CNN

May 14, 2024

0.1 MBKMUN001- ML CNN Image Classification

0.1.1 Preprocessing

[50]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from sklearn.metrics import confusion_matrix,accuracy_score, precision_score,␣
 ↪recall_score, f1_score
```

[51]:
```
def load_images_from_folder(folder):
    images = []
    for filename in os.listdir(folder):
        img = Image.open(os.path.join(folder, filename))
        if img is not None:
            images.append(np.array(img))
    return images
X = []
y = []
Dataset = '/home/nathan/Documents/EEE4114F/MBKMUN001_ML_Project/Dataset'
for i, folder_name in enumerate(os.listdir(Dataset)):
    folder_path = os.path.join(Dataset, folder_name)
    images = load_images_from_folder(folder_path)
    X.extend(images)
    y.extend([i] * len(images))
X = np.array(X)
y = np.array(y)
```

0.1.2 Model Training and evaluation

[52]:
```
print(X.shape)
print(y.shape)
# Split dataset 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)

# Convert labels to one-hot encoding
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Create CNN model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(5, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_test,␣
 ↪y_test))

# Make predictions
y_pred = np.argmax(model.predict(X_test), axis=-1)
```

```
(375, 28, 28, 3)
(375,)
Epoch 1/20
10/10 [==============================] - 0s 21ms/step - loss: 68.9233 -
accuracy: 0.1633 - val_loss: 25.7333 - val_accuracy: 0.1333
Epoch 2/20
10/10 [==============================] - 0s 10ms/step - loss: 12.9519 -
accuracy: 0.2300 - val_loss: 6.4919 - val_accuracy: 0.2933
Epoch 3/20
10/10 [==============================] - 0s 10ms/step - loss: 3.3271 - accuracy:
0.3267 - val_loss: 1.6977 - val_accuracy: 0.3600
Epoch 4/20
10/10 [==============================] - 0s 9ms/step - loss: 1.3783 - accuracy:
0.4267 - val_loss: 1.2558 - val_accuracy: 0.5200
Epoch 5/20
10/10 [==============================] - 0s 10ms/step - loss: 1.1885 - accuracy:
```

```
0.5167 - val_loss: 1.3267 - val_accuracy: 0.3200
Epoch 6/20
10/10 [==============================] - 0s 10ms/step - loss: 1.2007 - accuracy:
0.4733 - val_loss: 1.1126 - val_accuracy: 0.5867
Epoch 7/20
10/10 [==============================] - 0s 10ms/step - loss: 0.9755 - accuracy:
0.5933 - val_loss: 0.8634 - val_accuracy: 0.6133
Epoch 8/20
10/10 [==============================] - 0s 10ms/step - loss: 0.9513 - accuracy:
0.6367 - val_loss: 0.7626 - val_accuracy: 0.6933
Epoch 9/20
10/10 [==============================] - 0s 12ms/step - loss: 0.7760 - accuracy:
0.7033 - val_loss: 0.6112 - val_accuracy: 0.7733
Epoch 10/20
10/10 [==============================] - 0s 10ms/step - loss: 0.6817 - accuracy:
0.7833 - val_loss: 0.5482 - val_accuracy: 0.8533
Epoch 11/20
10/10 [==============================] - 0s 11ms/step - loss: 0.5928 - accuracy:
0.7900 - val_loss: 0.4950 - val_accuracy: 0.8667
Epoch 12/20
10/10 [==============================] - 0s 10ms/step - loss: 0.5647 - accuracy:
0.8267 - val_loss: 0.5188 - val_accuracy: 0.8267
Epoch 13/20
10/10 [==============================] - 0s 10ms/step - loss: 0.5764 - accuracy:
0.8100 - val_loss: 0.3839 - val_accuracy: 0.8400
Epoch 14/20
10/10 [==============================] - 0s 11ms/step - loss: 0.5191 - accuracy:
0.8267 - val_loss: 0.3966 - val_accuracy: 0.8800
Epoch 15/20
10/10 [==============================] - 0s 12ms/step - loss: 0.4935 - accuracy:
0.8467 - val_loss: 0.3834 - val_accuracy: 0.8533
Epoch 16/20
10/10 [==============================] - 0s 11ms/step - loss: 0.4793 - accuracy:
0.8267 - val_loss: 0.3924 - val_accuracy: 0.8933
Epoch 17/20
10/10 [==============================] - 0s 11ms/step - loss: 0.4518 - accuracy:
0.8367 - val_loss: 0.6690 - val_accuracy: 0.7467
Epoch 18/20
10/10 [==============================] - 0s 11ms/step - loss: 0.4717 - accuracy:
0.8333 - val_loss: 0.3700 - val_accuracy: 0.8800
Epoch 19/20
10/10 [==============================] - 0s 11ms/step - loss: 0.4213 - accuracy:
0.8467 - val_loss: 0.4210 - val_accuracy: 0.8667
Epoch 20/20
10/10 [==============================] - 0s 10ms/step - loss: 0.3872 - accuracy:
0.8767 - val_loss: 0.2789 - val_accuracy: 0.9067
```

0.1.3 Accuracy

[53]:
```
accuracy = accuracy_score(np.argmax(y_test, axis=-1), y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.9066666666666666
```

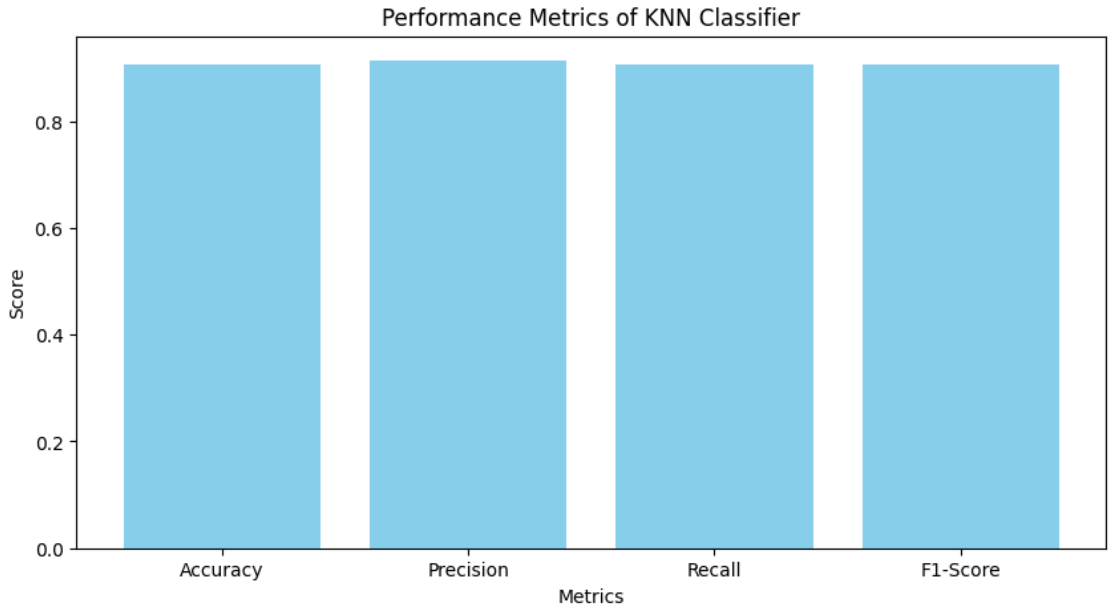
0.1.4 Precision, Recall, F1

[54]:
```
precision = precision_score(np.argmax(y_test, axis=-1), y_pred,␣
 ↪average='weighted')
recall = recall_score(np.argmax(y_test, axis=-1), y_pred, average='weighted')
f1 = f1_score(np.argmax(y_test, axis=-1), y_pred, average='weighted')


