Importing neccessary libraries

In [2]:
```
import os
import shutil
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings

import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D, Conv2D, MaxPool2D, BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, LearningRateScheduler
from tensorflow.keras.applications import ResNet50, InceptionV3

from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from tabulate import tabulate
warnings.filterwarnings('ignore')
```

Preparing train and test Image Generator

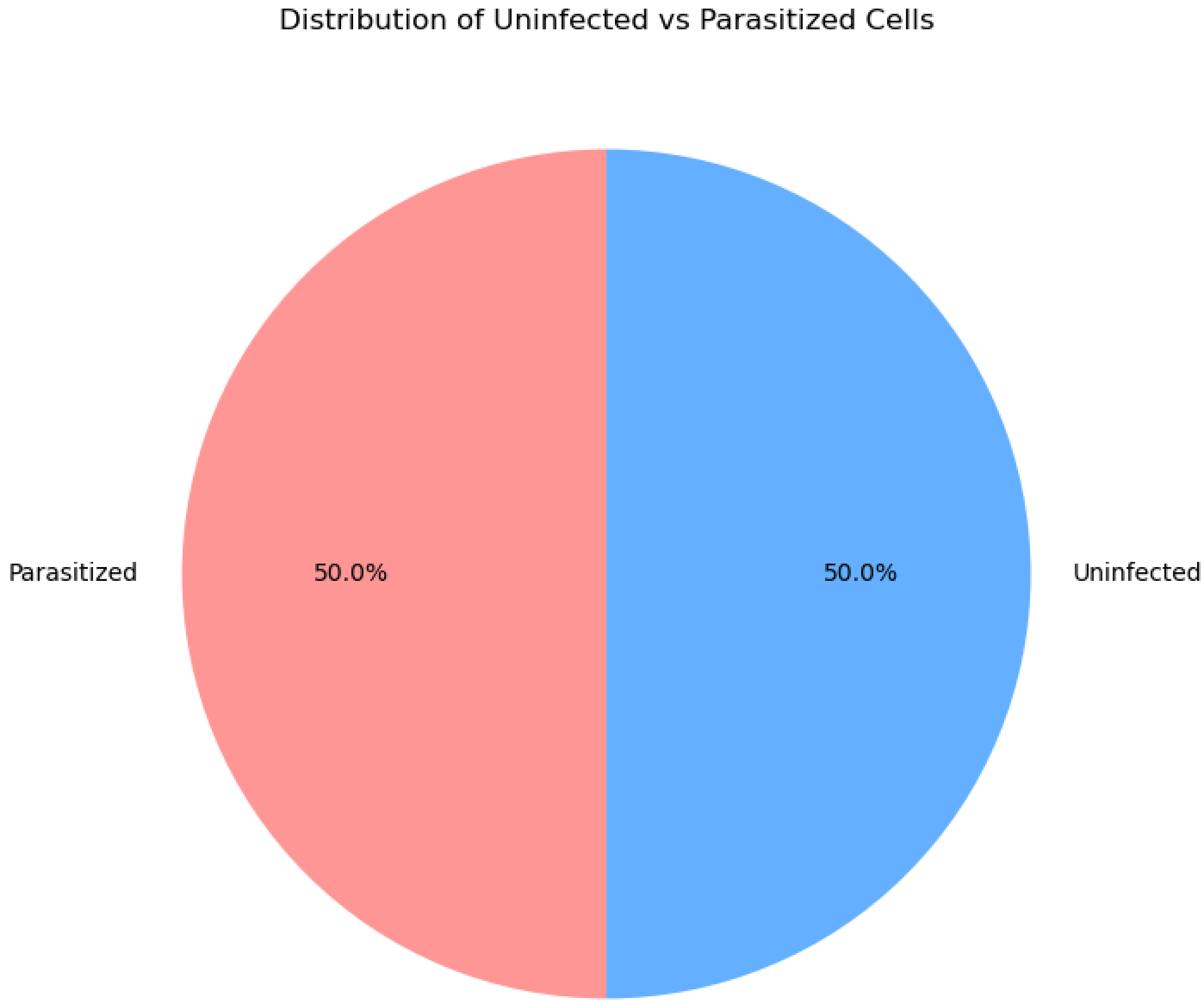
In [3]:
```
dataset_dir = './dataset/cell_images'
train_dir = './dataset/split/train'
val_dir = './dataset/split/validation'
test_dir = './dataset/split/test'

all_data = []
for class_label in ['Parasitized', 'Uninfected']:
    class_path = os.path.join(dataset_dir, class_label)
    for img in os.listdir(class_path):
        all_data.append((os.path.join(class_path, img), class_label))

data_df = pd.DataFrame(all_data, columns=['path', 'label'])
#data_sample = data_df.sample(n=2700, random_state=42)
data_sample = data_df

train_val_data, test_data = train_test_split(data_sample, test_size=0.2, stratify=data_sample['label'], random_state=42)
```

```
train_data, val_data = train_test_split(train_val_data, test_size=0.25, stratify=train_val_data['label'], random_state=42)

def copy_data(data_subset, target_dir):
    for _, row in data_subset.iterrows():
        class_dir = os.path.join(target_dir, row['label'])
        os.makedirs(class_dir, exist_ok=True)
        shutil.copy(row['path'], class_dir)

copy_data(train_data, train_dir)
copy_data(val_data, val_dir)
copy_data(test_data, test_dir)

datagen = ImageDataGenerator(rescale=1/255.0)

width=128
height=128

trainDatagen = datagen.flow_from_directory(
    train_dir,
    target_size=(width, height),
    class_mode='binary',
    batch_size=32
)

valDatagen = datagen.flow_from_directory(
    val_dir,
    target_size=(width, height),
    class_mode='binary',
    batch_size=32
)

testDatagen = datagen.flow_from_directory(
    test_dir,
    target_size=(width, height),
    class_mode='binary',
    batch_size=32,
    shuffle=False
)
```

```
Found 16758 images belonging to 2 classes.
Found 5670 images belonging to 2 classes.
Found 5676 images belonging to 2 classes.
```

In [13]:
```
# Count the number of images in each category
categories = ['Parasitized', 'Uninfected']
```

```
counts = [len(os.listdir(os.path.join(dataset_dir, category))) for category in categories]

# Generate the pie chart
plt.figure(figsize=(8, 8))
plt.pie(counts, labels=categories, autopct='%1.1f%%', startangle=90, colors=['#ff9999', '#66b3ff'])
plt.title('Distribution of Uninfected vs Parasitized Cells')
plt.show()
```

