ml-project

December 28, 2023

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import load_model

from sklearn.model_selection import train_test_split , StratifiedKFold
from sklearn.svm import SVC
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score
from sklearn.neighbors import KNeighborsClassifier
```

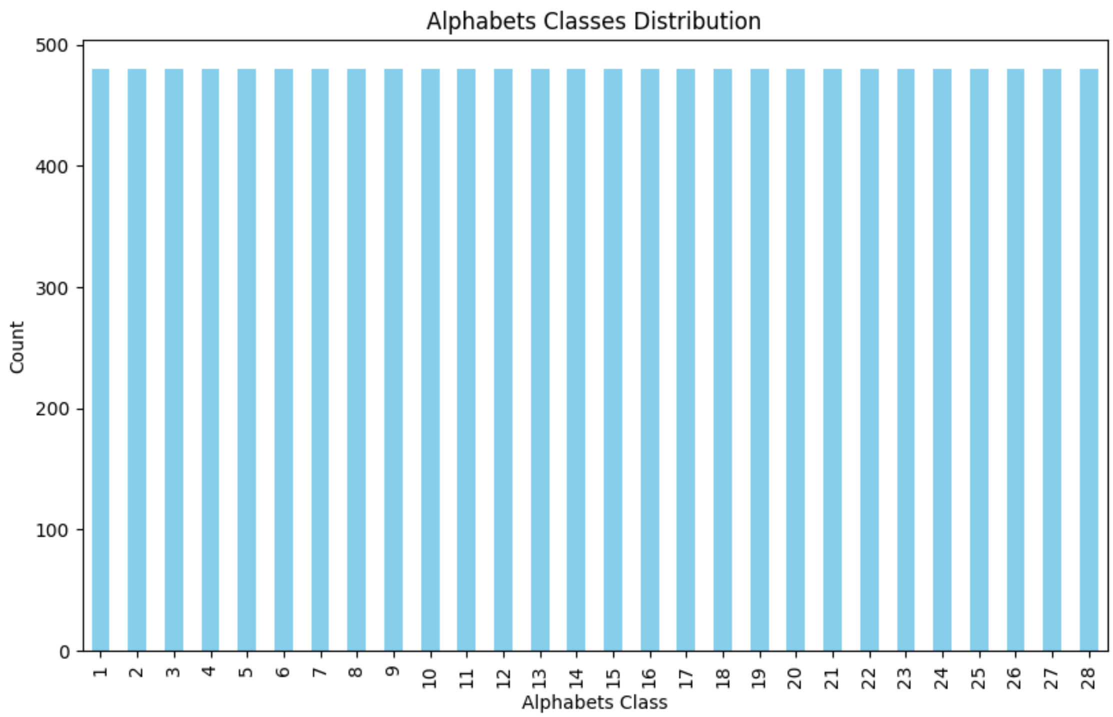
Data exploration and preparation

[2]:
```
df= pd.read_csv(r"C:\Users\Essam\Desktop\ML␣
 ↪Project\Arabic-Characters-Recognition\csvTrainImages 13440x1024.
 ↪csv",header=None)
df_labels=pd.read_csv(r"C:\Users\Essam\Desktop\ML␣
 ↪Project\Arabic-Characters-Recognition\csvTrainLabel 13440x1.csv",header=None␣
 ↪)
images = df.to_numpy()
normalized_images=images/255.

df_test = pd.read_csv(r"C:\Users\Essam\Desktop\ML␣
 ↪Project\Arabic-Characters-Recognition\csvTestImages 3360x1024.
 ↪csv",header=None)
df_labels_test = pd.read_csv(r"C:\Users\Essam\Desktop\ML␣
 ↪Project\Arabic-Characters-Recognition\csvTestLabel 3360x1.csv",header=None)
df_labels_test.to_numpy
images_t = df_test.to_numpy()
images_test=images_t/255.
```

