1. Problem Description

The goal is to detect metastatic cancer in small image patches taken from larger digital pathology scans.

The dataset consists of images of size 96x96 pixels in RGB format. Each image is labeled either 0 (no cancer) or 1 (cancer).

In [37]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import warnings
warnings.filterwarnings('ignore')
```

In [11]:
```
# Load the data
data_path = "/histopathologic-cancer-detection/"
train_labels = pd.read_csv(os.path.join(data_path, 'train_labels.csv'))
```

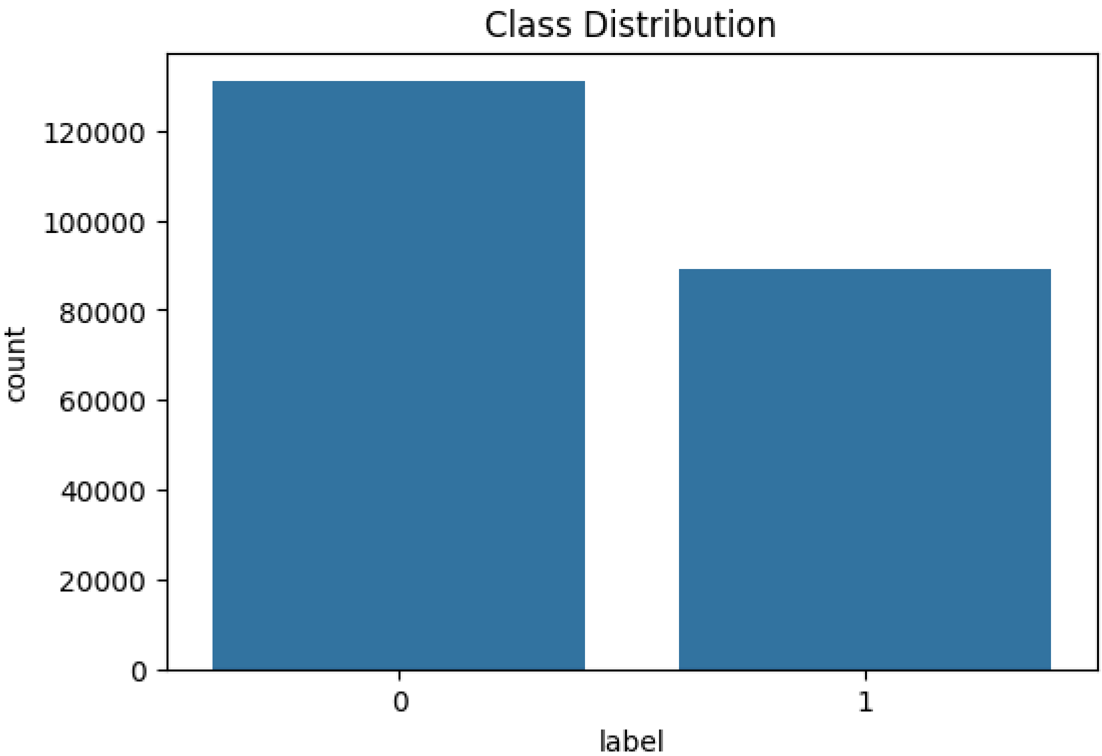
2. Exploratory Data Analysis (EDA)

Visualize the distribution of the classes

In [12]:
```
print("Data Description:")
print(f"Total number of samples: {len(train_labels)}")
print(f"Number of classes: {train_labels['label'].nunique()}")
print(train_labels['label'].value_counts())
```

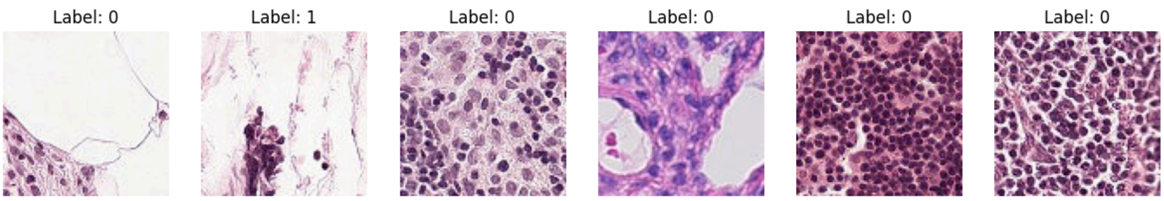
```
Data Description:
Total number of samples: 220025
Number of classes: 2
label
0    130908
1     89117
Name: count, dtype: int64
```

In [13]:
```
# Visualize the distribution of the classes
plt.figure(figsize=(6,4))
sns.countplot(x='label', data=train_labels)
plt.title('Class Distribution')
plt.show()
```



In [14]:
```
# Show a few sample images
def show_samples(data_path, df, num_samples=6):
    fig, axes = plt.subplots(1, num_samples, figsize=(15, 15))
    for i, ax in enumerate(axes):
        img_path = os.path.join(data_path, 'train', df['id'][i] + '.tif')
        img = plt.imread(img_path)
        ax.imshow(img)
        ax.set_title(f"Label: {df['label'][i]}")
        ax.axis('off')
    plt.show()