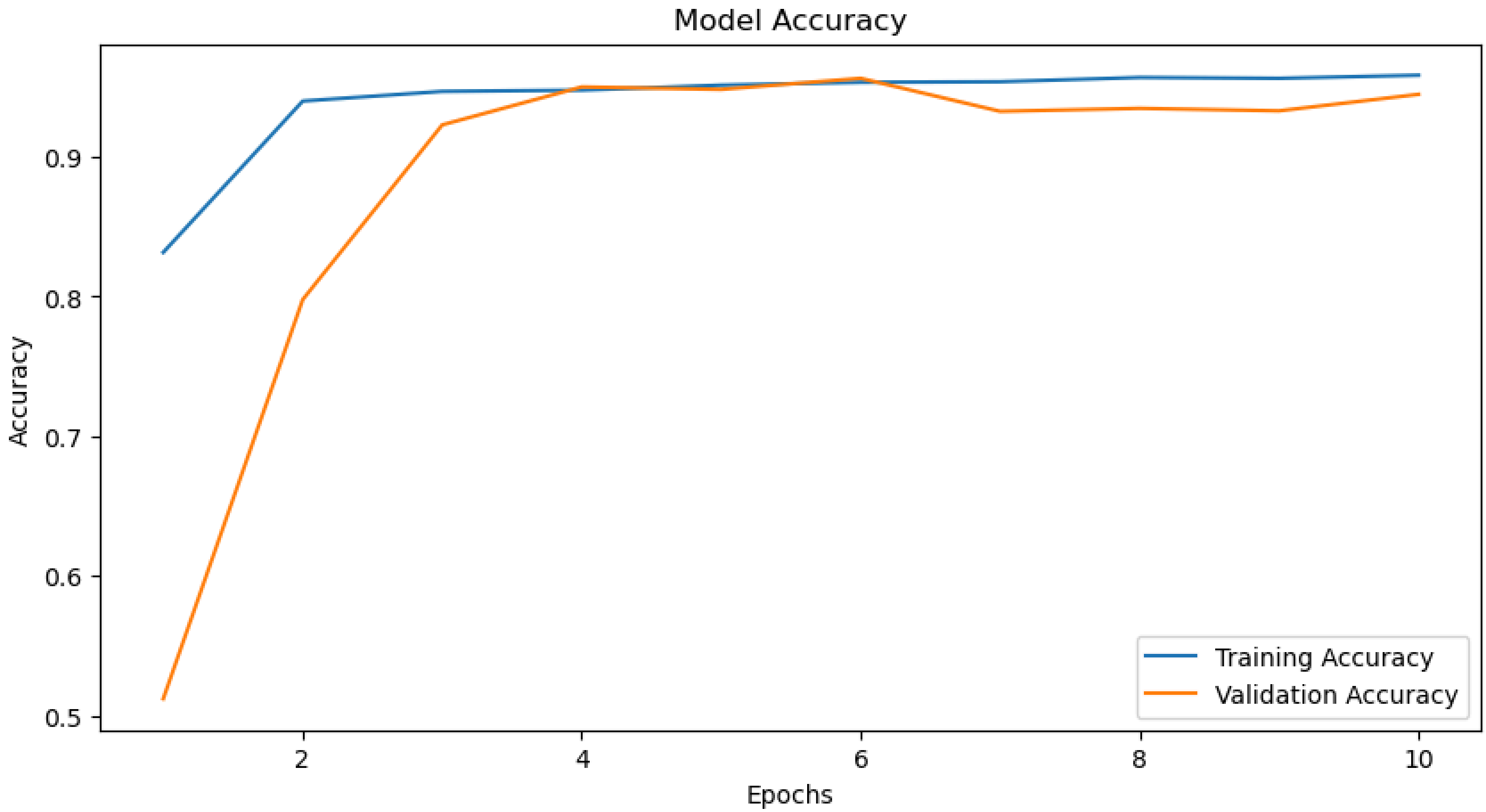

Distribution of Uninfected vs Parasitized Cells
Parasitized
50.0%
50.0%
Uninfected

