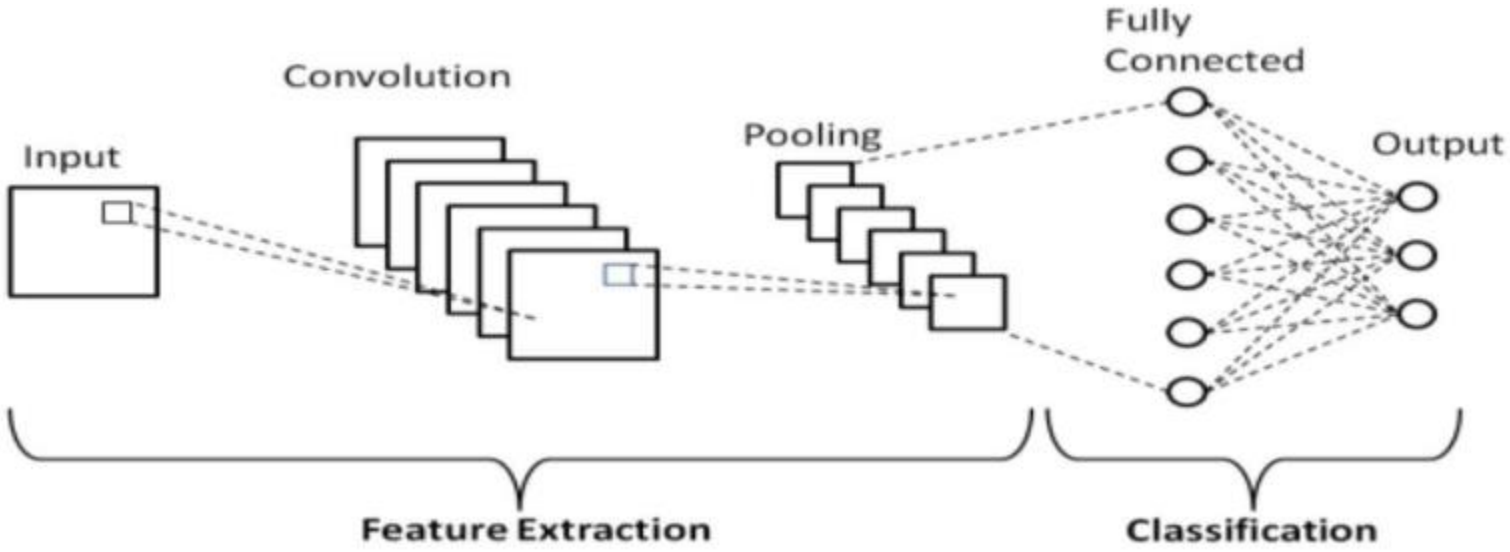
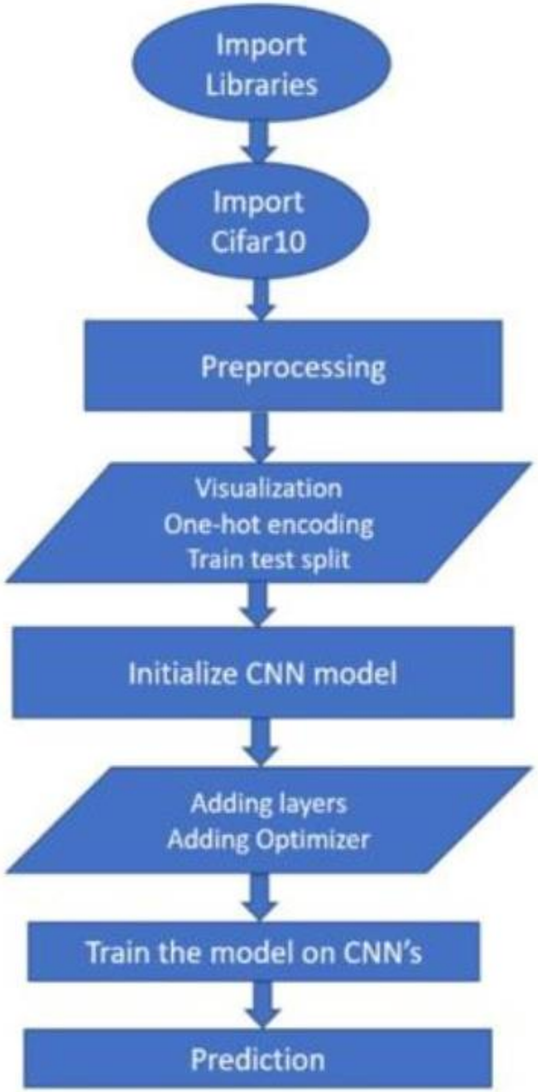
CIFAR10 IMAGE ANALYSIS