```
[3]: unique_classes = df_labels[0].unique()
     print("Unique Classes:", unique_classes)
```

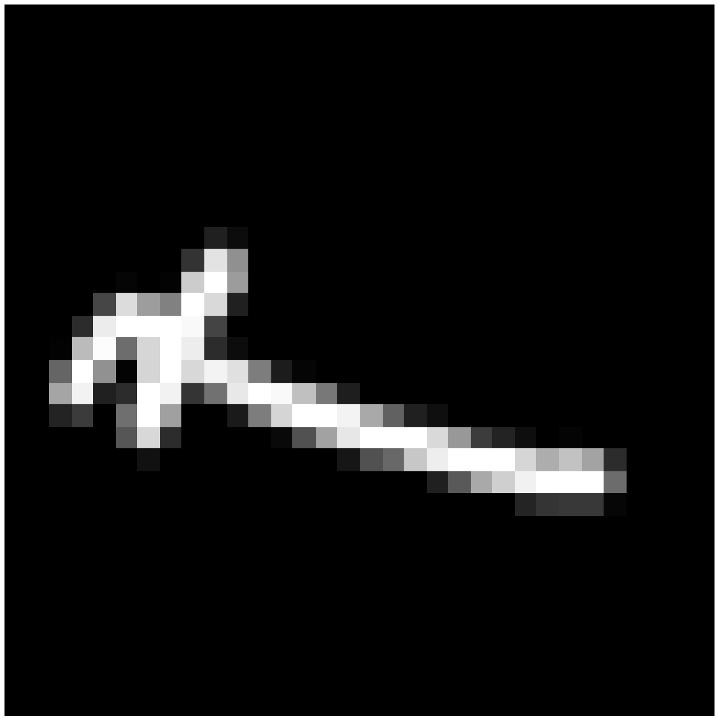
```
Unique Classes: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
22 23 24
 25 26 27 28]
```

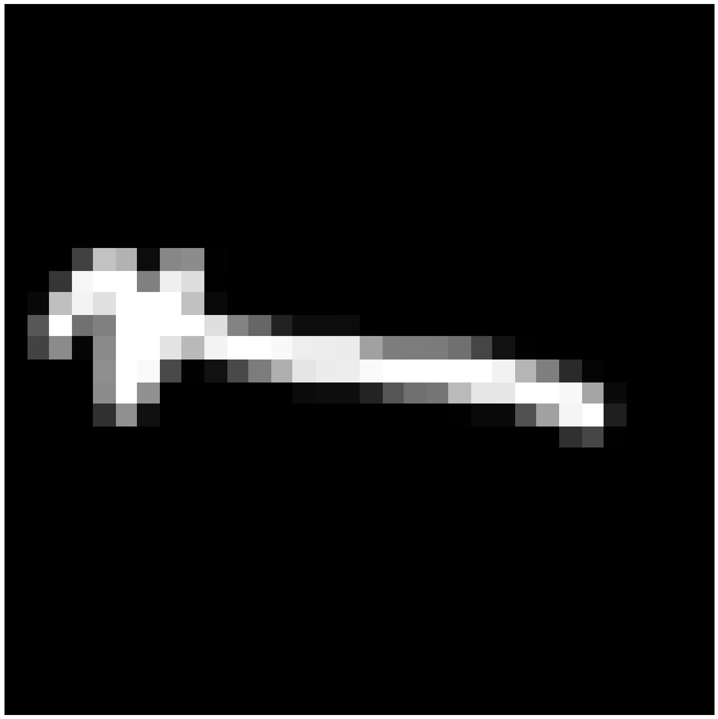
```
[4]: class_distribution = df_labels[0].value_counts()
     plt.figure(figsize=(10, 6))
     class_distribution = class_distribution.sort_index()
     class_distribution.plot(kind='bar', color='skyblue')
     plt.title('Alphabets Classes Distribution')
     plt.xlabel('Alphabets Class')
     plt.ylabel('Count')
     plt.show()
```

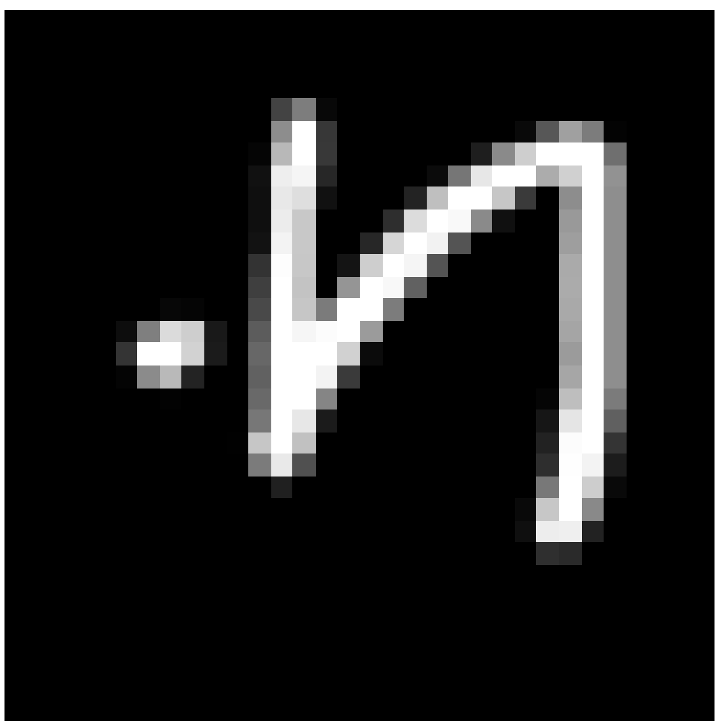


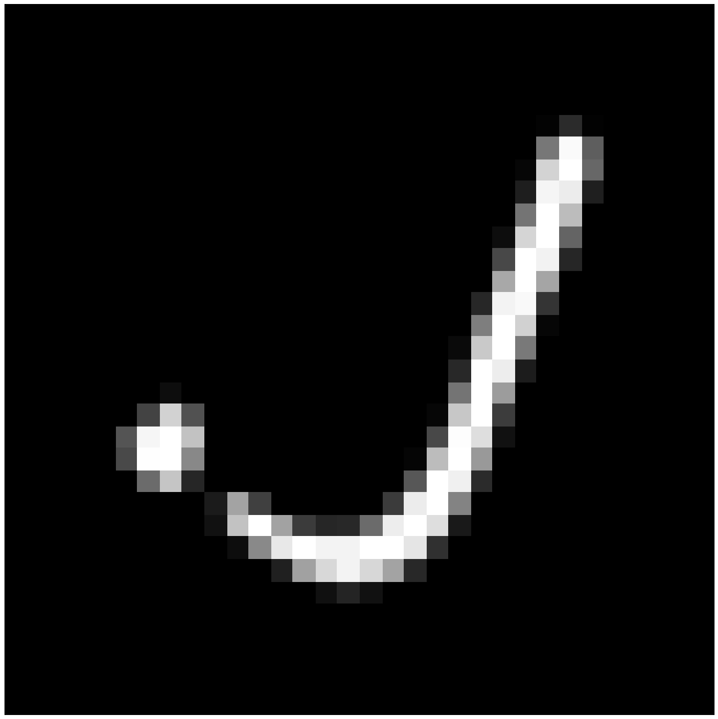

