final-aml-a4-q1-submit-1

December 12, 2023

We are trying to classify hand written English Alphabets,R G and S. Each class has 110 images, the hand written images have been taken from different people and pictures are clicked from different devices(mobiles) plus in different light setting which introduces variance in dataset. For the pretrained model we are using ResNet50 as its one of the best image classifiers (RGB)

[1]:
```
# Install necessary libraries
!pip install tensorflow matplotlib
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-
packages (2.14.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (3.7.1)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (23.5.26)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (16.0.6)
Requirement already satisfied: ml-dtypes==0.2.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: numpy>=1.23.5 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (1.23.5)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (23.2)
Requirement already satisfied:
protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3
in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.20.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-
```

```
packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.3.0)
Requirement already satisfied: typing-extensions>=3.6.6 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (4.5.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.34.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.59.3)
Requirement already satisfied: tensorboard<2.15,>=2.14 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.1)
Requirement already satisfied: tensorflow-estimator<2.15,>=2.14.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.0)
Requirement already satisfied: keras<2.15,>=2.14.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.14.0)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (4.45.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow)
(0.42.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.15,>=2.14->tensorflow) (2.17.3)
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.15,>=2.14->tensorflow) (1.0.0)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.15,>=2.14->tensorflow) (3.5.1)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.15,>=2.14->tensorflow) (2.31.0)
```

```
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.15,>=2.14->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.15,>=2.14->tensorflow) (3.0.1)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from google-
auth<3,>=1.6.3->tensorboard<2.15,>=2.14->tensorflow) (5.3.2)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.10/dist-packages (from google-
auth<3,>=1.6.3->tensorboard<2.15,>=2.14->tensorflow) (0.3.0)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-
packages (from google-auth<3,>=1.6.3->tensorboard<2.15,>=2.14->tensorflow) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from google-auth-
oauthlib<1.1,>=0.5->tensorboard<2.15,>=2.14->tensorflow) (1.3.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from
requests<3,>=2.21.0->tensorboard<2.15,>=2.14->tensorflow) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests<3,>=2.21.0->tensorboard<2.15,>=2.14->tensorflow) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from
requests<3,>=2.21.0->tensorboard<2.15,>=2.14->tensorflow) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from
requests<3,>=2.21.0->tensorboard<2.15,>=2.14->tensorflow) (2023.11.17)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/usr/local/lib/python3.10/dist-packages (from
werkzeug>=1.0.1->tensorboard<2.15,>=2.14->tensorflow) (2.1.3)
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in
/usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-
auth<3,>=1.6.3->tensorboard<2.15,>=2.14->tensorflow) (0.5.1)
Requirement already satisfied: oauthlib>=3.0.0 in
/usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-
auth-oauthlib<1.1,>=0.5->tensorboard<2.15,>=2.14->tensorflow) (3.2.2)
```

[1]:
```
# Import libraries
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image_dataset_from_directory
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

[3]:
```python
import warnings
warnings.filterwarnings('ignore')
```

[14]:
```python
# Connect to Google Drive
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call
drive.mount("/content/gdrive", force_remount=True).
```

[20]:
```python
# Define the path to your dataset on Google Drive
dataset_path = '/content/gdrive/My Drive/ClassificationOfHandwrittenText'
```

1. Take at least 100 images per class with at least 3 classes using your phone/camera (e.g. take photos of different types of trees, flowers or animals). Display 5 examples from each class. [10 points]

[24]:
```python
import os
import cv2
import numpy as np

dataset_path = '/content/gdrive/My Drive/ClassificationOfHandwrittenText'
image_list = []
labels = []

# Iterate through each class folder in the directory
for class_name in os.listdir(dataset_path):
    class_path = os.path.join(dataset_path, class_name)

    # Iterate through each file in the class folder
    for filename in os.listdir(class_path):
        if filename.endswith('.jpg') or filename.endswith('.png') or filename.
 ↪endswith('.jpeg'):
            # Construct the full path to the image file
            image_path = os.path.join(class_path, filename)

            # Read and resize the image
            image = cv2.imread(image_path)
            #image = cv2.resize(image, (400, 400))
            image = cv2.resize(image, (224, 224))
            image = image / 255.0

            if image is not None:
                image_list.append(image)

                # Obtain the label from the class folder name
                labels.append(class_name)
```

```python
# Convert the lists to NumPy arrays
image_list = np.array(image_list)
labels = np.array(labels)
```

[25]:
```python
image_list.shape
```

[25]:
```
(330, 224, 224, 3)
```

[26]:
```python
labels
```

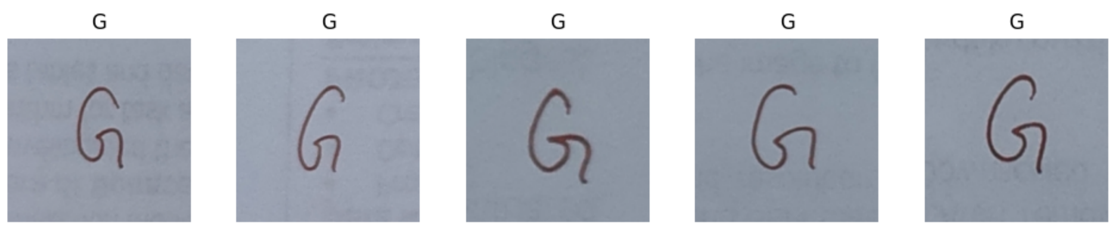
[26]:
```
array(['S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S',
       'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S',
       'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S',
       'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S',
       'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S',
       'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S',
       'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S',
       'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S',
       'S', 'S', 'S', 'S', 'S', 'S', 'R', 'R', 'R', 'R', 'R', 'R', 'R',
       'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R',
       'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R',
       'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R',
       'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R',
       'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R',
       'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R',
       'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R',
       'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'R', 'G',
       'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G',
       'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G',
       'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G',
       'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G',
       'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G',
       'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G',
       'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G',
       'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'G',
       'G', 'G', 'G', 'G', 'G'], dtype='<U1')
```

[27]:
```python
# Get the shape of the dataset
print(f"\n Classes in the dataset: {len(np.unique(labels))}")
print(f"\nShape of the dataset: {image_list.shape}") # heigth,width,channel
```

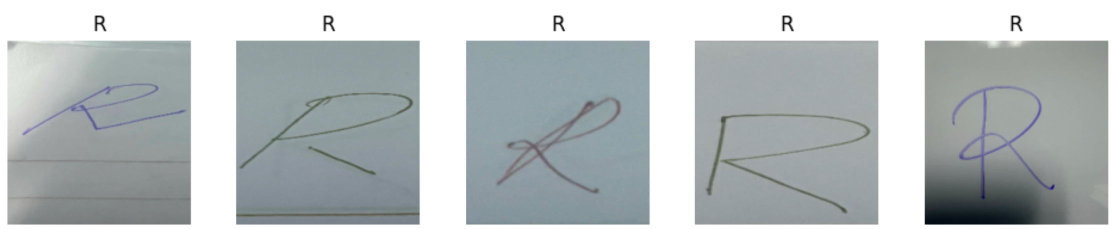
```
 Classes in the dataset: 3

