```
import tensorflow as tf

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
#IMPLEMENTATION OF KNN ON MNIST DATASET
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Flatten the images
x_train_flat = x_train.reshape(x_train.shape[0], -1)
x_test_flat = x_test.reshape(x_test.shape[0], -1)

# Create a KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)  # You can adjust the number of neighbors

# Train the classifier
knn.fit(x_train_flat, y_train)

# Make predictions on the test set
y_pred_knn = knn.predict(x_test_flat)

# Evaluate the accuracy
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print("Accuracy:", accuracy_knn)
```

```
Accuracy: 0.9688
```

```
from sklearn.metrics import classification_report, confusion_matrix

# Generate the classification report
print(classification_report(y_test, y_pred_knn))

# Generate the confusion matrix
print(confusion_matrix(y_test, y_pred_knn))
```

```
              precision    recall  f1-score   support

           0       0.96      0.99      0.98       980
           1       0.95      1.00      0.98      1135
           2       0.98      0.96      0.97      1032
           3       0.96      0.97      0.97      1010
           4       0.98      0.96      0.97       982
           5       0.97      0.97      0.97       892
           6       0.98      0.99      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.99      0.94      0.96       974
           9       0.96      0.95      0.95      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000

[[ 974    1    1    0    0    1    2    1    0    0]
 [   0 1133    2    0    0    0    0    0    0    0]
 [  11    8  991    2    1    0    1   15    3    0]
 [   0    3    3  976    1   13    1    6    3    4]
 [   3    7    0    0  944    0    4    2    1   21]
 [   5    0    0   12    2  862    4    1    2    4]
 [   5    3    0    0    3    2  945    0    0    0]
 [   0   22    4    0    3    0    0  988    0   11]
 [   8    3    5   13    6   12    5    5  913    4]
 [   5    7    3    9    7    3    1   10    2  962]]
```

```
from sklearn.metrics import classification_report, confusion_matrix

# Generate the classification report and store the result
report = classification_report(y_test, y_pred_knn, output_dict=True)

# Store precision, recall, and F1-score in variables
precision_knn = report['macro avg']['precision']
recall_knn = report['macro avg']['recall']
f1_score_knn = report['macro avg']['f1-score']

print("Precision:", precision_knn)
```

```
print("Recall:", recall_knn)
print("F1-score:", f1_score_knn)

# Generate the confusion matrix
print(confusion_matrix(y_test, y_pred_knn))
```

```
Precision: 0.9692753386570571
Recall: 0.9684705010297703
F1-score: 0.9687143421292884
[[ 974    1    1    0    0    1    2    1    0    0]
 [   0 1133    2    0    0    0    0    0    0    0]
 [  11    8  991    2    1    0    1   15    3    0]
 [   0    3    3  976    1   13    1    6    3    4]
 [   3    7    0    0  944    0    4    2    1   21]
 [   5    0    0   12    2  862    4    1    2    4]
 [   5    3    0    0    3    2  945    0    0    0]
 [   0   22    4    0    3    0    0  988    0   11]
 [   8    3    5   13    6   12    5    5  913    4]
 [   5    7    3    9    7    3    1   10    2  962]]
```

```
#IMPLEMENTATION OF ANN ON MNIST DATASET
import tensorflow
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras import layers, models
(x_train, y_train), (x_test, y_test) = tensorflow.keras.datasets.mnist.load_data()

# Define the ANN model
model = models.Sequential()
model.add(layers.Flatten(input_shape=(28, 28)))
model.add(layers.Dense(256, activation='relu'))  # Hidden layer 1
model.add(layers.Dense(128, activation='relu'))  # Hidden layer 2
model.add(layers.Dense(64, activation='relu'))   # Hidden layer 3
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()

# Train the model
history_ann = model.fit(x_train,
                    y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2,
                    verbose=2)

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)

# Make predictions on the test set
y_pred_ANN = model.predict(x_test)
y_pred_ANN_classes = np.argmax(y_pred_ANN, axis=1)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_c
  super().__init__(**kwargs)
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
flatten_2 (Flatten)	(None, 784)	0
dense_7 (Dense)	(None, 256)	200,960
dense_8 (Dense)	(None, 128)	32,896
dense_9 (Dense)	(None, 64)	8,256
dense_10 (Dense)	(None, 32)	2,080
dense_11 (Dense)	(None, 10)	330

