In [1]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import glob
from PIL import Image
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import  LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
#Intialised ImageDataGenerator from tensorflow module to preprocess the image
datagen = ImageDataGenerator(
        rotation_range=15,
        shear_range=0.2,
        horizontal_flip=True,
        featurewise_center=True,
        width_shift_range=0.05,
        height_shift_range=0.05,
        zoom_range=0.1,
        fill_mode='nearest')
```

In [3]:
```
# Path to all images files of dataset - each folder - alphabet contains 100 images
Patha="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/a/*.jpg"
Pathb="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/b/*.jpg"
Pathc="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/c/*.jpg"
Pathd="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/d/*.jpg"
Pathe="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/e/*.jpg"
Pathf="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/f/*.jpg"
Pathg="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/g/*.jpg"
Pathh="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/h/*.jpg"
Pathi="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/i/*.jpg"
Pathj="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/j/*.jpg"
Pathk="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/k/*.jpg"
Pathl="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/l/*.jpg"
Pathm="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/m/*.jpg"
Pathn="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/n/*.jpg"
Patho="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/o/*.jpg"
Pathp="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/p/*.jpg"
```

```
Pathq="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/q/*.jpg"
Pathr="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/r/*.jpg"
Paths="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/s/*.jpg"
Patht="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/t/*.jpg"
Pathu="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/u/*.jpg"
Pathv="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/v/*.jpg"
Pathw="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/w/*.jpg"
Pathx="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/x/*.jpg"
Pathy="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/y/*.jpg"
Pathz="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/dataset/z/*.jpg"
```

In [4]:

```
# import the data from each alphabet folder and club into final dataframe
def importing_data(path):
    sample = []
    for filename in glob.glob(path):
        img = Image.open(filename,'r')
        img = img.resize((128,128))
        sample.append(img)

    return sample

data_a = importing_data(Patha)
data_b = importing_data(Pathb)
data_c = importing_data(Pathc)
data_d = importing_data(Pathd)
data_e = importing_data(Pathe)
data_f = importing_data(Pathf)
data_g = importing_data(Pathg)
data_h = importing_data(Pathh)
data_i = importing_data(Pathi)
data_j = importing_data(Pathj)
data_k = importing_data(Pathk)
data_l = importing_data(Pathl)
data_m = importing_data(Pathm)
data_n = importing_data(Pathn)
data_o = importing_data(Patho)
data_p = importing_data(Pathp)
data_q = importing_data(Pathq)
data_r = importing_data(Pathr)
data_s = importing_data(Paths)
data_t = importing_data(Patht)
data_u = importing_data(Pathu)
data_v = importing_data(Pathv)
```

```
data_w = importing_data(Pathw)
data_x = importing_data(Pathx)
data_y = importing_data(Pathy)
data_z = importing_data(Pathz)
```

In [5]:
```
def data_import(a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z):
    df_data_a = pd.DataFrame({'image':a, 'label': 'a'})
    df_data_b = pd.DataFrame({'image':b, 'label': 'b'})
    df_data_c = pd.DataFrame({'image':c, 'label': 'c'})
    df_data_d = pd.DataFrame({'image':d, 'label': 'd'})
    df_data_e = pd.DataFrame({'image':e, 'label': 'e'})
    df_data_f = pd.DataFrame({'image':f, 'label': 'f'})
    df_data_g = pd.DataFrame({'image':g, 'label': 'g'})
    df_data_h = pd.DataFrame({'image':h, 'label': 'h'})
    df_data_i = pd.DataFrame({'image':i, 'label': 'i'})
    df_data_j = pd.DataFrame({'image':j, 'label': 'j'})
    df_data_k = pd.DataFrame({'image':k, 'label': 'k'})
    df_data_l = pd.DataFrame({'image':l, 'label': 'l'})
    df_data_m = pd.DataFrame({'image':m, 'label': 'm'})
    df_data_n = pd.DataFrame({'image':n, 'label': 'n'})
    df_data_o = pd.DataFrame({'image':o, 'label': 'o'})
    df_data_p = pd.DataFrame({'image':p, 'label': 'p'})
    df_data_q = pd.DataFrame({'image':q, 'label': 'q'})
    df_data_r = pd.DataFrame({'image':r, 'label': 'r'})
    df_data_s = pd.DataFrame({'image':s, 'label': 's'})
    df_data_t = pd.DataFrame({'image':t, 'label': 't'})
    df_data_u = pd.DataFrame({'image':u, 'label': 'u'})
    df_data_v = pd.DataFrame({'image':v, 'label': 'v'})
    df_data_w = pd.DataFrame({'image':w, 'label': 'w'})
    df_data_x = pd.DataFrame({'image':x, 'label': 'x'})
    df_data_y = pd.DataFrame({'image':y, 'label': 'y'})
    df_data_z = pd.DataFrame({'image':z, 'label': 'z'})



    final_data = [df_data_a, df_data_b, df_data_c, df_data_d, df_data_e, df_data_f, df_data_g, df_data_h, df_data_i, df_
    final_data = pd.concat(final_data)
    all_data = final_data['image']
    labels = final_data['label']
    all_data=np.stack(all_data,axis=0)
    labels = LabelBinarizer().fit_transform(labels)
    return all_data,labels
```

```
dataset,labels   = data_import(data_a,data_b,data_c,data_d,data_e,data_f,data_g,data_h,data_i,data_j,data_k,data_l,data_
```

In [6]:
```
dataset.shape
```

Out[6]:
```
(2600, 128, 128)
```

In [7]:
```
dataset=dataset.reshape(2600,128,128,1)
```

In [8]:
```
#augmentation method while preprocessing images
def augmentation(dataset,labels,counts):
    augs=[]
    augs_data=[]
    augs_labels=[]
    i=0
    for batch in datagen.flow(dataset,labels,
                              batch_size=260):
        i += 1
        augs.append(batch)
        if i>counts:
            break
    augs_data=[]
    for i in range(counts):
        data=augs[i][0]
        augs_data.append(data)
    x = np.vstack(augs_data)
    x = x/255.0

    for i in range(counts):
        label=augs[i][1]
        augs_labels.append(label)
    y = np.vstack(augs_labels)
    return x, y
```

In [9]:
```
#augment all the images in dataset
x,y = augmentation(dataset,labels,15)
y = np.where(y==1)[1]

from tensorflow.keras.utils import to_categorical
y_onehot = to_categorical(y)
```

```
x_train,x_test,y_train,y_test = train_test_split(x,y_onehot,test_size=0.26,random_state=2021)
```

In [10]:
```
#preprocess and train the data with cnn model layering
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Input, MaxPooling2D, Conv2D, Dropout
from tensorflow.keras import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

model = Sequential()
model.add(Conv2D(12, (5, 5), activation='relu', padding='same', input_shape=(128, 128, 1)))
model.add(MaxPooling2D(2,2))
model.add(Conv2D(24, (5, 5), activation='relu', padding='same',
                 kernel_regularizer =tf.keras.regularizers.l1(l=0.01)))
model.add(MaxPooling2D(2,2))

model.add(Conv2D(36, (5, 5), activation='relu', padding='same',
                 kernel_regularizer =tf.keras.regularizers.l2(l=0.01)))
model.add(MaxPooling2D(2,2))
model.add(Dropout(0.2))


model.add(Conv2D(48, (5, 5), activation='relu',
                 kernel_regularizer =tf.keras.regularizers.l2(l=0.01)))
model.add(MaxPooling2D(2,2))
model.add(Dropout(0.3))
model.add(Flatten())

m = model.output
m = Dense(120, activation = "relu")(m)
m = Dense(100, activation = "relu")(m)
m = Dropout(0.2)(m)
m = Dense(60, activation = "relu")(m)
m = Dense(60, activation = "relu")(m)
m = Dropout(0.2)(m)
final_layer =  Dense(26, activation = "softmax")(m)


cnn_model = Model(inputs=model.input, outputs=final_layer)
```

```
cnn_model.summary()

cnn_model.compile(optimizer=Adam(learning_rate=0.0001),loss='binary_crossentropy',metrics=['accuracy'])

callback = EarlyStopping(monitor='val_loss',patience=5,
                         restore_best_weights=True)

history=cnn_model.fit(x_train, y_train, validation_split=0.2,
                  epochs=50 , batch_size=64,callbacks=[callback])
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_input (InputLayer)   [(None, 128, 128, 1)]     0

 conv2d (Conv2D)             (None, 128, 128, 12)      312

 max_pooling2d (MaxPooling2D  (None, 64, 64, 12)       0
 )

 conv2d_1 (Conv2D)           (None, 64, 64, 24)        7224

 max_pooling2d_1 (MaxPooling  (None, 32, 32, 24)       0
 2D)

 conv2d_2 (Conv2D)           (None, 32, 32, 36)        21636

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 36)       0
 2D)

 dropout (Dropout)           (None, 16, 16, 36)        0

 conv2d_3 (Conv2D)           (None, 12, 12, 48)        43248

 max_pooling2d_3 (MaxPooling  (None, 6, 6, 48)         0
 2D)

 dropout_1 (Dropout)         (None, 6, 6, 48)          0

 flatten (Flatten)           (None, 1728)              0

 dense (Dense)               (None, 120)               207480

 dense_1 (Dense)             (None, 100)               12100

 dropout_2 (Dropout)         (None, 100)               0

 dense_2 (Dense)             (None, 60)                6060

 dense_3 (Dense)             (None, 60)                3660

 dropout_3 (Dropout)         (None, 60)                0
```

```
 dense_4 (Dense)             (None, 26)                1586

=================================================================
Total params: 303,306
Trainable params: 303,306
Non-trainable params: 0
_________________________________________________________________
Epoch 1/50
37/37 [==============================] - 18s 446ms/step - loss: 4.1377 - accuracy: 0.0342 - val_loss: 3.8877 - val_accu
racy: 0.0363
Epoch 2/50
37/37 [==============================] - 18s 491ms/step - loss: 3.6440 - accuracy: 0.0364 - val_loss: 3.3415 - val_accu
racy: 0.0398
Epoch 3/50
37/37 [==============================] - 19s 498ms/step - loss: 3.1745 - accuracy: 0.0394 - val_loss: 2.8767 - val_accu
racy: 0.0398
Epoch 4/50
37/37 [==============================] - 17s 461ms/step - loss: 2.7742 - accuracy: 0.0347 - val_loss: 2.5300 - val_accu
racy: 0.0363
Epoch 5/50
37/37 [==============================] - 18s 475ms/step - loss: 2.4476 - accuracy: 0.0399 - val_loss: 2.2349 - val_accu
racy: 0.0363
Epoch 6/50
37/37 [==============================] - 17s 463ms/step - loss: 2.1613 - accuracy: 0.0399 - val_loss: 1.9744 - val_accu
racy: 0.0363
Epoch 7/50
37/37 [==============================] - 17s 467ms/step - loss: 1.9081 - accuracy: 0.0373 - val_loss: 1.7405 - val_accu
racy: 0.0363
Epoch 8/50
37/37 [==============================] - 17s 468ms/step - loss: 1.6753 - accuracy: 0.0381 - val_loss: 1.5240 - val_accu
racy: 0.0363
Epoch 9/50
37/37 [==============================] - 19s 510ms/step - loss: 1.4692 - accuracy: 0.0342 - val_loss: 1.3309 - val_accu
racy: 0.0363
Epoch 10/50
37/37 [==============================] - 19s 512ms/step - loss: 1.2832 - accuracy: 0.0399 - val_loss: 1.1546 - val_accu
racy: 0.0363
Epoch 11/50
37/37 [==============================] - 17s 470ms/step - loss: 1.1113 - accuracy: 0.0342 - val_loss: 0.9948 - val_accu
racy: 0.0363
Epoch 12/50
37/37 [==============================] - 19s 509ms/step - loss: 0.9564 - accuracy: 0.0347 - val_loss: 0.8506 - val_accu
```