Shape of the dataset: (330, 224, 224, 3)
```

```
[28]: # Function to display images
def display_images(images, class_name):
    plt.figure(figsize=(12, 2))
    for i in range(min(5, len(images))):
        plt.subplot(1, 5, i + 1)
        plt.imshow(images[i])
        plt.title(f"{class_name}")
        plt.axis("off")
    plt.show()

# Display 5 images from each class
unique_labels = np.unique(labels)
print(unique_labels)
for label in unique_labels:
    label_images = image_list[labels == label]
    print(f"Number of images for class {label}: {len(label_images)}")  # Debugging statement
    display_images(label_images, label)
```

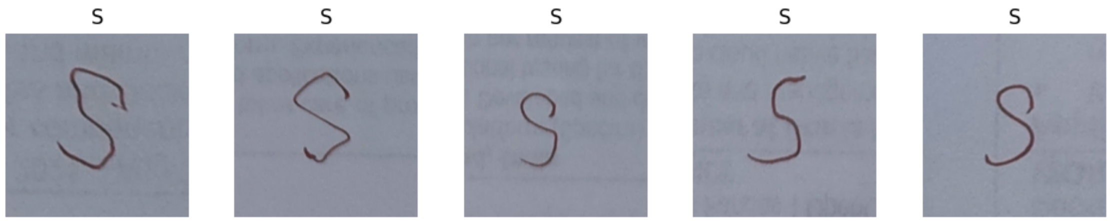
```
['G' 'R' 'S']
Number of images for class G: 110
```




```
Number of images for class R: 110
```




```
Number of images for class S: 110
```



The above displays first few images in each of the 3 classes.

2. Split the images into a training set, a validation set, and a test set. [5 points]

[29]:
```python
# # Assuming y_train contains labels like ['class1', 'class2', ...]
# label_mapping = {'G': 0, 'R': 1, 'S': 2}  # Add all your class labels
from sklearn.preprocessing import OrdinalEncoder
# # Convert labels to numeric
# labels_new = np.array([label_mapping[label] for label in labels])

labels_2d = labels.reshape(-1, 1)

# Fit and transform using OrdinalEncoder
encoder = OrdinalEncoder()
labels_numeric = encoder.fit_transform(labels_2d)

# Flatten the resulting 2D array to get a 1D array of numeric labels
labels_numeric = labels_numeric.flatten()

# Now, labels_numeric contains the numeric representation of your original␣
 ↪class names
print(labels_numeric)
```

```
[2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

[30]:

```python
# Check the data type of y_train_numeric
print(f"labels_new_numeric dtype: {labels_numeric.dtype}")
```

```
labels_new_numeric dtype: float64
```

[31]:

```python
labels_numeric
```

[31]:

```
array([2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0.])
```

[ ]:

```python
img_size = (32, 32)
batch_size = 32
#Data Augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    rotation_range=45,
    brightness_range=[0.5, 1.5],
    validation_split=0.2
)

train_generator = train_datagen.flow_from_directory(
    dataset_path + '/Training data',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
```

```
)

validation_generator = train_datagen.flow_from_directory(
    dataset_path + '/Training data',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'
)
```

[37]:
```
# Split the data into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(image_list, labels_numeric,␣
 ↪test_size=0.333, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,␣
 ↪random_state=42)
```

Spliting the dataset into train,test and validation Training set: 66.67% Validation set: 16.67% Test set: 16.67%

[38]:
```
# Print the shape of the resulting sets
print(f"Training set: {X_train.shape}, {y_train.shape}")
print(f"Validation set: {X_val.shape}, {y_val.shape}")
print(f"Test set: {X_test.shape}, {y_test.shape}")
```

```
Training set: (220, 224, 224, 3), (220,)
Validation set: (55, 224, 224, 3), (55,)
Test set: (55, 224, 224, 3), (55,)
```

[39]:
```
# Print the split ratios in percentage form
total_samples = len(image_list)
train_ratio = len(X_train) / total_samples * 100
val_ratio = len(X_val) / total_samples * 100
test_ratio = len(X_test) / total_samples * 100

print(f"Training set: {train_ratio:.2f}%")
print(f"Validation set: {val_ratio:.2f}%")
print(f"Test set: {test_ratio:.2f}%")
```

```
Training set: 66.67%
Validation set: 16.67%
Test set: 16.67%
```

Build the input pipeline, including the appropriate preprocessing operations, and add data augmentation. [10 points]

[40]:
```
# Define data augmentation operations
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomFlip("horizontal"),
```

```
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
    tf.keras.layers.experimental.preprocessing.RandomZoom(0.2),
])

# Apply data augmentation to training data
augmented_train_data = tf.data.Dataset.from_tensor_slices((X_train, y_train))
augmented_train_data = augmented_train_data.map(lambda x, y:
 ↪(data_augmentation(x, training=True), y))
```

Data Augumentation performs: 1. introduces randomness by horizontally flipping images during training. 2. randomly rotates the images by a maximum angle of 0.2 radians 3. random zooming to the images, with a maximum zoom factor of 0.2.

[41]:
```
# Create a function to preprocess images, including converting to grayscale
def preprocess_image(image, label):
    # Convert to float32
    image = tf.image.convert_image_dtype(image, tf.float32)

    # Convert to grayscale
    # image = tf.image.rgb_to_grayscale(image)

    # Resize images to the desired size
    image = tf.image.resize(image, (224, 224))

    return image, label
```

Image processing : convert to float and resize, no need to apply greyscaling as ResNet50 takes RGB images.

[42]:
```
# Create datasets for training, validation, and test sets
train_dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train))
val_dataset = tf.data.Dataset.from_tensor_slices((X_val, y_val))
test_dataset = tf.data.Dataset.from_tensor_slices((X_test, y_test))

# Apply the preprocess_image function to each dataset
train_dataset = train_dataset.map(preprocess_image)
val_dataset = val_dataset.map(preprocess_image)
test_dataset = test_dataset.map(preprocess_image)
```

[43]:
```
# Batch the datasets
batch_size = 32
train_dataset = train_dataset.shuffle(buffer_size=len(X_train)).
 ↪batch(batch_size).prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
val_dataset = val_dataset.batch(batch_size).prefetch(buffer_size=tf.data.
 ↪experimental.AUTOTUNE)
test_dataset = test_dataset.batch(batch_size).prefetch(buffer_size=tf.data.
 ↪experimental.AUTOTUNE)
```

```
# Verify the shape of the batches
for images, labels in train_dataset.take(1):
    print("Batch shape:", images.shape)
```

```
Batch shape: (32, 224, 224, 3)
```