```
 Total params: 244,522 (955.16 KB)
 Trainable params: 244,522 (955.16 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/10
1500/1500 - 12s - 8ms/step - accuracy: 0.7970 - loss: 0.8430 - val_accuracy: 0.9229 - val_loss: 0.2982
Epoch 2/10
1500/1500 - 21s - 14ms/step - accuracy: 0.9322 - loss: 0.2580 - val_accuracy: 0.9452 - val_loss: 0.2147
Epoch 3/10
1500/1500 - 15s - 10ms/step - accuracy: 0.9489 - loss: 0.1881 - val_accuracy: 0.9487 - val_loss: 0.1976
Epoch 4/10
1500/1500 - 19s - 13ms/step - accuracy: 0.9588 - loss: 0.1506 - val_accuracy: 0.9611 - val_loss: 0.1509
Epoch 5/10
1500/1500 - 13s - 9ms/step - accuracy: 0.9653 - loss: 0.1250 - val_accuracy: 0.9559 - val_loss: 0.1633
Epoch 6/10
1500/1500 - 9s - 6ms/step - accuracy: 0.9711 - loss: 0.1046 - val_accuracy: 0.9578 - val_loss: 0.1713
Epoch 7/10
1500/1500 - 9s - 6ms/step - accuracy: 0.9753 - loss: 0.0910 - val_accuracy: 0.9665 - val_loss: 0.1404
Epoch 8/10
1500/1500 - 10s - 7ms/step - accuracy: 0.9779 - loss: 0.0846 - val_accuracy: 0.9712 - val_loss: 0.1200
Epoch 9/10
1500/1500 - 10s - 6ms/step - accuracy: 0.9815 - loss: 0.0711 - val_accuracy: 0.9685 - val_loss: 0.1339
Epoch 10/10
1500/1500 - 8s - 5ms/step - accuracy: 0.9828 - loss: 0.0660 - val_accuracy: 0.9672 - val_loss: 0.1240
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9634 - loss: 0.1506
Test accuracy: 0.9686999917030334
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
```

```
# Generate the classification report and store the result
report_ann = classification_report(y_test, y_pred_ANN_classes, output_dict=True) # Use predicted classes instead of probabilities

# Store precision, recall, and F1-score in variables
precision_ann = report_ann['macro avg']['precision']
recall_ann = report_ann['macro avg']['recall']
f1_score_ann = report_ann['macro avg']['f1-score']
accuracy_ann = test_acc  # Assuming test_acc is the accuracy from model.evaluate

print("ANN Precision:", precision_ann)
print("ANN Recall:", recall_ann)
print("ANN F1-score:", f1_score_ann)
print("ANN Accuracy:", accuracy_ann)
```

```
ANN Precision: 0.9698358994567979
ANN Recall: 0.9693013081111405
ANN F1-score: 0.9693865758462759
ANN Accuracy: 0.9697999954223633
```

```
#IMPLEMENTATION OF SUPPORT VECTOR MACHINE CLASSIFIER ON MNIST DATASET
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix

# Load the MNIST dataset
print("Loading MNIST dataset...")
mnist = fetch_openml('mnist_784')

# Extract the data and target values
X, y = mnist['data'], mnist['target']

# Convert target to integers (if needed)
y = y.astype(np.int8)
```

```
# Normalize the data (pixel values are from 0 to 255, we scale them to 0 to 1)
X = X / 255.0

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Optionally, standardize the features (SVM performs better with standardized data)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the SVM classifier
svm_classifier = SVC(kernel='linear', random_state=42)  # RBF kernel is commonly used for SVMs, but you can try others like 'linear'

# Train the SVM classifier
print("Training SVM classifier...")
svm_classifier.fit(X_train_scaled, y_train)

# Make predictions on the test set
print("Predicting on the test set...")
y_pred_svm = svm_classifier.predict(X_test_scaled)