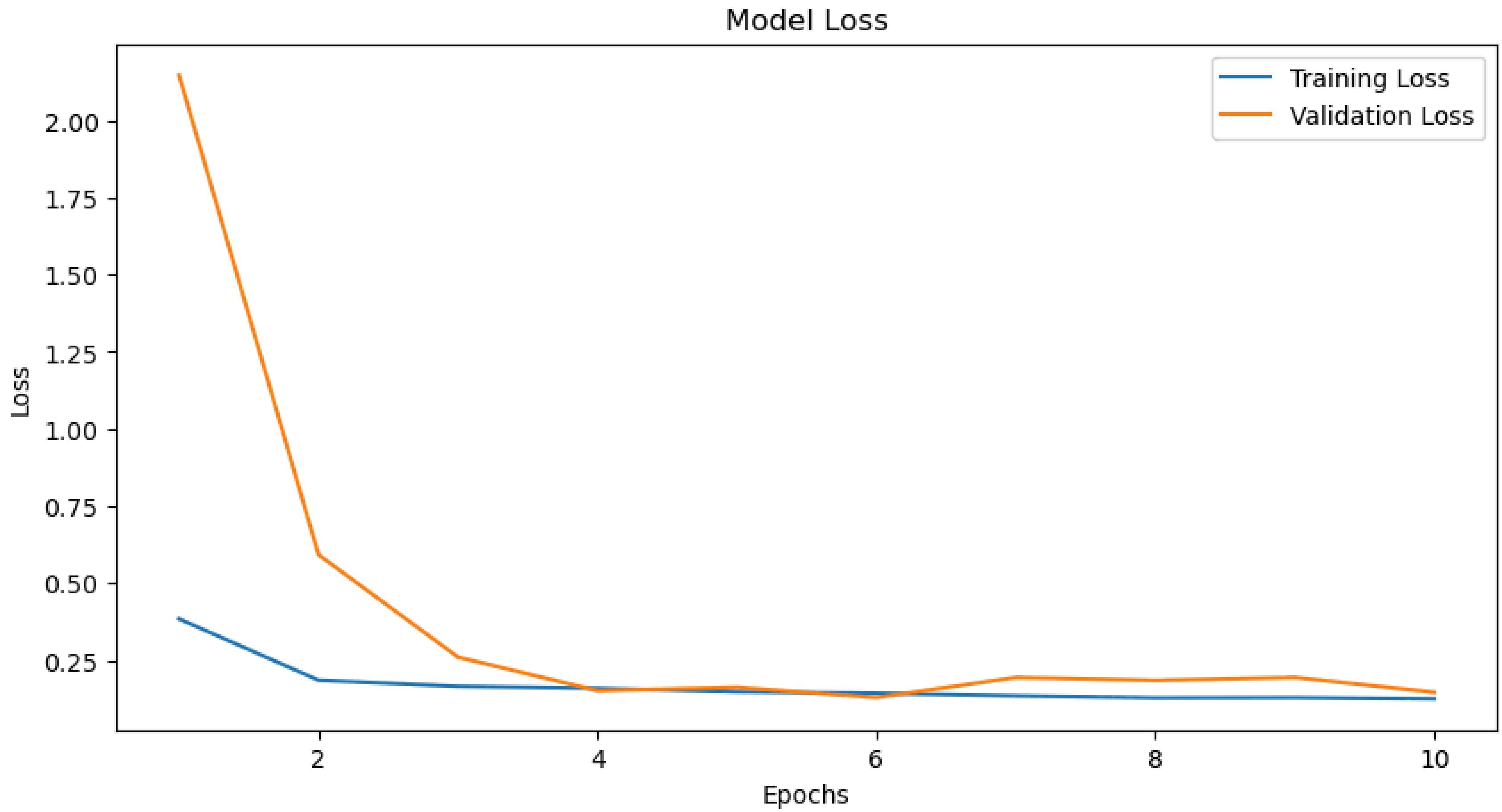
CNN Model

In [4]:
```
cnn_model = Sequential()

cnn_model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPool2D(2, 2))
cnn_model.add(Dropout(0.2))

cnn_model.add(Conv2D(64, (3, 3), activation='relu'))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPool2D(2, 2))
cnn_model.add(Dropout(0.3))

cnn_model.add(Conv2D(128, (3, 3), activation='relu'))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPool2D(2, 2))
cnn_model.add(Dropout(0.4))

cnn_model.add(GlobalAveragePooling2D())
cnn_model.add(Dense(128, activation='relu'))
cnn_model.add(Dropout(0.5))
cnn_model.add(Dense(1, activation='sigmoid'))

cnn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

early_stop = EarlyStopping(monitor='val_loss', patience=5)

def scheduler(epoch, lr):
    return lr * 0.9 if epoch > 3 else lr

lr_scheduler = LearningRateScheduler(scheduler)

# Train the model
cnn_history = cnn_model.fit(
    x=trainDatagen,
    steps_per_epoch=len(trainDatagen),
    epochs=10,
    validation_data=valDatagen,
    validation_steps=len(valDatagen),
    callbacks=[early_stop, lr_scheduler]
)

# Plot learning curves
def plotLearningCurve(history):
    epochs = range(1, len(history.history['accuracy']) + 1)
```

```
    # Plot accuracy
    plt.figure(figsize=(10, 5))
    plt.plot(epochs, history.history['accuracy'], label='Training Accuracy')
    plt.plot(epochs, history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend(loc='best')
    plt.show()
    # Plot loss
    plt.figure(figsize=(10, 5))
    plt.plot(epochs, history.history['loss'], label='Training Loss')
    plt.plot(epochs, history.history['val_loss'], label='Validation Loss')
    plt.title('Model Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend(loc='best')
    plt.show()

plotLearningCurve(cnn_history)
```

```
Epoch 1/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 354s 670ms/step - accuracy: 0.7301 - loss: 0.5358 - val_accuracy: 0.5122 - val_loss: 2.1460
- learning_rate: 0.0010
Epoch 2/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 167s 319ms/step - accuracy: 0.9398 - loss: 0.1863 - val_accuracy: 0.7979 - val_loss: 0.5924
- learning_rate: 0.0010
Epoch 3/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 172s 328ms/step - accuracy: 0.9464 - loss: 0.1666 - val_accuracy: 0.9229 - val_loss: 0.2606
- learning_rate: 0.0010
Epoch 4/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 201s 383ms/step - accuracy: 0.9478 - loss: 0.1625 - val_accuracy: 0.9501 - val_loss: 0.1520
- learning_rate: 0.0010
Epoch 5/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 174s 332ms/step - accuracy: 0.9516 - loss: 0.1532 - val_accuracy: 0.9483 - val_loss: 0.1627
- learning_rate: 9.0000e-04
Epoch 6/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 161s 307ms/step - accuracy: 0.9518 - loss: 0.1486 - val_accuracy: 0.9563 - val_loss: 0.1298
- learning_rate: 8.1000e-04
Epoch 7/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 161s 307ms/step - accuracy: 0.9554 - loss: 0.1362 - val_accuracy: 0.9326 - val_loss: 0.1947
- learning_rate: 7.2900e-04
Epoch 8/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 161s 307ms/step - accuracy: 0.9573 - loss: 0.1285 - val_accuracy: 0.9347 - val_loss: 0.1852
- learning_rate: 6.5610e-04
Epoch 9/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 160s 305ms/step - accuracy: 0.9572 - loss: 0.1267 - val_accuracy: 0.9330 - val_loss: 0.1947
- learning_rate: 5.9049e-04
Epoch 10/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 165s 314ms/step - accuracy: 0.9559 - loss: 0.1280 - val_accuracy: 0.9448 - val_loss: 0.1467
- learning_rate: 5.3144e-04
```


Model Accuracy
Accuracy
0.9
0.8
0.7
0.6
0.5
2
4
6
8
10
Epochs
Training Accuracy
Validation Accuracy



CNN Testing

In [5]:

```
val_predictions = (cnn_model.predict(testDatagen) > 0.5).astype("int32")
val_true_labels = testDatagen.classes

cnn_accuracy = accuracy_score(val_true_labels, val_predictions)
cnn_precision = precision_score(val_true_labels, val_predictions)
cnn_recall = recall_score(val_true_labels, val_predictions)
cnn_f1 = f1_score(val_true_labels, val_predictions)

print(f"Accuracy: {cnn_accuracy}")
print(f"Precision: {cnn_precision}")
```

```
print(f"Recall: {cnn_recall}")
print(f"F1 Score: {cnn_f1}")
```

```
178/178 ━━━━━━━━━━━━━━━━━━━━ 67s 374ms/step
Accuracy: 0.94538407329105
Precision: 0.9182389937106918
Recall: 0.9777934437786394
F1 Score: 0.9470809149880506
```