[44]:
```
X_train = np.array(X_train)
y_train = np.array(y_train)
print(f"X_train dtype: {X_train.dtype}")
print(f"y_train dtype: {y_train.dtype}")
```

```
X_train dtype: float64
y_train dtype: float64
```

4. Fine-tune a pretrained model of your choice on this dataset (the one you created in part 3). Report classification accuracy and give a few examples of correct/incorrect classification (show a few images that were correctly/incorrectly classified). [10 points]

[2]:
```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
import matplotlib.pyplot as plt
import numpy as np
```

[46]:
```
# Load the ResNet50 model with pre-trained weights
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224,␣
 ↪224, 3))

# Freeze the pre-trained layers
for layer in base_model.layers:
    layer.trainable = False

# Build your custom model on top of the pre-trained ResNet50
model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(3, activation='softmax')
])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 1s 0us/step
```

[47]:
```python
# Compile the model
model.compile(optimizer=Adam(learning_rate=1e-4),␣
↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model on your training dataset
history = model.fit(train_dataset, validation_data=val_dataset, epochs=20)
```

```
Epoch 1/20
7/7 [==============================] - 70s 10s/step - loss: 1.2872 - accuracy:
0.3591 - val_loss: 1.1085 - val_accuracy: 0.2909
Epoch 2/20
7/7 [==============================] - 53s 8s/step - loss: 1.3045 - accuracy:
0.3409 - val_loss: 1.1240 - val_accuracy: 0.2909
Epoch 3/20
7/7 [==============================] - 53s 8s/step - loss: 1.2275 - accuracy:
0.3773 - val_loss: 1.1119 - val_accuracy: 0.2909
Epoch 4/20
7/7 [==============================] - 53s 8s/step - loss: 1.2749 - accuracy:
0.3136 - val_loss: 1.0981 - val_accuracy: 0.3091
Epoch 5/20
7/7 [==============================] - 54s 8s/step - loss: 1.2469 - accuracy:
0.3409 - val_loss: 1.0945 - val_accuracy: 0.4727
Epoch 6/20
7/7 [==============================] - 53s 8s/step - loss: 1.1950 - accuracy:
0.3636 - val_loss: 1.1189 - val_accuracy: 0.2909
Epoch 7/20
7/7 [==============================] - 54s 8s/step - loss: 1.2374 - accuracy:
0.3227 - val_loss: 1.1004 - val_accuracy: 0.3091
Epoch 8/20
7/7 [==============================] - 54s 8s/step - loss: 1.2016 - accuracy:
0.3364 - val_loss: 1.0921 - val_accuracy: 0.3273
Epoch 9/20
7/7 [==============================] - 52s 8s/step - loss: 1.2227 - accuracy:
0.3136 - val_loss: 1.0897 - val_accuracy: 0.4545
Epoch 10/20
7/7 [==============================] - 61s 9s/step - loss: 1.1702 - accuracy:
0.3136 - val_loss: 1.0922 - val_accuracy: 0.3091
Epoch 11/20
7/7 [==============================] - 54s 8s/step - loss: 1.1626 - accuracy:
0.3545 - val_loss: 1.1013 - val_accuracy: 0.3091
Epoch 12/20
7/7 [==============================] - 52s 7s/step - loss: 1.1776 - accuracy:
0.3364 - val_loss: 1.1170 - val_accuracy: 0.2909
Epoch 13/20
7/7 [==============================] - 53s 8s/step - loss: 1.1933 - accuracy:
0.3455 - val_loss: 1.0978 - val_accuracy: 0.3455
Epoch 14/20
```

```
7/7 [==============================] - 52s 7s/step - loss: 1.1459 - accuracy:
0.3455 - val_loss: 1.0964 - val_accuracy: 0.3455
Epoch 15/20
7/7 [==============================] - 61s 9s/step - loss: 1.1052 - accuracy:
0.4227 - val_loss: 1.0921 - val_accuracy: 0.3455
Epoch 16/20
7/7 [==============================] - 60s 9s/step - loss: 1.1294 - accuracy:
0.3545 - val_loss: 1.0903 - val_accuracy: 0.3636
Epoch 17/20
7/7 [==============================] - 60s 9s/step - loss: 1.0829 - accuracy:
0.4000 - val_loss: 1.0885 - val_accuracy: 0.4000
Epoch 18/20
7/7 [==============================] - 52s 7s/step - loss: 1.0896 - accuracy:
0.3909 - val_loss: 1.0896 - val_accuracy: 0.3455
Epoch 19/20
7/7 [==============================] - 61s 9s/step - loss: 1.1318 - accuracy:
0.3682 - val_loss: 1.0967 - val_accuracy: 0.3455
Epoch 20/20
7/7 [==============================] - 52s 8s/step - loss: 1.1705 - accuracy:
0.2864 - val_loss: 1.1022 - val_accuracy: 0.3636
```

[48]:
```
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(test_dataset)
print(f'Test accuracy: {test_acc}')

# Make predictions on the test set
predictions = model.predict(test_dataset)
predicted_labels = np.argmax(predictions, axis=1)
```

```
2/2 [==============================] - 17s 5s/step - loss: 1.0894 - accuracy:
0.4000
Test accuracy: 0.4000000059604645
2/2 [==============================] - 17s 4s/step
```

- Train Accuracy : 0.4
- Val Accuarcy : 0.45
- Test Accuarcy : 0.40

[49]:
```
predicted_labels
```

[49]:
```
array([1, 1, 1, 1, 1, 1, 1, 2, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

[50]:
```
y_test
```

[50]:
```
array([0., 1., 2., 0., 2., 2., 0., 2., 2., 0., 1., 0., 2., 1., 2., 1., 1.,
       0., 0., 1., 2., 1., 0., 0., 1., 0., 0., 1., 1., 0., 1., 2., 1., 0.,
```

```
       2., 2., 0., 1., 2., 1., 1., 2., 0., 1., 0., 0., 1., 0., 1., 2., 2.,
       0., 1., 2., 2.])
```

[51]:
```
y_train
```

[51]:
```
array([1., 0., 0., 1., 1., 2., 1., 2., 0., 0., 2., 1., 2., 2., 2., 1., 1.,
       2., 2., 2., 0., 2., 2., 2., 0., 2., 0., 1., 1., 1., 2., 0., 1., 0.,
       1., 0., 1., 0., 1., 0., 1., 2., 0., 1., 0., 1., 1., 1., 0., 0., 0.,
       1., 1., 1., 1., 2., 1., 1., 0., 1., 0., 0., 1., 2., 2., 2., 0., 2.,
       1., 1., 1., 0., 2., 0., 2., 0., 2., 1., 0., 1., 2., 2., 2., 1., 1.,
       0., 1., 1., 2., 2., 0., 1., 1., 2., 2., 0., 1., 2., 0., 1., 0., 2.,
       0., 0., 0., 2., 1., 0., 1., 1., 2., 1., 2., 2., 0., 2., 1., 2., 0.,
       0., 2., 0., 1., 0., 2., 0., 0., 0., 2., 1., 2., 0., 1., 0., 0., 0.,
       1., 2., 1., 1., 0., 1., 0., 2., 2., 0., 2., 2., 1., 2., 0., 0., 1.,
       0., 0., 1., 1., 0., 0., 0., 0., 0., 1., 1., 2., 1., 1., 0., 2., 2.,
       2., 2., 2., 1., 2., 0., 2., 2., 0., 0., 2., 2., 0., 1., 0., 1., 0.,
       0., 2., 2., 1., 1., 0., 1., 1., 2., 2., 0., 0., 2., 1., 0., 1., 0.,
       0., 0., 1., 1., 1., 2., 2., 1., 1., 0., 2., 1., 2., 2., 0., 2.])
```

[52]:
```
label_mapping = {'G': 0, 'R': 1, 'S': 2}
true_labels = np.array([key for key, value in label_mapping.items() if value ==
 ↪numeric_label] for numeric_label in y_test)
true_labels
```

[52]:
```
array(<generator object <genexpr> at 0x7dd8586e2650>, dtype=object)
```

[53]:
```
# Report classification metrics
print(classification_report(y_test, predicted_labels))
print(confusion_matrix(y_test, predicted_labels))
```