# Evaluate the classifier
print("Evaluation Results:")
print(confusion_matrix(y_test, y_pred_svm))
print(classification_report(y_test, y_pred_svm))
```

```
Loading MNIST dataset...
/usr/local/lib/python3.10/dist-packages/sklearn/datasets/_openml.py:1022: FutureWarning: The default value of `parser` will change 1
  warn(
Training SVM classifier...
Predicting on the test set...
Evaluation Results:
[[1299    1    4    1    3   14   10    2    7    2]
 [   0 1563    5    7    1    3    0    5   12    4]
 [   8   13 1269   22   16    8   10    7   22    5]
 [   3    3   35 1303    3   46    1    9   20   10]
 [   5    3   14    2 1215    4    5    6    3   38]
 [  12   10   12   52    9 1125   21    1   20   11]
 [  11    3   29    2   17   31 1300    1    2    0]
 [   1    6   24   13   23    7    0 1405    2   22]
 [  15   26   27   54    6   45   10   10 1151   13]
 [   7   11   10   18   50    8    0   40   11 1265]]
              precision    recall  f1-score   support

           0       0.95      0.97      0.96      1343
           1       0.95      0.98      0.97      1600
           2       0.89      0.92      0.90      1380
           3       0.88      0.91      0.90      1433
           4       0.90      0.94      0.92      1295
           5       0.87      0.88      0.88      1273
           6       0.96      0.93      0.94      1396
           7       0.95      0.93      0.94      1503
           8       0.92      0.85      0.88      1357
           9       0.92      0.89      0.91      1420