print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-Score: {f1}")
```

```
Precision: 0.9132731092436976
Recall: 0.9066666666666666
F1-Score: 0.9063377737904991
```

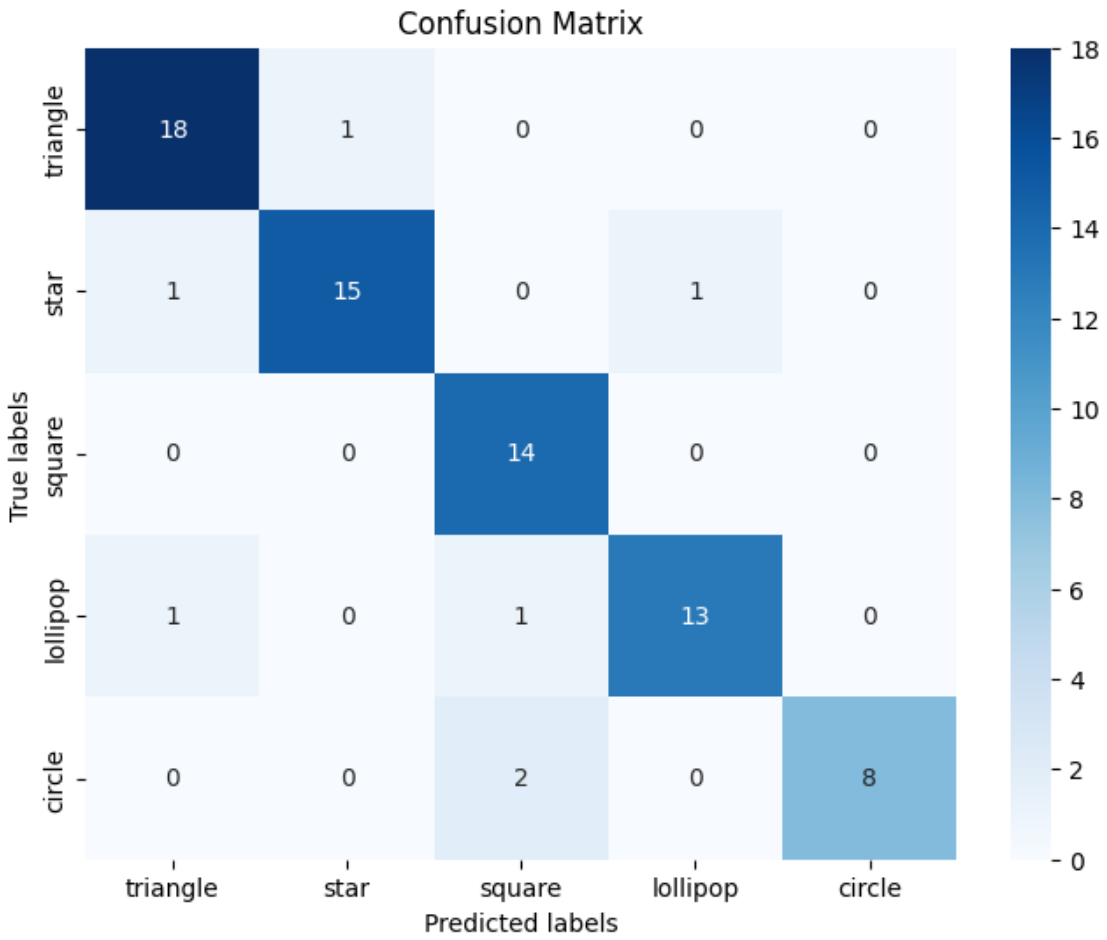
0.1.5 Classification Report

[55]:
```
metrics = ['Accuracy', 'Precision', 'Recall', 'F1-Score']
values = [accuracy, precision, recall, f1]