```
              precision    recall  f1-score   support

         0.0       0.75      0.16      0.26        19
         1.0       0.36      0.95      0.52        19
         2.0       1.00      0.06      0.11        17

    accuracy                           0.40        55
   macro avg       0.70      0.39      0.30        55
weighted avg       0.69      0.40      0.30        55

[[ 3 16  0]
 [ 1 18  0]
 [ 0 16  1]]
```

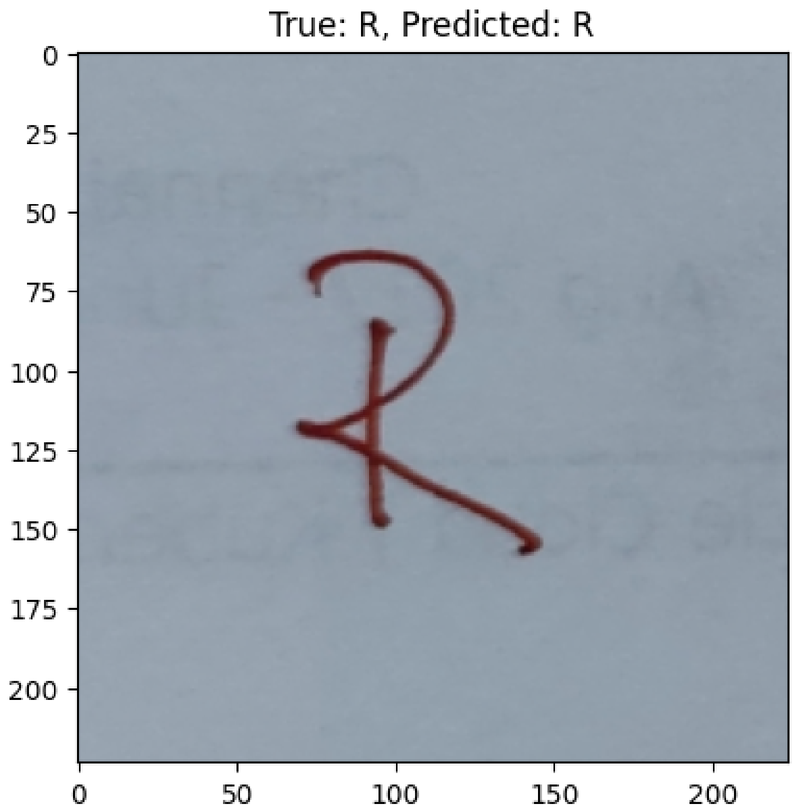
1. For class 0 (G): 0 true positives, 6 false positives, and 0 false negatives.
2. For class 1 (R): 5 true positives, 0 false positives, and 0 false negatives.
3. For class 2 (S): 0 true positives, 5 false positives, and 0 false negatives.

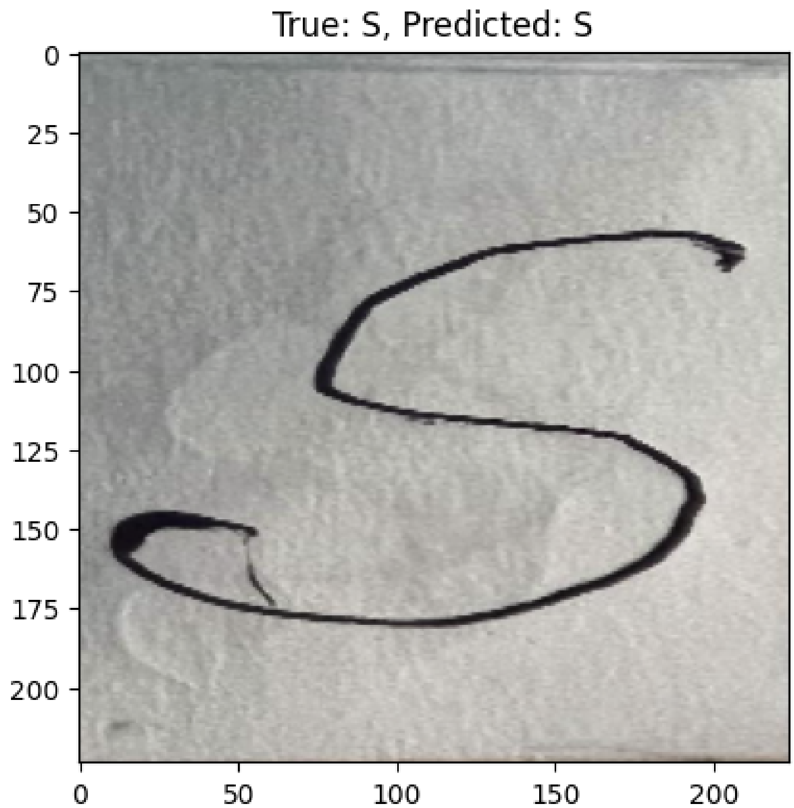
Interpretation:
The model is correctly identifying all instances of class 1 (R), but it's not predicting class
The overall accuracy is low, and the model seems to have a bias towards predicting class 1 (R)

