.1769 - val_
accuracy: 0.6250 - val_loss: 0.8974
Epoch 4/10
163/163 ━━━━━━━━━━━━━━━━━━━━ 97s 576ms/step - accuracy: 0.9242 - loss: 0.1762 - val_
accuracy: 0.8125 - val_loss: 0.4453
Epoch 5/10
163/163 ━━━━━━━━━━━━━━━━━━━━ 97s 573ms/step - accuracy: 0.9376 - loss: 0.1580 - val_
accuracy: 0.8750 - val_loss: 0.5180
Epoch 6/10
163/163 ━━━━━━━━━━━━━━━━━━━━ 127s 759ms/step - accuracy: 0.9427 - loss: 0.1511 - val
_accuracy: 0.6875 - val_loss: 0.5395
Epoch 7/10
163/163 ━━━━━━━━━━━━━━━━━━━━ 100s 593ms/step - accuracy: 0.9376 - loss: 0.1481 - val
_accuracy: 0.8125 - val_loss: 0.2721
Epoch 8/10
163/163 ━━━━━━━━━━━━━━━━━━━━ 103s 613ms/step - accuracy: 0.9444 - loss: 0.1431 - val
_accuracy: 0.6250 - val_loss: 1.5458
Epoch 9/10
163/163 ━━━━━━━━━━━━━━━━━━━━ 108s 646ms/step - accuracy: 0.9512 - loss: 0.1316 - val
_accuracy: 0.6250 - val_loss: 1.1715
Epoch 10/10
163/163 ━━━━━━━━━━━━━━━━━━━━ 127s 751ms/step - accuracy: 0.9491 - loss: 0.1308 - val
_accuracy: 0.6250 - val_loss: 0.9418
```

In [13]:

```python
# 5. Evaluate the model on the test set
test_loss, test_acc = model.evaluate(test_generator)
print(f"Test Accuracy: {test_acc:.3f}")

# 6. Plot training & validation accuracy
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.show()
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 8s 390ms/step - accuracy: 0.8440 - loss: 0.5190
Test Accuracy: 0.864
```



In [14]:
```
# Make predictions on the test data
predictions = model.predict(test_generator)

# Get the class labels for binary classification (NORMAL = 0, PNEUMONIA = 1)
class_labels = list(test_generator.class_indices.keys())
```

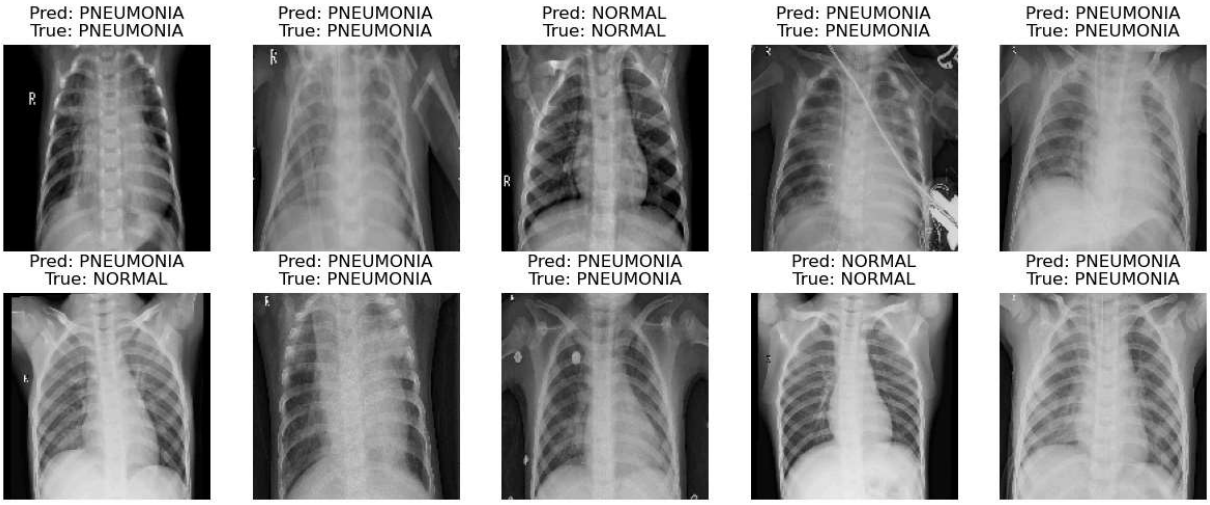
```
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 332ms/step
```

In [17]:
```
# Function to plot images with their predictions
def show_predictions(generator, num_images=10):
    plt.figure(figsize=(15, 15))

    for i in range(num_images):
        # Get a batch of test images and labels
        img_batch, label_batch = next(generator)  # Use next() to get images and la
        img = img_batch[0]  # Get the first image in the batch
        label = label_batch[0]  # Get the first label in the batch

        # Predict on the single image
        prediction = model.predict(np.expand_dims(img, axis=0))  # Make prediction
        predicted_class = (prediction > 0.5).astype("int32")[0][0]

        # Plot the image and show prediction
        plt.subplot(5, 5, i + 1)
        plt.imshow(img)
        plt.title(f'Pred: {class_labels[predicted_class]}\nTrue: {class_labels[int(
        plt.axis('off')
```

```
    plt.show()
```

In [18]:
```
# Show 10 predictions
show_predictions(test_generator, num_images=10)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 127ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
```



In [ ]:
```
```