plt.figure(figsize=(10, 5))

# Plotting the metrics
plt.bar(metrics, values, color='skyblue')

# Adding labels and title
plt.xlabel('Metrics')
plt.ylabel('Score')
plt.title('Performance Metrics of KNN Classifier')

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



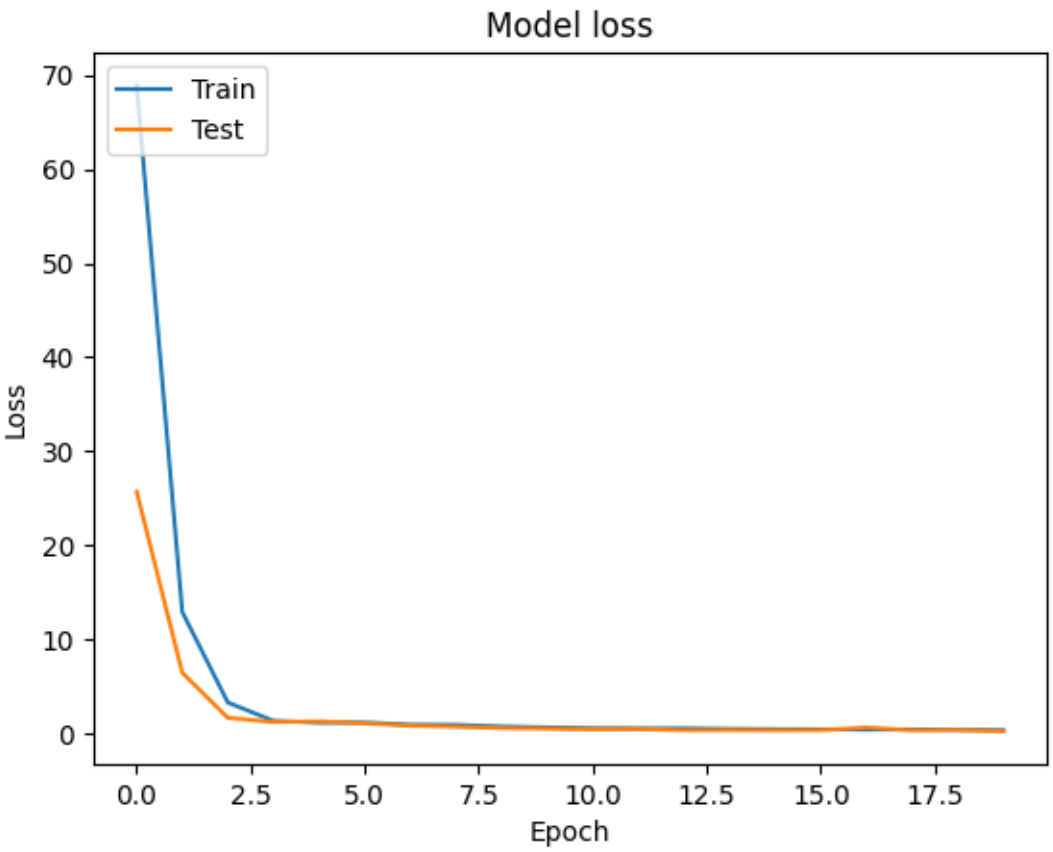
0.1.6 Confusion Matrix

[56]:
```
conf_matrix = confusion_matrix(np.argmax(y_test, axis=-1), y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=os.
 ↪listdir(Dataset), yticklabels=os.listdir(Dataset))
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```



0.1.7 Model Loss

[57]:
```
# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```


Model loss
Train
Test
Loss
Epoch
70
60
50
40
30
20
10
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5