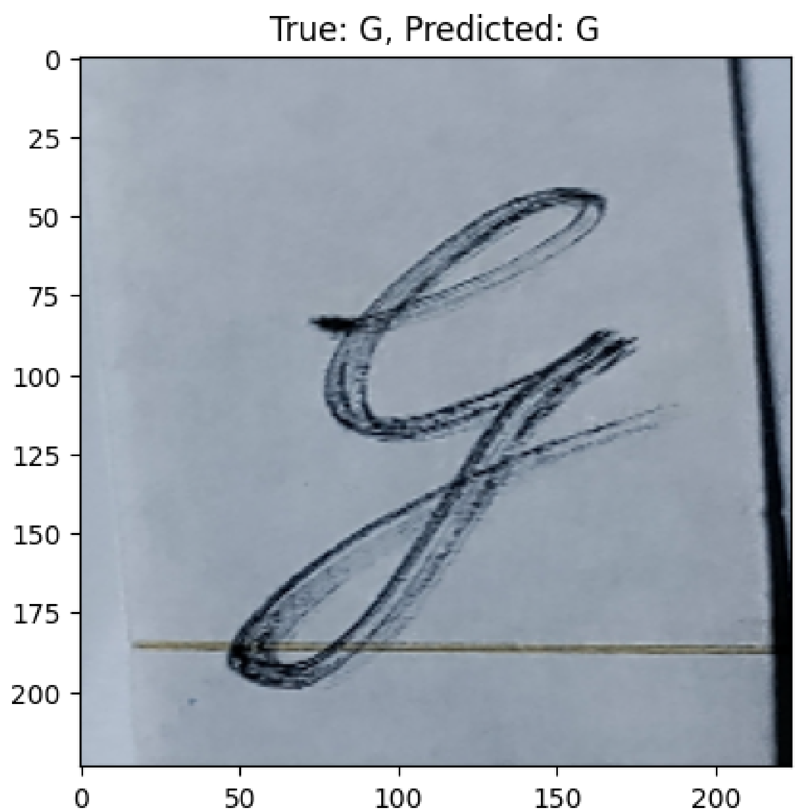
Pretrained model might not perform as expected, and a small training set is one of the factors

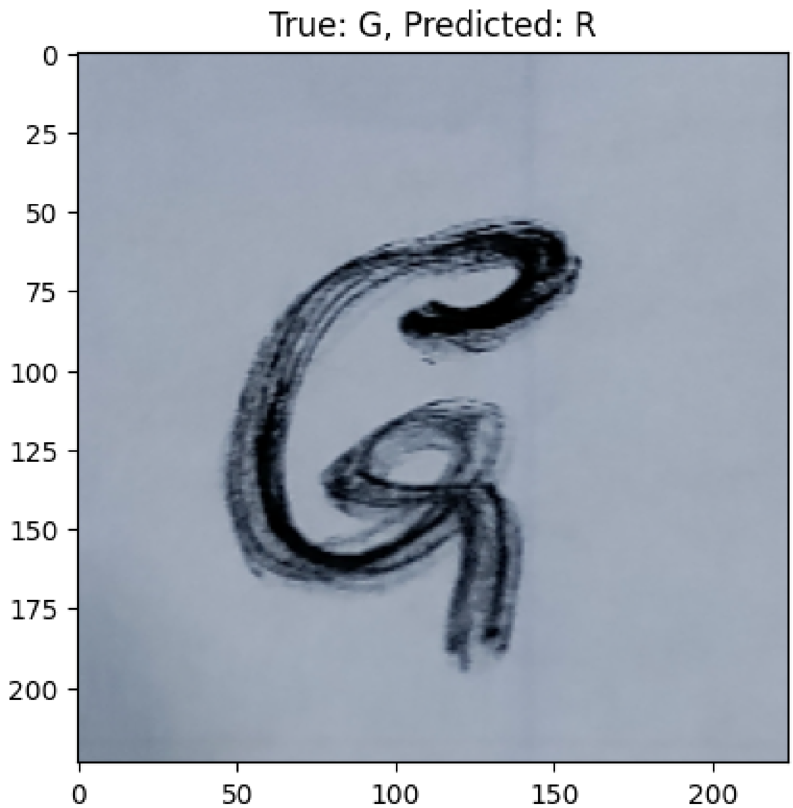
[54]:

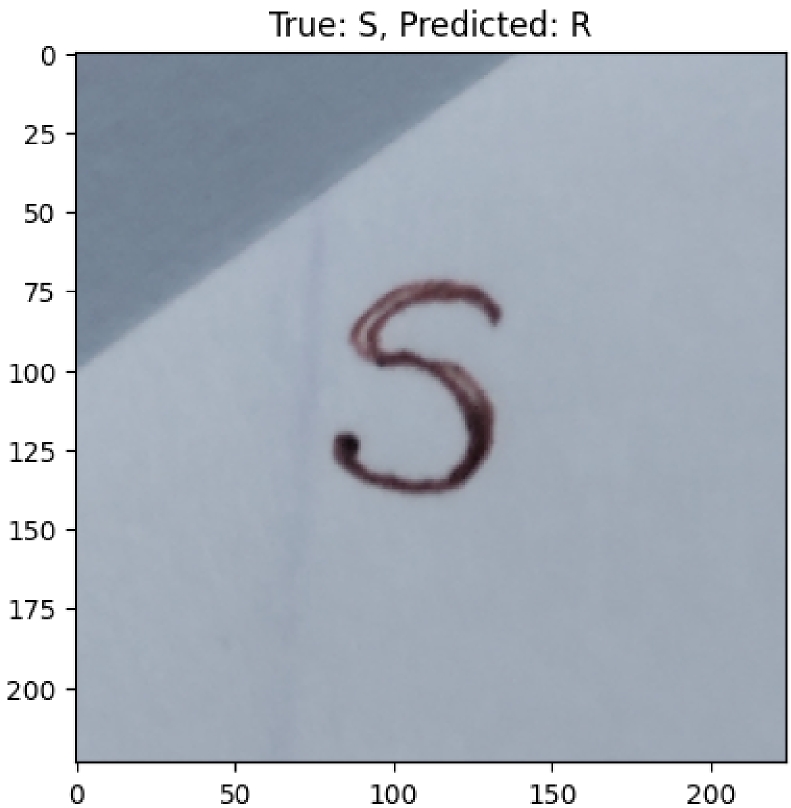
```python
# Assuming label_mapping is already defined
label_mapping_inverse = {value: key for key, value in label_mapping.items()}

# Show a few examples of correct and incorrect classifications
correct_indices = np.where(y_test == predicted_labels)[0]
incorrect_indices = np.where(y_test != predicted_labels)[0]

# Display a few correct predictions
for i in range(min(3, len(correct_indices))):
    index = correct_indices[i]
    plt.imshow(X_test[index])
    true_label_name = label_mapping_inverse[y_test[index]]
    predicted_label_name = label_mapping_inverse[predicted_labels[index]]
    plt.title(f'True: {true_label_name}, Predicted: {predicted_label_name}')
    plt.show()