```
racy: 0.0381
Epoch 13/50
37/37 [==============================] - 18s 479ms/step - loss: 0.8171 - accuracy: 0.0407 - val_loss: 0.7224 - val_accu
racy: 0.0398
Epoch 14/50
37/37 [==============================] - 17s 450ms/step - loss: 0.6925 - accuracy: 0.0381 - val_loss: 0.6102 - val_accu
racy: 0.0398
Epoch 15/50
37/37 [==============================] - 17s 470ms/step - loss: 0.5851 - accuracy: 0.0347 - val_loss: 0.5070 - val_accu
racy: 0.0398
Epoch 16/50
37/37 [==============================] - 19s 509ms/step - loss: 0.4887 - accuracy: 0.0433 - val_loss: 0.4209 - val_accu
racy: 0.0398
Epoch 17/50
37/37 [==============================] - 18s 496ms/step - loss: 0.4102 - accuracy: 0.0351 - val_loss: 0.3482 - val_accu
racy: 0.0398
Epoch 18/50
37/37 [==============================] - 19s 508ms/step - loss: 0.3402 - accuracy: 0.0433 - val_loss: 0.2881 - val_accu
racy: 0.0398
Epoch 19/50
37/37 [==============================] - 18s 500ms/step - loss: 0.2855 - accuracy: 0.0373 - val_loss: 0.2408 - val_accu
racy: 0.0398
Epoch 20/50
37/37 [==============================] - 17s 453ms/step - loss: 0.2437 - accuracy: 0.0407 - val_loss: 0.2058 - val_accu
racy: 0.0398
Epoch 21/50
37/37 [==============================] - 17s 466ms/step - loss: 0.2133 - accuracy: 0.0412 - val_loss: 0.1831 - val_accu
racy: 0.0398
Epoch 22/50
37/37 [==============================] - 18s 499ms/step - loss: 0.1974 - accuracy: 0.0377 - val_loss: 0.1732 - val_accu
racy: 0.0398
Epoch 23/50
37/37 [==============================] - 17s 457ms/step - loss: 0.1910 - accuracy: 0.0373 - val_loss: 0.1704 - val_accu
racy: 0.0346
Epoch 24/50
37/37 [==============================] - 19s 509ms/step - loss: 0.1884 - accuracy: 0.0373 - val_loss: 0.1687 - val_accu
racy: 0.0346
Epoch 25/50
37/37 [==============================] - 18s 482ms/step - loss: 0.1865 - accuracy: 0.0368 - val_loss: 0.1679 - val_accu
racy: 0.0346
Epoch 26/50
37/37 [==============================] - 17s 462ms/step - loss: 0.1838 - accuracy: 0.0472 - val_loss: 0.1672 - val_accu
racy: 0.0346
```

```
Epoch 27/50
37/37 [==============================] - 17s 461ms/step - loss: 0.1835 - accuracy: 0.0325 - val_loss: 0.1666 - val_accu
racy: 0.0398
Epoch 28/50
37/37 [==============================] - 17s 460ms/step - loss: 0.1829 - accuracy: 0.0360 - val_loss: 0.1662 - val_accu
racy: 0.0398
Epoch 29/50
37/37 [==============================] - 17s 465ms/step - loss: 0.1835 - accuracy: 0.0399 - val_loss: 0.1658 - val_accu
racy: 0.0398
Epoch 30/50
37/37 [==============================] - 17s 462ms/step - loss: 0.1810 - accuracy: 0.0407 - val_loss: 0.1656 - val_accu
racy: 0.0398
Epoch 31/50
37/37 [==============================] - 17s 462ms/step - loss: 0.1814 - accuracy: 0.0429 - val_loss: 0.1653 - val_accu
racy: 0.0398
Epoch 32/50
37/37 [==============================] - 17s 470ms/step - loss: 0.1810 - accuracy: 0.0351 - val_loss: 0.1657 - val_accu
racy: 0.0398
Epoch 33/50
37/37 [==============================] - 18s 488ms/step - loss: 0.1790 - accuracy: 0.0468 - val_loss: 0.1656 - val_accu
racy: 0.0346
Epoch 34/50
37/37 [==============================] - 19s 508ms/step - loss: 0.1793 - accuracy: 0.0390 - val_loss: 0.1650 - val_accu
racy: 0.0346
Epoch 35/50
37/37 [==============================] - 20s 535ms/step - loss: 0.1790 - accuracy: 0.0364 - val_loss: 0.1649 - val_accu
racy: 0.0346
Epoch 36/50
37/37 [==============================] - 20s 542ms/step - loss: 0.1775 - accuracy: 0.0477 - val_loss: 0.1648 - val_accu
racy: 0.0346
Epoch 37/50
37/37 [==============================] - 18s 486ms/step - loss: 0.1791 - accuracy: 0.0351 - val_loss: 0.1646 - val_accu
racy: 0.0346
Epoch 38/50
37/37 [==============================] - 19s 508ms/step - loss: 0.1782 - accuracy: 0.0338 - val_loss: 0.1645 - val_accu
racy: 0.0346
Epoch 39/50
37/37 [==============================] - 19s 521ms/step - loss: 0.1778 - accuracy: 0.0472 - val_loss: 0.1645 - val_accu
racy: 0.0346
Epoch 40/50
37/37 [==============================] - 18s 491ms/step - loss: 0.1774 - accuracy: 0.0364 - val_loss: 0.1644 - val_accu
racy: 0.0346
Epoch 41/50
```

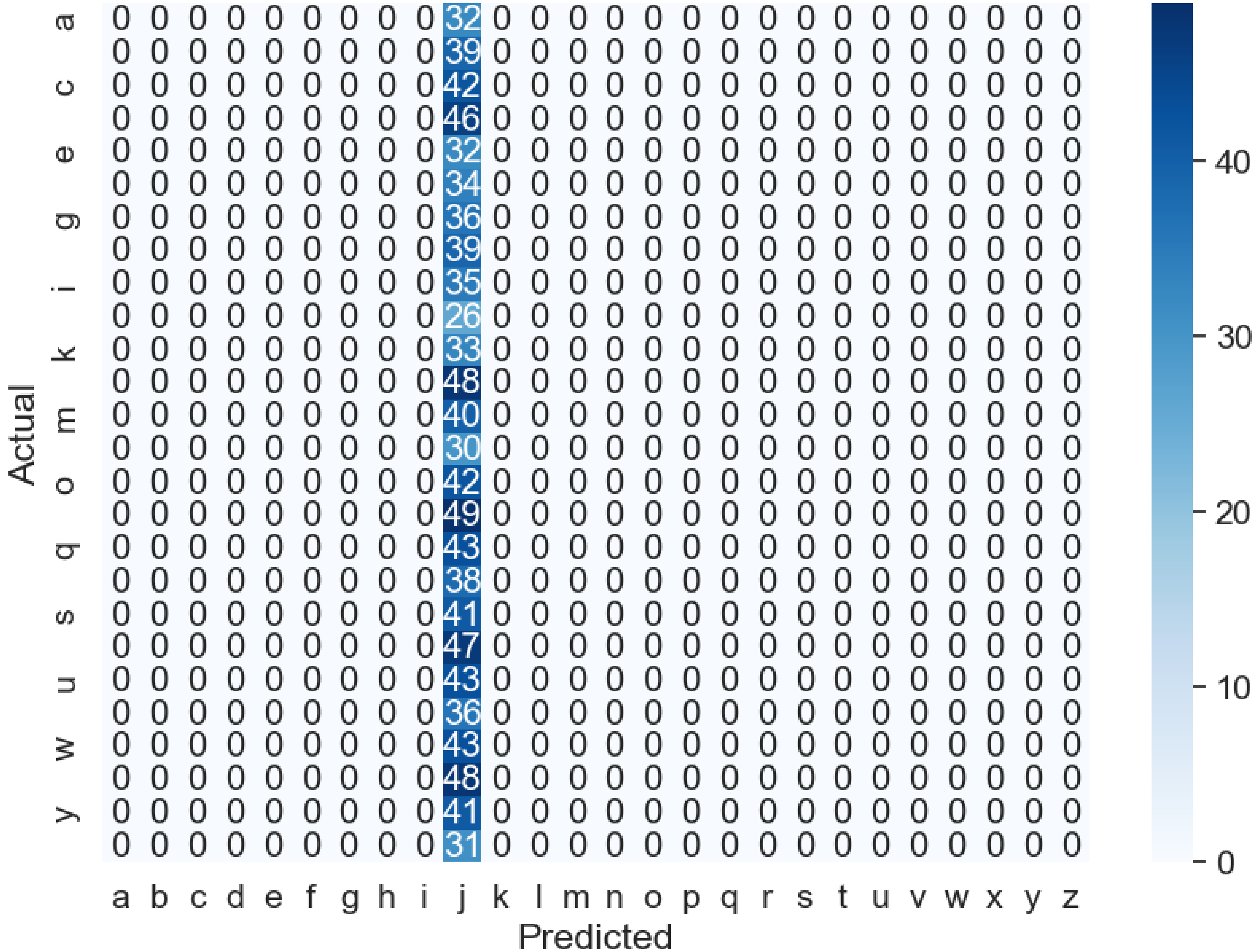
```
37/37 [==============================] - 19s 505ms/step - loss: 0.1767 - accuracy: 0.0407 - val_loss: 0.1644 - val_accu
racy: 0.0346
Epoch 42/50
37/37 [==============================] - 19s 512ms/step - loss: 0.1763 - accuracy: 0.0464 - val_loss: 0.1643 - val_accu
racy: 0.0346
Epoch 43/50
37/37 [==============================] - 20s 536ms/step - loss: 0.1770 - accuracy: 0.0394 - val_loss: 0.1644 - val_accu
racy: 0.0346
Epoch 44/50
37/37 [==============================] - 20s 531ms/step - loss: 0.1763 - accuracy: 0.0360 - val_loss: 0.1645 - val_accu
racy: 0.0346
Epoch 45/50
37/37 [==============================] - 19s 517ms/step - loss: 0.1759 - accuracy: 0.0433 - val_loss: 0.1644 - val_accu
racy: 0.0346
Epoch 46/50
37/37 [==============================] - 19s 526ms/step - loss: 0.1762 - accuracy: 0.0347 - val_loss: 0.1643 - val_accu
racy: 0.0346
Epoch 47/50
37/37 [==============================] - 21s 569ms/step - loss: 0.1759 - accuracy: 0.0420 - val_loss: 0.1643 - val_accu
racy: 0.0346
Epoch 48/50
37/37 [==============================] - 19s 523ms/step - loss: 0.1751 - accuracy: 0.0438 - val_loss: 0.1643 - val_accu
racy: 0.0346
Epoch 49/50
37/37 [==============================] - 20s 550ms/step - loss: 0.1756 - accuracy: 0.0373 - val_loss: 0.1642 - val_accu
racy: 0.0346
Epoch 50/50
37/37 [==============================] - 20s 545ms/step - loss: 0.1760 - accuracy: 0.0338 - val_loss: 0.1641 - val_accu
racy: 0.0346
```

In [11]:

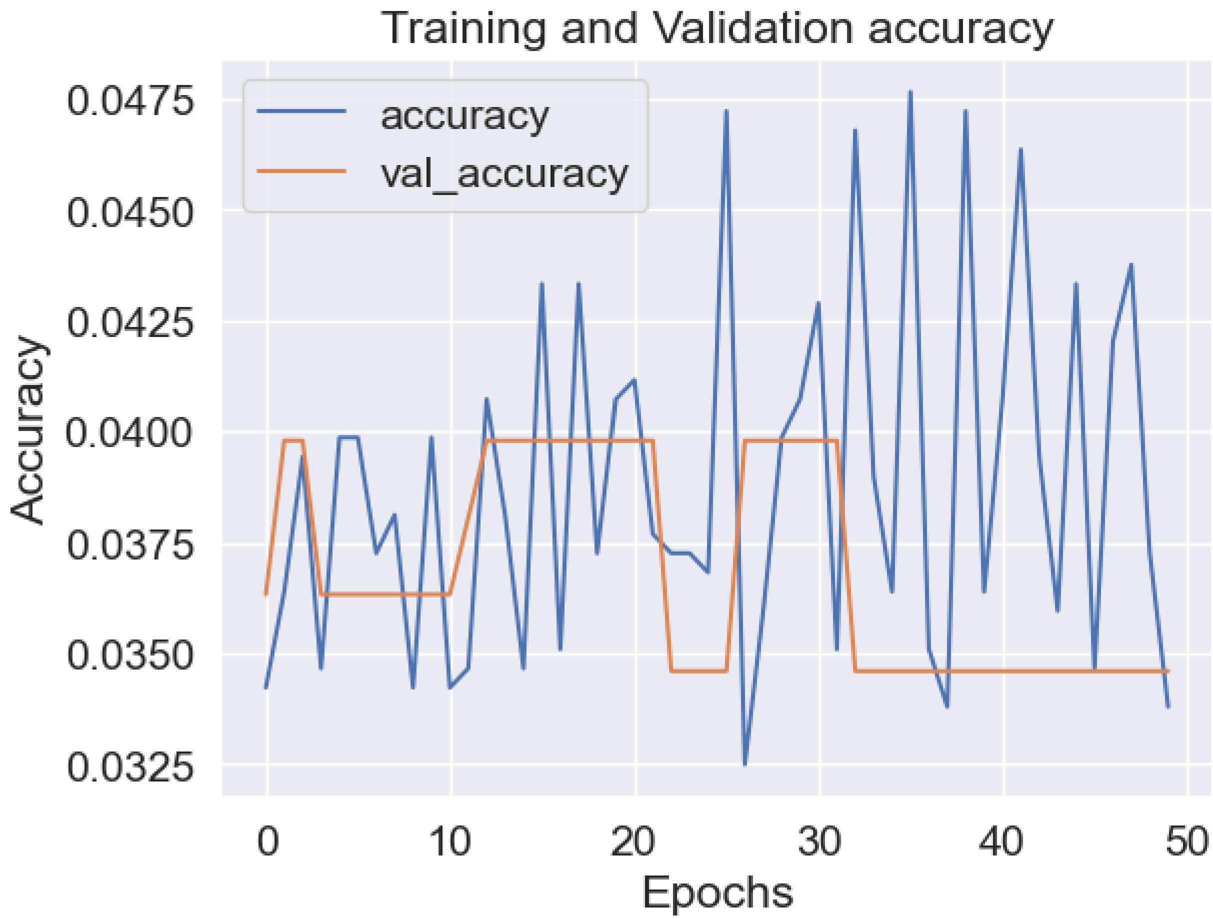
```
#get dataset and fit into algo and  confusion matrix, classification report
import seaborn as sn
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