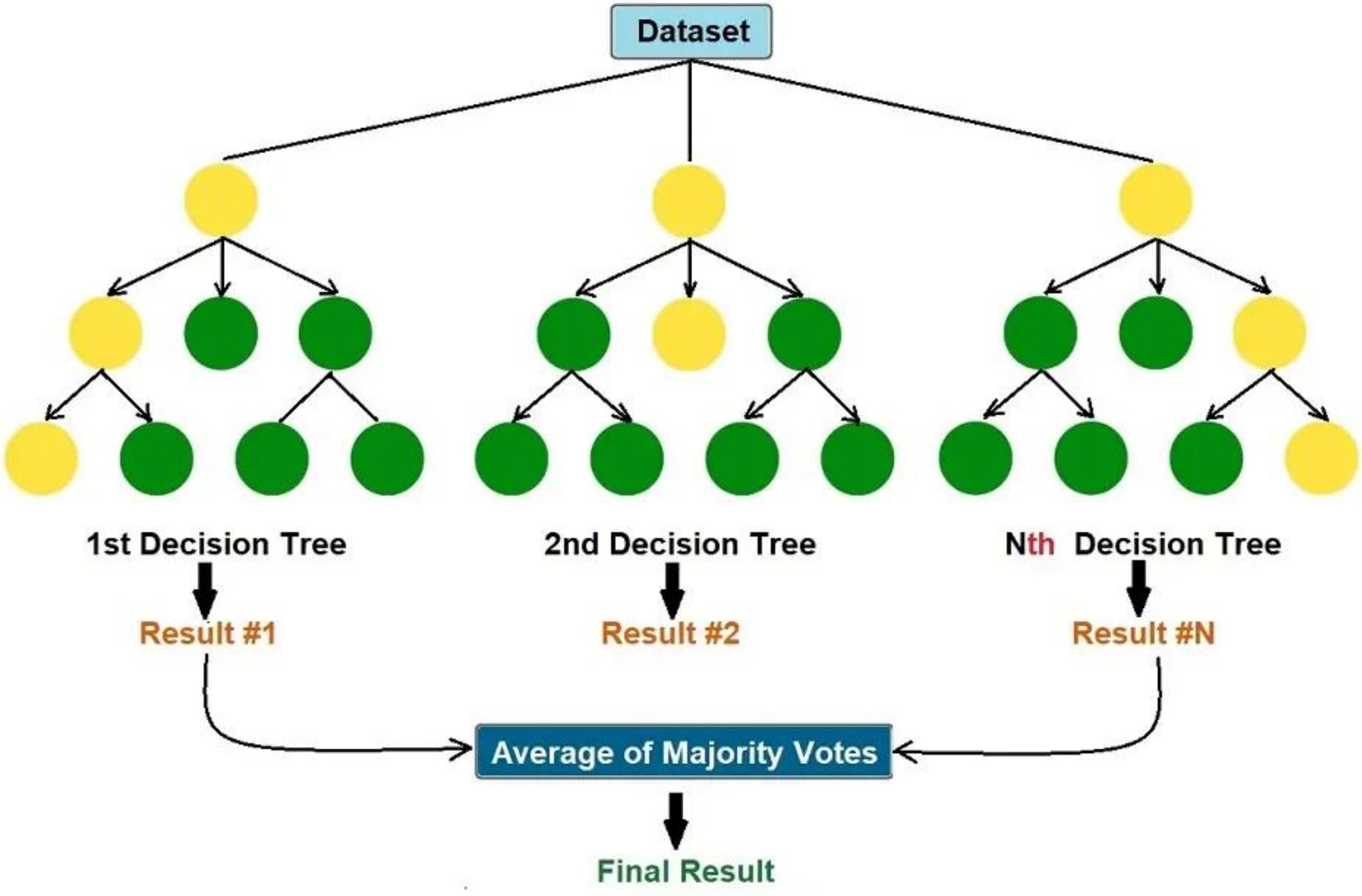
- This database contains;
- 60,000 images
- separated by 10 target classes,
- each a section containing 6000 images of 32 * 32 shapes.
- This database contains images of low-resolution (32 * 32),
- which allows researchers to experiment with new algorithms.

