CNN Model

In [2]:

```
import numpy as np
import tensorflow as tf
from tensorflow import keras

# Generate random training data (placeholders - replace with actual data)
image_height = 128  # Replace with the desired image height
image_width = 128  # Replace with the desired image width
num_channels = 3  # Replace with the number of image channels (e.g., 3 for RGB)
num_classes = 4  # Replace with the number of vegetation classes

# Define the CNN model architecture
model = keras.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(image_height, image_width, num_channels)),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(num_classes, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# Generate random training data (placeholders - replace with actual data)
train_images = np.random.rand(100, image_height, image_width, num_channels)
train_labels = np.random.randint(num_classes, size=(100,))

# Train the model
model.fit(train_images, train_labels, epochs=10, batch_size=32)

# Generate random testing data (placeholders - replace with actual data)
test_images = np.random.rand(20, image_height, image_width, num_channels)
test_labels = np.random.randint(num_classes, size=(20,))

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
```

```
print('Test accuracy:', test_acc)

# Generate random new images (placeholders - replace with actual data)
new_images = np.random.rand(5, image_height, image_width, num_channels)

# Classify new images
predictions = model.predict(new_images)
```

```
Epoch 1/10
C:\Users\Owner\anaconda3\lib\site-packages\keras\backend.py:5612: UserWarning: "`sparse_categorical_crossentropy` recei
ved `from_logits=True`, but the `output` argument was produced by a Softmax activation and thus does not represent logi
ts. Was this intended?
  output, from_logits = _get_logits(
4/4 [==============================] - 1s 153ms/step - loss: 3.7482 - accuracy: 0.3200
Epoch 2/10
4/4 [==============================] - 1s 142ms/step - loss: 1.6449 - accuracy: 0.3000
Epoch 3/10
4/4 [==============================] - 1s 143ms/step - loss: 1.3685 - accuracy: 0.3700
Epoch 4/10
4/4 [==============================] - 1s 149ms/step - loss: 1.3769 - accuracy: 0.4100
Epoch 5/10
4/4 [==============================] - 1s 140ms/step - loss: 1.3330 - accuracy: 0.3400
Epoch 6/10
4/4 [==============================] - 1s 138ms/step - loss: 1.3187 - accuracy: 0.4000
Epoch 7/10
4/4 [==============================] - 1s 130ms/step - loss: 1.2455 - accuracy: 0.3800
Epoch 8/10
4/4 [==============================] - 1s 144ms/step - loss: 1.1538 - accuracy: 0.5100
Epoch 9/10
4/4 [==============================] - 1s 130ms/step - loss: 1.0882 - accuracy: 0.6900
Epoch 10/10
4/4 [==============================] - 1s 130ms/step - loss: 1.0130 - accuracy: 0.7800
1/1 [==============================] - 0s 159ms/step - loss: 1.8984 - accuracy: 0.2000
Test accuracy: 0.20000000298023224
1/1 [==============================] - 0s 81ms/step
```

Random Forest Model

In [3]:

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier

# Generate random training data (placeholders - replace with actual data)
image_height = 128  # Replace with the desired image height
```

```
image_width = 128  # Replace with the desired image width
num_channels = 3  # Replace with the number of image channels (e.g., 3 for RGB)
num_classes = 4  # Replace with the number of vegetation classes

# Generate random training data (placeholders - replace with actual data)
train_images = np.random.rand(100, image_height * image_width * num_channels)
train_labels = np.random.randint(num_classes, size=(100,))

# Create a Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100)

# Train the classifier
rf_classifier.fit(train_images, train_labels)

# Generate random testing data (placeholders - replace with actual data)
test_images = np.random.rand(20, image_height * image_width * num_channels)
test_labels = np.random.randint(num_classes, size=(20,))

# Evaluate the classifier
test_accuracy = rf_classifier.score(test_images, test_labels)
print('Test accuracy:', test_accuracy)

# Generate random new images (placeholders - replace with actual data)
new_images = np.random.rand(5, image_height * image_width * num_channels)

# Classify new images
predictions = rf_classifier.predict(new_images)
```

```
Test accuracy: 0.2
```

Support Vector Machine Model

In [4]:
```
import numpy as np
from sklearn.svm import SVC