```
[5]: def display_image(image):
         image_size = 32
         images = image.reshape((image_size, image_size))
         plt.imshow(images, cmap='gray')
         plt.axis('off')
         plt.show()
```

[6]:
```
display_image(images[0])
display_image(images[3])
display_image(images[54])
display_image(images[80])
```









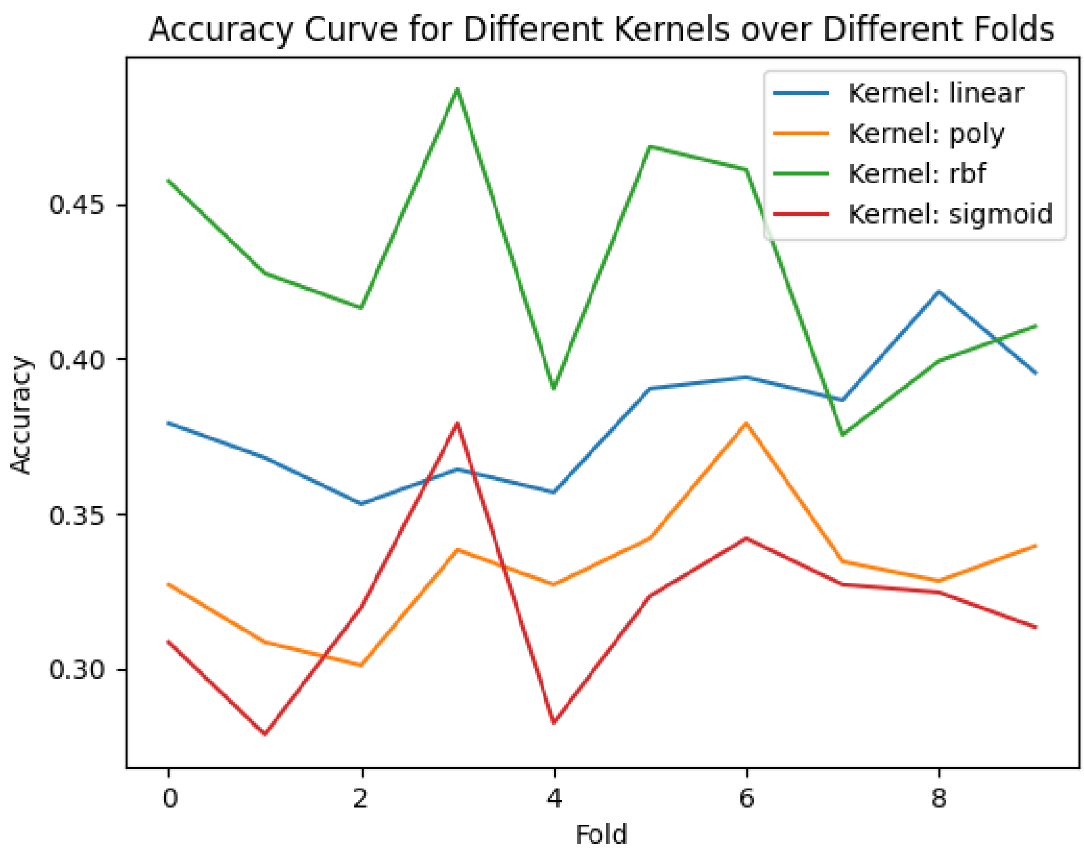
[7]:
```
df_labels = df_labels.to_numpy()
df_labels_test = df_labels_test.to_numpy()
print(df_labels.dtype)
print(df_labels_test.dtype)
```

```
int64
int64
```

First Experiment Support Vector Machine (SVM)

[8]:
```
X_train, X_test, y_train, y_test = train_test_split(normalized_images,␣
↪df_labels, test_size=0.2, random_state=42)
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
kernel_accuracies = {kernel: [] for kernel in kernels}
for train_index, val_index in kf.split(X_test, y_test):
    X_val_train, X_val_test = X_test[train_index], X_test[val_index]
    y_val_train, y_val_test = y_test[train_index], y_test[val_index]
    for kernel in kernels:
        svm_model = SVC(kernel=kernel)
        svm_model.fit(X_val_train, y_val_train.ravel())
        y_val_pred = svm_model.predict(X_val_test)
        accuracy = accuracy_score(y_val_test.ravel(), y_val_pred)
        kernel_accuracies[kernel].append(accuracy)
```

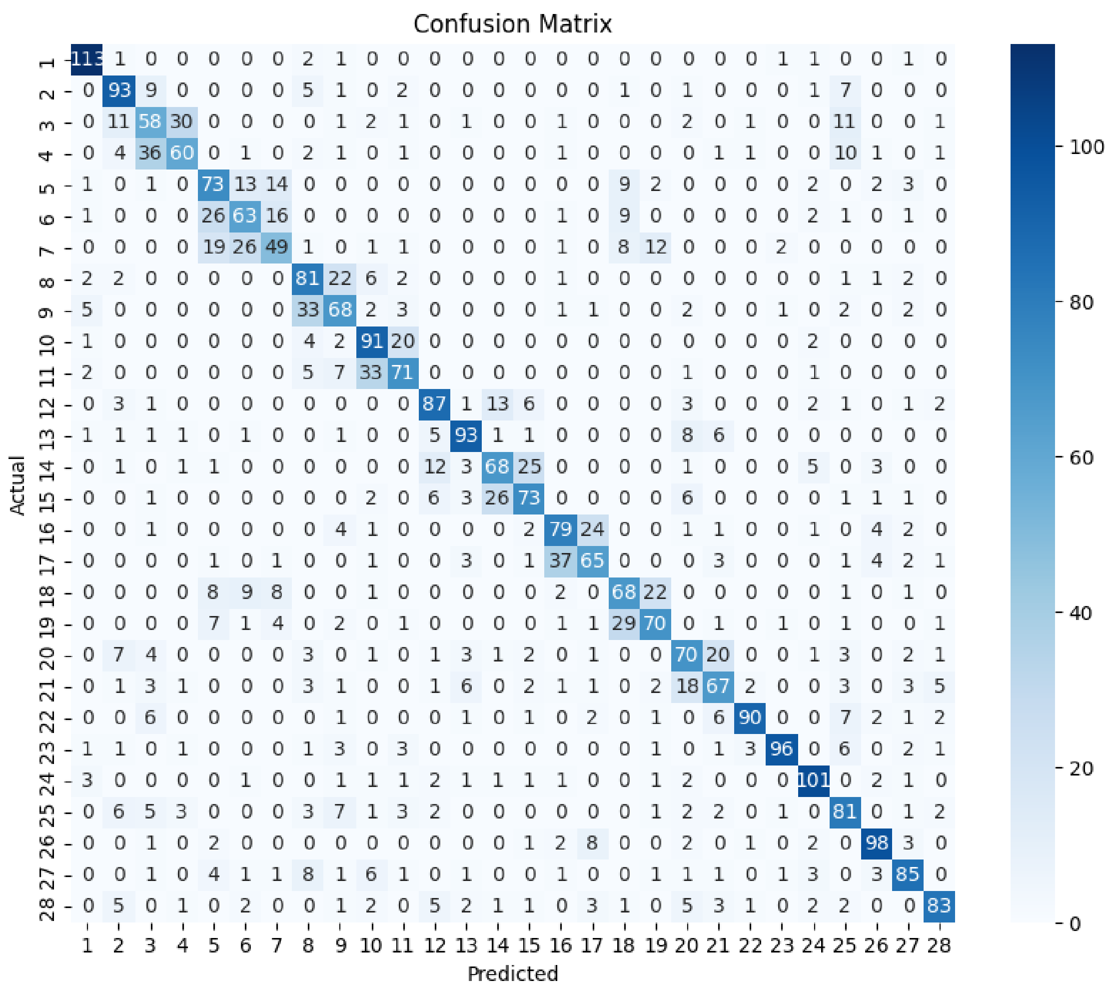
```
for kernel, accuracies in kernel_accuracies.items():
    plt.plot(accuracies, label=f'Kernel: {kernel}')
plt.xlabel('Fold')
plt.ylabel('Accuracy')
plt.title('Accuracy Curve for Different Kernels over Different Folds')
plt.legend()
plt.show()
```



After splitting test data into 10 K-folds and test each kernel in each fold , we found that when kernel = 'rbf' ,The model performace is the highest

[9]:
```
# Train an SVM model on training data
SVM_model = SVC(kernel='rbf', C=1.0, random_state=42)
SVM_model.fit(normalized_images, df_labels.ravel())