    accuracy                           0.92     14000
   macro avg       0.92      0.92      0.92     14000
weighted avg       0.92      0.92      0.92     14000
```

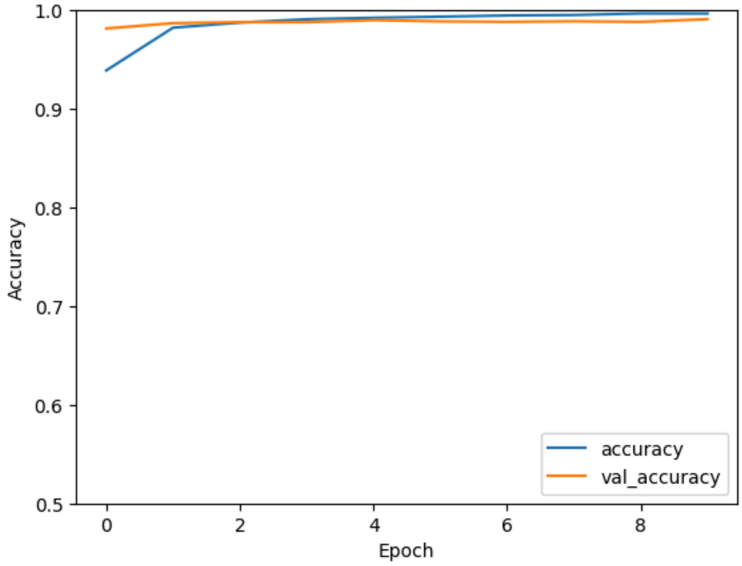
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Calculate accuracy
accuracy_svm = accuracy_score(y_test, y_pred_svm)

# Calculate precision (macro-averaged)
precision_svm = precision_score(y_test, y_pred_svm, average='macro')

# Calculate recall (macro-averaged)
recall_svm = recall_score(y_test, y_pred_svm, average='macro')

# Calculate F1-score (macro-averaged)
f1_svm = f1_score(y_test, y_pred_svm, average='macro')

print("SVM Accuracy:", accuracy_svm)
print("SVM Precision:", precision_svm)
print("SVM Recall:", recall_svm)
print("SVM F1-score:", f1_svm)
```

```
SVM Accuracy: 0.9210714285714285
SVM Precision: 0.9203851248826431
```

```
SVM Recall: 0.9199990891902636
SVM F1-score: 0.9198935548264234
```

```
# IMPLEMENTATION OF CONVOLUTIONAL NEURAL NETWORKS
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Preprocess the data
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

# Split the training data into training and validation sets
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=42)

# Define the CNN model
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(x_train, y_train, epochs=10, batch_size=64,
                    validation_data=(x_val, y_val))

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)

# Plot the training and validation accuracy
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 58s 74ms/step - accuracy: 0.8590 - loss: 0.4650 - val_accuracy: 0.9815 - val_loss: 0.0620
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 81s 73ms/step - accuracy: 0.9804 - loss: 0.0615 - val_accuracy: 0.9868 - val_loss: 0.0420
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 51s 69ms/step - accuracy: 0.9881 - loss: 0.0383 - val_accuracy: 0.9879 - val_loss: 0.0387
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 83s 70ms/step - accuracy: 0.9911 - loss: 0.0298 - val_accuracy: 0.9878 - val_loss: 0.0374
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 48s 64ms/step - accuracy: 0.9924 - loss: 0.0226 - val_accuracy: 0.9899 - val_loss: 0.0338
Epoch 6/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 82s 64ms/step - accuracy: 0.9937 - loss: 0.0184 - val_accuracy: 0.9887 - val_loss: 0.0376
Epoch 7/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 84s 66ms/step - accuracy: 0.9954 - loss: 0.0143 - val_accuracy: 0.9882 - val_loss: 0.0434
Epoch 8/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 82s 67ms/step - accuracy: 0.9964 - loss: 0.0111 - val_accuracy: 0.9887 - val_loss: 0.0387
Epoch 9/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 49s 65ms/step - accuracy: 0.9974 - loss: 0.0084 - val_accuracy: 0.9882 - val_loss: 0.0470
Epoch 10/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 83s 67ms/step - accuracy: 0.9969 - loss: 0.0088 - val_accuracy: 0.9909 - val_loss: 0.0350
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - accuracy: 0.9899 - loss: 0.0387
Test accuracy: 0.9919999837875366
```




```
accuracy = history.history['accuracy']
print(accuracy)
```

```
[0.9389166831970215, 0.9822708368301392, 0.9874374866485596, 0.9907916784286499, 0.9922916889190674, 0.9934375286102295, 0.99464583
```

```
import numpy as np
# Make predictions on the test set
y_pred_cnn = model.predict(x_test)
y_pred_cnn_classes = np.argmax(y_pred_cnn, axis=1)

# Generate the confusion matrix
confusion_cnn = confusion_matrix(y_test, y_pred_cnn_classes)
print("CNN Confusion Matrix:\n", confusion_cnn)

# Generate the classification report
report_cnn = classification_report(y_test, y_pred_cnn_classes)
print("CNN Classification Report:\n", report_cnn)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step
CNN Confusion Matrix:
 [[ 976    1    1    0    0    0    0    1    1    0]
 [   0 1126    0    1    0    1    2    1    4    0]
 [   0    3 1024    0    1    0    0    2    2    0]
 [   0    0    0 1006    0    4    0    0    0    0]
 [   0    0    0    0  973    0    2    0    0    7]
 [   1    0    0    5    0  883    1    1    0    1]
 [   4    1    0    1    2    5  943    0    2    0]
 [   0    5    0    1    0    0    0 1017    1    4]
 [   0    0    0    1    0    0    0    1  969    3]
 [   1    0    0    1    1    1    0    1    1 1003]]
CNN Classification Report:
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       980
           1       0.99      0.99      0.99      1135
           2       1.00      0.99      1.00      1032
           3       0.99      1.00      0.99      1010
           4       1.00      0.99      0.99       982
           5       0.99      0.99      0.99       892
           6       0.99      0.98      0.99       958
           7       0.99      0.99      0.99      1028
           8       0.99      0.99      0.99       974
           9       0.99      0.99      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

```
# Generate the classification report and store the result
report_cnn = classification_report(y_test, y_pred_cnn_classes, output_dict=True)