# Generate random training data (placeholders - replace with actual data)
image_height = 128  # Replace with the desired image height
image_width = 128  # Replace with the desired image width
num_channels = 3  # Replace with the number of image channels (e.g., 3 for RGB)
num_classes = 4  # Replace with the number of vegetation classes

# Generate random training data (placeholders - replace with actual data)
train_images = np.random.rand(100, image_height * image_width * num_channels)
```

```
train_labels = np.random.randint(num_classes, size=(100,))

# Create an SVM classifier
svm_classifier = SVC(kernel='linear')

# Train the classifier
svm_classifier.fit(train_images, train_labels)

# Generate random testing data (placeholders - replace with actual data)
test_images = np.random.rand(20, image_height * image_width * num_channels)
test_labels = np.random.randint(num_classes, size=(20,))

# Evaluate the classifier
test_accuracy = svm_classifier.score(test_images, test_labels)
print('Test accuracy:', test_accuracy)

# Generate random new images (placeholders - replace with actual data)
new_images = np.random.rand(5, image_height * image_width * num_channels)

# Classify new images
predictions = svm_classifier.predict(new_images)
```

```
Test accuracy: 0.25
```

Assessing the CNN, RF & SVM models using accuracy scores, classification reports and confusion matrices

In [5]:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import tensorflow as tf
from tensorflow import keras
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

# Generate random data (placeholders - replace with actual data)
image_height = 128  # Replace with the desired image height
image_width = 128  # Replace with the desired image width
num_channels = 3  # Replace with the number of image channels (e.g., 3 for RGB)
num_classes = 4  # Replace with the number of vegetation classes

# Generate random data (placeholders - replace with actual data)
```

```
data = np.random.rand(1000, image_height, image_width, num_channels)
labels = np.random.randint(num_classes, size=(1000,))

# Split the data into training and testing sets
train_data, test_data, train_labels, test_labels = train_test_split(data, labels, test_size=0.2, random_state=42)

# CNN model evaluation
cnn_model = keras.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(image_height, image_width, num_channels)),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(num_classes, activation='softmax')
])
cnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
cnn_model.fit(train_data, train_labels, epochs=10, batch_size=32)
cnn_predictions = np.argmax(cnn_model.predict(test_data), axis=-1)
cnn_accuracy = accuracy_score(test_labels, cnn_predictions)
cnn_classification_report = classification_report(test_labels, cnn_predictions)
cnn_confusion_matrix = confusion_matrix(test_labels, cnn_predictions)

# Random Forest model evaluation
rf_model = RandomForestClassifier()
rf_model.fit(train_data.reshape(len(train_data), -1), train_labels)
rf_predictions = rf_model.predict(test_data.reshape(len(test_data), -1))
rf_accuracy = accuracy_score(test_labels, rf_predictions)
rf_classification_report = classification_report(test_labels, rf_predictions)
rf_confusion_matrix = confusion_matrix(test_labels, rf_predictions)

# SVM model evaluation
svm_model = SVC(kernel='linear')
svm_model.fit(train_data.reshape(len(train_data), -1), train_labels)
svm_predictions = svm_model.predict(test_data.reshape(len(test_data), -1))
svm_accuracy = accuracy_score(test_labels, svm_predictions)
svm_classification_report = classification_report(test_labels, svm_predictions)
svm_confusion_matrix = confusion_matrix(test_labels, svm_predictions)