1 - Convolutional Neural Networks (CNN) Analysis Simplified

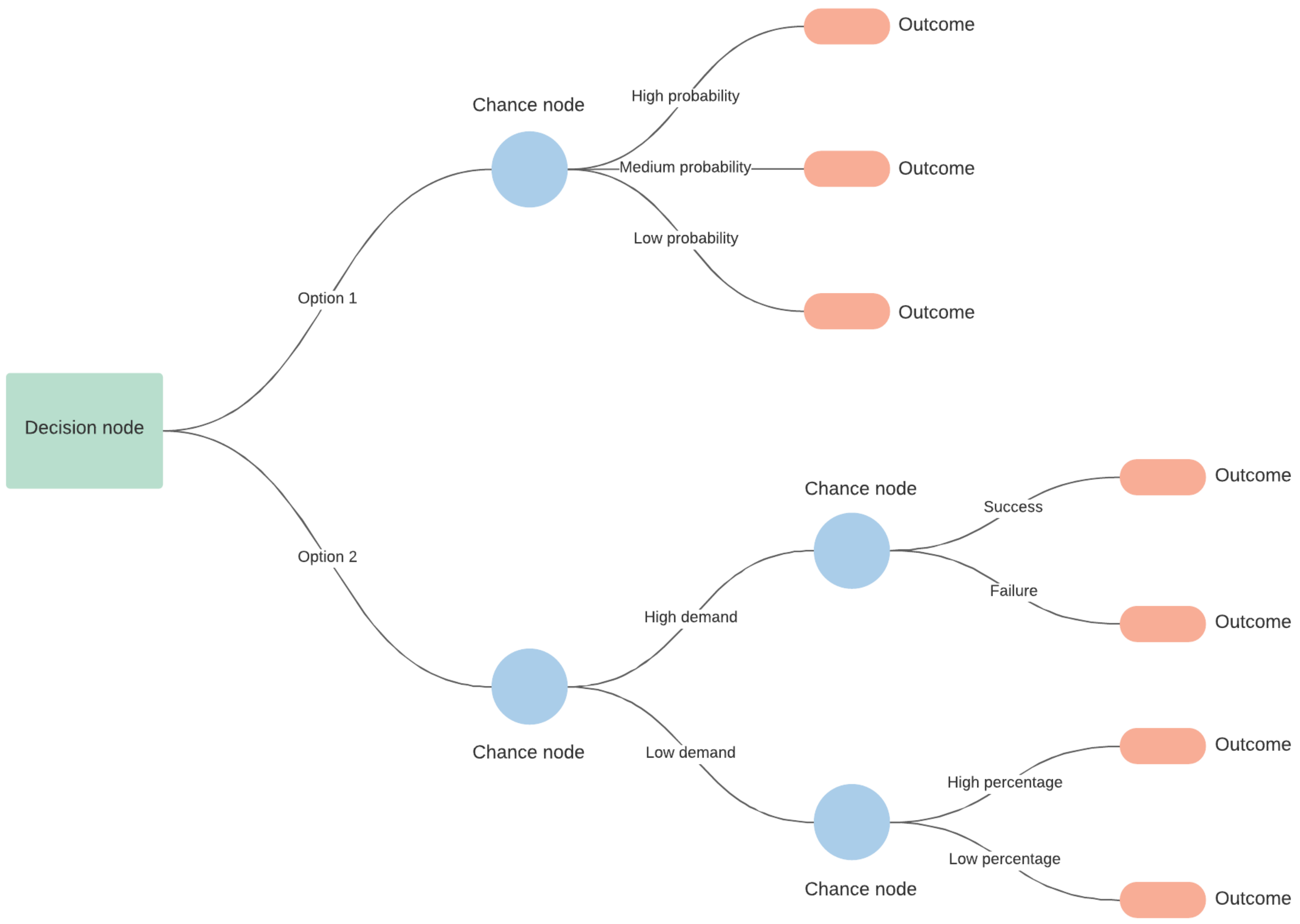




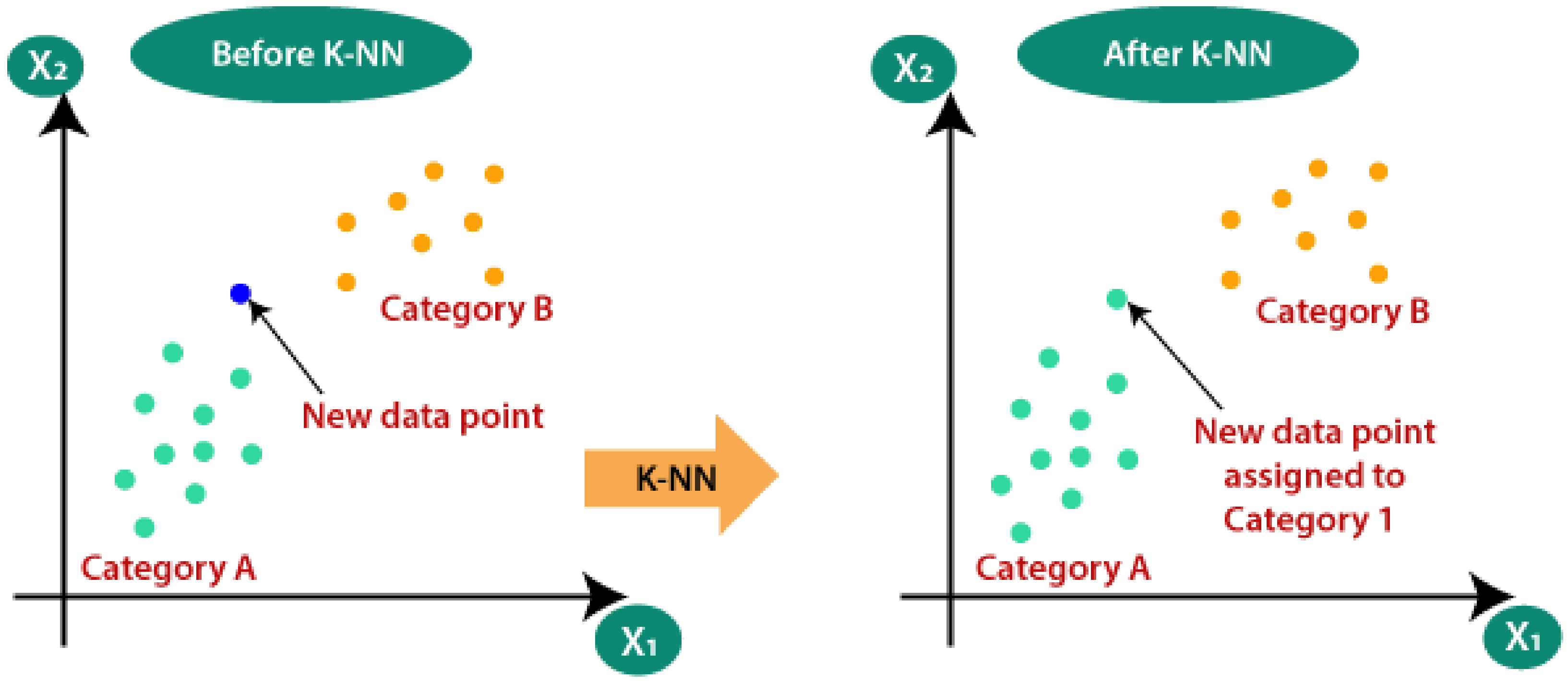
2 – Random Forest Analysis Simplified



3 – Decision Tree Analysis Simplified



4 – K-Nearest Neighbour Analysis Simplified



Coding

```python
# matris işleme kütüphanelerini yüklüyoruz
import numpy as np
import pandas as pd

#görselleştirme kütüphanesini yüklüyoruz
import matplotlib.pyplot as plt

#yapay zeka modelleri kütüphanelerini yüklüyoruz
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from sklearn.preprocessing import StandardScaler

# CIFAR-10 veri setini yükleyip verileri train/test olarak ikiye ayırıyoruz
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# CNN Modeli için veriyi yeniden şekillendirip normalize ediyoruz
X_train_cnn = X_train.astype('float32') / 255.0
X_test_cnn = X_test.astype('float32') / 255.0
y_train_cnn = to_categorical(y_train, 10)
y_test_cnn = to_categorical(y_test, 10)
```

```python
# Düzleştirilmiş verileri normalize ediyoruz
scaler = StandardScaler()
X_train_flat = X_train.reshape(X_train.shape[0], -1)
X_test_flat = X_test.reshape(X_test.shape[0], -1)
X_train_flat = scaler.fit_transform(X_train_flat)
X_test_flat = scaler.transform(X_test_flat)

# Etiketleri tek boyutlu hale getiriyoruz
y_train = y_train.flatten()
y_test = y_test.flatten()

# CNN modelini oluşturuyoruz
cnn_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

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

Coding Continued

```
# CNN Modelini eğitip değerlendiriyoruz
print("Training CNN...")
cnn_model.fit(X_train_cnn, y_train_cnn, epochs=10, validation_data=(X_test_cnn, y_test_cnn), verbose=2)
cnn_y_pred = cnn_model.predict(X_test_cnn).argmax(axis=1)

cnn_report = classification_report(y_test, cnn_y_pred, output_dict=True)
cnn_accuracy = cnn_report["accuracy"]
cnn_precision = np.mean([cnn_report[str(i)]["precision"] for i in range(10)])
cnn_recall = np.mean([cnn_report[str(i)]["recall"] for i in range(10)])
cnn_f1_score = np.mean([cnn_report[str(i)]["f1-score"] for i in range(10)])

metrics = pd.DataFrame(columns=["Model", "Accuracy", "Precision", "Recall", "F1-Score"])
cnn_metrics = pd.DataFrame([{
    "Model": "CNN",
    "Accuracy": cnn_accuracy,
    "Precision": cnn_precision,
    "Recall": cnn_recall,
    "F1-Score": cnn_f1_score
}])
metrics = pd.concat([metrics, cnn_metrics], ignore_index=True)

print(f"\nPerformance for CNN:")
print(classification_report(y_test, cnn_y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, cnn_y_pred))
print("\n" + "-"*50 + "\n")
```

Coding Continued

```python
# CNN Modelinin sonuçlarını diğer ML algoritmaları ile kıyaslamak için 3 farklı algoritma tanımlıyoruz
models = {
    "Random Forest": RandomForestClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "k-NN": KNeighborsClassifier()
}

# Diğer modelleri tek tek eğitip score ları hesaplıyoruz
for model_name, model in models.items():
    print(f"Training {model_name}...")