# Test the model on the test data
predictions_svm = SVM_model.predict(images_test)

# confusion matrix
conf_matrix = confusion_matrix(df_labels_test.ravel(), predictions_svm)
```

```
plt.figure(figsize=(10,8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
 ↪xticklabels=unique_classes, yticklabels=unique_classes)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# average f1-score
average_f1_score_svm = f1_score(df_labels_test.ravel(), predictions_svm,
 ↪average='micro')
print("Average F1 Score:", average_f1_score_svm)
```




```
Average F1 Score: 0.6529761904761905
```

Second Experiment KNN

[10]:
```
X_train, X_val, y_train, y_val = train_test_split(normalized_images, df_labels,␣
↪test_size=0.1, random_state=42,shuffle=True)
```

[11]:
```
encoder = OneHotEncoder(sparse=False, categories='auto')
y_train_onehot = encoder.fit_transform(y_train)
y_val_onehot = encoder.transform(y_val)
```

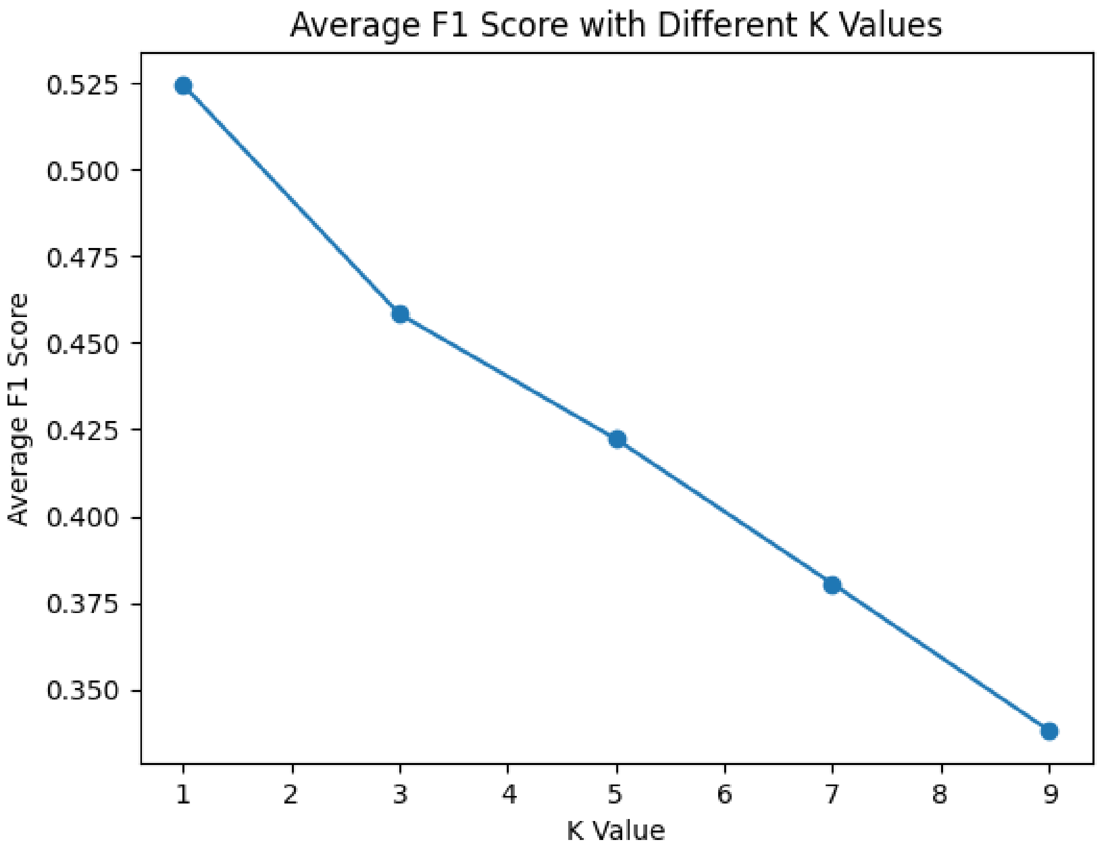
```
c:\Users\Essam\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\preprocessing\_encoders.py:972: FutureWarning: `sparse` was
renamed to `sparse_output` in version 1.2 and will be removed in 1.4.
`sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

[12]:
```
def train_and_evaluate_knn(X_train, y_train, X_val, y_val, k_values):
    f1_scores = []

    for k in k_values:
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)
        y_val_pred = knn.predict(X_val)
        f1 = f1_score(y_val, y_val_pred, average='weighted')
        f1_scores.append(f1)
        print(f"K={k}, F1 Score (weighted): {f1}")

    return f1_scores
```

[13]:
```
k_values = [1, 3, 5, 7, 9]
average_f1_scores = train_and_evaluate_knn(X_train, y_train_onehot, X_val,␣
↪y_val_onehot, k_values)
```

```
K=1, F1 Score (weighted): 0.524647332275423
K=3, F1 Score (weighted): 0.4583997805530425
K=5, F1 Score (weighted): 0.422321550594203
K=7, F1 Score (weighted): 0.3805427289780064
K=9, F1 Score (weighted): 0.3381732377759577
```

[14]:
```
plt.plot(k_values, average_f1_scores, marker='o')
plt.title('Average F1 Score with Different K Values')
plt.xlabel('K Value')
plt.ylabel('Average F1 Score')
plt.show()
```



[15]:
```
best_k = k_values[np.argmax(average_f1_scores)]
print(f"The best K value is: {best_k}")

final_knn_model = KNeighborsClassifier(n_neighbors=best_k)
final_knn_model.fit(X_train, y_train_onehot)
```

```
The best K value is: 1
```