# Print the accuracy scores
print("CNN Accuracy:", cnn_accuracy)
print("Random Forest Accuracy:", rf_accuracy)
print("SVM Accuracy:", svm_accuracy)

# Print classification reports
```

```
print("CNN Classification Report:\n", cnn_classification_report)
print("Random Forest Classification Report:\n", rf_classification_report)
print("SVM Classification Report:\n", svm_classification_report)

# Print confusion matrices
print("CNN Confusion Matrix:\n", cnn_confusion_matrix)
print("Random Forest Confusion Matrix:\n", rf_confusion_matrix)
print("SVM Confusion Matrix:\n", svm_confusion_matrix)
```

```
Epoch 1/10
25/25 [==============================] - 5s 183ms/step - loss: 2.4244 - accuracy: 0.2438
Epoch 2/10
25/25 [==============================] - 5s 184ms/step - loss: 1.3907 - accuracy: 0.2537
Epoch 3/10
25/25 [==============================] - 5s 193ms/step - loss: 1.3875 - accuracy: 0.2937
Epoch 4/10
25/25 [==============================] - 5s 185ms/step - loss: 1.3862 - accuracy: 0.2225
Epoch 5/10
25/25 [==============================] - 5s 184ms/step - loss: 1.3860 - accuracy: 0.2537
Epoch 6/10
25/25 [==============================] - 5s 187ms/step - loss: 1.3857 - accuracy: 0.2550
Epoch 7/10
25/25 [==============================] - 5s 189ms/step - loss: 1.3856 - accuracy: 0.2387
Epoch 8/10
25/25 [==============================] - 5s 194ms/step - loss: 1.3857 - accuracy: 0.2475
Epoch 9/10
25/25 [==============================] - 5s 209ms/step - loss: 1.3855 - accuracy: 0.2463
Epoch 10/10
25/25 [==============================] - 5s 207ms/step - loss: 1.3854 - accuracy: 0.2600
7/7 [==============================] - 0s 40ms/step
```

```
C:\Users\Owner\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to c
ontrol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Owner\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to c
ontrol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Owner\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to c
ontrol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
CNN Accuracy: 0.24
Random Forest Accuracy: 0.265
SVM Accuracy: 0.185
CNN Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        55
           1       0.24      1.00      0.39        48
           2       0.00      0.00      0.00        50
           3       0.00      0.00      0.00        47

    accuracy                           0.24       200
   macro avg       0.06      0.25      0.10       200
weighted avg       0.06      0.24      0.09       200

Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.31      0.35      0.32        55
           1       0.25      0.33      0.28        48
           2       0.20      0.08      0.11        50
           3       0.26      0.30      0.28        47

    accuracy                           0.27       200
   macro avg       0.25      0.26      0.25       200
weighted avg       0.26      0.27      0.25       200

SVM Classification Report:
              precision    recall  f1-score   support

           0       0.28      0.31      0.29        55
           1       0.15      0.17      0.16        48
           2       0.00      0.00      0.00        50
           3       0.18      0.26      0.21        47

    accuracy                           0.18       200
   macro avg       0.15      0.18      0.17       200
weighted avg       0.16      0.18      0.17       200

CNN Confusion Matrix:
 [[ 0 55  0  0]
 [ 0 48  0  0]
 [ 0 50  0  0]
 [ 0 47  0  0]]
Random Forest Confusion Matrix:
```

```
 [[19 18  5 13]
 [11 16  8 13]
 [16 17  4 13]
 [16 14  3 14]]
SVM Confusion Matrix:
 [[17 12  8 18]
 [17  8  7 16]
 [13 17  0 20]
 [14 16  5 12]]
```