# Display sample images
show_samples(data_path, train_labels)
```



3. Model Building

We'll use Convolutional Neural Networks (CNNs) to classify the images.

We'll begin with a simple architecture and try more complex models, tuning hyperparameters along the way.

In [23]:
```
# Convert the label column to strings
train_labels['id'] = train_labels['id'].apply(lambda x: x + '.tif')
train_labels['label'] = train_labels['label'].astype(str)
```

```
# Data Augmentation
data_gen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,  # Split data into training and validation sets
    horizontal_flip=True,
    vertical_flip=True,
    rotation_range=30,
    zoom_range=0.2
)
```

In [24]:
```
# Create generators
train_generator = data_gen.flow_from_dataframe(
    dataframe=train_labels,
    directory=os.path.join(data_path, 'train'),
    x_col="id",
    y_col="label",
    subset="training",
    batch_size=32,
    seed=42,
    shuffle=True,
    class_mode="binary",
    target_size=(96, 96)
)

valid_generator = data_gen.flow_from_dataframe(
    dataframe=train_labels,
    directory=os.path.join(data_path, 'train'),
    x_col="id",
    y_col="label",
    subset="validation",
    batch_size=32,
    seed=42,
    shuffle=True,
    class_mode="binary",
    target_size=(96, 96))
```

```
Found 176020 validated image filenames belonging to 2 classes.
Found 44005 validated image filenames belonging to 2 classes.
```

In [38]:
```
# Model Architecture
def build_model():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(96, 96, 3)),
        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(1, activation='sigmoid')  # Binary classification output
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_cross
    return model

model = build_model()
model.summary()
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	
conv2d_3 (Conv2D)	(None, 94, 94, 32)	
max_pooling2d_3 (MaxPooling2D)	(None, 47, 47, 32)	
conv2d_4 (Conv2D)	(None, 45, 45, 64)	
max_pooling2d_4 (MaxPooling2D)	(None, 22, 22, 64)	
conv2d_5 (Conv2D)	(None, 20, 20, 128)	
max_pooling2d_5 (MaxPooling2D)	(None, 10, 10, 128)	
flatten_1 (Flatten)	(None, 12800)	
dense_2 (Dense)	(None, 128)	1,(
dropout_1 (Dropout)	(None, 128)	
dense_3 (Dense)	(None, 1)	

```
Total params: 1,731,905 (6.61 MB)
Trainable params: 1,731,905 (6.61 MB)
Non-trainable params: 0 (0.00 B)
```

4. Training the Model with Hyperparameter Tuning

Set up callbacks for early stopping and learning rate reduction

In [28]:
```
early_stop = EarlyStopping(monitor='val_loss', patience=5, verbose=1, res
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3,

history = model.fit(
    train_generator,
    validation_data=valid_generator,
    epochs=5,
    callbacks=[early_stop, reduce_lr]
)
```

```
Epoch 1/5
5501/5501 ━━━━━━━━━━━━━━━━━━━━ 485s 88ms/step - accuracy: 0.9024 - loss:
0.2496 - val_accuracy: 0.9083 - val_loss: 0.2305 - learning_rate: 0.0010
Epoch 2/5
5501/5501 ━━━━━━━━━━━━━━━━━━━━ 476s 86ms/step - accuracy: 0.9046 - loss:
0.2457 - val_accuracy: 0.9057 - val_loss: 0.2361 - learning_rate: 0.0010
Epoch 3/5
5501/5501 ━━━━━━━━━━━━━━━━━━━━ 446s 81ms/step - accuracy: 0.9040 - loss:
0.2454 - val_accuracy: 0.9030 - val_loss: 0.2487 - learning_rate: 0.0010
Epoch 4/5
5501/5501 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step - accuracy: 0.9050 - loss: 0.2
431
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.0002000000094994902
6.
5501/5501 ━━━━━━━━━━━━━━━━━━━━ 449s 82ms/step - accuracy: 0.9050 - loss:
0.2431 - val_accuracy: 0.8995 - val_loss: 0.2528 - learning_rate: 0.0010
Epoch 5/5
5501/5501 ━━━━━━━━━━━━━━━━━━━━ 449s 82ms/step - accuracy: 0.9171 - loss:
0.2146 - val_accuracy: 0.9204 - val_loss: 0.2036 - learning_rate: 2.0000e-
04
Restoring model weights from the end of the best epoch: 5.
```