# Store precision, recall, and F1-score in variables
precision_cnn = report_cnn['macro avg']['precision']
recall_cnn = report_cnn['macro avg']['recall']
f1_score_cnn = report_cnn['macro avg']['f1-score']
accuracy_cnn = test_acc

print("CNN Precision:", precision_cnn)
print("CNN Recall:", recall_cnn)
print("CNN F1-score:", f1_score_cnn)
print("CNN Accuracy:", accuracy_cnn)
```

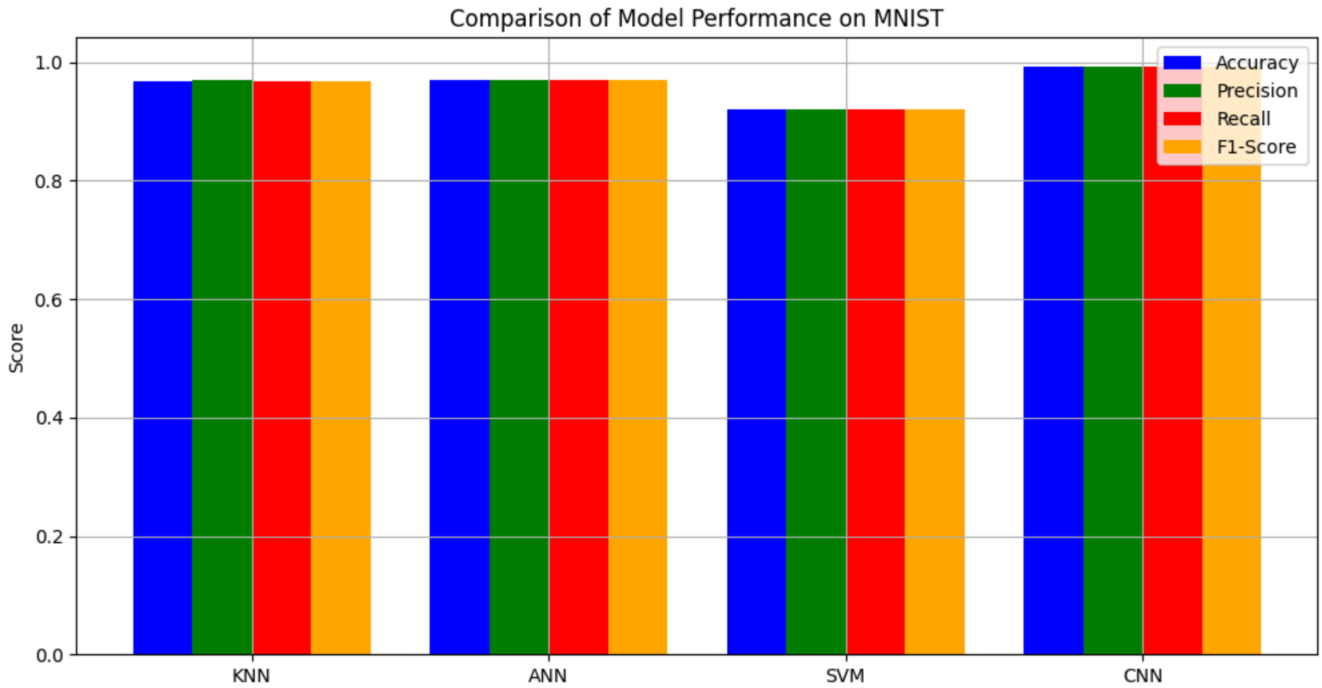
```
CNN Precision: 0.9919801194684963
CNN Recall: 0.991958390131788
CNN F1-score: 0.9919603503846947
CNN Accuracy: 0.9919999837875366
```

Generate | compare and plot accuracy, precision, recall anf f1 score of the above models | Close

< 1 of 1 > Undo Changes Use code with caution

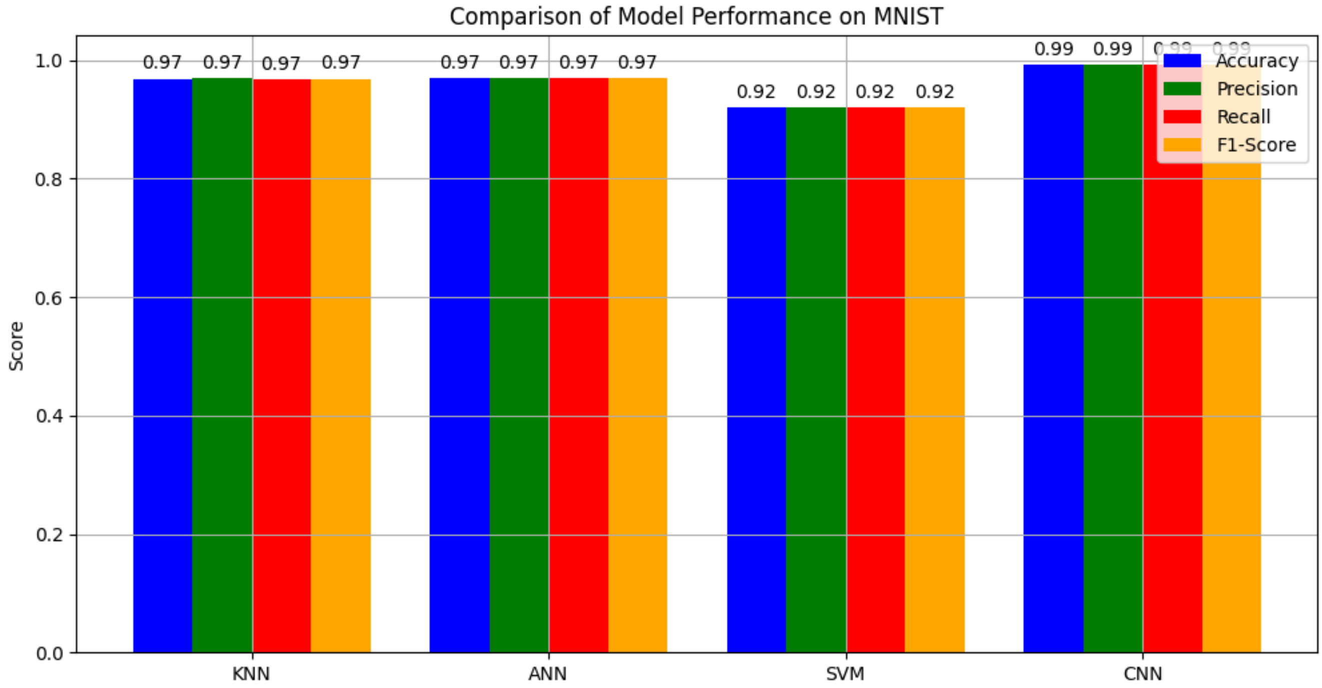
```
import matplotlib.pyplot as plt