# Display a few incorrect predictions
for i in range(min(3, len(incorrect_indices))):
    index = incorrect_indices[i]
    plt.imshow(X_test[index])
    true_label_name = label_mapping_inverse[y_test[index]]
    predicted_label_name = label_mapping_inverse[predicted_labels[index]]
    plt.title(f'True: {true_label_name}, Predicted: {predicted_label_name}')
    plt.show()
```

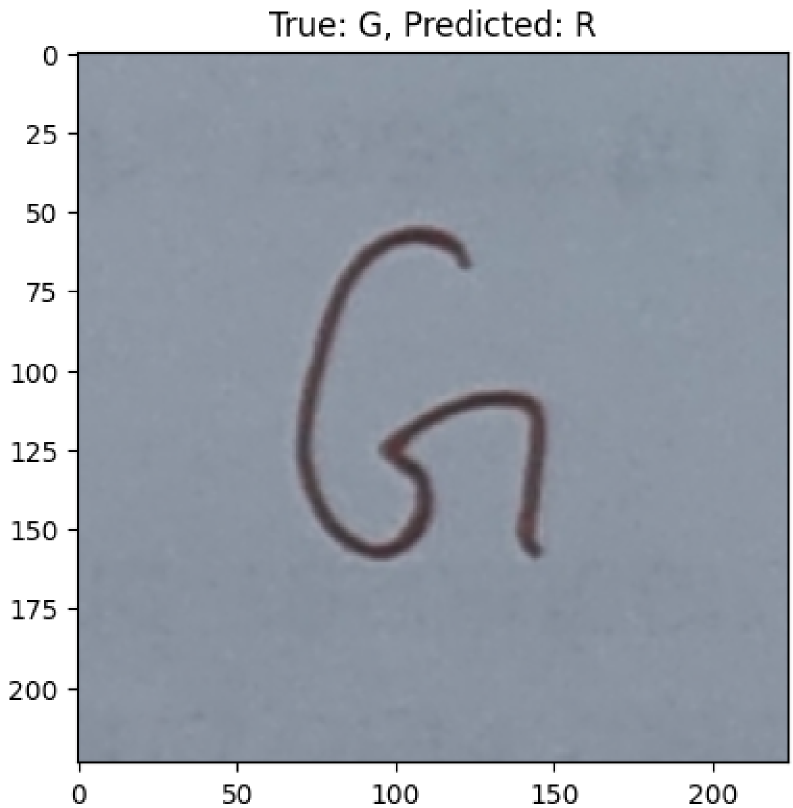

True: R, Predicted: R
0
25
50
75
100
125
150
175
200
0
50
100
150
200


True: S, Predicted: S
0
25
50
75
100
125
150
175
200
0
50
100
150
200


True: G, Predicted: G
0
25
50
75
100
125
150
175
200
0
50
100
150
200


True: G, Predicted: R
0
25
50
75
100
125
150
175
200
0
50
100
150
200


True: S, Predicted: R
0
25
50
75
100
125
150
175
200
0
50
100
150
200



Above shows some images classified correctly and incorrect classification as well

5. Train from scratch (without pretraining) a deep neural network that contains convolutional layers on this dataset (the one you created in part 3). Report classification accuracy and give a few examples of correct/incorrect classification (show a few images that were correctly/incorrectly classified). Note: The objective of this question is to illustrate that training deep networks from scratch requires a lot of data so it is ok if your classification accuracy is low. [15 points]

[3]:
```
# simple convolutional neural network
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(3, activation='softmax')  # have 3 classes
])
```

[4]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896

 max_pooling2d (MaxPooling2  (None, 111, 111, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 54, 54, 64)        0
 g2D)

 flatten (Flatten)           (None, 186624)            0

 dense (Dense)               (None, 128)               23888000

 dense_1 (Dense)             (None, 3)                 387

=================================================================
Total params: 23907779 (91.20 MB)
Trainable params: 23907779 (91.20 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[56]:

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

# Train the model
history = model.fit(X_train, y_train, validation_data=(X_val, y_val),␣
 ↪epochs=10, batch_size=32)
```

```
Epoch 1/10
7/7 [==============================] - 24s 3s/step - loss: 7.4050 - accuracy:
0.3545 - val_loss: 1.5673 - val_accuracy: 0.2909
Epoch 2/10
7/7 [==============================] - 32s 5s/step - loss: 1.2765 - accuracy:
0.3409 - val_loss: 1.1571 - val_accuracy: 0.2909
Epoch 3/10
7/7 [==============================] - 22s 3s/step - loss: 1.1239 - accuracy:
0.3955 - val_loss: 1.1272 - val_accuracy: 0.2909
Epoch 4/10
7/7 [==============================] - 23s 3s/step - loss: 1.0144 - accuracy:
0.4182 - val_loss: 1.1243 - val_accuracy: 0.4182
Epoch 5/10
7/7 [==============================] - 25s 3s/step - loss: 0.8495 - accuracy:
```

```
0.6409 - val_loss: 0.9111 - val_accuracy: 0.6182
Epoch 6/10
7/7 [==============================] - 22s 3s/step - loss: 0.6664 - accuracy:
0.7364 - val_loss: 0.9274 - val_accuracy: 0.5818
Epoch 7/10
7/7 [==============================] - 24s 3s/step - loss: 0.4590 - accuracy:
0.8045 - val_loss: 0.7204 - val_accuracy: 0.7455
Epoch 8/10
7/7 [==============================] - 24s 3s/step - loss: 0.2477 - accuracy:
0.9500 - val_loss: 0.8036 - val_accuracy: 0.6545
Epoch 9/10
7/7 [==============================] - 22s 3s/step - loss: 0.1833 - accuracy:
0.9136 - val_loss: 0.9213 - val_accuracy: 0.7091
Epoch 10/10
7/7 [==============================] - 23s 3s/step - loss: 0.0621 - accuracy:
0.9864 - val_loss: 0.9028 - val_accuracy: 0.6182
```

[57]:

```
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc}')
```

```
2/2 [==============================] - 2s 848ms/step - loss: 1.0020 - accuracy:
0.6909
Test accuracy: 0.6909090876579285
```

[58]:

```
# Make predictions on the test set
predictions = model.predict(X_test)
predicted_labels = np.argmax(predictions, axis=1)
```

```
2/2 [==============================] - 2s 814ms/step
```