predictions = cnn_model.predict(x_test)
pred_labels = np.argmax(predictions, axis = 1)
tests=np.argmax(y_test,axis=1)

conf_mx = confusion_matrix(tests, pred_labels)
conf_mx

heat_cm = pd.DataFrame(conf_mx, columns=("a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t
```

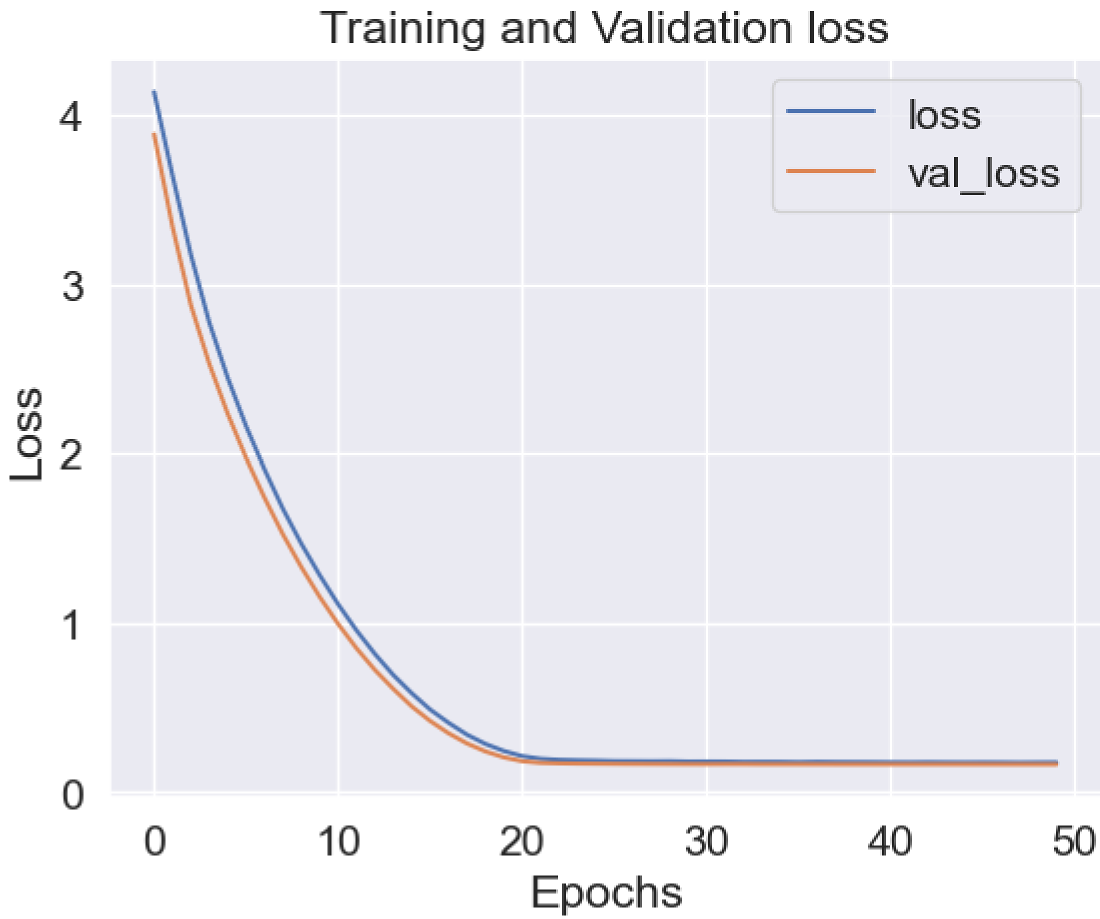
```
                                  "w","x","y","z"), index =("a","b","c","d","e","f","g","h","i","j","k","l","m","n","o"
                                                                     ,"p","q","r","s","t","u","v","w","x","y","z"))
heat_cm.index.name = 'Actual'
heat_cm.columns.name = 'Predicted'
plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)
sn.heatmap(heat_cm, cmap="Blues", annot=True, annot_kws={"size": 16},fmt='g')
plt.show()


print(classification_report(tests, pred_labels))

history_df = pd.DataFrame(history.history)

plt.plot(history_df.loc[:, ['accuracy']], label='accuracy')
plt.plot(history_df.loc[:, ['val_accuracy']], label='val_accuracy')
plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.plot(history_df.loc[:, ['loss']], label='loss')
plt.plot(history_df.loc[:, ['val_loss']], label='val_loss')

plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
32/32 [==============================] - 2s 58ms/step
```


Actual
a
c
e
g
i
k
m
o
q
s
u
w
y
a b c d e f g h i j k l m n o p q r s t u v w x y z
Predicted
0
10
20
30
40

	precision	recall	f1-score	support
0	0.00	0.00	0.00	32
1	0.00	0.00	0.00	39
2	0.00	0.00	0.00	42
3	0.00	0.00	0.00	46
4	0.00	0.00	0.00	32
5	0.00	0.00	0.00	34
6	0.00	0.00	0.00	36
7	0.00	0.00	0.00	39
8	0.00	0.00	0.00	35
9	0.03	1.00	0.05	26
10	0.00	0.00	0.00	33
11	0.00	0.00	0.00	48
12	0.00	0.00	0.00	40
13	0.00	0.00	0.00	30
14	0.00	0.00	0.00	42
15	0.00	0.00	0.00	49
16	0.00	0.00	0.00	43
17	0.00	0.00	0.00	38
18	0.00	0.00	0.00	41
19	0.00	0.00	0.00	47
20	0.00	0.00	0.00	43
21	0.00	0.00	0.00	36
22	0.00	0.00	0.00	43
23	0.00	0.00	0.00	48
24	0.00	0.00	0.00	41
25	0.00	0.00	0.00	31
accuracy			0.03	1014
macro avg	0.00	0.04	0.00	1014
weighted avg	0.00	0.03	0.00	1014


Training and Validation accuracy
accuracy
val_accuracy
Accuracy
Epochs
0.0475
0.0450
0.0425
0.0400
0.0375
0.0350
0.0325
0
10
20
30
40
50

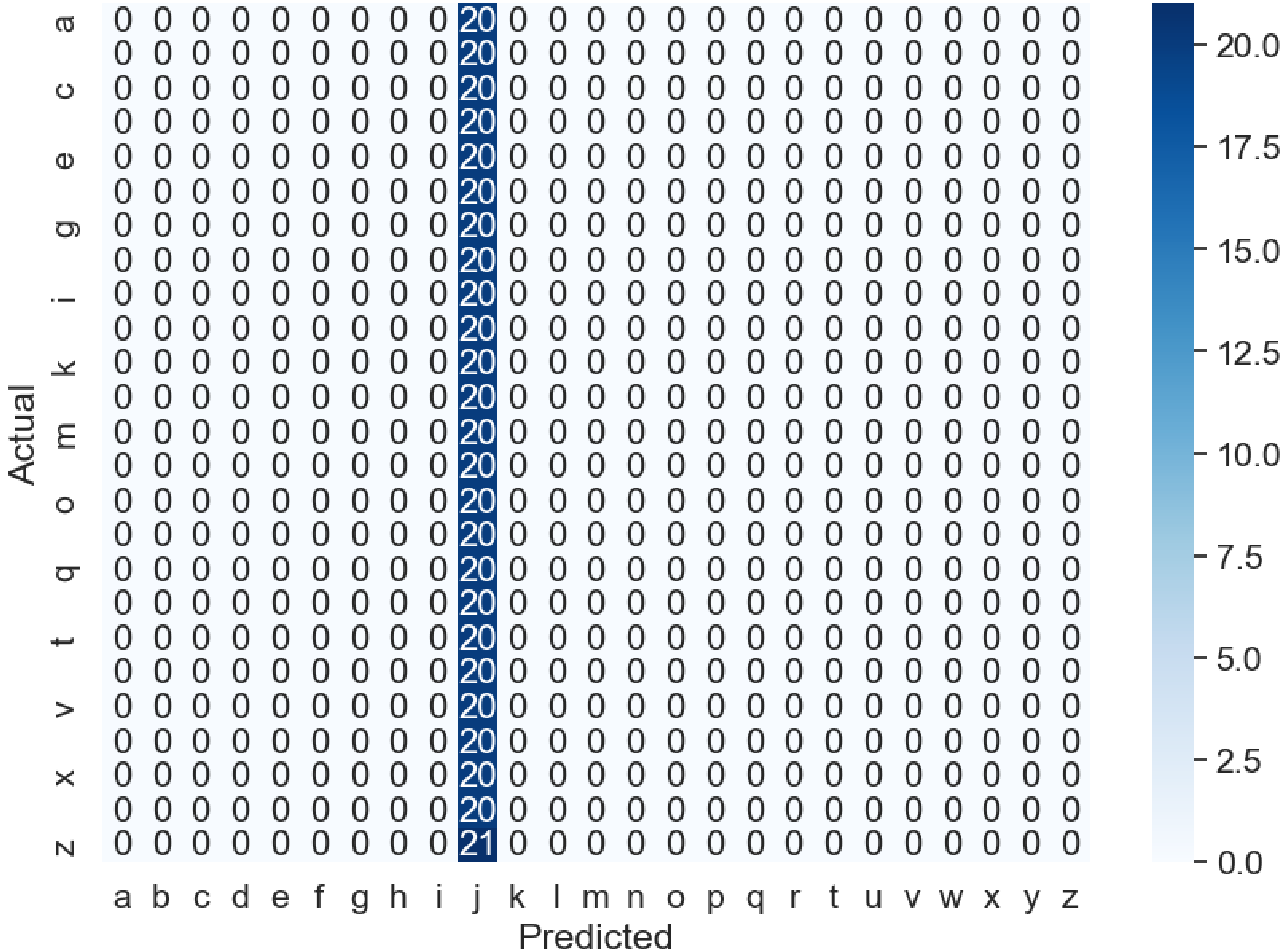


In [12]:
```
# all the test data is loaded in test folder, were imported and clubbed into final df
externala="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/a/*.jpg"
externalb="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/b/*.jpg"
externalc="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/c/*.jpg"
externald="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/d/*.jpg"
externale="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/e/*.jpg"
externalf="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/f/*.jpg"
externalg="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/g/*.jpg"
externalh="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/h/*.jpg"
externali="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/i/*.jpg"
externalj="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/j/*.jpg"
externalk="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/k/*.jpg"
externall="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/l/*.jpg"
externalm="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/m/*.jpg"
```

```
externaln="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/n/*.jpg"
externalo="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/o/*.jpg"
externalp="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/p/*.jpg"
externalq="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/q/*.jpg"
externalr="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/r/*.jpg"
externals="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/s/*.jpg"
externalt="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/t/*.jpg"
externalu="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/u/*.jpg"
externalv="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/v/*.jpg"
externalw="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/w/*.jpg"
externalx="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/x/*.jpg"
externaly="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/y/*.jpg"
externalz="C:/Users/thipp/Fall2022/CS5710_13469/Sign Language for Alphabets/test/z/*.jpg"

external_test_a = importing_data(externala)
external_test_b = importing_data(externalb)
external_test_c = importing_data(externalc)
external_test_d= importing_data(externald)
external_test_e= importing_data(externale)
external_test_f= importing_data(externalf)
external_test_g= importing_data(externalg)
external_test_h= importing_data(externalh)
external_test_i= importing_data(externali)
external_test_j= importing_data(externalj)
external_test_k= importing_data(externalk)
external_test_l= importing_data(externall)
external_test_m= importing_data(externalm)
external_test_n= importing_data(externaln)
external_test_o= importing_data(externalo)
external_test_p= importing_data(externalp)
external_test_q= importing_data(externalq)
external_test_r= importing_data(externalr)
external_test_s= importing_data(externals)
external_test_t= importing_data(externalt)
external_test_u= importing_data(externalu)
external_test_v= importing_data(externalv)
external_test_w= importing_data(externalw)
external_test_x= importing_data(externalx)
external_test_y= importing_data(externaly)
external_test_z= importing_data(externalz)


external_test_data,external_test_label   = data_import(external_test_a,external_test_b,external_test_c,external_test_d,
```