# Data for the bar chart
models = ['KNN', 'ANN', 'SVM', 'CNN']
accuracies = [accuracy_knn, accuracy_ann, accuracy_svm, accuracy_cnn]
precisions = [precision_knn, precision_ann, precision_svm, precision_cnn]
recalls = [recall_knn, recall_ann, recall_svm, recall_cnn]
f1_scores = [f1_score_knn, f1_score_ann, f1_svm, f1_score_cnn]

# Set the width of the bars
bar_width = 0.2

# Create the figure and axes
fig, ax = plt.subplots(figsize=(12, 6))

# Calculate the x positions for the bars
x = range(len(models))
x_accuracy = [i - 1.5 * bar_width for i in x]
x_precision = [i - 0.5 * bar_width for i in x]
x_recall = [i + 0.5 * bar_width for i in x]
x_f1_score = [i + 1.5 * bar_width for i in x]

# Create the bar plots
ax.bar(x_accuracy, accuracies, width=bar_width, label='Accuracy', color='blue')
ax.bar(x_precision, precisions, width=bar_width, label='Precision', color='green')
ax.bar(x_recall, recalls, width=bar_width, label='Recall', color='red')
ax.bar(x_f1_score, f1_scores, width=bar_width, label='F1-Score', color='orange')

# Set the x-axis ticks and labels
ax.set_xticks(x)
ax.set_xticklabels(models)

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

# Set the title
ax.set_title('Comparison of Model Performance on MNIST')

# Add a legend
ax.legend()
```

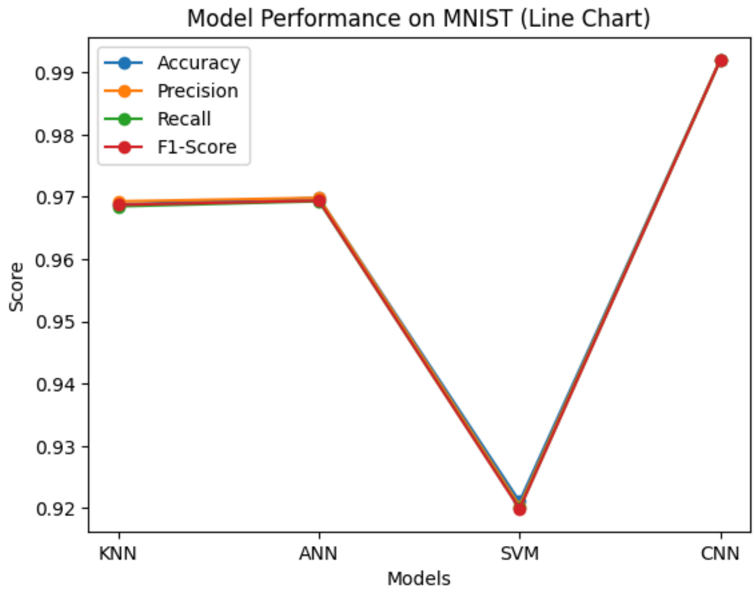
```
# Add gridlines
ax.grid(True)

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




```
import matplotlib.pyplot as plt

# Data for the bar chart
models = ['KNN', 'ANN', 'SVM', 'CNN']
accuracies = [accuracy_knn, accuracy_ann, accuracy_svm, accuracy_cnn]
precisions = [precision_knn, precision_ann, precision_svm, precision_cnn]
recalls = [recall_knn, recall_ann, recall_svm, recall_cnn]
f1_scores = [f1_score_knn, f1_score_ann, f1_svm, f1_score_cnn]

# Create the figure and axes
fig, ax = plt.subplots(figsize=(12, 6))

# Calculate the x positions for the bars
x = range(len(models))
x_accuracy = [i - 1.5 * bar_width for i in x]
x_precision = [i - 0.5 * bar_width for i in x]
x_recall = [i + 0.5 * bar_width for i in x]
x_f1_score = [i + 1.5 * bar_width for i in x]

# Create the bar plots
rects_accuracy = ax.bar(x_accuracy, accuracies, width=bar_width, label='Accuracy', color='blue')
rects_precision = ax.bar(x_precision, precisions, width=bar_width, label='Precision', color='green')
rects_recall = ax.bar(x_recall, recalls, width=bar_width, label='Recall', color='red')
rects_f1_score = ax.bar(x_f1_score, f1_scores, width=bar_width, label='F1-Score', color='orange')


# Add data labels to the bars
def autolabel(rects):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(round(height, 2)),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

autolabel(rects_accuracy)
autolabel(rects_precision)
autolabel(rects_recall)
autolabel(rects_f1_score)

# Set the x-axis ticks and labels
ax.set_xticks(x)
ax.set_xticklabels(models)
```