Assessing the CNN, RF & SVM models using precision, recall, and F1-score

In [6]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score
import tensorflow as tf
from tensorflow import keras
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

# Generate random data (placeholders - replace with actual data)
image_height = 128  # Replace with the desired image height
image_width = 128  # Replace with the desired image width
num_channels = 3  # Replace with the number of image channels (e.g., 3 for RGB)
num_classes = 4  # Replace with the number of vegetation classes

# Generate random data (placeholders - replace with actual data)
data = np.random.rand(1000, image_height, image_width, num_channels)
labels = np.random.randint(num_classes, size=(1000,))

# Split the data into training and testing sets
train_data, test_data, train_labels, test_labels = train_test_split(data, labels, test_size=0.2, random_state=42)

# CNN model evaluation
cnn_model = keras.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(image_height, image_width, num_channels)),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(64, activation='relu'),
```

```
    keras.layers.Dense(num_classes, activation='softmax')
])
cnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
cnn_model.fit(train_data, train_labels, epochs=10, batch_size=32)
cnn_predictions = np.argmax(cnn_model.predict(test_data), axis=-1)
cnn_precision = precision_score(test_labels, cnn_predictions, average='weighted')
cnn_recall = recall_score(test_labels, cnn_predictions, average='weighted')
cnn_f1 = f1_score(test_labels, cnn_predictions, average='weighted')

# Random Forest model evaluation
rf_model = RandomForestClassifier()
rf_model.fit(train_data.reshape(len(train_data), -1), train_labels)
rf_predictions = rf_model.predict(test_data.reshape(len(test_data), -1))
rf_precision = precision_score(test_labels, rf_predictions, average='weighted')
rf_recall = recall_score(test_labels, rf_predictions, average='weighted')
rf_f1 = f1_score(test_labels, rf_predictions, average='weighted')