    model.fit(X_train_flat, y_train)
    y_pred = model.predict(X_test_flat)

    report = classification_report(y_test, y_pred, output_dict=True)
    accuracy = report["accuracy"]
    precision = np.mean([report[str(i)]["precision"] for i in range(10)])
    recall = np.mean([report[str(i)]["recall"] for i in range(10)])
    f1_score = np.mean([report[str(i)]["f1-score"] for i in range(10)])

    model_metrics = pd.DataFrame([{
        "Model": model_name,
        "Accuracy": accuracy,
        "Precision": precision,
        "Recall": recall,
        "F1-Score": f1_score
    }])
```

Coding Continued

```
    metrics = pd.concat([metrics, model_metrics], ignore_index=True)

    print(f"\nPerformance for {model_name}:")
    print(classification_report(y_test, y_pred))
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))
    print("\n" + "-"*50 + "\n")

# Performans skorlarını birbirleriyle kıyaslamak için tabloyu yazdırıyoruz
print(metrics)

# Performans skorlarını görselleştiriyoruz
metrics.set_index("Model", inplace=True)

fig, axs = plt.subplots(2, 2, figsize=(15, 10))

metrics["Accuracy"].plot(kind="bar", ax=axs[0, 0], color='blue', title="Accuracy")
metrics["Precision"].plot(kind="bar", ax=axs[0, 1], color='green', title="Precision")
metrics["Recall"].plot(kind="bar", ax=axs[1, 0], color='red', title="Recall")
metrics["F1-Score"].plot(kind="bar", ax=axs[1, 1], color='yellow', title="F1-Score")

for ax in axs.flat:
    ax.set_ylim(0, 1)
    ax.set_xlabel("Model")
    ax.set_ylabel("Score")
    ax.grid(True)

plt.tight_layout()
plt.show()
```

Training CNN

```
Training CNN...
Epoch 1/10
1563/1563 - 73s - loss: 1.4004 - accuracy: 0.5006 - val_loss: 1.1772 - val_accuracy: 0.5781 - 73s/epoch - 47ms/step
Epoch 2/10
1563/1563 - 69s - loss: 1.0460 - accuracy: 0.6352 - val_loss: 1.1462 - val_accuracy: 0.6031 - 69s/epoch - 44ms/step
Epoch 3/10
1563/1563 - 68s - loss: 0.9123 - accuracy: 0.6817 - val_loss: 1.0072 - val_accuracy: 0.6462 - 68s/epoch - 44ms/step
Epoch 4/10
1563/1563 - 68s - loss: 0.8151 - accuracy: 0.7163 - val_loss: 0.9099 - val_accuracy: 0.6868 - 68s/epoch - 43ms/step
Epoch 5/10
1563/1563 - 69s - loss: 0.7428 - accuracy: 0.7400 - val_loss: 0.8761 - val_accuracy: 0.6992 - 69s/epoch - 44ms/step
Epoch 6/10
1563/1563 - 68s - loss: 0.6725 - accuracy: 0.7649 - val_loss: 0.8928 - val_accuracy: 0.6989 - 68s/epoch - 44ms/step
Epoch 7/10
1563/1563 - 70s - loss: 0.6133 - accuracy: 0.7840 - val_loss: 0.9014 - val_accuracy: 0.7037 - 70s/epoch - 44ms/step
Epoch 8/10
1563/1563 - 72s - loss: 0.5526 - accuracy: 0.8071 - val_loss: 0.9112 - val_accuracy: 0.7018 - 72s/epoch - 46ms/step
Epoch 9/10
1563/1563 - 68s - loss: 0.4963 - accuracy: 0.8248 - val_loss: 0.9574 - val_accuracy: 0.6948 - 68s/epoch - 44ms/step
Epoch 10/10
1563/1563 - 66s - loss: 0.4462 - accuracy: 0.8450 - val_loss: 0.9859 - val_accuracy: 0.7029 - 66s/epoch - 42ms/step
313/313 [==============================] - 4s 13ms/step
```

Performance for CNN