```
                                                       external_test_e,external_test_f,external_test_g,external_test_h,
                                                       external_test_i,external_test_j,external_test_k,external_test_l,
                                                       external_test_m,external_test_n,external_test_o,external_test_p,
                                                       external_test_q,external_test_r,external_test_s,external_test_t,
                                                       external_test_u,external_test_v,external_test_w,external_test_x,
                                                      external_test_y,external_test_z)

print(external_test_data.shape)
external_test_data=external_test_data.reshape(501,128,128,1)
external_test_data = external_test_data /255.0

external_test_pred = cnn_model.predict(external_test_data)
external_pred_labels = np.argmax(external_test_pred, axis = 1)
external_tests=np.argmax(external_test_label,axis=1)


external_conf_mx = confusion_matrix(external_tests, external_pred_labels)
heat_cm = pd.DataFrame(external_conf_mx,
                       columns=("a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","t","u","v","w"
                       ,index =("a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","t","u","v","w"
heat_cm.index.name = 'Actual'
heat_cm.columns.name = 'Predicted'
plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)
sn.heatmap(heat_cm, cmap="Blues", annot=True, annot_kws={"size": 16},fmt='g')
plt.show()
```

```
(501, 128, 128)
16/16 [==============================] - 1s 70ms/step
```



In [13]:

```
x_for_ml = model.predict(x_train)
x_test_ml = model.predict(x_test)
y_train_ml = np.where(y_train==1)[1]
y_test_ml = np.where(y_test==1)[1]
```

```
external_data_ml = model.predict(external_test_data)
external_label_ml = np.where(external_test_label==1)[1]
```

```
91/91 [==============================] - 6s 63ms/step
32/32 [==============================] - 2s 62ms/step
16/16 [==============================] - 1s 59ms/step
```

In [14]:
```
#common method to execute various machine learning algorithms and their performances
def machine_learning(algorithm):
    algorithm.fit(x_for_ml, y_train_ml)

    prediction_algo = algorithm.predict(x_test_ml)
    cm_algo = confusion_matrix(y_test_ml, prediction_algo)
    heat_cm = pd.DataFrame(cm_algo, columns=("a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p",
                                             "q","r","s","t","u","v","w","x","y","z"),
                                             index=("a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p"
                                                    ,"q","r","s","t","u","v","w","x","y","z"))
    heat_cm.index.name = 'Actual'
    heat_cm.columns.name = 'Predicted'
    plt.figure(figsize = (10,7))
    sn.set(font_scale=1.4)
    sn.heatmap(heat_cm, cmap="Blues", annot=True, annot_kws={"size": 16},fmt='g')
    cm_algo=plt.show()

    report = print(classification_report(y_test_ml, prediction_algo))
    algo_accuracy =  accuracy_score(y_test_ml, prediction_algo)

    prediction_self_algo = algorithm.predict(external_data_ml)
    cm_self_algo = confusion_matrix(external_label_ml, prediction_self_algo)
    algo_test_accuracy = accuracy_score(external_label_ml, prediction_self_algo)
    heat_cm = pd.DataFrame(cm_self_algo, columns=("a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","
                       index =("a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u",
    heat_cm.index.name = 'Actual'
    heat_cm.columns.name = 'Predicted'
    plt.figure(figsize = (10,7))
    sn.set(font_scale=1.4)
    sn.heatmap(heat_cm, cmap="Blues", annot=True, annot_kws={"size": 16},fmt='g')
    cm_self_algo=plt.show()
    return cm_algo, report, cm_self_algo, algo_accuracy, algo_test_accuracy
```

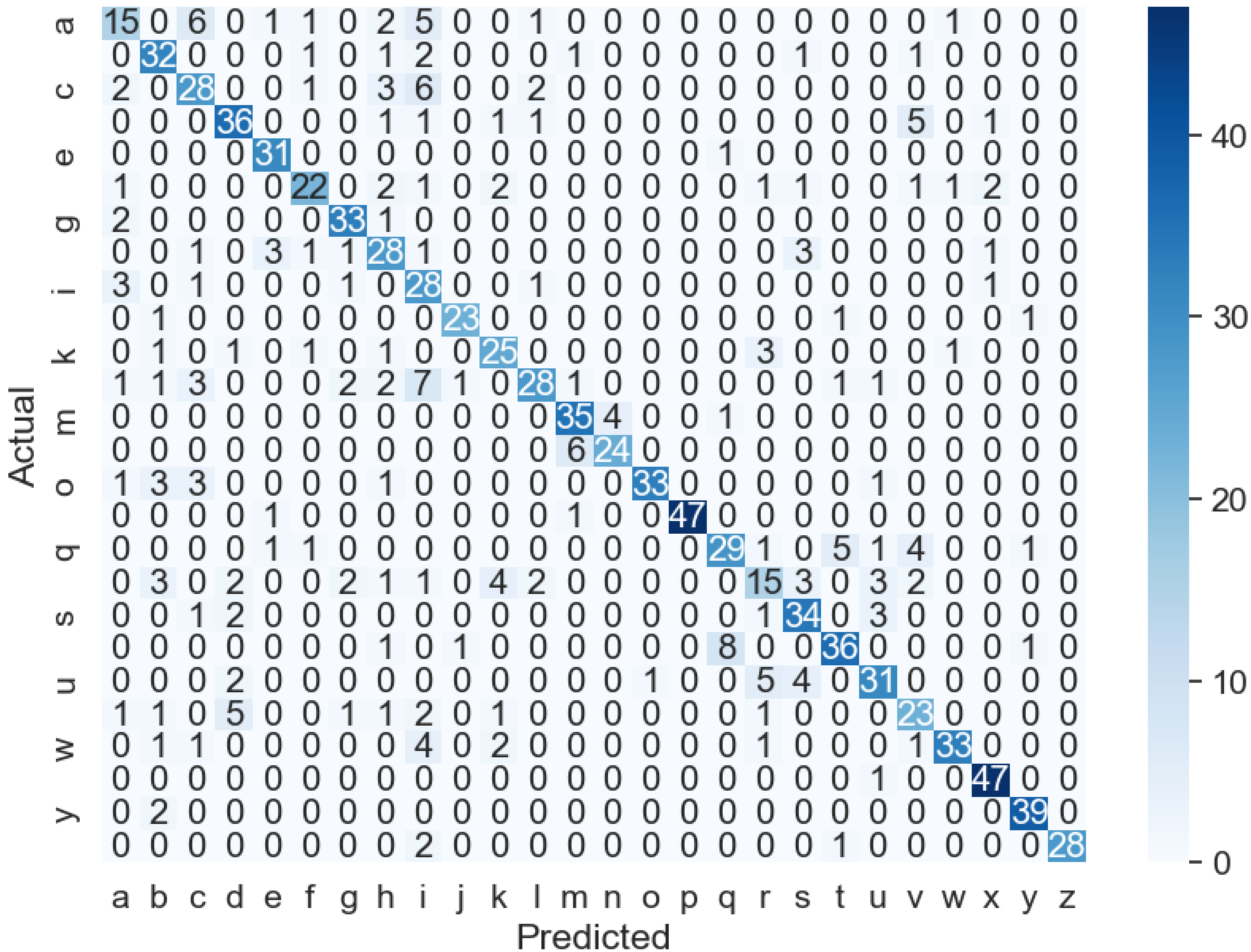
In [15]:
```
def machine_learning1(algorithm):
    #fit the training data into ML algorithm
    algorithm.fit(x_for_ml, y_train_ml)
```

```
    #Predict the data using test
    prediction_algo = algorithm.predict(x_test_ml)
    #confustion matrix
    cm_algo = confusion_matrix(y_test_ml, prediction_algo)
    #plotting the confusion matrix
    heat_cm = pd.DataFrame(cm_algo, columns=("a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p",
                                             "q","r","s","t","u","v","w","x","y","z"),
                                             index=("a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p"
                                                    ,"q","r","s","t","u","v","w","x","y","z"))
    heat_cm.index.name = 'Actual'
    heat_cm.columns.name = 'Predicted'
    plt.figure(figsize = (10,7))
    sn.set(font_scale=1.4)
    sn.heatmap(heat_cm, cmap="Blues", annot=True, annot_kws={"size": 16},fmt='g')
    cm_algo=plt.show()
    #classification report of training data
    report = print(classification_report(y_test_ml, prediction_algo))
    #accuracy of training data
    algo_accuracy =  accuracy_score(y_test_ml, prediction_algo)


    #predicting the external test dataset
    prediction_self_algo = algorithm.predict(external_data_ml)
    #test dataset confusion matrix
    cm_self_algo = confusion_matrix(external_label_ml, prediction_self_algo)
    #test dataset accuracy
    algo_test_accuracy = accuracy_score(external_label_ml, prediction_self_algo)
    heat_cm = pd.DataFrame(cm_self_algo, columns=("a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","
                           index =("a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","t","u","v",
    heat_cm.index.name = 'Actual'
    heat_cm.columns.name = 'Predicted'
    plt.figure(figsize = (10,7))
    sn.set(font_scale=1.4)
    sn.heatmap(heat_cm, cmap="Blues", annot=True, annot_kws={"size": 16},fmt='g')
    cm_self_algo=plt.show()
    return cm_algo, report, cm_self_algo, algo_accuracy, algo_test_accuracy
```

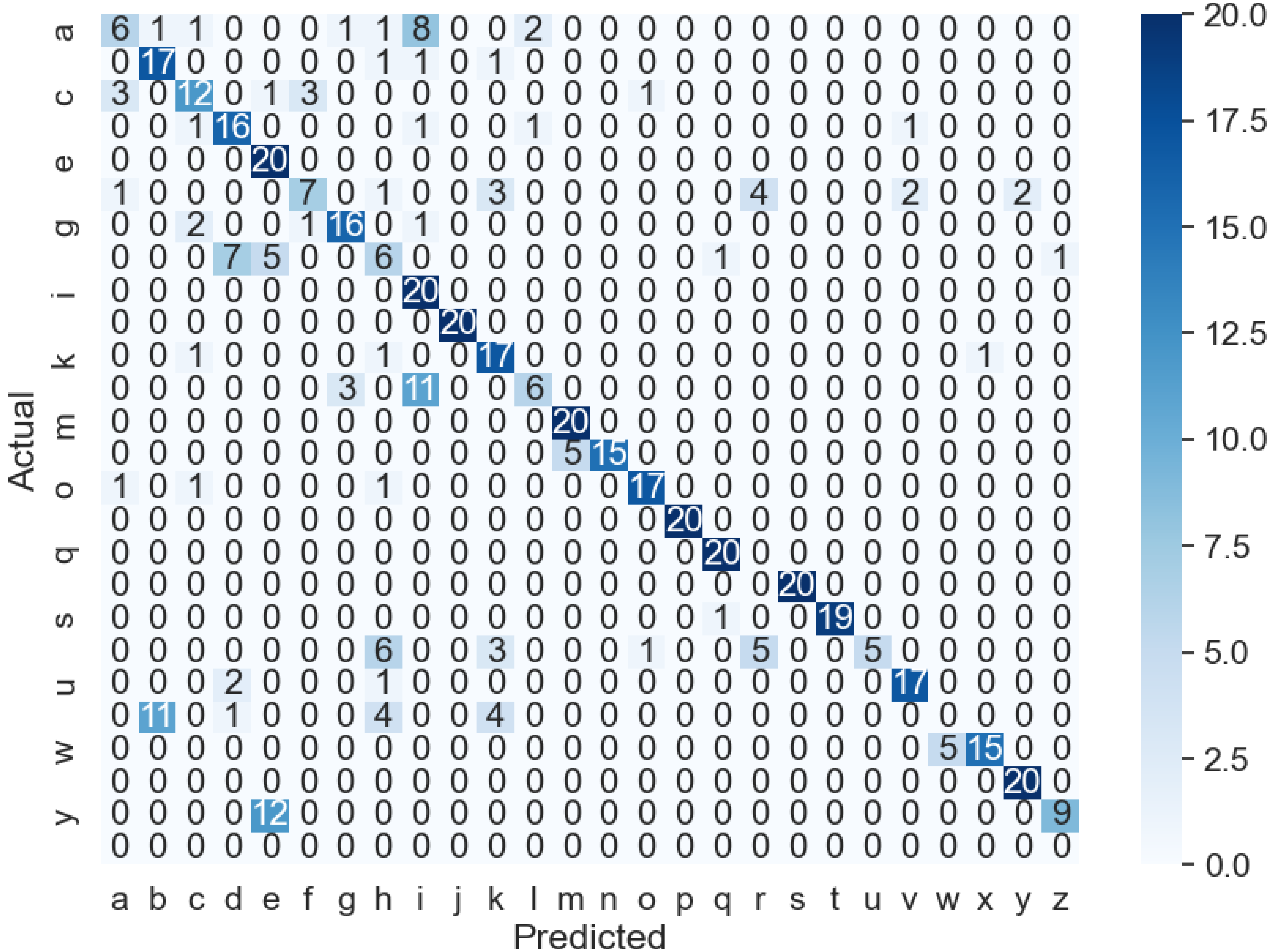
In [16]:

```
#XgBoost Algorithm
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
xgb = XGBClassifier()

xgboost = XGBClassifier(n_estimators=50,learning_rate=0.1,
```

```
                    max_depth=2,reg_lambda=0.1)

xgb_cm, xgb_report, xgb_external_cm, xgb_acc, xgb_ext_acc = machine_learning(xgboost)
```

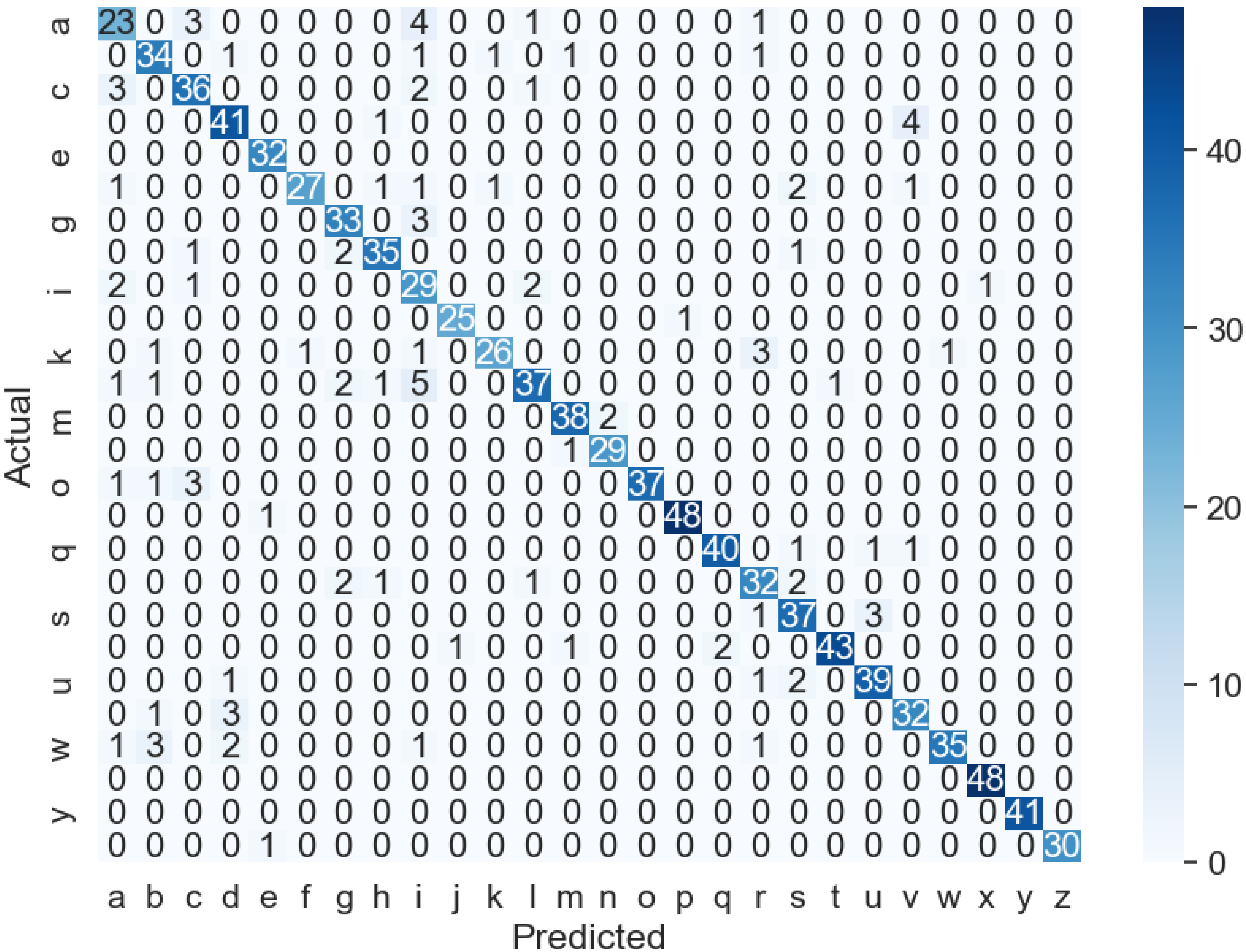


```
              precision    recall  f1-score   support

           0       0.58      0.47      0.52        32
           1       0.71      0.82      0.76        39
           2       0.64      0.67      0.65        42
           3       0.75      0.78      0.77        46
           4       0.84      0.97      0.90        32
           5       0.79      0.65      0.71        34
           6       0.82      0.92      0.87        36
           7       0.62      0.72      0.67        39
           8       0.47      0.80      0.59        35
           9       0.92      0.88      0.90        26
          10       0.71      0.76      0.74        33
          11       0.80      0.58      0.67        48
          12       0.80      0.88      0.83        40
          13       0.86      0.80      0.83        30
          14       0.97      0.79      0.87        42
          15       1.00      0.96      0.98        49
          16       0.74      0.67      0.71        43
          17       0.54      0.39      0.45        38
          18       0.74      0.83      0.78        41
          19       0.82      0.77      0.79        47
          20       0.76      0.72      0.74        43
          21       0.62      0.64      0.63        36
          22       0.92      0.77      0.84        43
          23       0.90      0.98      0.94        48
          24       0.93      0.95      0.94        41
          25       1.00      0.90      0.95        31

    accuracy                           0.77      1014
   macro avg       0.78      0.77      0.77      1014
weighted avg       0.78      0.77      0.77      1014
```

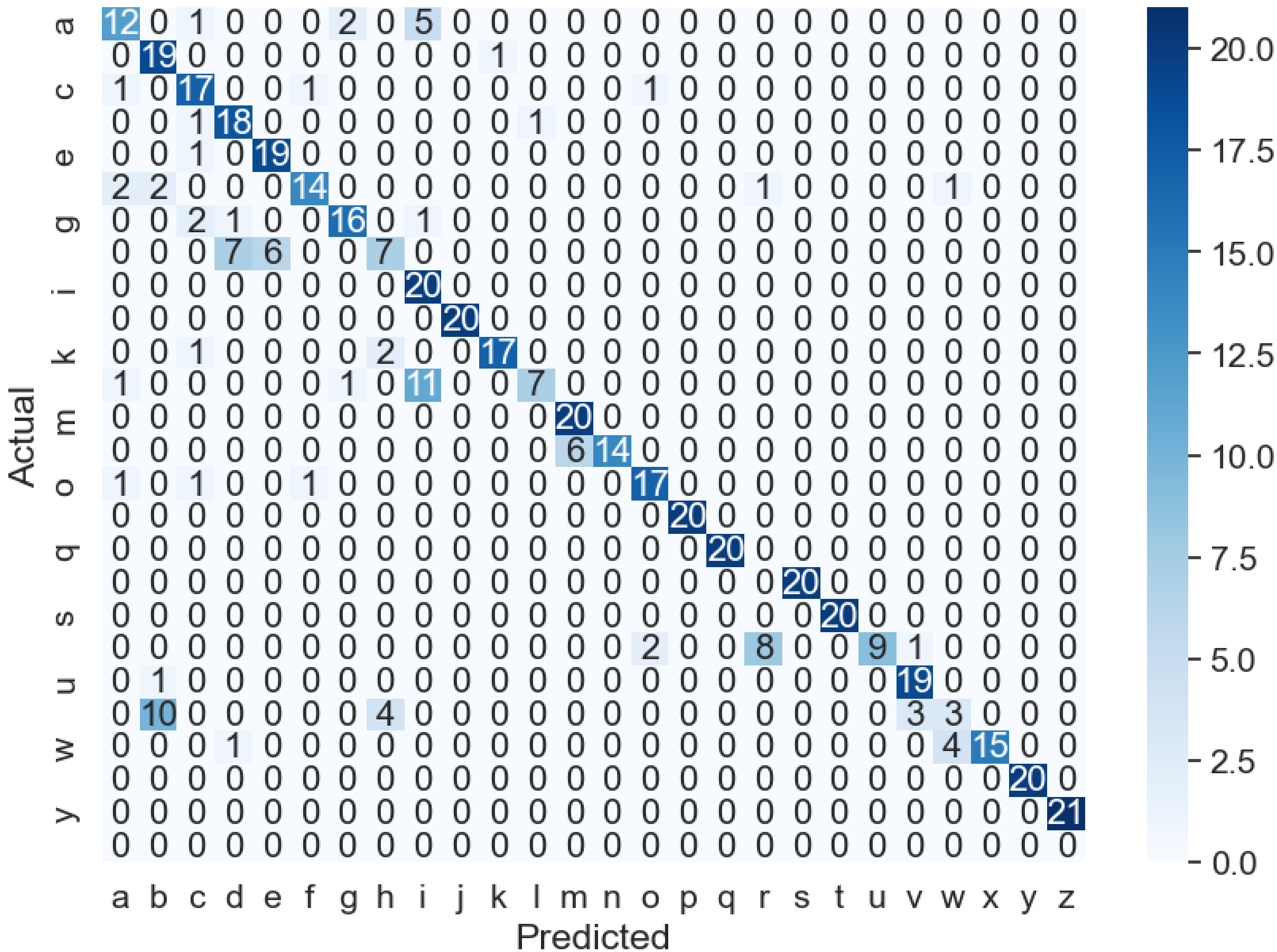


In [17]:

```
#%% LGBM
from lightgbm import LGBMClassifier
lgbm = LGBMClassifier(n_estimators=500,random_state=2021)