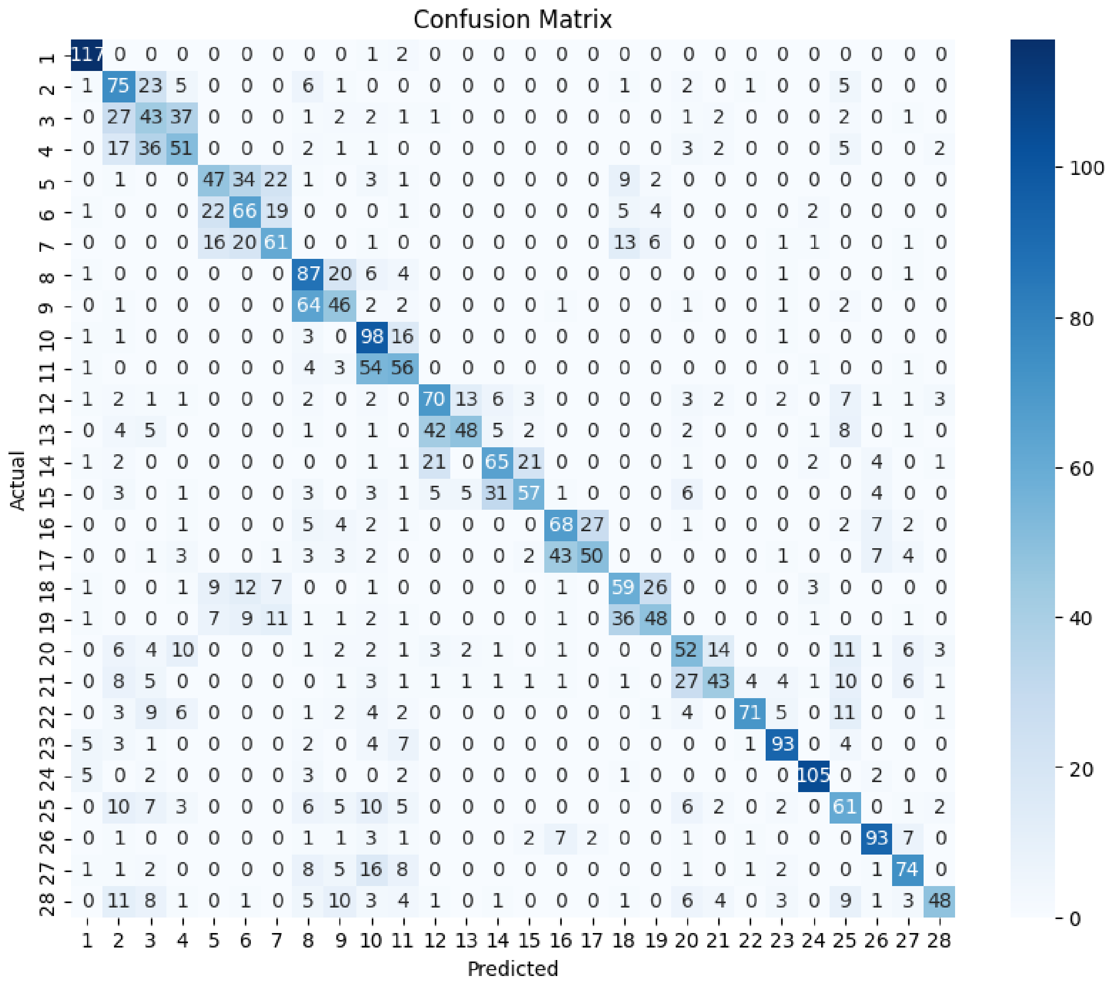
[15]:
```
KNeighborsClassifier(n_neighbors=1)
```

[16]:
```
y_test_onehot = encoder.transform(df_labels_test)
y_test_pred = final_knn_model.predict(images_test)
f1_test = f1_score(y_test_onehot, y_test_pred, average='weighted')
```

[17]:
```
print(f"\nTest F1 Score (weighted): {f1_test}")
```

```

Test F1 Score (weighted): 0.5508777566941211
```

[18]:
```
y_test_true_onehot = encoder.transform(df_labels_test)
conf_matrix = confusion_matrix(y_test_true_onehot.argmax(axis=1), y_test_pred.
 ↪argmax(axis=1))

plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',␣
 ↪xticklabels=unique_classes, yticklabels=unique_classes)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Third Experiment Neural Network (NN)

[19]:
```
def build_model_1(input_shape):
    model = Sequential()
```

```
    model.add(tf.keras.layers.Dense(32, input_shape=input_shape,␣
↪activation='relu'))
    model.add(tf.keras.layers.Dense(64, activation='relu'))
    model.add(tf.keras.layers.Dense(32, activation='relu'))
    model.add(tf.keras.layers.Dense(512, activation='relu'))
    model.add(tf.keras.layers.Dense(28, activation='softmax'))
    return model

def build_model_2(input_shape):
    model = Sequential()
    model.add(tf.keras.layers.Dense(128, input_shape=input_shape,␣
↪activation='relu'))
    model.add(tf.keras.layers.Dense(128, activation='relu'))
    model.add(tf.keras.layers.Dense(420, activation='relu'))
    model.add(tf.keras.layers.Dense(512, activation='relu'))
    model.add(tf.keras.layers.Dense(512, activation='relu'))
    model.add(tf.keras.layers.Dense(612, activation='relu'))
    model.add(tf.keras.layers.Dense(612, activation='relu'))
    model.add(tf.keras.layers.Dense(28, activation='softmax'))
    return model
```

[20]:
```
X_train, X_val, y_train, y_val = train_test_split(normalized_images, df_labels,␣
↪test_size=0.05, random_state=42,shuffle=True)
```

[21]:
```
encoder = OneHotEncoder(sparse=False, categories='auto')
y_train_onehot = encoder.fit_transform(y_train)
y_val_onehot = encoder.transform(y_val)
```

```
c:\Users\Essam\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\preprocessing\_encoders.py:972: FutureWarning: `sparse` was
renamed to `sparse_output` in version 1.2 and will be removed in 1.4.
`sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