```
Performance for CNN:
              precision    recall  f1-score   support

           0       0.76      0.76      0.76      1000
           1       0.86      0.77      0.81      1000
           2       0.65      0.54      0.59      1000
           3       0.48      0.51      0.49      1000
           4       0.73      0.56      0.64      1000
           5       0.59      0.63      0.61      1000
           6       0.74      0.81      0.77      1000
           7       0.77      0.75      0.76      1000
           8       0.74      0.86      0.79      1000
           9       0.74      0.83      0.79      1000

    accuracy                           0.70     10000
   macro avg       0.71      0.70      0.70     10000
weighted avg       0.71      0.70      0.70     10000

Confusion Matrix:
[[759  13  34  29   9   8  10   7  93  38]
 [ 17 768   8  14   3   7  10   2  48 123]
 [ 69   7 540  88  62  72  83  37  27  15]
 [ 22   3  60 514  38 175  86  36  33  33]
 [ 22   7  71  98 563  73  55  80  23   8]
 [ 14   3  40 185  28 634  26  41  17  12]
 [  4   8  34  79  19  19 805   9  13  10]
 [ 20   5  28  44  40  77   7 752   6  21]
 [ 54  18   8  17   3   5   2   5 859  29]
 [ 18  60   9  12   2   5   7   9  43 835]]
```

Performance for RandomForest

```
Performance for Random Forest:
              precision    recall  f1-score   support

           0       0.55      0.58      0.56      1000
           1       0.53      0.54      0.54      1000
           2       0.37      0.32      0.34      1000
           3       0.34      0.28      0.31      1000
           4       0.39      0.40      0.39      1000
           5       0.41      0.39      0.40      1000
           6       0.47      0.55      0.50      1000
           7       0.50      0.45      0.47      1000
           8       0.58      0.61      0.60      1000
           9       0.47      0.55      0.51      1000

    accuracy                           0.47     10000
   macro avg       0.46      0.47      0.46     10000
weighted avg       0.46      0.47      0.46     10000

Confusion Matrix:
[[577  35  42  20  32  18  26  25 167  58]
 [ 33 543  15  34  18  29  45  36  56 191]
 [102  36 317  76 154  81 116  66  24  28]
 [ 47  39  84 284  82 176 134  63  23  68]
 [ 54  18 150  62 395  45 145  88  24  19]
 [ 37  27  90 152  82 390  76  82  27  37]
 [ 11  36  80  80 114  56 550  32   6  35]
 [ 56  44  38  58 107  88  48 446  24  91]
 [ 80  85  18  32  19  36  14  22 615  79]
 [ 48 157  14  42  20  21  28  38  87 545]]
```

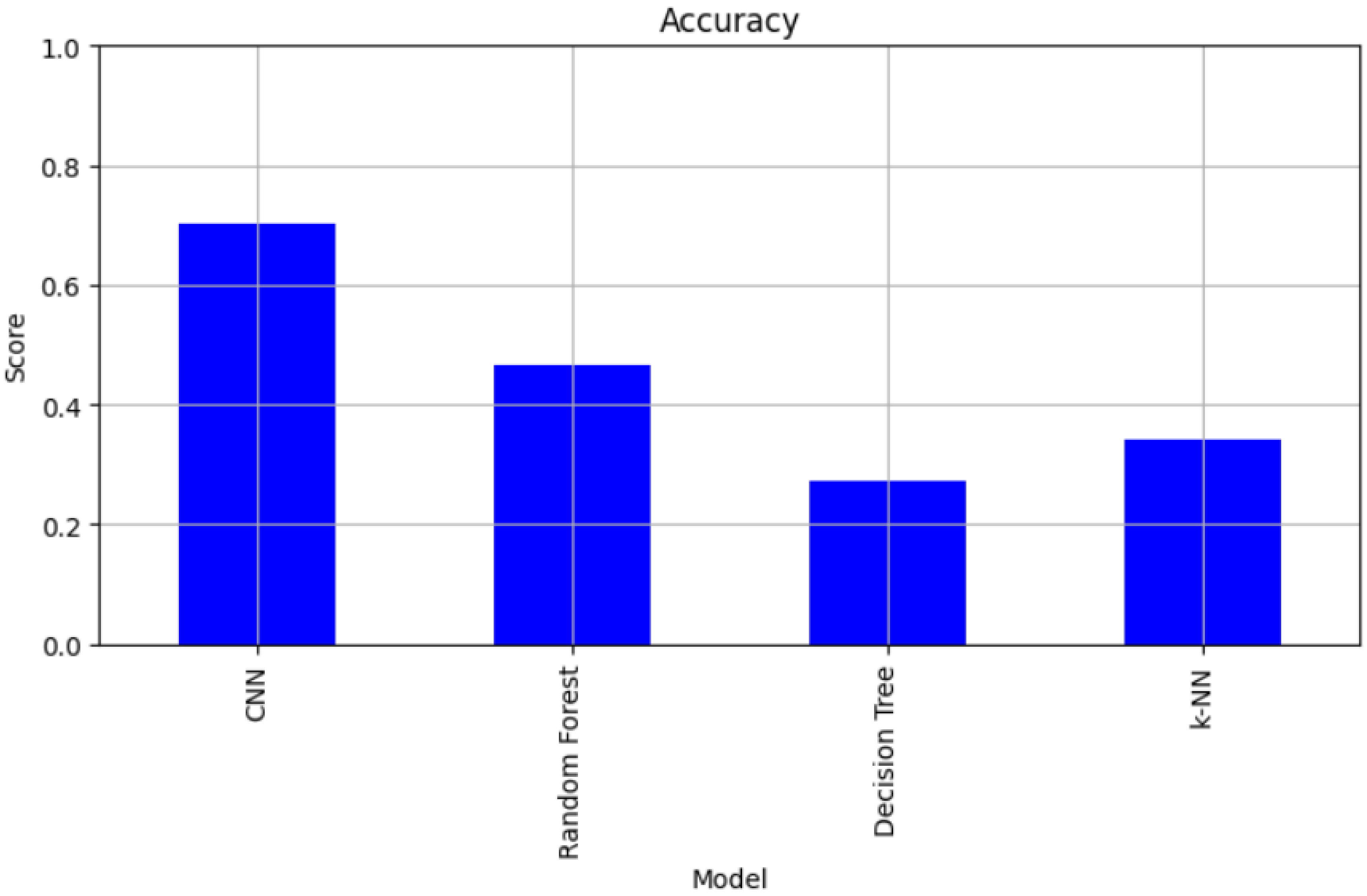
Performance for DecisionTree