# SVM model evaluation
svm_model = SVC(kernel='linear')
svm_model.fit(train_data.reshape(len(train_data), -1), train_labels)
svm_predictions = svm_model.predict(test_data.reshape(len(test_data), -1))
svm_precision = precision_score(test_labels, svm_predictions, average='weighted')
svm_recall = recall_score(test_labels, svm_predictions, average='weighted')
svm_f1 = f1_score(test_labels, svm_predictions, average='weighted')

# Print precision, recall, and F1-score
print("CNN Precision:", cnn_precision)
print("CNN Recall:", cnn_recall)
print("CNN F1-score:", cnn_f1)

print("Random Forest Precision:", rf_precision)
print("Random Forest Recall:", rf_recall)
print("Random Forest F1-score:", rf_f1)

print("SVM Precision:", svm_precision)
print("SVM Recall:", svm_recall)
print("SVM F1-score:", svm_f1)
```

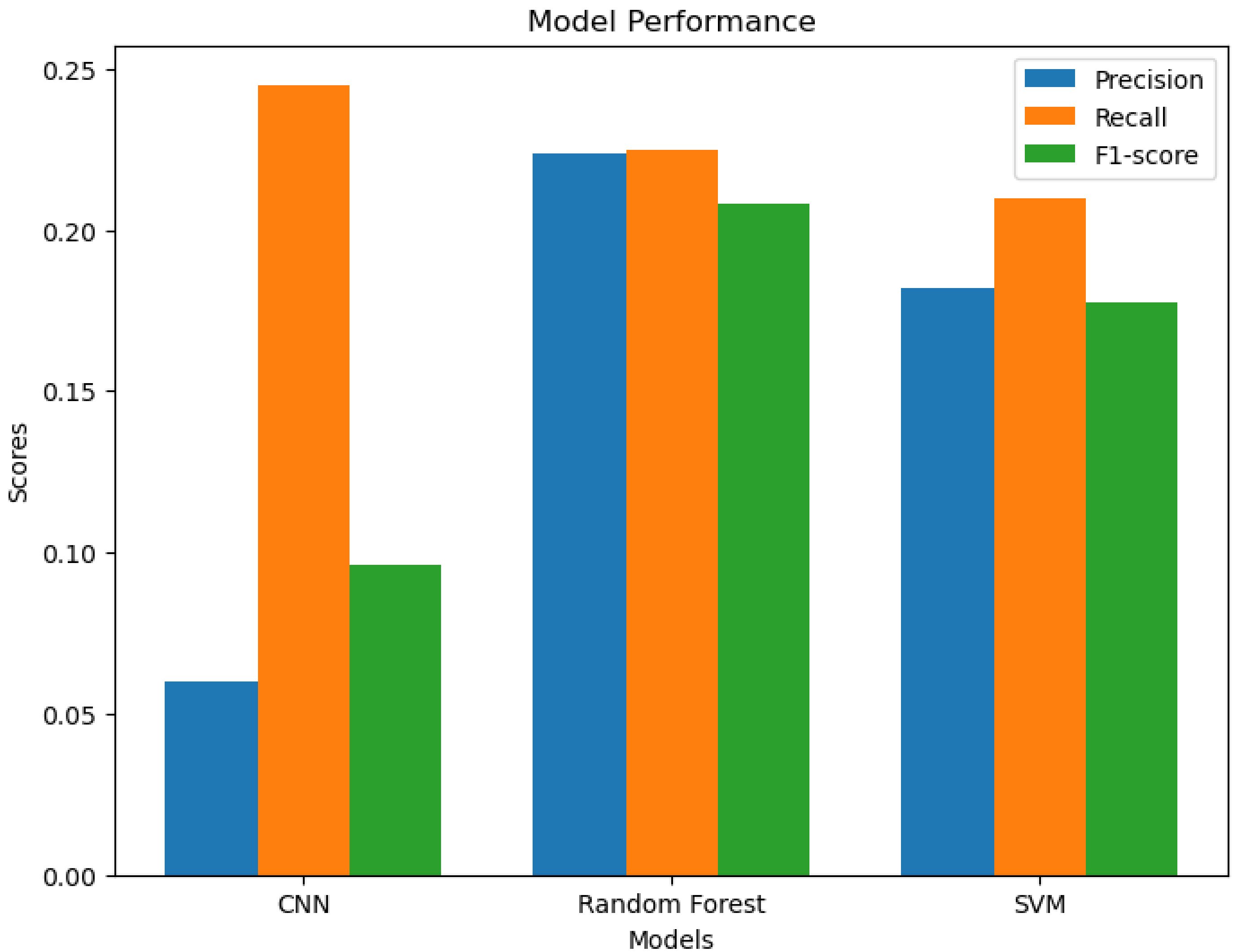
```
Epoch 1/10
25/25 [==============================] - 5s 182ms/step - loss: 1.6489 - accuracy: 0.2438
Epoch 2/10
25/25 [==============================] - 5s 202ms/step - loss: 1.3860 - accuracy: 0.2725
Epoch 3/10
25/25 [==============================] - 5s 200ms/step - loss: 1.3857 - accuracy: 0.2750
Epoch 4/10
25/25 [==============================] - 5s 207ms/step - loss: 1.3854 - accuracy: 0.2750
Epoch 5/10
25/25 [==============================] - 5s 218ms/step - loss: 1.3852 - accuracy: 0.2750
Epoch 6/10
25/25 [==============================] - 5s 203ms/step - loss: 1.3879 - accuracy: 0.2663
Epoch 7/10
25/25 [==============================] - 5s 207ms/step - loss: 1.3846 - accuracy: 0.2750
Epoch 8/10
25/25 [==============================] - 6s 229ms/step - loss: 1.3847 - accuracy: 0.2750
Epoch 9/10
25/25 [==============================] - 5s 202ms/step - loss: 1.3846 - accuracy: 0.2750
Epoch 10/10
25/25 [==============================] - 5s 206ms/step - loss: 1.3846 - accuracy: 0.2750
7/7 [==============================] - 0s 51ms/step
C:\Users\Owner\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision i
s ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
CNN Precision: 0.060024999999999995
CNN Recall: 0.245
CNN F1-score: 0.09642570281124498
Random Forest Precision: 0.2238982500192729
Random Forest Recall: 0.225
Random Forest F1-score: 0.20806160506160506
SVM Precision: 0.18166508112679033
SVM Recall: 0.21
SVM F1-score: 0.17736003694612967
```

Visualization

In [14]:

```
import matplotlib.pyplot as plt

# Precision, Recall, and F1-score values
models = ['CNN', 'Random Forest', 'SVM']
precision_scores = [cnn_precision, rf_precision, svm_precision]
recall_scores = [cnn_recall, rf_recall, svm_recall]
```

```
f1_scores = [cnn_f1, rf_f1, svm_f1]

# Bar plot
fig, ax = plt.subplots(figsize=(8, 6))
x = np.arange(len(models))
width = 0.25

# Precision
ax.bar(x - width, precision_scores, width, label='Precision')
# Recall
ax.bar(x, recall_scores, width, label='Recall')
# F1-score
ax.bar(x + width, f1_scores, width, label='F1-score')

# Customize the plot
ax.set_xlabel('Models')
ax.set_ylabel('Scores')
ax.set_title('Model Performance')
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()

# Display the plot
plt.show()
```



Alternative visualization

In [15]:

```
import pandas as pd

# Create a dataframe
results = pd.DataFrame({
    'Model': models,
    'Precision': precision_scores,
```

```
    'Recall': recall_scores,
    'F1-score': f1_scores
})

# Print the dataframe
print(results)
```

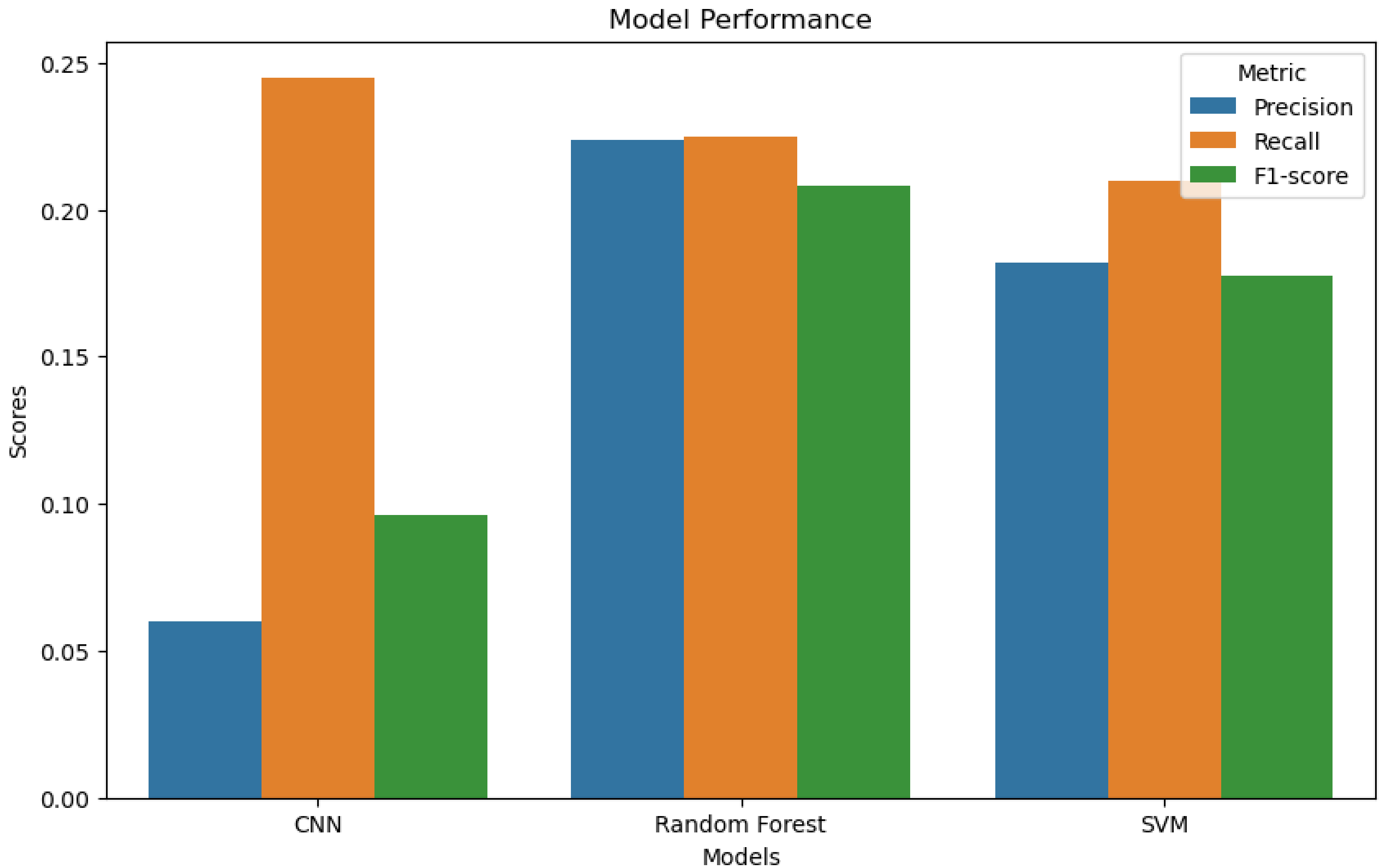
```
           Model  Precision  Recall  F1-score
0            CNN   0.060025   0.245  0.096426
1  Random Forest   0.223898   0.225  0.208062
2            SVM   0.181665   0.210  0.177360
```

Advanced visualization

In [16]:

```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

# Precision, Recall, and F1-score values
models = ['CNN', 'Random Forest', 'SVM']
precision_scores = [cnn_precision, rf_precision, svm_precision]
recall_scores = [cnn_recall, rf_recall, svm_recall]
f1_scores = [cnn_f1, rf_f1, svm_f1]

# Create a dataframe
df = pd.DataFrame({
    'Model': models,
    'Precision': precision_scores,
    'Recall': recall_scores,
    'F1-score': f1_scores
})