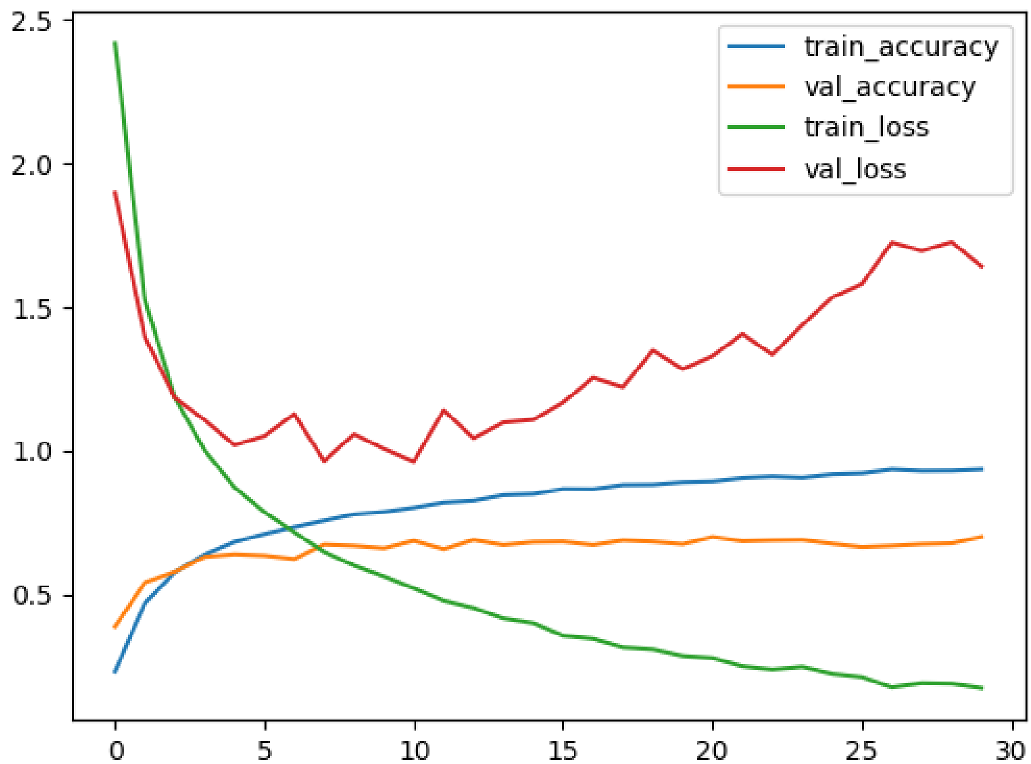
[22]:
```
def train_and_plot(model, X_train, y_train, X_val, y_val, model_name):
    model.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

    # Add a ModelCheckpoint callback to save the best model
    checkpoint = ModelCheckpoint(model_name, monitor='val_accuracy',␣
↪save_best_only=True)

    history = model.fit(X_train, y_train, epochs=30, batch_size=32,␣
↪validation_data=(X_val, y_val), callbacks=[checkpoint])

    # Plot training and validation curves
    plt.plot(history.history['accuracy'], label='train_accuracy')
```

```python
    plt.plot(history.history['val_accuracy'], label='val_accuracy')
    plt.plot(history.history['loss'], label='train_loss')
    plt.plot(history.history['val_loss'], label='val_loss')
    plt.legend()
    plt.show()

# Reshape input data if needed
input_shape = X_train.shape[1:]

# Example usage
model_1 = build_model_1(input_shape)
train_and_plot(model_1, X_train, y_train_onehot, X_val, y_val_onehot, 'model_1.
 ↪h5')

model_2 = build_model_2(input_shape)
train_and_plot(model_2, X_train, y_train_onehot, X_val, y_val_onehot, 'model_2.
 ↪h5')
```

```
Epoch 1/30
399/399 [==============================] - 5s 7ms/step - loss: 2.4199 -
accuracy: 0.2321 - val_loss: 1.8990 - val_accuracy: 0.3884
Epoch 2/30
 12/399 […] - ETA: 1s - loss: 1.7318 - accuracy:
0.3906

c:\Users\Essam\AppData\Local\Programs\Python\Python310\lib\site-
packages\keras\src\engine\training.py:3000: UserWarning: You are saving your
model as an HDF5 file via `model.save()`. This file format is considered legacy.
We recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(

399/399 [==============================] - 2s 5ms/step - loss: 1.5265 -
accuracy: 0.4698 - val_loss: 1.3974 - val_accuracy: 0.5417
Epoch 3/30
399/399 [==============================] - 2s 5ms/step - loss: 1.1870 -
accuracy: 0.5775 - val_loss: 1.1854 - val_accuracy: 0.5789
Epoch 4/30
399/399 [==============================] - 2s 5ms/step - loss: 1.0020 -
accuracy: 0.6390 - val_loss: 1.1078 - val_accuracy: 0.6310
Epoch 5/30
399/399 [==============================] - 2s 5ms/step - loss: 0.8733 -
accuracy: 0.6831 - val_loss: 1.0206 - val_accuracy: 0.6399
Epoch 6/30
399/399 [==============================] - 2s 5ms/step - loss: 0.7870 -
accuracy: 0.7107 - val_loss: 1.0522 - val_accuracy: 0.6354
Epoch 7/30
399/399 [==============================] - 2s 5ms/step - loss: 0.7170 -
```