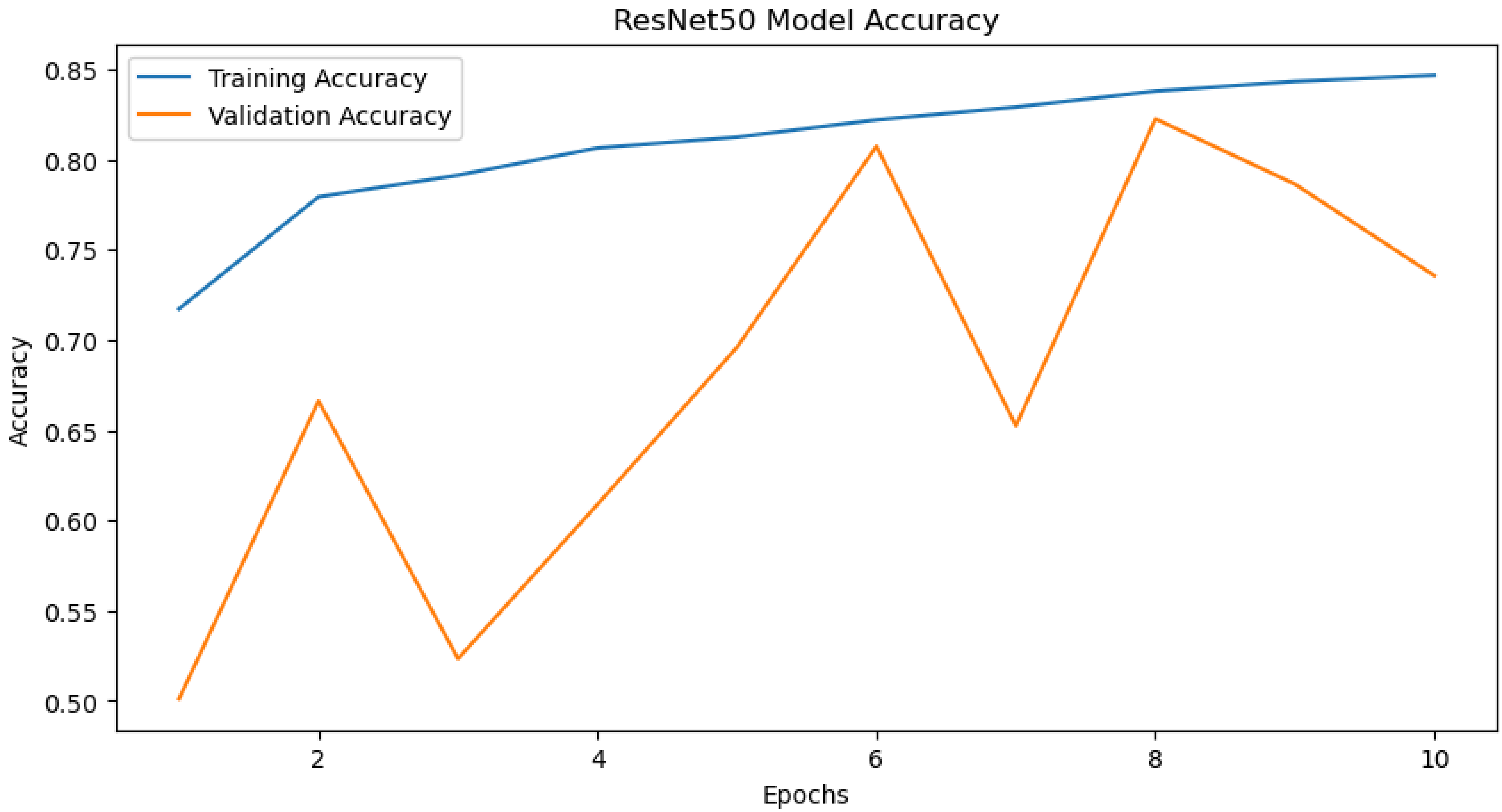
Resnet Model

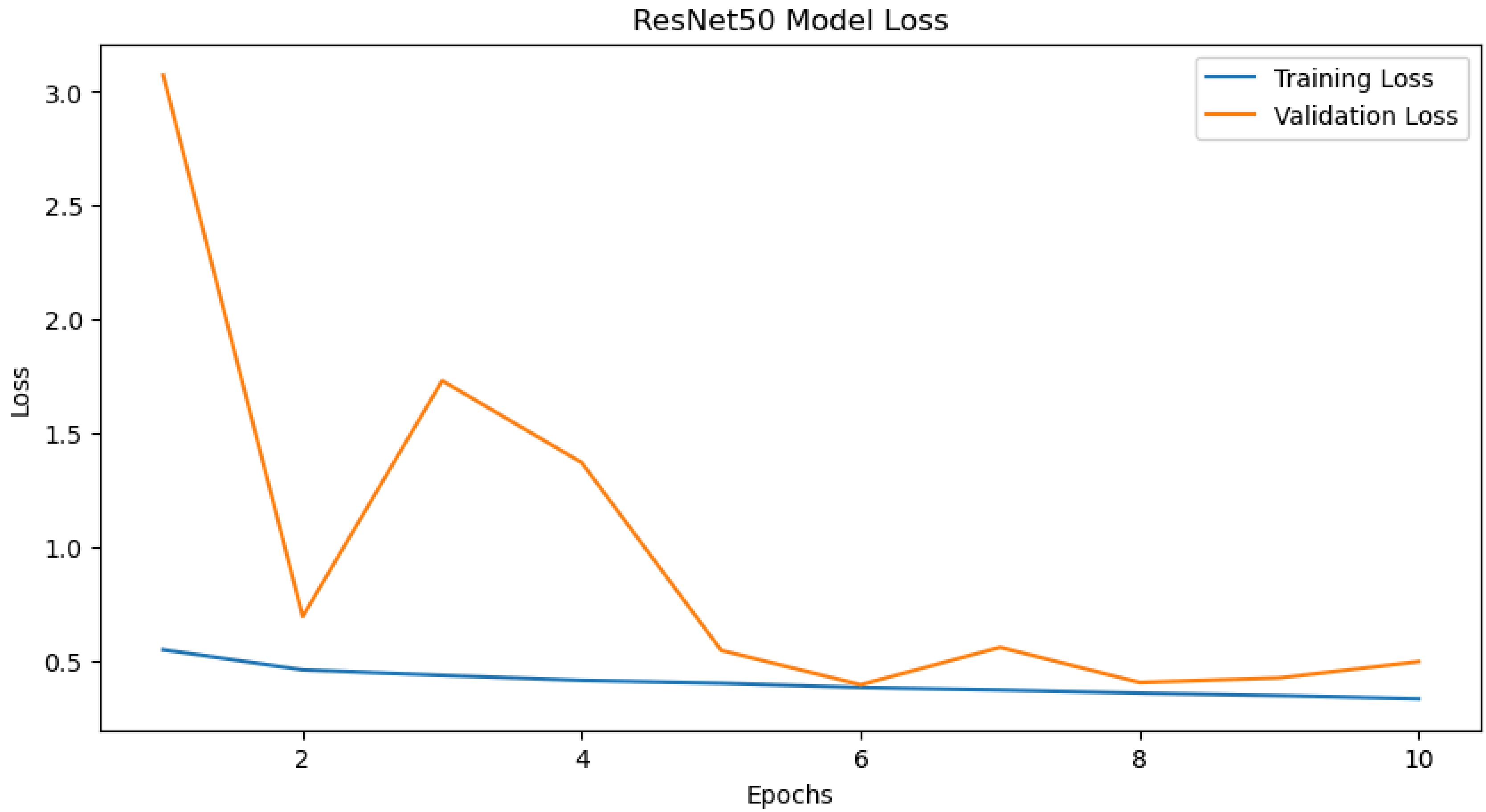
In [6]:

```
base_resnet = ResNet50(weights='imagenet', include_top=False, input_shape=(128, 128, 3))

for layer in base_resnet.layers[:140]:
    layer.trainable = False
for layer in base_resnet.layers[140:]:
    layer.trainable = True

resnet_model = Sequential([
    base_resnet,
    GlobalAveragePooling2D(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

resnet_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4), loss='binary_crossentropy', metrics=['accurac

early_stop = EarlyStopping(monitor='val_loss', patience=5)

def scheduler(epoch, lr):
    return lr * 0.9 if epoch > 5 else lr

lr_scheduler = LearningRateScheduler(scheduler)

resnet_history = resnet_model.fit(
    x=trainDatagen,
    steps_per_epoch=len(trainDatagen),
    epochs=10,
    validation_data=valDatagen,
    validation_steps=len(valDatagen),
    callbacks=[early_stop, lr_scheduler]
)
```

```
# Plot learning curves
def plotLearningCurve(history, title_prefix="ResNet50"):
    epochs = range(1, len(history.history['accuracy']) + 1)
    # Plot accuracy
    plt.figure(figsize=(10, 5))
    plt.plot(epochs, history.history['accuracy'], label='Training Accuracy')
    plt.plot(epochs, history.history['val_accuracy'], label='Validation Accuracy')
    plt.title(f'{title_prefix} Model Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend(loc='best')
    plt.show()
    # Plot loss
    plt.figure(figsize=(10, 5))
    plt.plot(epochs, history.history['loss'], label='Training Loss')
    plt.plot(epochs, history.history['val_loss'], label='Validation Loss')
    plt.title(f'{title_prefix} Model Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend(loc='best')
    plt.show()

plotLearningCurve(resnet_history, "ResNet50")
```

```
Epoch 1/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 462s 857ms/step - accuracy: 0.6761 - loss: 0.5996 - val_accuracy: 0.5012 - val_loss: 3.0720
- learning_rate: 1.0000e-04
Epoch 2/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 418s 798ms/step - accuracy: 0.7755 - loss: 0.4647 - val_accuracy: 0.6663 - val_loss: 0.6935
- learning_rate: 1.0000e-04