lgbm_cm, lgbm_report,lgbm_cm_external,lgb_acc, lgb_ext_acc = machine_learning(lgbm)
```

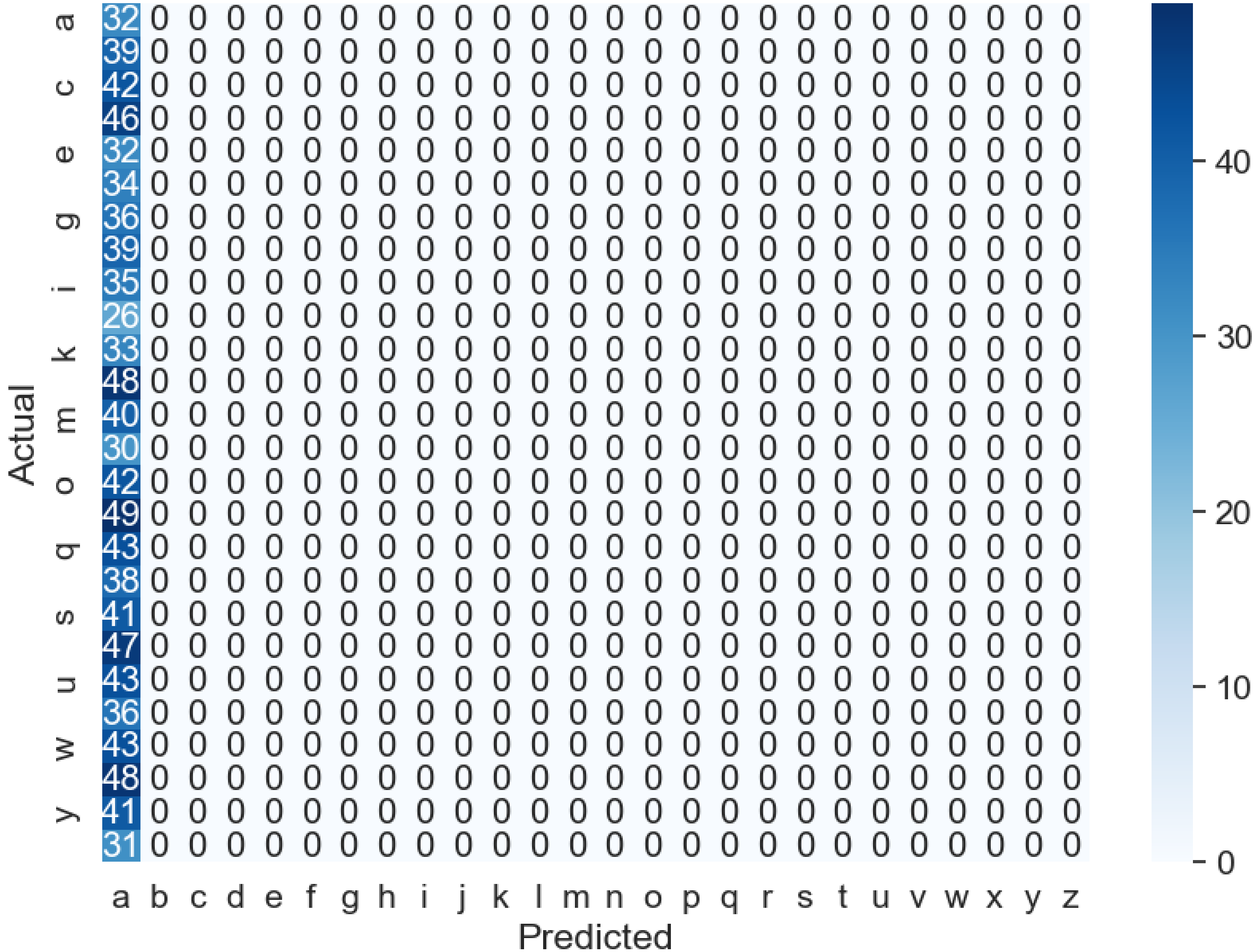

Actual
a c e g i k m o q s u w y
23 0 3 0 0 0 0 0 4 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 34 0 1 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
3 0 36 0 0 0 0 0 2 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 41 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0
0 0 0 0 32 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 27 0 1 1 0 1 0 0 0 0 0 0 0 2 0 0 1 0 0 0 0
0 0 0 0 0 0 33 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 2 35 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
2 0 1 0 0 0 0 0 29 0 0 2 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 25 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 1 0 0 1 0 26 0 0 0 0 0 0 3 0 0 0 0 1 0 0 0
1 1 0 0 0 0 2 1 5 0 0 37 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 38 2 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 29 0 0 0 0 0 0 0 0 0 0 0 0
1 1 3 0 0 0 0 0 0 0 0 0 0 0 37 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 48 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 40 0 1 0 1 1 0 0 0 0
0 0 0 0 0 0 2 1 0 0 0 1 0 0 0 0 0 32 2 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 37 0 3 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 2 0 0 43 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 0 39 0 0 0 0 0
0 1 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 32 0 0 0 0
1 3 0 2 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 35 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 48 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 41 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 30
a b c d e f g h i j k l m n o p q r s t u v w x y z
Predicted
40
30
20
10
0

	precision	recall	f1-score	support
0	0.72	0.72	0.72	32
1	0.83	0.87	0.85	39
2	0.82	0.86	0.84	42
3	0.85	0.89	0.87	46
4	0.94	1.00	0.97	32
5	0.96	0.79	0.87	34
6	0.85	0.92	0.88	36
7	0.90	0.90	0.90	39
8	0.62	0.83	0.71	35
9	0.96	0.96	0.96	26
10	0.93	0.79	0.85	33
11	0.88	0.77	0.82	48
12	0.93	0.95	0.94	40
13	0.94	0.97	0.95	30
14	1.00	0.88	0.94	42
15	0.98	0.98	0.98	49
16	0.95	0.93	0.94	43
17	0.80	0.84	0.82	38
18	0.82	0.90	0.86	41
19	0.98	0.91	0.95	47
20	0.91	0.91	0.91	43
21	0.84	0.89	0.86	36
22	0.97	0.81	0.89	43
23	0.98	1.00	0.99	48
24	1.00	1.00	1.00	41
25	1.00	0.97	0.98	31
accuracy			0.89	1014
macro avg	0.90	0.89	0.89	1014
weighted avg	0.90	0.89	0.90	1014

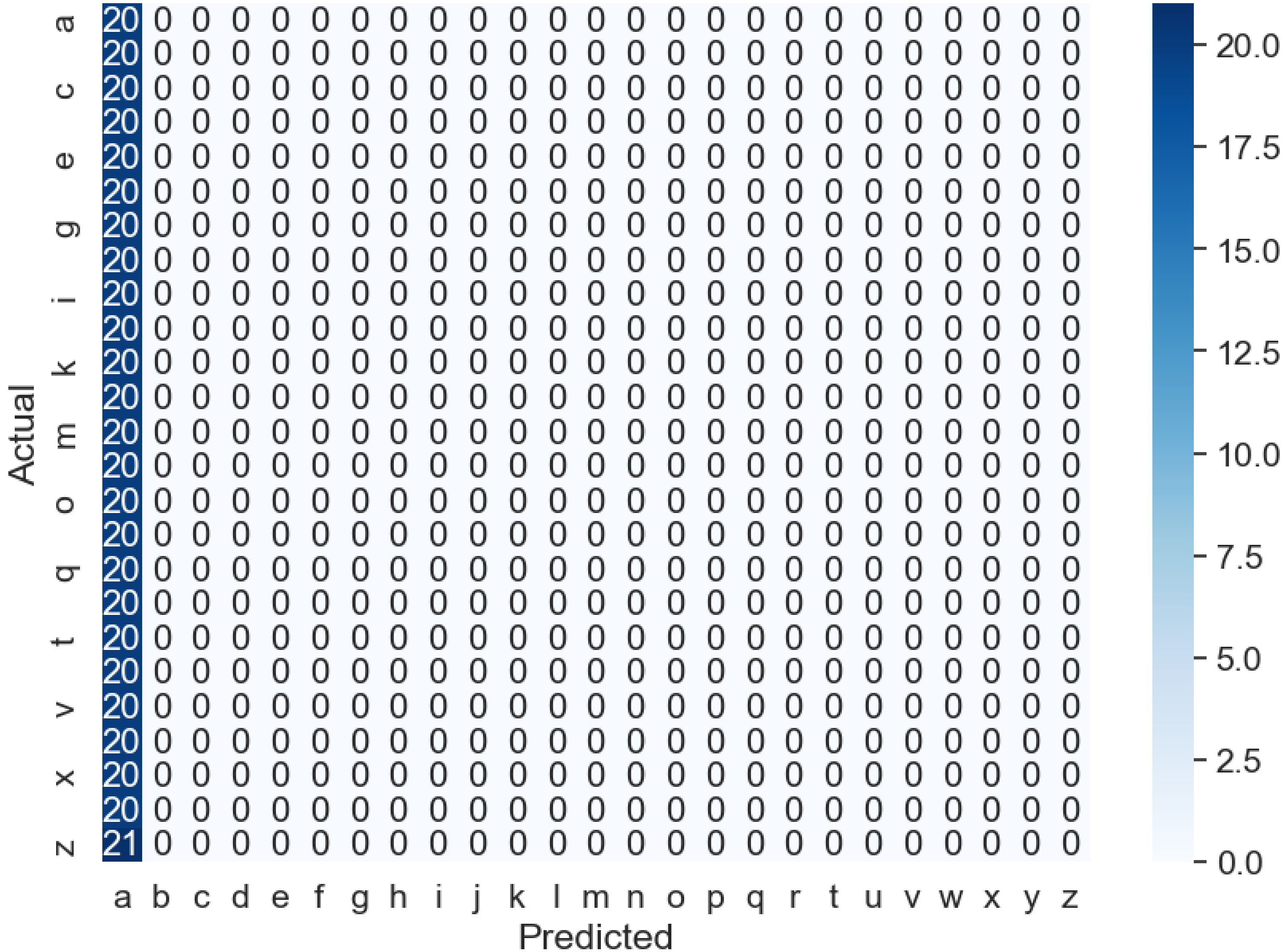


In [18]:

```python
#Support vector machine
from sklearn.svm import SVC
SVCClf = SVC(kernel = 'linear',gamma = 'scale', shrinking = False,)