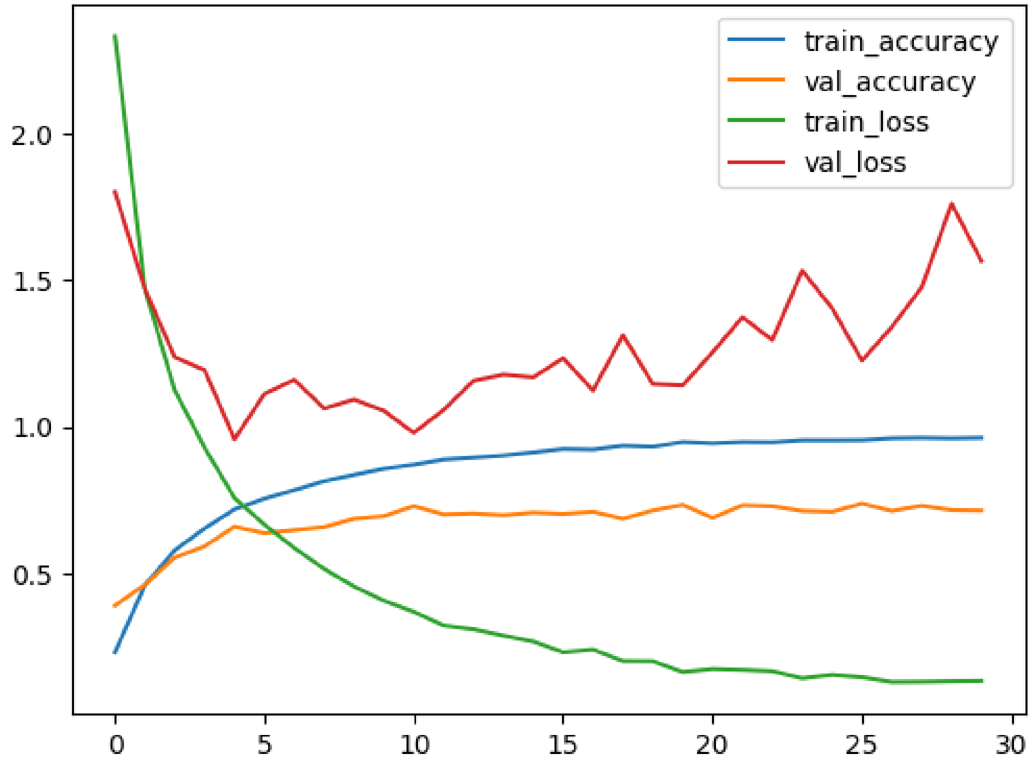
```
accuracy: 0.7350 - val_loss: 1.1286 - val_accuracy: 0.6235
Epoch 8/30
399/399 [==============================] - 2s 5ms/step - loss: 0.6481 -
accuracy: 0.7573 - val_loss: 0.9649 - val_accuracy: 0.6741
Epoch 9/30
399/399 [==============================] - 2s 5ms/step - loss: 0.6012 -
accuracy: 0.7792 - val_loss: 1.0590 - val_accuracy: 0.6696
Epoch 10/30
399/399 [==============================] - 2s 5ms/step - loss: 0.5630 -
accuracy: 0.7878 - val_loss: 1.0073 - val_accuracy: 0.6607
Epoch 11/30
399/399 [==============================] - 2s 5ms/step - loss: 0.5219 -
accuracy: 0.8023 - val_loss: 0.9632 - val_accuracy: 0.6875
Epoch 12/30
399/399 [==============================] - 2s 5ms/step - loss: 0.4788 -
accuracy: 0.8203 - val_loss: 1.1419 - val_accuracy: 0.6577
Epoch 13/30
399/399 [==============================] - 2s 5ms/step - loss: 0.4534 -
accuracy: 0.8268 - val_loss: 1.0447 - val_accuracy: 0.6905
Epoch 14/30
399/399 [==============================] - 2s 5ms/step - loss: 0.4171 -
accuracy: 0.8464 - val_loss: 1.0994 - val_accuracy: 0.6726
Epoch 15/30
399/399 [==============================] - 2s 5ms/step - loss: 0.4003 -
accuracy: 0.8502 - val_loss: 1.1090 - val_accuracy: 0.6830
Epoch 16/30
399/399 [==============================] - 2s 5ms/step - loss: 0.3567 -
accuracy: 0.8677 - val_loss: 1.1691 - val_accuracy: 0.6845
Epoch 17/30
399/399 [==============================] - 2s 5ms/step - loss: 0.3461 -
accuracy: 0.8670 - val_loss: 1.2553 - val_accuracy: 0.6726
Epoch 18/30
399/399 [==============================] - 2s 5ms/step - loss: 0.3161 -
accuracy: 0.8818 - val_loss: 1.2227 - val_accuracy: 0.6890
Epoch 19/30
399/399 [==============================] - 2s 5ms/step - loss: 0.3093 -
accuracy: 0.8827 - val_loss: 1.3501 - val_accuracy: 0.6845
Epoch 20/30
399/399 [==============================] - 2s 5ms/step - loss: 0.2863 -
accuracy: 0.8913 - val_loss: 1.2851 - val_accuracy: 0.6756
Epoch 21/30
399/399 [==============================] - 2s 5ms/step - loss: 0.2792 -
accuracy: 0.8940 - val_loss: 1.3305 - val_accuracy: 0.7009
Epoch 22/30
399/399 [==============================] - 2s 5ms/step - loss: 0.2502 -
accuracy: 0.9060 - val_loss: 1.4078 - val_accuracy: 0.6860
Epoch 23/30
399/399 [==============================] - 2s 5ms/step - loss: 0.2385 -
```

```
accuracy: 0.9113 - val_loss: 1.3352 - val_accuracy: 0.6890
Epoch 24/30
399/399 [==============================] - 2s 6ms/step - loss: 0.2484 -
accuracy: 0.9069 - val_loss: 1.4385 - val_accuracy: 0.6905
Epoch 25/30
399/399 [==============================] - 2s 6ms/step - loss: 0.2238 -
accuracy: 0.9185 - val_loss: 1.5346 - val_accuracy: 0.6771
Epoch 26/30
399/399 [==============================] - 2s 5ms/step - loss: 0.2124 -
accuracy: 0.9219 - val_loss: 1.5815 - val_accuracy: 0.6652
Epoch 27/30
399/399 [==============================] - 2s 5ms/step - loss: 0.1770 -
accuracy: 0.9355 - val_loss: 1.7262 - val_accuracy: 0.6696
Epoch 28/30
399/399 [==============================] - 2s 5ms/step - loss: 0.1918 -
accuracy: 0.9305 - val_loss: 1.6968 - val_accuracy: 0.6756
Epoch 29/30
399/399 [==============================] - 2s 5ms/step - loss: 0.1900 -
accuracy: 0.9311 - val_loss: 1.7274 - val_accuracy: 0.6786
Epoch 30/30
399/399 [==============================] - 2s 5ms/step - loss: 0.1745 -
accuracy: 0.9355 - val_loss: 1.6434 - val_accuracy: 0.7009
```



```
Epoch 1/30
399/399 [==============================] - 16s 32ms/step - loss: 2.3313 -
accuracy: 0.2310 - val_loss: 1.8002 - val_accuracy: 0.3899
Epoch 2/30
399/399 [==============================] - 13s 32ms/step - loss: 1.4707 -
accuracy: 0.4592 - val_loss: 1.4700 - val_accuracy: 0.4613
Epoch 3/30
399/399 [==============================] - 12s 31ms/step - loss: 1.1242 -
accuracy: 0.5786 - val_loss: 1.2371 - val_accuracy: 0.5536
Epoch 4/30
399/399 [==============================] - 12s 31ms/step - loss: 0.9282 -
accuracy: 0.6530 - val_loss: 1.1929 - val_accuracy: 0.5923
Epoch 5/30
399/399 [==============================] - 13s 32ms/step - loss: 0.7568 -
accuracy: 0.7184 - val_loss: 0.9571 - val_accuracy: 0.6592
Epoch 6/30
399/399 [==============================] - 12s 30ms/step - loss: 0.6661 -
accuracy: 0.7546 - val_loss: 1.1119 - val_accuracy: 0.6369
Epoch 7/30