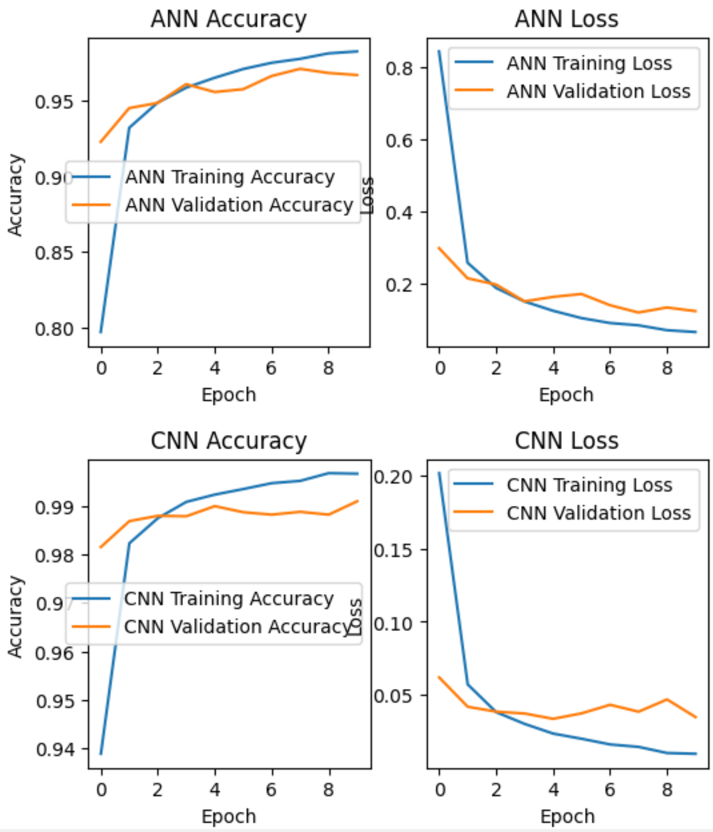
```
# Set the y-axis label
ax.set_ylabel('Score')

# Set the title
ax.set_title('Comparison of Model Performance on MNIST')

# Add a legend
ax.legend()

# Add gridlines
ax.grid(True)


# OTHER TYPE OF PLOT - LINE CHART

# Create a new figure and axes for the line chart
fig_line, ax_line = plt.subplots()

# Plot the accuracy scores for each model as a line chart
ax_line.plot(models, accuracies, marker='o', label='Accuracy')
ax_line.plot(models, precisions, marker='o', label='Precision')
ax_line.plot(models, recalls, marker='o', label='Recall')
ax_line.plot(models, f1_scores, marker='o', label='F1-Score')

# Add labels and title
ax_line.set_xlabel('Models')
ax_line.set_ylabel('Score')
ax_line.set_title('Model Performance on MNIST (Line Chart)')

# Add a legend
ax_line.legend()

# Show the line chart
plt.show()
```






```
import matplotlib.pyplot as plt

plt.figure(figsize=(6,3))

# Plot training and validation accuracy for ANN
plt.subplot(1, 2, 1)
plt.plot(history_ann.history['accuracy'], label='ANN Training Accuracy')
plt.plot(history_ann.history['val_accuracy'], label='ANN Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('ANN Accuracy')
plt.legend()

# Plot training and validation loss for ANN
plt.subplot(1, 2, 2)
plt.plot(history_ann.history['loss'], label='ANN Training Loss')
plt.plot(history_ann.history['val_loss'], label='ANN Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('ANN Loss')
plt.legend()

plt.show()


plt.figure(figsize=(6,3))

# Plot training and validation accuracy for CNN
```

```
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='CNN Training Accuracy')
plt.plot(history.history['val_accuracy'], label='CNN Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('CNN Accuracy')
plt.legend()

# Plot training and validation loss for CNN
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='CNN Training Loss')
plt.plot(history.history['val_loss'], label='CNN Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('CNN Loss')
plt.legend()

plt.show()
```




```
import matplotlib.pyplot as plt

# Extract accuracy values for ANN and CNN
ann_accuracy = history_ann.history['accuracy']
cnn_accuracy = history.history['accuracy']

# Create a range of epochs
epochs = range(1, len(ann_accuracy) + 1)

# Plot the bar graph
plt.figure(figsize=(10, 6))
plt.bar(epochs, ann_accuracy, width=0.4, label='ANN', align='center')
plt.bar([x + 0.4 for x in epochs], cnn_accuracy, width=0.4, label='CNN', align='center')

plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Epochs vs Accuracy (ANN vs CNN)')
plt.xticks(epochs)
plt.legend()
plt.show()
```