SVCClf_cm, SVCClf_report,SVCClf_cm_external, svc_acc, svc_ext_acc = machine_learning1(SVCClf)
```

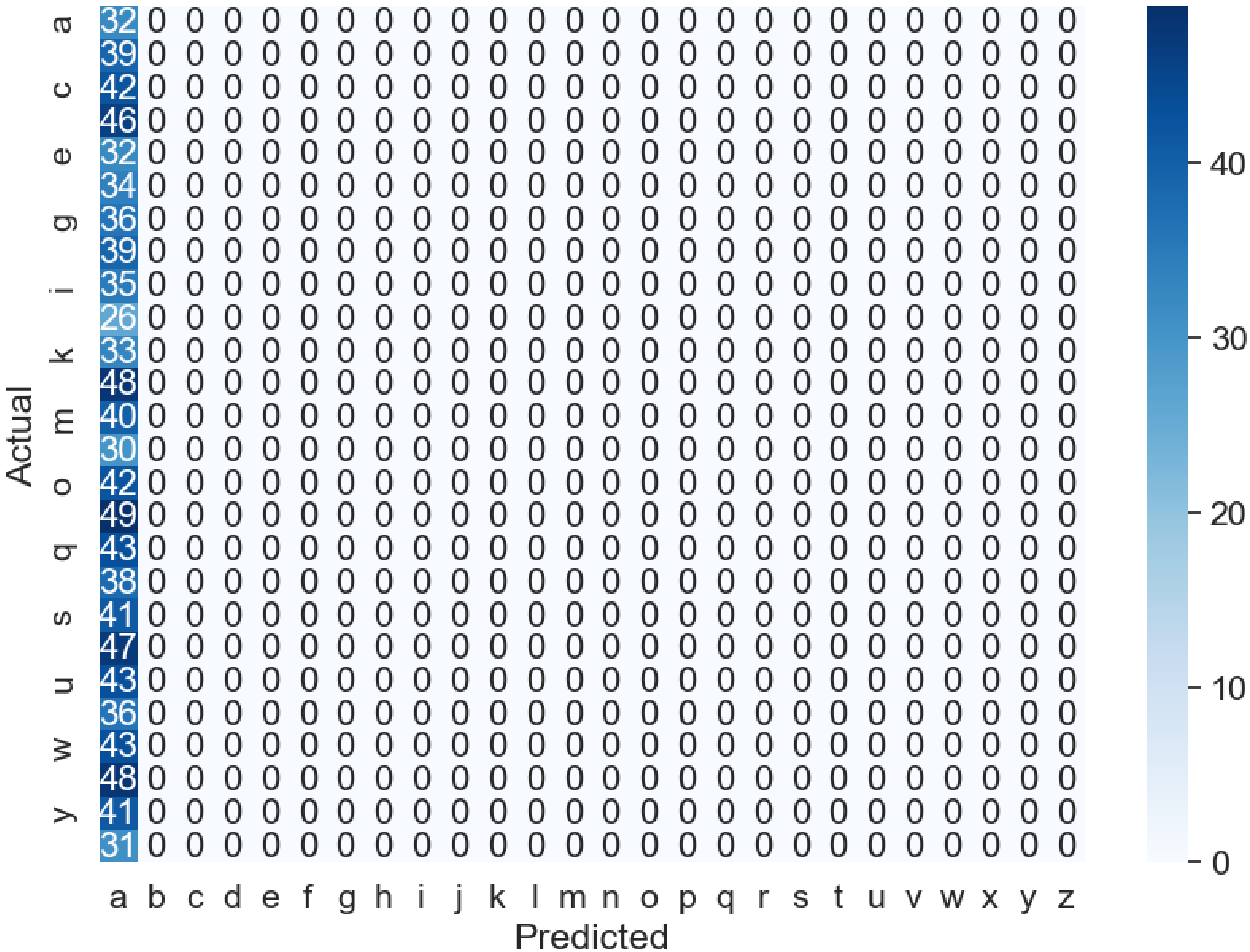

Actual
a
c
e
g
i
k
m
o
q
s
u
w
y
32
39
42
46
32
34
36
39
35
26
33
48
40
30
42
49
43
38
41
47
43
36
43
48
41
31
a b c d e f g h i j k l m n o p q r s t u v w x y z
Predicted
40
30
20
10
0

	precision	recall	f1-score	support
0	0.03	1.00	0.06	32
1	0.00	0.00	0.00	39
2	0.00	0.00	0.00	42
3	0.00	0.00	0.00	46
4	0.00	0.00	0.00	32
5	0.00	0.00	0.00	34
6	0.00	0.00	0.00	36
7	0.00	0.00	0.00	39
8	0.00	0.00	0.00	35
9	0.00	0.00	0.00	26
10	0.00	0.00	0.00	33
11	0.00	0.00	0.00	48
12	0.00	0.00	0.00	40
13	0.00	0.00	0.00	30
14	0.00	0.00	0.00	42
15	0.00	0.00	0.00	49
16	0.00	0.00	0.00	43
17	0.00	0.00	0.00	38
18	0.00	0.00	0.00	41
19	0.00	0.00	0.00	47
20	0.00	0.00	0.00	43
21	0.00	0.00	0.00	36
22	0.00	0.00	0.00	43
23	0.00	0.00	0.00	48
24	0.00	0.00	0.00	41
25	0.00	0.00	0.00	31
accuracy			0.03	1014
macro avg	0.00	0.04	0.00	1014
weighted avg	0.00	0.03	0.00	1014

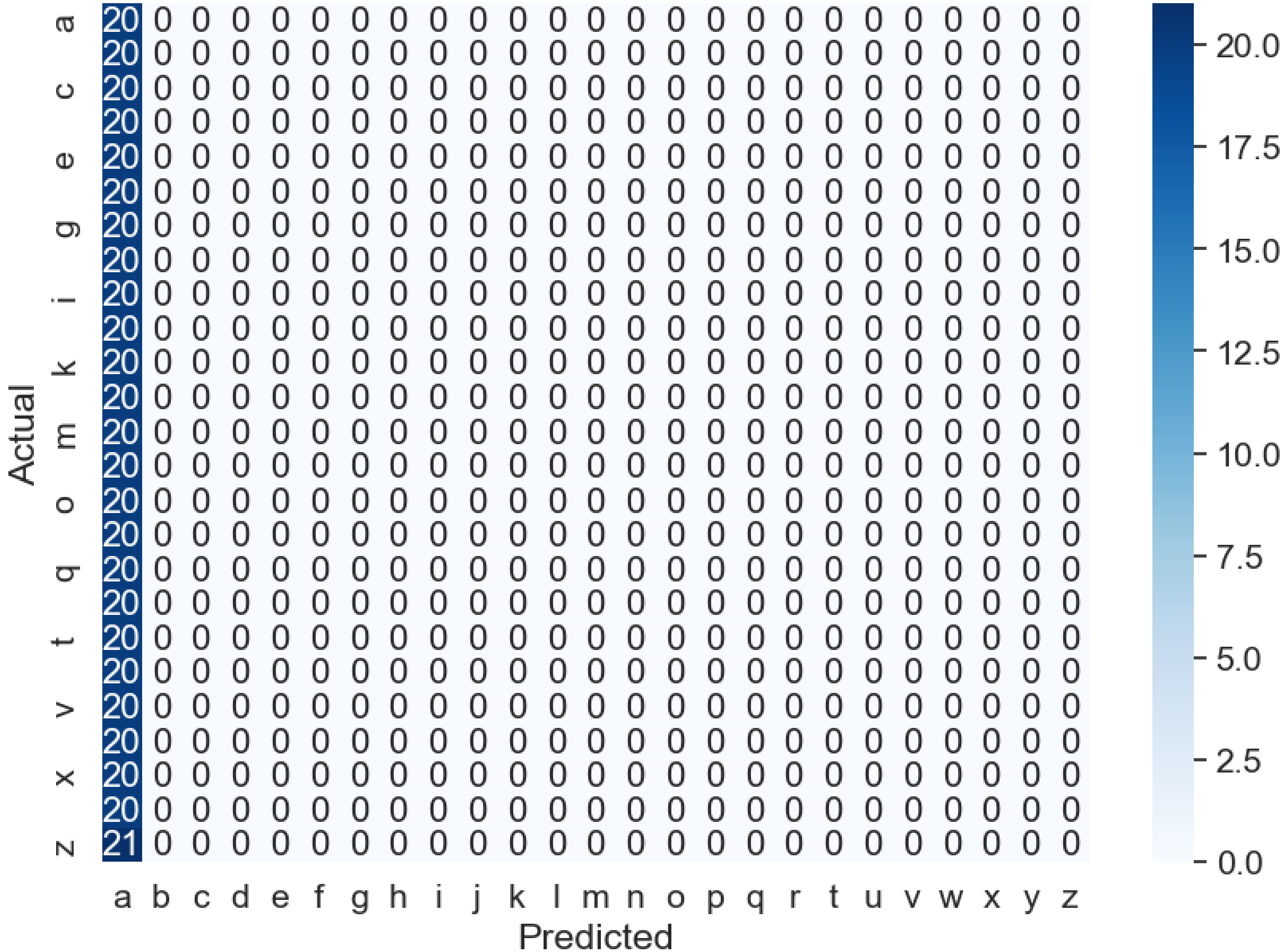


In [19]:

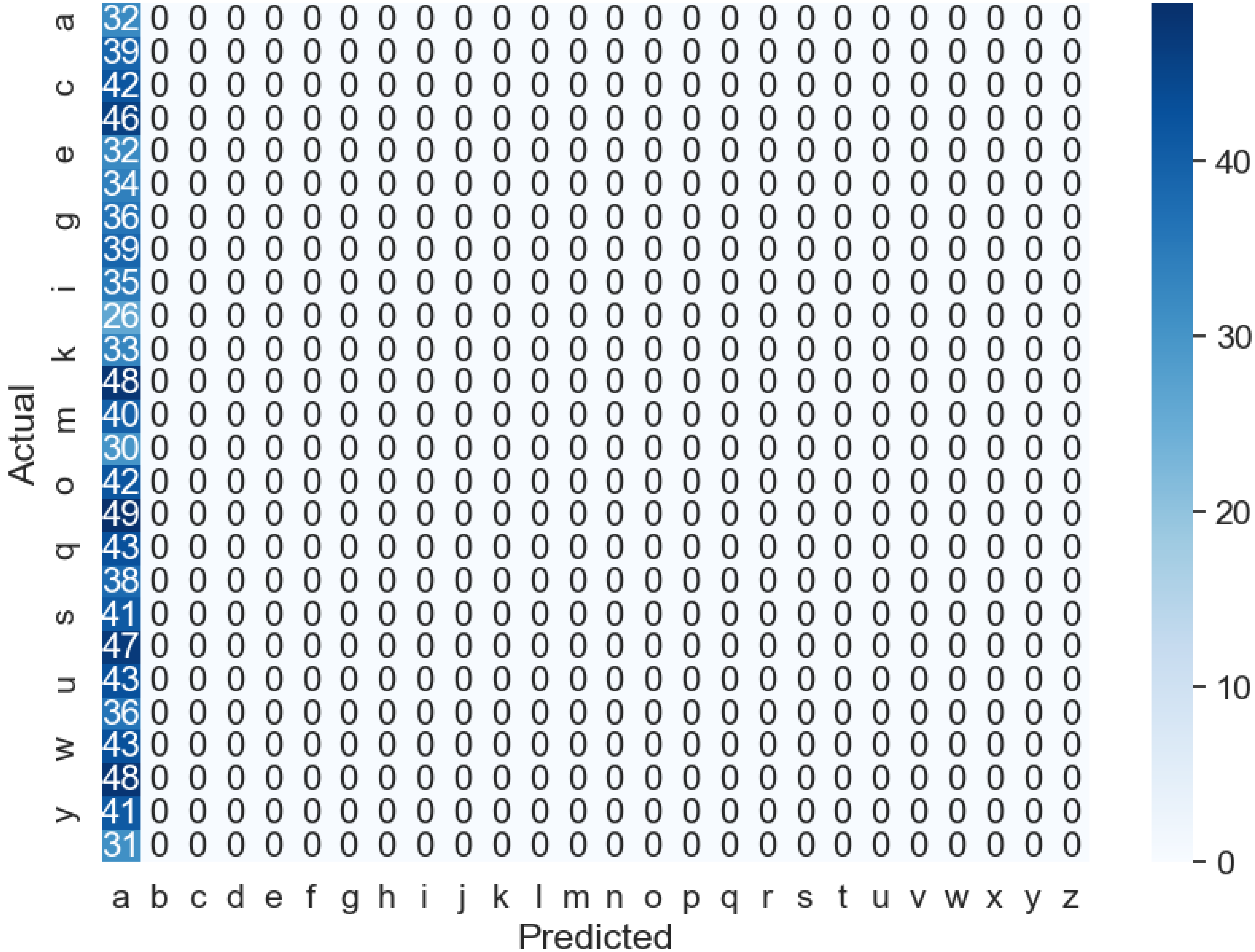
```
#Decision Tree
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(criterion="gini", random_state=42,max_depth=3, min_samples_leaf=5)
clf_cm, clf_report,clf_cm_external, clf_acc, clf_ext_acc = machine_learning1(clf)
```


Actual
a
c
e
g
i
k
m
o
q
s
u
w
y
a b c d e f g h i j k l m n o p q r s t u v w x y z
Predicted
40
30
20
10
0

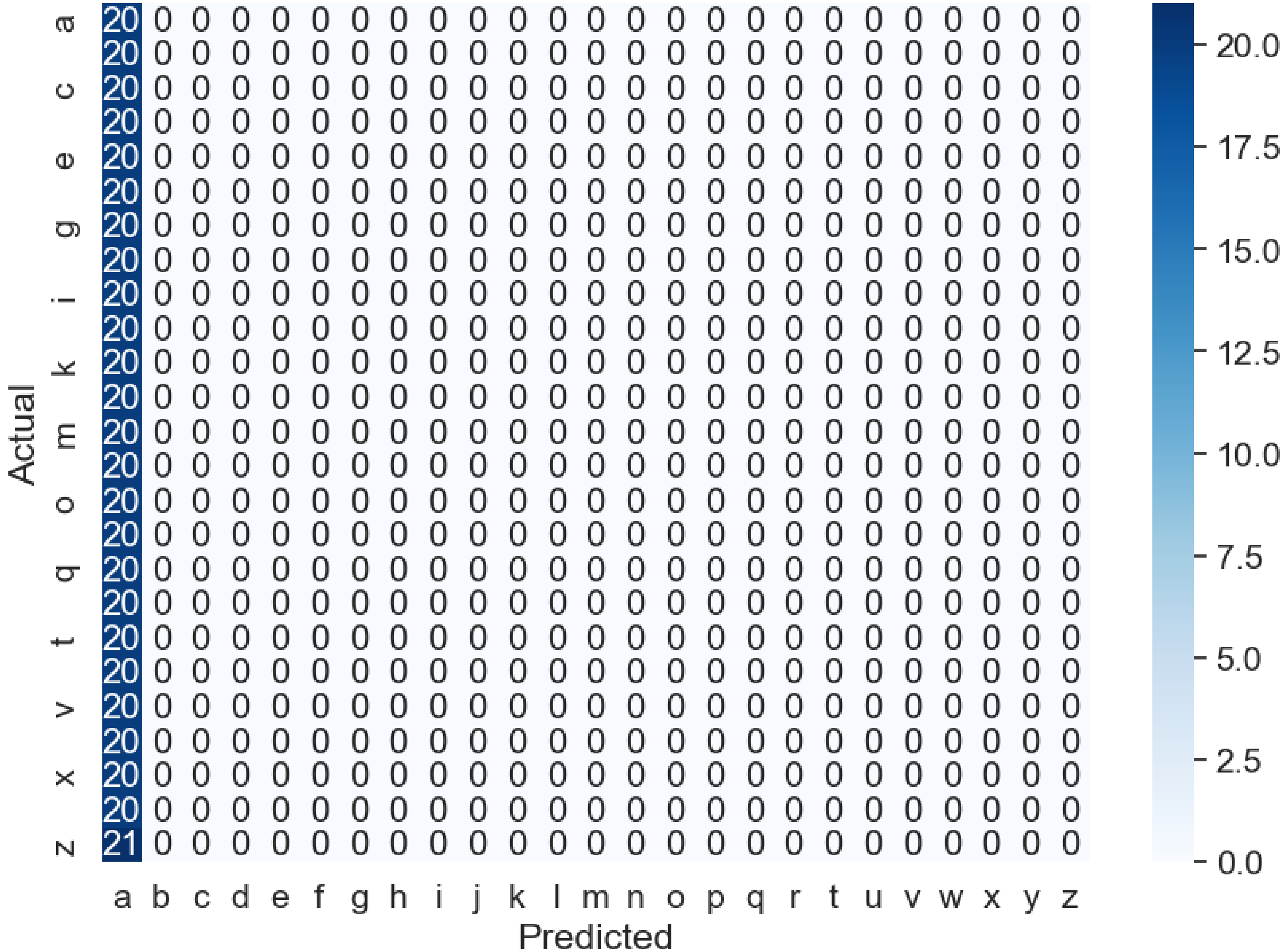
```
              precision    recall  f1-score   support