Epoch 3/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 406s 774ms/step - accuracy: 0.7915 - loss: 0.4363 - val_accuracy: 0.5235 - val_loss: 1.7296
- learning_rate: 1.0000e-04
Epoch 4/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 398s 759ms/step - accuracy: 0.8068 - loss: 0.4094 - val_accuracy: 0.6090 - val_loss: 1.3696
- learning_rate: 1.0000e-04
Epoch 5/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 396s 755ms/step - accuracy: 0.8112 - loss: 0.4048 - val_accuracy: 0.6961 - val_loss: 0.5445
- learning_rate: 1.0000e-04
Epoch 6/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 398s 759ms/step - accuracy: 0.8277 - loss: 0.3790 - val_accuracy: 0.8076 - val_loss: 0.3932
- learning_rate: 1.0000e-04
Epoch 7/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 408s 778ms/step - accuracy: 0.8296 - loss: 0.3713 - val_accuracy: 0.6526 - val_loss: 0.5579
- learning_rate: 9.0000e-05
Epoch 8/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 400s 763ms/step - accuracy: 0.8404 - loss: 0.3549 - val_accuracy: 0.8228 - val_loss: 0.4031
- learning_rate: 8.1000e-05
Epoch 9/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 401s 765ms/step - accuracy: 0.8446 - loss: 0.3409 - val_accuracy: 0.7866 - val_loss: 0.4232
- learning_rate: 7.2900e-05
Epoch 10/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 399s 763ms/step - accuracy: 0.8499 - loss: 0.3259 - val_accuracy: 0.7358 - val_loss: 0.4947
- learning_rate: 6.5610e-05
```


ResNet50 Model Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epochs
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
2
4
6
8
10



Resnet Testing

In [7]:
```
val_predictions = (resnet_model.predict(testDatagen) > 0.5).astype("int32")
val_true = valDatagen.classes

resnet_accuracy = accuracy_score(val_true_labels, val_predictions)
resnet_precision = precision_score(val_true_labels, val_predictions)
resnet_recall = recall_score(val_true_labels, val_predictions)
resnet_f1 = f1_score(val_true_labels, val_predictions)

print(f"ResNet50 Accuracy: {resnet_accuracy}")
print(f"ResNet50 Precision: {resnet_precision}")
```

```
print(f"ResNet50 Recall: {resnet_recall}")
print(f"ResNet50 F1 Score: {resnet_f1}")
```