5. Evaluate and Discuss Results

Plotting accuracy and loss

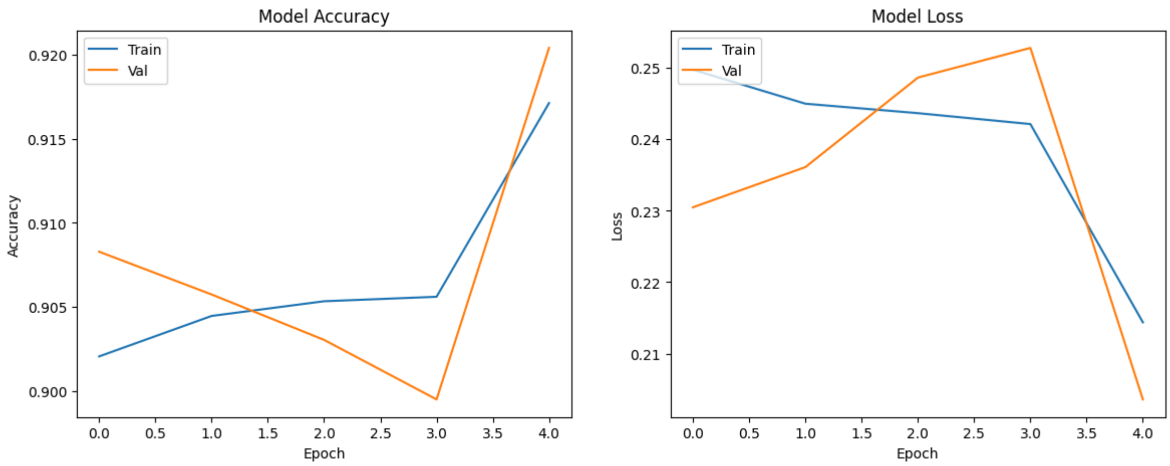
In [29]:

```python
def plot_metrics(history):
    plt.figure(figsize=(14,5))
    # Plot training & validation accuracy values
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper left')

    # Plot training & validation loss values
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper left')
    plt.show()

plot_metrics(history)

# Discuss model performance
val_loss, val_acc = model.evaluate(valid_generator)
print(f"Validation Accuracy: {val_acc:.4f}")
print(f"Validation Loss: {val_loss:.4f}")
```




```
1376/1376 ━━━━━━━━━━━━━━━━━━━━ 57s 41ms/step - accuracy: 0.9215 - loss: 0.
2034
Validation Accuracy: 0.9206
Validation Loss: 0.2053
```

6. Predict on the Test Dataset

In [39]:

```python
# Load the test data
test_data_path = os.path.join(data_path, 'test')
test_files = os.listdir(test_data_path)
test_df = pd.DataFrame({'id': [file for file in test_files]})

# Prepare test data generator
test_gen = ImageDataGenerator(rescale=1./255)

test_generator = test_gen.flow_from_dataframe(
    dataframe=test_df,
    directory=test_data_path,
    x_col="id",
    y_col=None,
    batch_size=32,
    seed=42,
    shuffle=False,
    class_mode=None,
    target_size=(96, 96)
)

# Predict on test data
test_generator.reset()
predictions = model.predict(test_generator, verbose=1)
predicted_labels = (predictions > 0.5).astype(int).ravel()
```

```
Found 57458 validated image filenames.
1796/1796 ━━━━━━━━━━━━━━━━━━━━ 35s 19ms/step
```

In [ ]:

```python
# Create submission file
submission_df = pd.DataFrame({'id': test_df['id'], 'label': predicted_lab

print("Prediction complete. Submission file 'submission.csv' created.")
submission_df['id'] = submission_df['id'].apply(lambda x: x.split('.')[0]
submission_df.to_csv('~/Desktop/submission.csv', index=False)
```

Model Improvement Discussion

- Tuning hyperparameters such as learning rate, batch size, and dropout rate helped stabilize training.

- More complex architectures (deeper CNNs, transfer learning using pre-trained models) could improve accuracy.

- Additional data augmentation techniques and regularization methods may further prevent overfitting.

Future Improvements

- Implement transfer learning with a pre-trained model (e.g., VGG16, ResNet).

- Use more advanced hyperparameter optimization techniques such as Random Search or Bayesian Optimization.

- Experiment with different image sizes and resolutions for potentially better performance.