- Train accuracy : 0.98
- Val accuracy : 0.74
- Test accuracy : 0.69

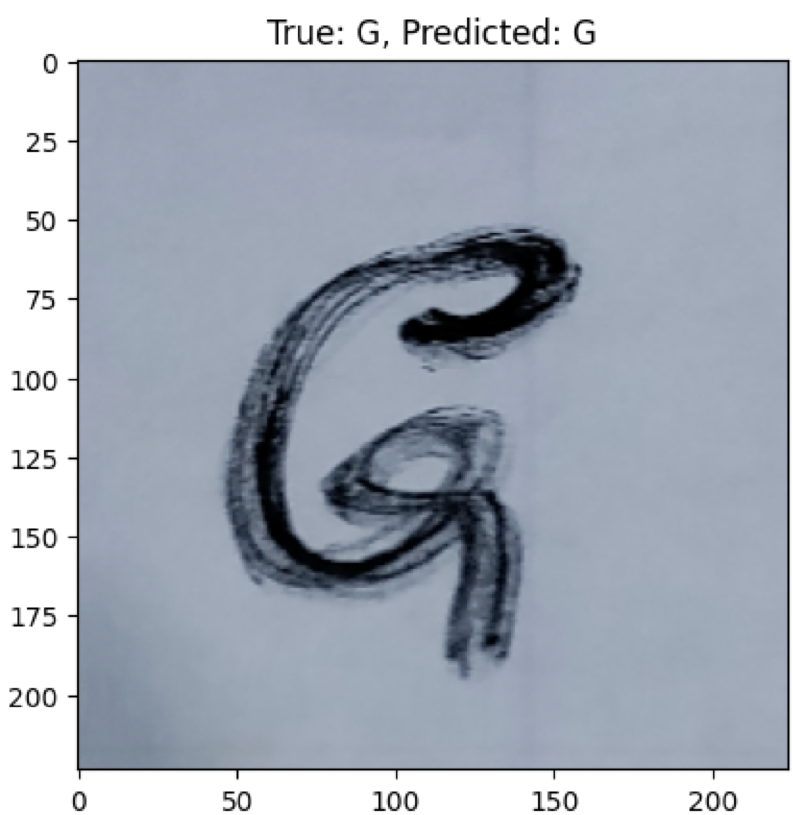
[59]:

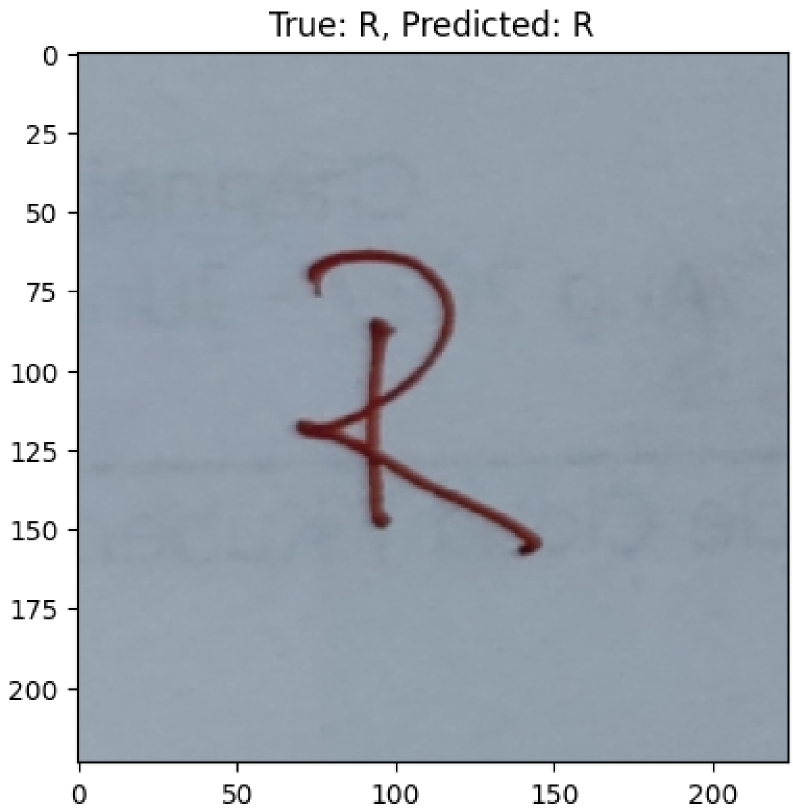
```
# Compute the confusion matrix
conf_matrix = confusion_matrix(y_test, predicted_labels)
print('Confusion Matrix:')
print(conf_matrix)
```

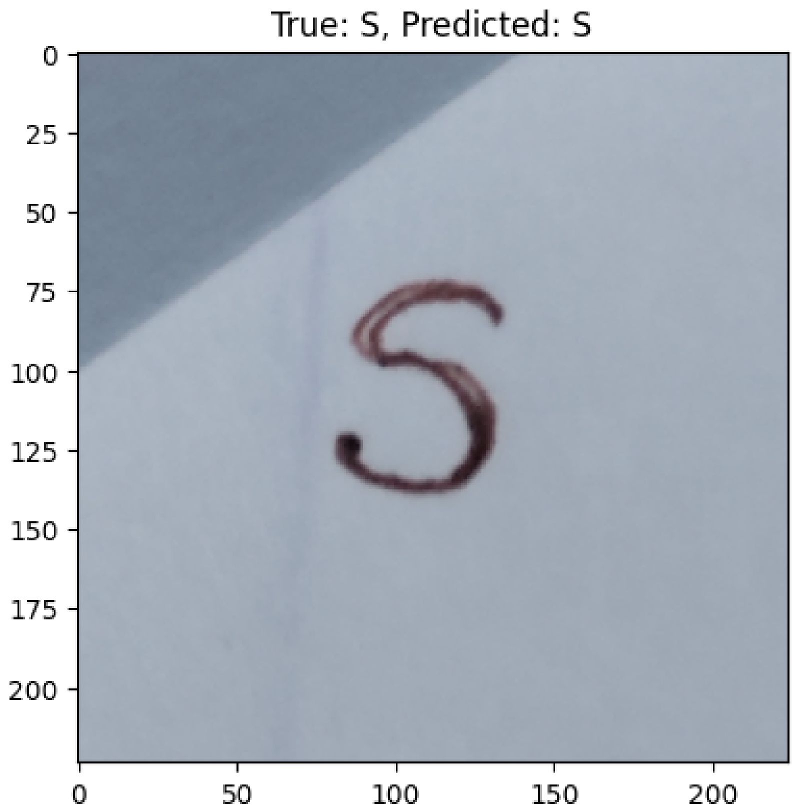
```
Confusion Matrix:
[[13  2  4]
 [ 3 12  4]
 [ 3  1 13]]
```

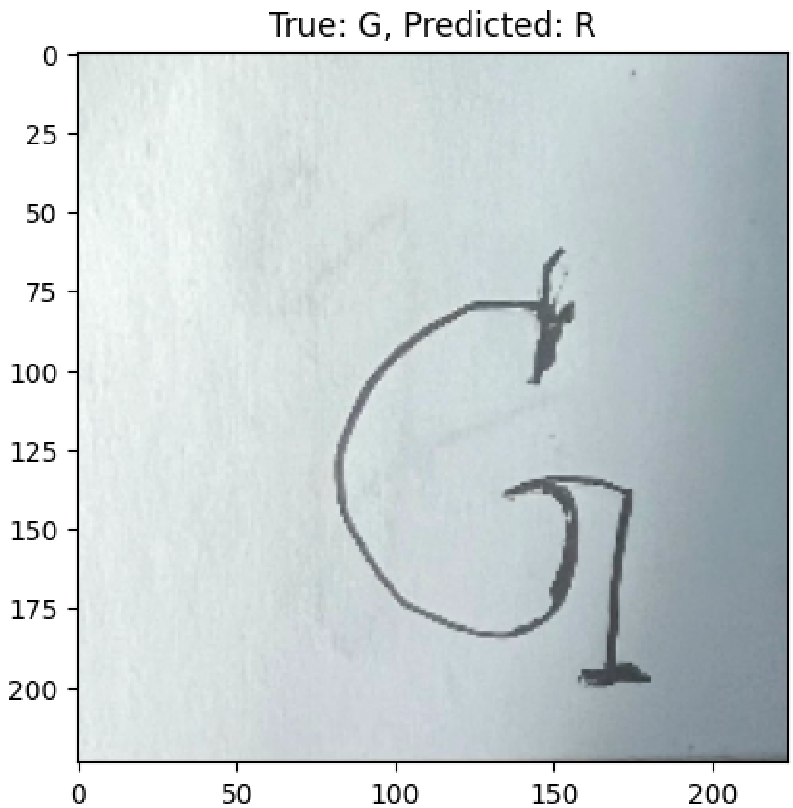
The model correctly predicted 13 instances for Class 1, 12 instances for Class 2, and 13 instances for Class 3. However, it exhibited misclassifications, such as 2 instances of Class 1 being predicted as Class 2, 3 instances of Class 2 being predicted as Class 1, and 4 instances of Class 3 being predicted as Class 1.