# Melt the dataframe
df_melted = df.melt(id_vars='Model', var_name='Metric', value_name='Score')

# Create the plot
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='Score', hue='Metric', data=df_melted)

# Add labels and title
plt.xlabel('Models')
plt.ylabel('Scores')
plt.title('Model Performance')
```

```
# Show the plot
plt.show()
```



More Advanced visualizations

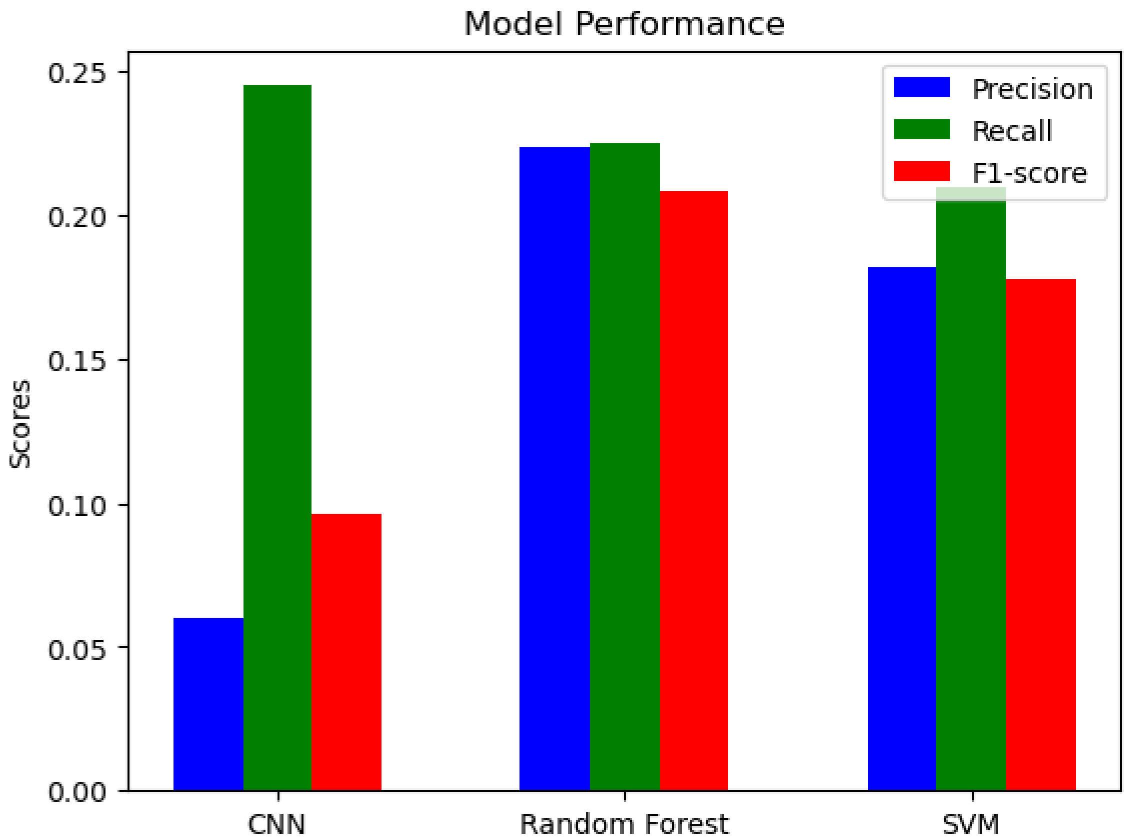
In [17]:
```
import plotly.graph_objects as go
```

```
# Create the trace for each metric
precision_trace = go.Bar(
    x=models,
    y=precision_scores,
    name='Precision'
)
recall_trace = go.Bar(
    x=models,
    y=recall_scores,
    name='Recall'
)
f1_trace = go.Bar(
    x=models,
    y=f1_scores,
    name='F1-score'
)

# Create the data list
data = [precision_trace, recall_trace, f1_trace]

# Set the layout
layout = go.Layout(
    title='Model Performance',
    xaxis=dict(title='Models'),
    yaxis=dict(title='Scores'),
    barmode='group'
)

# Create the figure
fig = go.Figure(data=data, layout=layout)

# Show the figure
fig.show()
```

Other visualizations - Using Pyplot

In [18]:
```
import numpy as np
import matplotlib.pyplot as plt

# Precision, Recall, and F1-score values
models = ['CNN', 'Random Forest', 'SVM']
precision_scores = [cnn_precision, rf_precision, svm_precision]
recall_scores = [cnn_recall, rf_recall, svm_recall]
```

```
f1_scores = [cnn_f1, rf_f1, svm_f1]