```
Performance for Decision Tree:
              precision    recall  f1-score   support

           0       0.34      0.36      0.35      1000
           1       0.29      0.27      0.28      1000
           2       0.21      0.22      0.22      1000
           3       0.19      0.18      0.18      1000
           4       0.22      0.23      0.22      1000
           5       0.23      0.22      0.22      1000
           6       0.29      0.29      0.29      1000
           7       0.27      0.26      0.27      1000
           8       0.38      0.40      0.39      1000
           9       0.29      0.28      0.28      1000

    accuracy                           0.27     10000
   macro avg       0.27      0.27      0.27     10000
weighted avg       0.27      0.27      0.27     10000

Confusion Matrix:
[[356  66  84  59  60  46  38  60 151  80]
 [ 78 272  65  63  71  53  56  64 108 170]
 [ 90  52 224  88 144 109 118  85  41  49]
 [ 66  58 117 182 106 130 129  89  61  62]
 [ 66  40 154  89 229 108 114 109  47  44]
 [ 61  49  98 156  91 219 109 103  65  49]
 [ 40  49 127 119 145  87 291  64  30  48]
 [ 73  68  86  95 105  98  68 262  55  90]
 [132 110  52  53  43  40  28  50 401  91]
 [ 80 168  54  78  46  58  47  85 107 277]]
```

Performance for k-NN

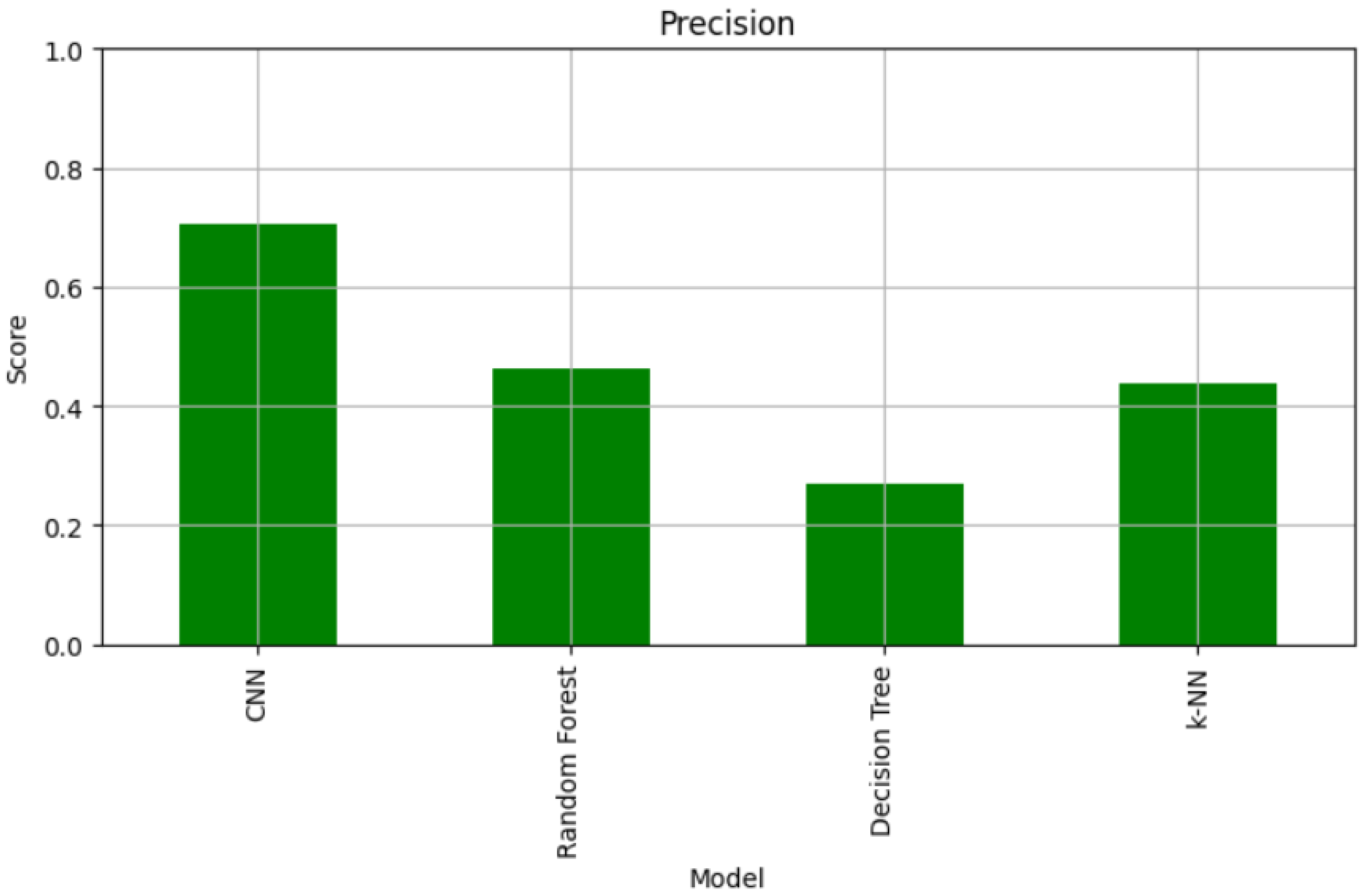
```
Performance for k-NN:
              precision    recall  f1-score   support