399/399 [==============================] - 13s 33ms/step - loss: 0.5866 -
accuracy: 0.7836 - val_loss: 1.1602 - val_accuracy: 0.6473
Epoch 8/30
399/399 [==============================] - 13s 32ms/step - loss: 0.5150 -
accuracy: 0.8142 - val_loss: 1.0621 - val_accuracy: 0.6577
Epoch 9/30
399/399 [==============================] - 13s 32ms/step - loss: 0.4550 -
accuracy: 0.8357 - val_loss: 1.0926 - val_accuracy: 0.6860
Epoch 10/30
399/399 [==============================] - 13s 31ms/step - loss: 0.4064 -
accuracy: 0.8565 - val_loss: 1.0543 - val_accuracy: 0.6949
Epoch 11/30
399/399 [==============================] - 12s 31ms/step - loss: 0.3687 -
accuracy: 0.8706 - val_loss: 0.9790 - val_accuracy: 0.7292
Epoch 12/30
399/399 [==============================] - 12s 31ms/step - loss: 0.3220 -
accuracy: 0.8878 - val_loss: 1.0574 - val_accuracy: 0.7009
Epoch 13/30
399/399 [==============================] - 12s 30ms/step - loss: 0.3090 -
accuracy: 0.8946 - val_loss: 1.1569 - val_accuracy: 0.7039
Epoch 14/30
399/399 [==============================] - 11s 28ms/step - loss: 0.2873 -
accuracy: 0.9016 - val_loss: 1.1777 - val_accuracy: 0.6979
Epoch 15/30
399/399 [==============================] - 12s 30ms/step - loss: 0.2672 -
accuracy: 0.9123 - val_loss: 1.1688 - val_accuracy: 0.7068
Epoch 16/30
399/399 [==============================] - 12s 30ms/step - loss: 0.2308 -
accuracy: 0.9240 - val_loss: 1.2337 - val_accuracy: 0.7024
```

```
Epoch 17/30
399/399 [==============================] - 12s 31ms/step - loss: 0.2399 -
accuracy: 0.9222 - val_loss: 1.1235 - val_accuracy: 0.7098
Epoch 18/30
399/399 [==============================] - 12s 31ms/step - loss: 0.2008 -
accuracy: 0.9352 - val_loss: 1.3126 - val_accuracy: 0.6860
Epoch 19/30
399/399 [==============================] - 12s 31ms/step - loss: 0.2001 -
accuracy: 0.9318 - val_loss: 1.1457 - val_accuracy: 0.7143
Epoch 20/30
399/399 [==============================] - 13s 32ms/step - loss: 0.1634 -
accuracy: 0.9477 - val_loss: 1.1411 - val_accuracy: 0.7336
Epoch 21/30
399/399 [==============================] - 13s 31ms/step - loss: 0.1736 -
accuracy: 0.9433 - val_loss: 1.2537 - val_accuracy: 0.6890
Epoch 22/30
399/399 [==============================] - 13s 33ms/step - loss: 0.1707 -
accuracy: 0.9477 - val_loss: 1.3736 - val_accuracy: 0.7321
Epoch 23/30
399/399 [==============================] - 13s 32ms/step - loss: 0.1662 -
accuracy: 0.9468 - val_loss: 1.2956 - val_accuracy: 0.7292
Epoch 24/30
399/399 [==============================] - 13s 31ms/step - loss: 0.1419 -
accuracy: 0.9540 - val_loss: 1.5324 - val_accuracy: 0.7128
Epoch 25/30
399/399 [==============================] - 12s 31ms/step - loss: 0.1539 -
accuracy: 0.9539 - val_loss: 1.4043 - val_accuracy: 0.7098
Epoch 26/30
399/399 [==============================] - 12s 30ms/step - loss: 0.1455 -
accuracy: 0.9545 - val_loss: 1.2255 - val_accuracy: 0.7381
Epoch 27/30
399/399 [==============================] - 13s 33ms/step - loss: 0.1295 -
accuracy: 0.9608 - val_loss: 1.3402 - val_accuracy: 0.7128
Epoch 28/30
399/399 [==============================] - 13s 33ms/step - loss: 0.1302 -
accuracy: 0.9621 - val_loss: 1.4754 - val_accuracy: 0.7307
Epoch 29/30
399/399 [==============================] - 13s 32ms/step - loss: 0.1320 -
accuracy: 0.9605 - val_loss: 1.7603 - val_accuracy: 0.7158
Epoch 30/30
399/399 [==============================] - 12s 31ms/step - loss: 0.1333 -
accuracy: 0.9622 - val_loss: 1.5646 - val_accuracy: 0.7143
```



[23]:
```
best_model = load_model('model_2.h5')
y_pred_probabilities = best_model.predict(images_test)
y_pred_classes = np.argmax(y_pred_probabilities, axis=1)
y_pred_classes+=1
```

```
105/105 [==============================] - 1s 11ms/step
```

[24]:
```
accuracy = accuracy_score(df_labels_test, y_pred_classes)
f1 = f1_score(df_labels_test, y_pred_classes, average='weighted')

print("Accuracy:", accuracy)
print("\nF1 Score (weighted):", f1)
```

```
Accuracy: 0.7282738095238095

F1 Score (weighted): 0.7281577943800837
```

[25]:
```
conf_matrix = confusion_matrix(df_labels_test, y_pred_classes)
plt.figure(figsize=(10,8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',␣
 ↪xticklabels=unique_classes, yticklabels=unique_classes)
plt.title('Confusion Matrix')
```

```
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
plt.show()
plt.show()
```