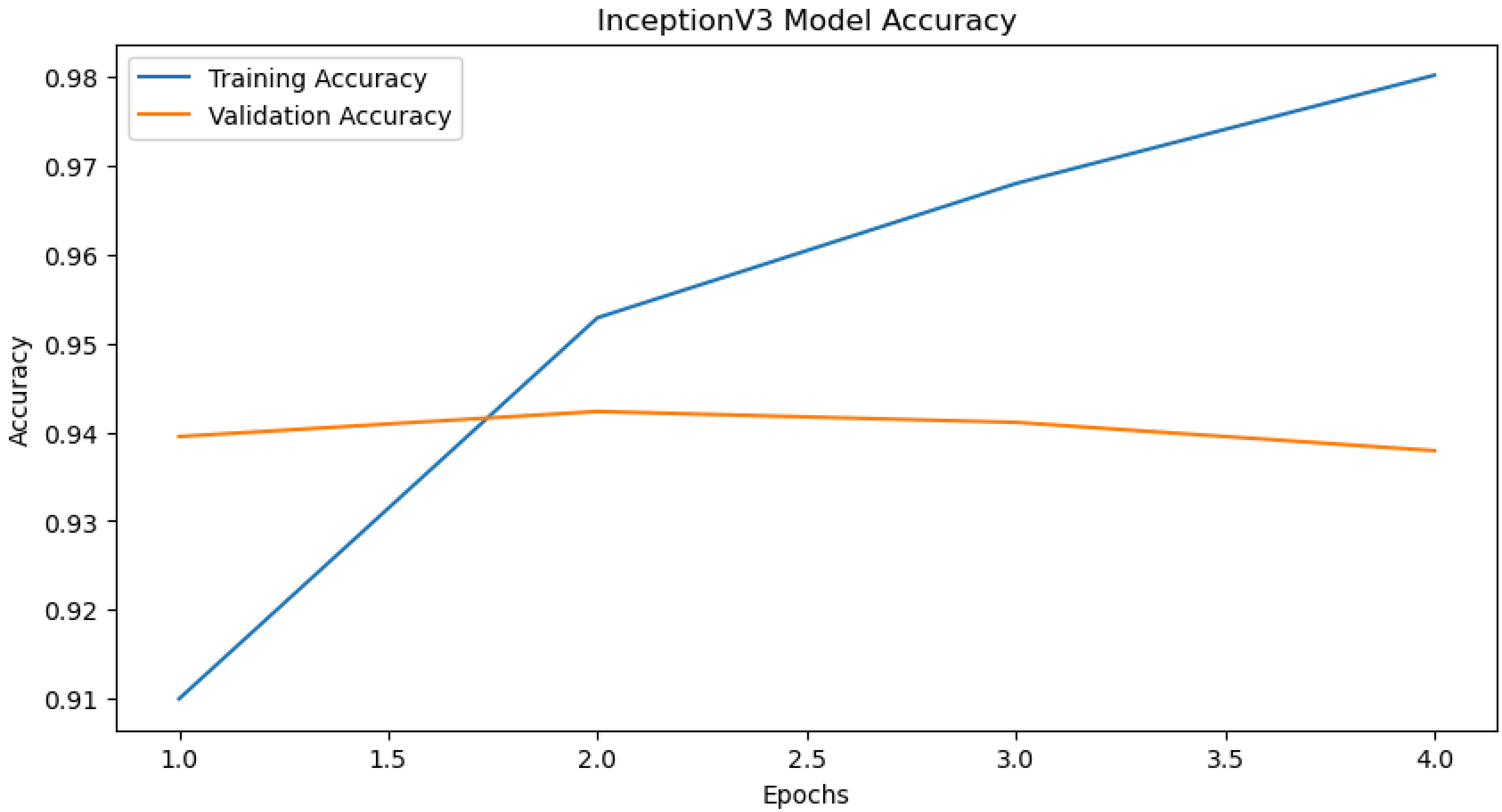
```
178/178 ━━━━━━━━━━━━━━━━━━━━ 75s 409ms/step
ResNet50 Accuracy: 0.7371388301620859
ResNet50 Precision: 0.9153798641136504
ResNet50 Recall: 0.522382798731054
ResNet50 F1 Score: 0.6651705565529623
```