           0       0.38      0.54      0.45      1000
           1       0.67      0.21      0.32      1000
           2       0.22      0.44      0.29      1000
           3       0.30      0.23      0.26      1000
           4       0.25      0.52      0.33      1000
           5       0.39      0.22      0.28      1000
           6       0.36      0.26      0.30      1000
           7       0.69      0.22      0.33      1000
           8       0.40      0.66      0.50      1000
           9       0.73      0.13      0.23      1000

    accuracy                           0.34     10000
   macro avg       0.44      0.34      0.33     10000
weighted avg       0.44      0.34      0.33     10000

Confusion Matrix:
[[539   5 111  16  60   5  25   6 231   2]
 [139 209 109  45 150  34  54   7 231  22]
 [107   3 437  53 236  37  60  11  52   4]
 [ 66   6 242 225 183 109 103  13  50   3]
 [ 70   2 255  40 521  18  35  13  44   2]
 [ 76   4 228 150 191 219  68  13  46   5]
 [ 27   3 265  68 311  40 259   1  25   1]
 [ 91   7 190  46 273  53  59 218  59   4]
 [144  14  42  40  63  14  11   9 656   7]
 [155  61 108  62 125  26  51  25 253 134]]
```

Scores / Comparison



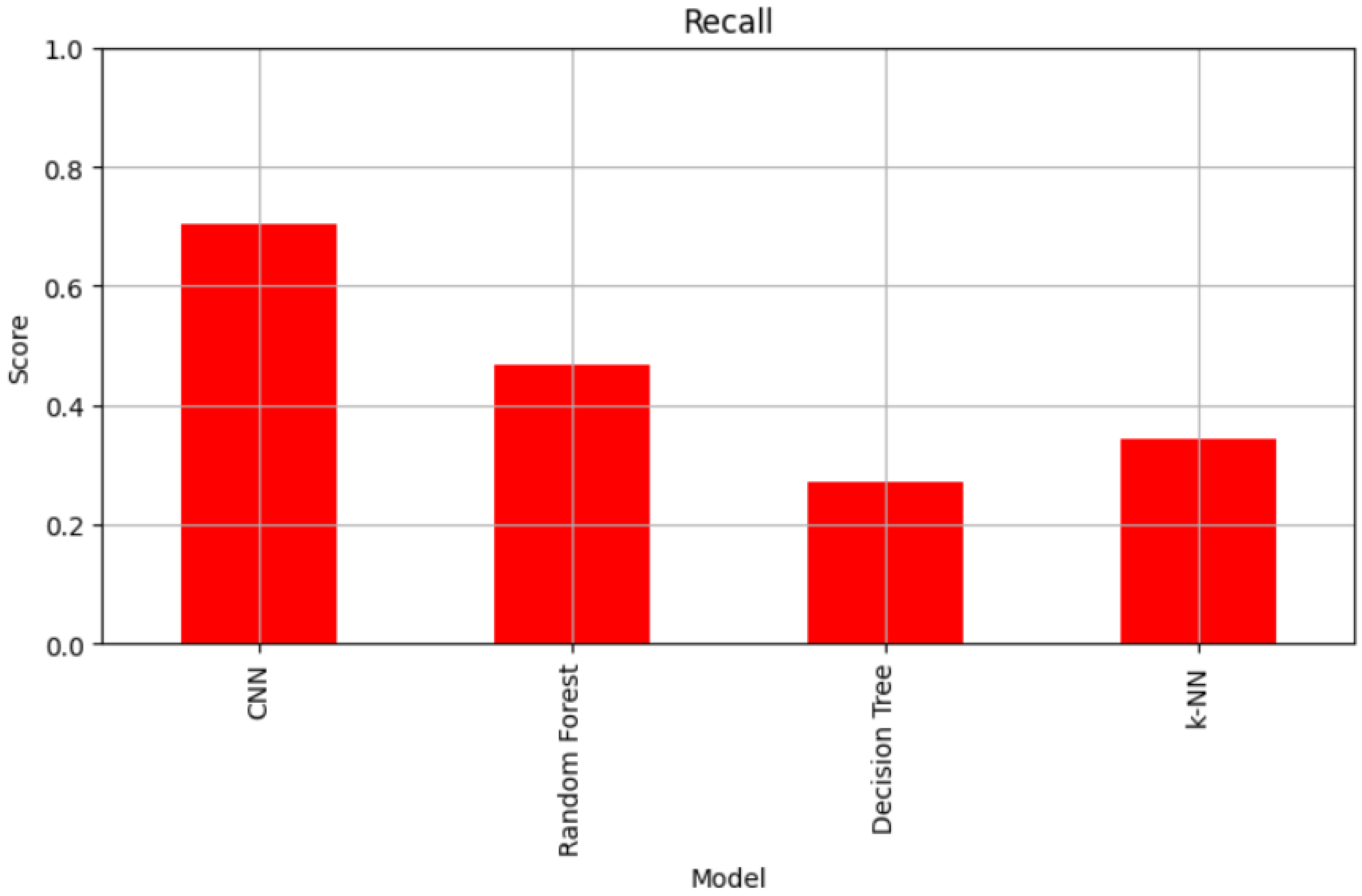
CNN has the best *accuracy* comparing to other ML algorithms

Scores / Comparison



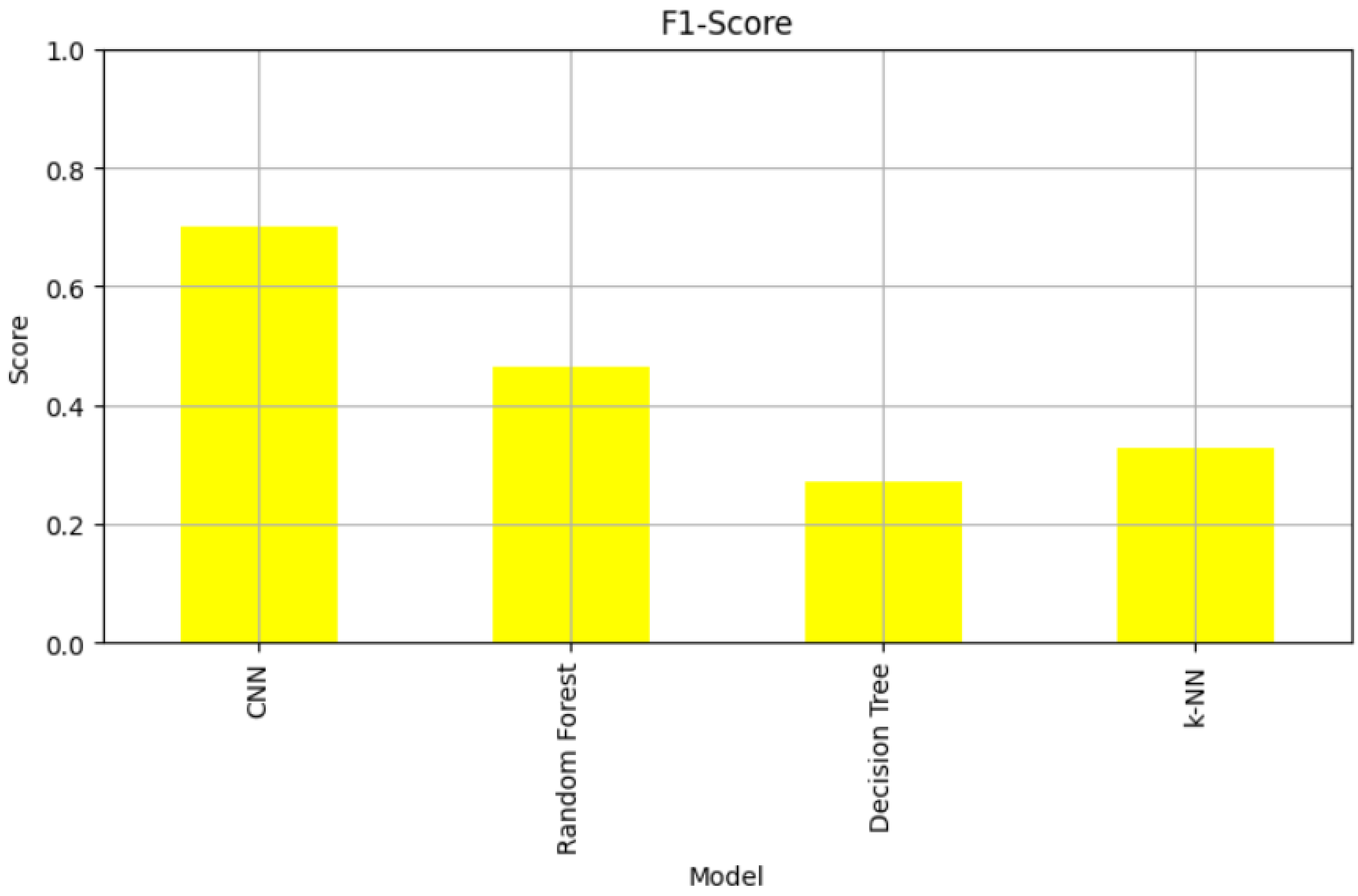
CNN has the best <u>*precision*</u> comparing to other ML algorithms

Scores / Comparison



CNN has the best *recall* comparing to other ML algorithms

Scores / Comparison



CNN has the best *F1-Score* comparing to other ML algorithms

Comments

In final words, CNN has better scores comparing to other ML algorithms for CIFAR10 image dataset analysis

So what can we do to have a better accuracy?

Increasing *number of epochs* gives us better accuracy for train/test data as seen in chart below