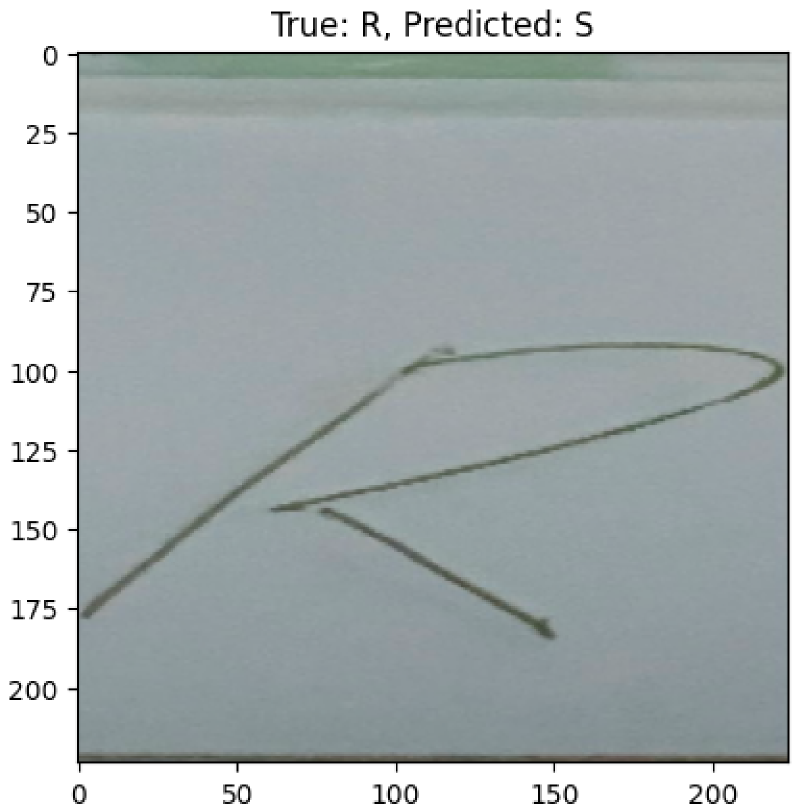
[60]:
```
numeric_to_letter_mapping = {0: 'G', 1: 'R', 2: 'S'}
# Display a few correct predictions
correct_indices = np.where(predicted_labels == y_test)[0]
for i in range(min(3, len(correct_indices))):
    index = correct_indices[i]
    plt.imshow(X_test[index])
    true_label_name = numeric_to_letter_mapping[y_test[index]]  # Use your␣
↪mapping from previous steps
    predicted_label_name = numeric_to_letter_mapping[predicted_labels[index]]
    plt.title(f'True: {true_label_name}, Predicted: {predicted_label_name}')
    plt.show()

# Display a few incorrect predictions
incorrect_indices = np.where(predicted_labels != y_test)[0]
for i in range(min(3, len(incorrect_indices))):
    index = incorrect_indices[i]
    plt.imshow(X_test[index])
    true_label_name = numeric_to_letter_mapping[y_test[index]]  # Use your␣
↪mapping from previous steps
    predicted_label_name = numeric_to_letter_mapping[predicted_labels[index]]
    plt.title(f'True: {true_label_name}, Predicted: {predicted_label_name}')
    plt.show()
```

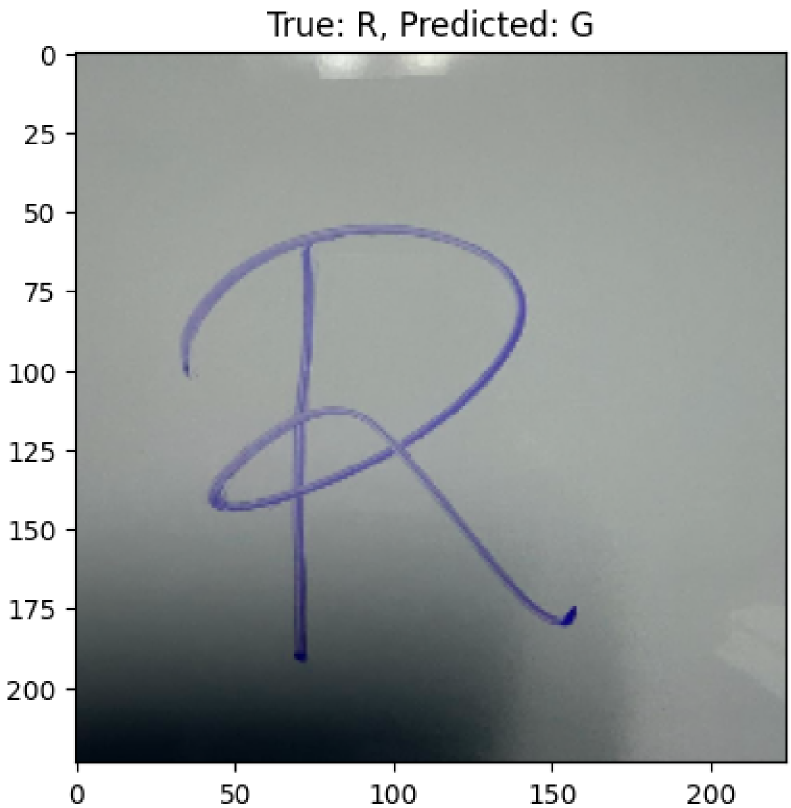



True: R, Predicted: R
0
25
50
75
100
125
150
175
200
0
50
100
150
200


True: S, Predicted: S
0
25
50
75
100
125
150
175
200
0
50
100
150
200


True: G, Predicted: R
0
25
50
75
100
125
150
175
200
0
50
100
150
200


True: R, Predicted: S
0
25
50
75
100
125
150
175
200
0
50
100
150
200


True: R, Predicted: G
0
25
50
75
100
125
150
175
200
0
50
100
150
200