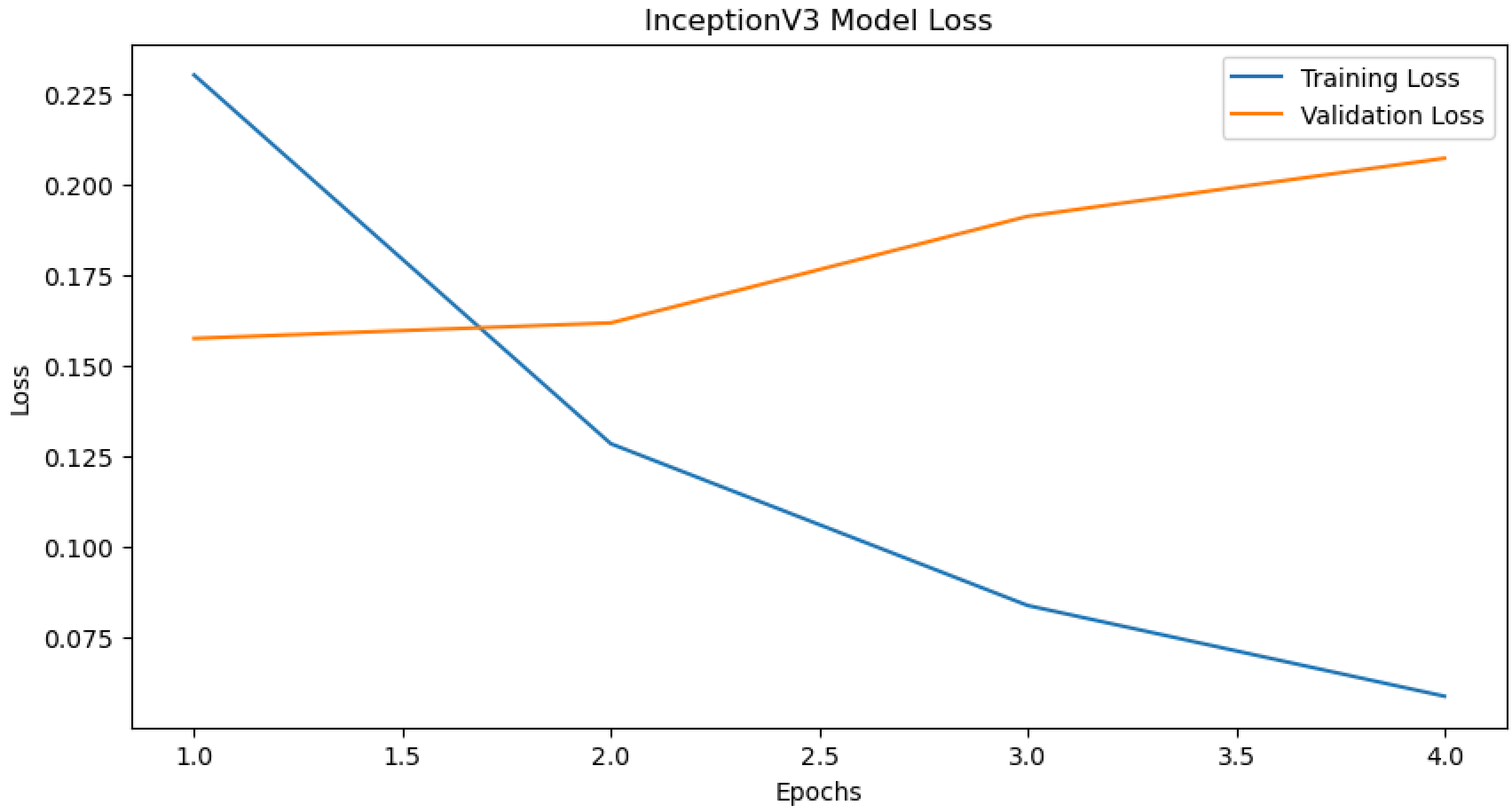
Inception V3 Model

In [8]:
```
base_inception = InceptionV3(weights='imagenet', include_top=False, input_shape=(128, 128, 3))

for layer in base_inception.layers[:249]:
    layer.trainable = False
for layer in base_inception.layers[249:]:
    layer.trainable = True

inception_model = Sequential([
    base_inception,
    GlobalAveragePooling2D(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

inception_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),loss='binary_crossentropy',metrics=['accura

early_stop = EarlyStopping(monitor='val_loss', patience=3)

inception_history = inception_model.fit(
    x=trainDatagen,
    steps_per_epoch=len(trainDatagen),
    epochs=10,
    validation_data=valDatagen,
    validation_steps=len(valDatagen),
    callbacks=[early_stop]
)

# Plot learning curves
def plotLearningCurve(history, title_prefix="InceptionV3"):
    epochs = range(1, len(history.history['accuracy']) + 1)
    # Plot accuracy
```

```
    plt.figure(figsize=(10, 5))
    plt.plot(epochs, history.history['accuracy'], label='Training Accuracy')
    plt.plot(epochs, history.history['val_accuracy'], label='Validation Accuracy')
    plt.title(f'{title_prefix} Model Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend(loc='best')
    plt.show()
    # Plot loss
    plt.figure(figsize=(10, 5))
    plt.plot(epochs, history.history['loss'], label='Training Loss')
    plt.plot(epochs, history.history['val_loss'], label='Validation Loss')
    plt.title(f'{title_prefix} Model Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend(loc='best')
    plt.show()

plotLearningCurve(inception_history, "InceptionV3")
```

```
Epoch 1/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 233s 419ms/step - accuracy: 0.8792 - loss: 0.2962 - val_accuracy: 0.9395 - val_loss: 0.1575
Epoch 2/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 223s 426ms/step - accuracy: 0.9532 - loss: 0.1275 - val_accuracy: 0.9423 - val_loss: 0.1618
Epoch 3/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 219s 418ms/step - accuracy: 0.9707 - loss: 0.0791 - val_accuracy: 0.9411 - val_loss: 0.1912
Epoch 4/10
524/524 ━━━━━━━━━━━━━━━━━━━━ 220s 420ms/step - accuracy: 0.9826 - loss: 0.0507 - val_accuracy: 0.9379 - val_loss: 0.2072
```


InceptionV3 Model Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epochs
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0.91
1.0
1.5
2.0
2.5
3.0
3.5
4.0



Inception V3 Testing

In [9]:
```
val_predictions = (inception_model.predict(testDatagen) > 0.5).astype("int32")
val_true_labels = testDatagen.classes

inception_accuracy = accuracy_score(val_true_labels, val_predictions)
inception_precision = precision_score(val_true_labels, val_predictions)
inception_recall = recall_score(val_true_labels, val_predictions)
inception_f1 = f1_score(val_true_labels, val_predictions)

print(f"InceptionV3 Accuracy: {inception_accuracy}")
print(f"InceptionV3 Precision: {inception_precision}")
```

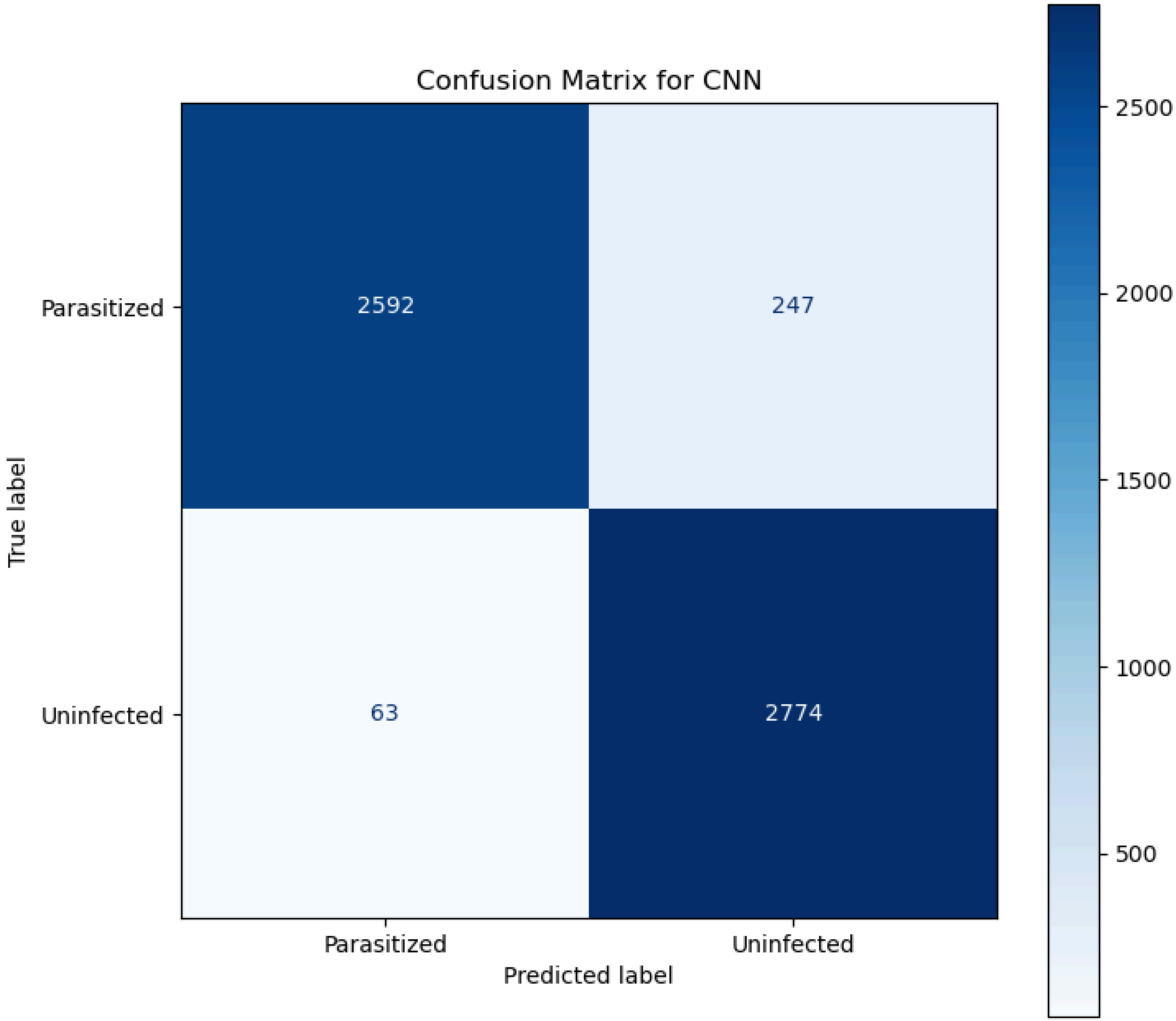
```
print(f"InceptionV3 Recall: {inception_recall}")
print(f"InceptionV3 F1 Score: {inception_f1}")
```

```
178/178 ━━━━━━━━━━━━━━━━━━━━ 42s 222ms/step
InceptionV3 Accuracy: 0.9342847075405215
InceptionV3 Precision: 0.9095744680851063
InceptionV3 Recall: 0.9643990130419458
InceptionV3 F1 Score: 0.9361847733105219
```