           0       0.03      1.00      0.06        32
           1       0.00      0.00      0.00        39
           2       0.00      0.00      0.00        42
           3       0.00      0.00      0.00        46
           4       0.00      0.00      0.00        32
           5       0.00      0.00      0.00        34
           6       0.00      0.00      0.00        36
           7       0.00      0.00      0.00        39
           8       0.00      0.00      0.00        35
           9       0.00      0.00      0.00        26
          10       0.00      0.00      0.00        33
          11       0.00      0.00      0.00        48
          12       0.00      0.00      0.00        40
          13       0.00      0.00      0.00        30
          14       0.00      0.00      0.00        42
          15       0.00      0.00      0.00        49
          16       0.00      0.00      0.00        43
          17       0.00      0.00      0.00        38
          18       0.00      0.00      0.00        41
          19       0.00      0.00      0.00        47
          20       0.00      0.00      0.00        43
          21       0.00      0.00      0.00        36
          22       0.00      0.00      0.00        43
          23       0.00      0.00      0.00        48
          24       0.00      0.00      0.00        41
          25       0.00      0.00      0.00        31

    accuracy                           0.03      1014
   macro avg       0.00      0.04      0.00      1014
weighted avg       0.00      0.03      0.00      1014
```




```
In [20]: #LogisticRegression
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression(random_state = 0)
logreg_cm, logreg_report,logreg_cm_external, logreg_acc, logreg_ext_acc = machine_learning1(logreg)
```

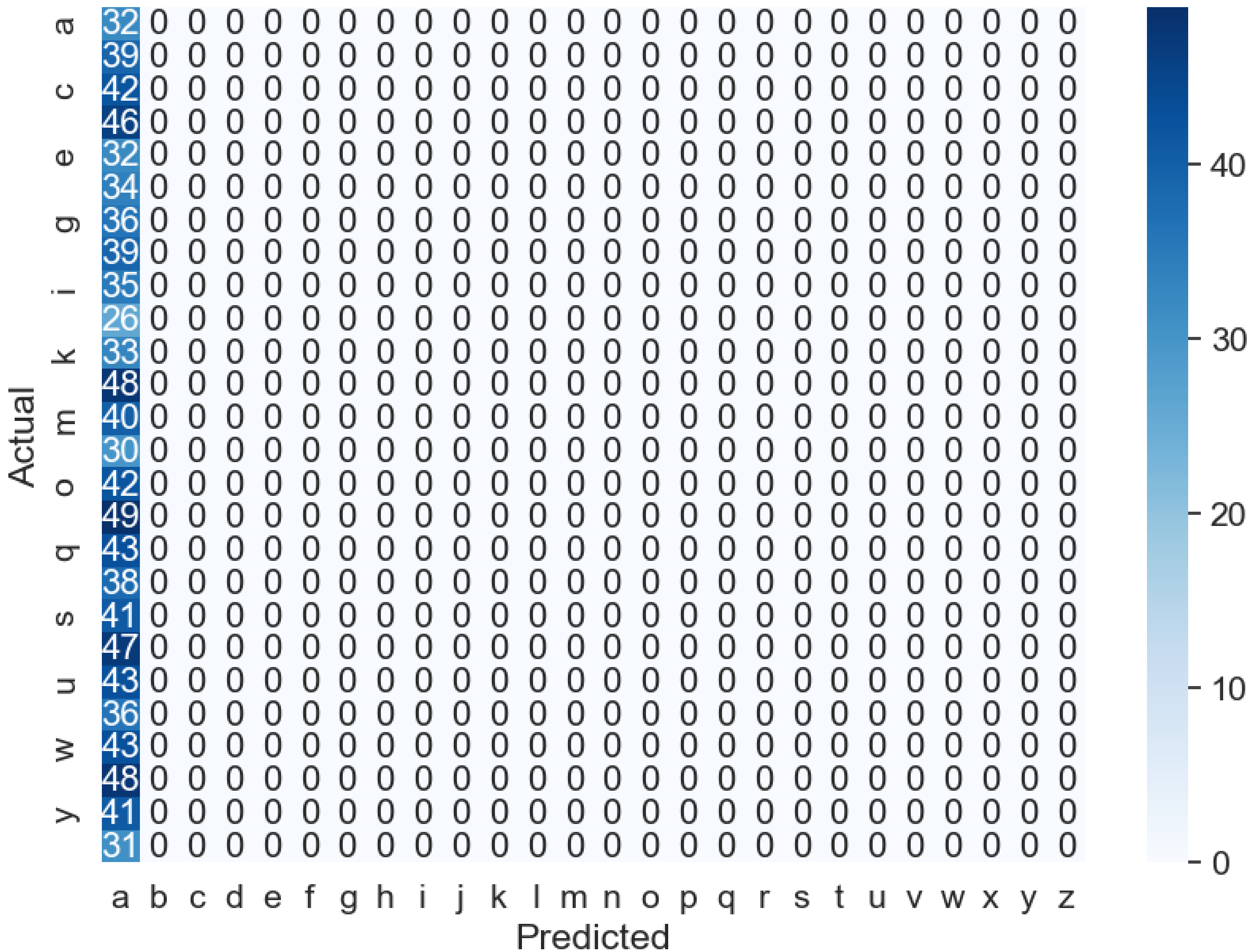

Actual
a
c
e
g
i
k
m
o
q
s
u
w
y
32 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
39 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
42 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
46 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
32 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
34 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
36 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
39 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
35 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
26 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
33 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
48 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
40 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 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
42 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
49 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
43 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
38 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
41 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
47 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
43 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
36 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
43 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
48 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
41 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
31 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
a b c d e f g h i j k l m n o p q r s t u v w x y z
Predicted
40
30
20
10
0

```
              precision    recall  f1-score   support

           0       0.03      1.00      0.06        32
           1       0.00      0.00      0.00        39
           2       0.00      0.00      0.00        42
           3       0.00      0.00      0.00        46
           4       0.00      0.00      0.00        32
           5       0.00      0.00      0.00        34
           6       0.00      0.00      0.00        36
           7       0.00      0.00      0.00        39
           8       0.00      0.00      0.00        35
           9       0.00      0.00      0.00        26
          10       0.00      0.00      0.00        33
          11       0.00      0.00      0.00        48
          12       0.00      0.00      0.00        40
          13       0.00      0.00      0.00        30
          14       0.00      0.00      0.00        42
          15       0.00      0.00      0.00        49
          16       0.00      0.00      0.00        43
          17       0.00      0.00      0.00        38
          18       0.00      0.00      0.00        41
          19       0.00      0.00      0.00        47
          20       0.00      0.00      0.00        43
          21       0.00      0.00      0.00        36
          22       0.00      0.00      0.00        43
          23       0.00      0.00      0.00        48
          24       0.00      0.00      0.00        41
          25       0.00      0.00      0.00        31

    accuracy                           0.03      1014
   macro avg       0.00      0.04      0.00      1014
weighted avg       0.00      0.03      0.00      1014
```

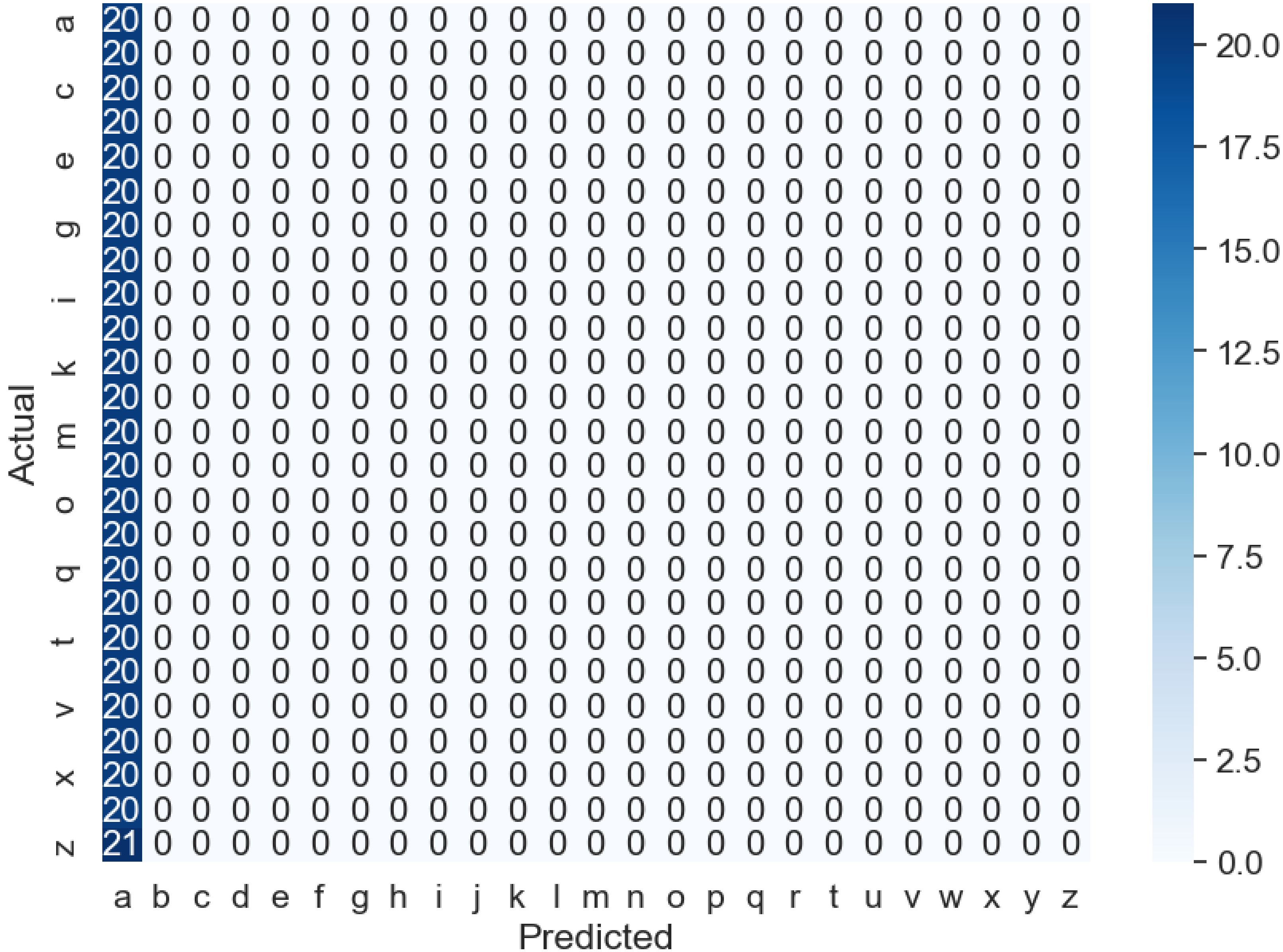


In [21]:

```
#RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(n_estimators = 100)
rf_clf_cm, rf_clf_report,rf_clf_cm_external, rf_acc, rf_ext_acc = machine_learning1(rf_clf)
```


Actual
Predicted
a b c d e f g h i j k l m n o p q r s t u v w x y z
a c e g i k m o q s u w y
0 10 20 30 40

	precision	recall	f1-score	support
0	0.03	1.00	0.06	32
1	0.00	0.00	0.00	39
2	0.00	0.00	0.00	42
3	0.00	0.00	0.00	46
4	0.00	0.00	0.00	32
5	0.00	0.00	0.00	34
6	0.00	0.00	0.00	36
7	0.00	0.00	0.00	39
8	0.00	0.00	0.00	35
9	0.00	0.00	0.00	26
10	0.00	0.00	0.00	33
11	0.00	0.00	0.00	48
12	0.00	0.00	0.00	40
13	0.00	0.00	0.00	30
14	0.00	0.00	0.00	42
15	0.00	0.00	0.00	49
16	0.00	0.00	0.00	43
17	0.00	0.00	0.00	38
18	0.00	0.00	0.00	41
19	0.00	0.00	0.00	47
20	0.00	0.00	0.00	43
21	0.00	0.00	0.00	36
22	0.00	0.00	0.00	43
23	0.00	0.00	0.00	48
24	0.00	0.00	0.00	41
25	0.00	0.00	0.00	31
accuracy			0.03	1014
macro avg	0.00	0.04	0.00	1014
weighted avg	0.00	0.03	0.00	1014



In [22]:
```python
#Accuary of each alogirthm displaying as table
res_table = pd.DataFrame({
    'Model': ['XGBoost','LGBM','SVC','Decision Tree','Logistic Regression','Random Forest'],
    'Train Score': [xgb_acc,lgb_acc,svc_acc,clf_acc,logreg_acc,rf_acc],
```

```
    'Test Score' : [xgb_ext_acc,lgb_ext_acc,svc_ext_acc,clf_ext_acc,logreg_ext_acc,rf_ext_acc]})
res_table.sort_values(by='Train Score', ascending=False)
```

Out[22]:

	Model	Train Score	Test Score
1	LGBM	0.894477	0.566866
0	XGBoost	0.772189	0.518962
2	SVC	0.031558	0.039920
3	Decision Tree	0.031558	0.039920
4	Logistic Regression	0.031558	0.039920
5	Random Forest	0.031558	0.039920