# Set the width of the bars
bar_width = 0.2

# Set the positions of the bars on the x-axis
r1 = np.arange(len(models))
r2 = [x + bar_width for x in r1]
r3 = [x + bar_width for x in r2]

# Create the figure and subplots
fig, ax = plt.subplots()

# Plot the precision scores
ax.bar(r1, precision_scores, color='b', width=bar_width, label='Precision')
# Plot the recall scores
ax.bar(r2, recall_scores, color='g', width=bar_width, label='Recall')
# Plot the F1-scores
ax.bar(r3, f1_scores, color='r', width=bar_width, label='F1-score')

# Set the x-axis tick positions and labels
ax.set_xticks([r + bar_width for r in range(len(models))])
ax.set_xticklabels(models)

# Set the y-axis label
ax.set_ylabel('Scores')

# Set the plot title
ax.set_title('Model Performance')

# Add a legend
ax.legend()

# Show the plot
plt.show()
```



Other visualizations - Using Bokeh:

In [19]:
```
from bokeh.plotting import figure, show
from bokeh.models import ColumnDataSource
from bokeh.palettes import Category10
from bokeh.transform import dodge

# Precision, Recall, and F1-score values
models = ['CNN', 'Random Forest', 'SVM']
precision_scores = [cnn_precision, rf_precision, svm_precision]
recall_scores = [cnn_recall, rf_recall, svm_recall]
f1_scores = [cnn_f1, rf_f1, svm_f1]

# Set the width of the bars
bar_width = 0.2
```

```
# Set the x-axis labels and positions
x = models
x_pos = [-0.2, 0.0, 0.2]

# Create a ColumnDataSource
source = ColumnDataSource(data=dict(x=x, precision=precision_scores, recall=recall_scores, f1=f1_scores))

# Create the figure
p = figure(x_range=models, y_range=(0, 1), plot_height=400, title='Model Performance',
           toolbar_location=None, tools='')

# Plot the precision bars
p.vbar(x=dodge('x', -bar_width, range=p.x_range), top='precision', width=bar_width, source=source,
       color=Category10[3][0], legend_label='Precision')

# Plot the recall bars
p.vbar(x=dodge('x', 0, range=p.x_range), top='recall', width=bar_width, source=source,
       color=Category10[3][1], legend_label='Recall')

# Plot the F1-score bars
p.vbar(x=dodge('x', bar_width, range=p.x_range), top='f1', width=bar_width, source=source,
       color=Category10[3][2], legend_label='F1-score')

# Set the y-axis label
p.yaxis.axis_label = 'Scores'

# Add a legend
p.legend.location = 'top_right'
p.legend.orientation = 'horizontal'

# Show the plot
show(p)
```

Other visualizations - Using Altair:

Packages needed to be installed

In [ ]:

```
#pip install altair_saver
#pip install altair_viewer
```

```
#pip install altair
#pip install -U altair_viewer
```

In [20]:
```
import pandas as pd
import altair as alt
import altair_saver

# Create a DataFrame with the model scores
data = pd.DataFrame({
    'Model': ['CNN', 'Random Forest', 'SVM'],
    'Precision': [cnn_precision, rf_precision, svm_precision],
    'Recall': [cnn_recall, rf_recall, svm_recall],
    'F1-score': [cnn_f1, rf_f1, svm_f1]
})

# Melt the DataFrame to convert it to long format
melted_data = data.melt('Model', var_name='Metric', value_name='Score')

# Define the colors for the metrics
colors = {
    'Precision': '#1f77b4',
    'Recall': '#ff7f0e',
    'F1-score': '#2ca02c'
}

# Create the Altair chart
chart = alt.Chart(melted_data).mark_bar().encode(
    x='Model',
    y='Score',
    color=alt.Color('Metric', scale=alt.Scale(domain=list(colors.keys()), range=list(colors.values()))),
    column='Metric'
).properties(
    width=200,
    height=300
).configure_axis(
    labelFontSize=12,
    titleFontSize=14
).configure_legend(
    titleFontSize=14,
    labelFontSize=12
)

# Save the chart as an HTML file
altair_saver.save(chart, 'chart.html')
```

```python
# Open the HTML file in a web browser
import webbrowser
webbrowser.open('chart.html')
```

Out[20]:

```
True
```