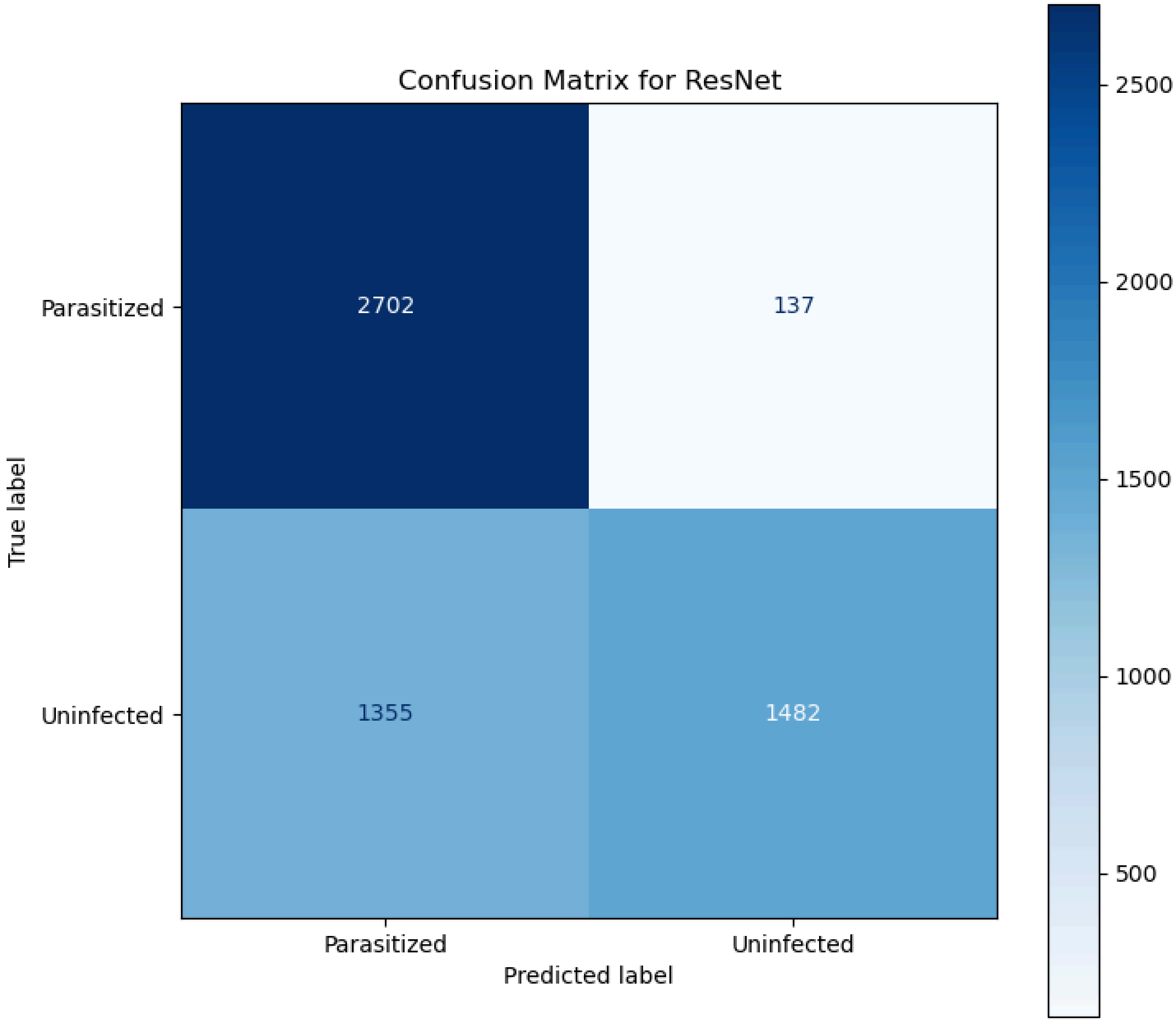
In [10]:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

def plot_confusion_matrix(test_generator, model, model_name):
    # Predict the classes
    predictions = (model.predict(test_generator) > 0.5).astype("int32")
    true_labels = test_generator.classes

    # Compute confusion matrix
    cm = confusion_matrix(true_labels, predictions)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=test_generator.class_indices.keys())

    # Plot the confusion matrix
    plt.figure(figsize=(8, 8))
    disp.plot(cmap=plt.cm.Blues, ax=plt.gca())
    plt.title(f'Confusion Matrix for {model_name}')
    plt.savefig(f'{model_name}_confusion_matrix.png')
    plt.show()

plot_confusion_matrix(testDatagen, cnn_model, "CNN")
plot_confusion_matrix(testDatagen, resnet_model, "ResNet")
plot_confusion_matrix(testDatagen, inception_model, "InceptionV3")
```

```
178/178 ━━━━━━━━━━━━━━━━━━━━ 10s 56ms/step
```




```
178/178 ━━━━━━━━━━━━━━━━━━━━ 39s 216ms/step
```




```
178/178 ━━━━━━━━━━━━━━━━━━━━ 20s 113ms/step
```

In [12]:
```
model_scores = {
    "Model": ["CNN", "ResNet50", "InceptionV3"],
    "Accuracy": [cnn_accuracy, resnet_accuracy, inception_accuracy],
    "Precision": [cnn_precision, resnet_precision, inception_precision],
    "Recall": [cnn_recall, resnet_recall, inception_recall],
```

```
    "F1 Score": [cnn_f1, resnet_f1, inception_f1]
}

scores_df = pd.DataFrame(model_scores)

print(tabulate(scores_df, headers="keys", tablefmt="fancy_grid", floatfmt=".2f"))
```

	Model	Accuracy	Precision	Recall	F1 Score
0	CNN	0.95	0.92	0.98	0.95
1	ResNet50	0.74	0.92	0.52	0.67
2	InceptionV3	0.93	0.91	0.96	0.94

Conclusion: CNN and InceptionV3 are the best models while CNN is slightly better than InceptionV3. ResNet50 needs a lot of improvements if it is to be effective on this dataset.

In [ ]: