In [8]:

```
import tensorflow as tf
```

In [9]:

```
from tensorflow.keras.utils import to_categorical
```

In [10]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from PIL import Image
import os
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
from tensorflow.keras import layers
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

In [11]:

```
data=[]
labels=[]
classes=43
cur_path='C:\\Users\\is-os\\Desktop\\archive'
```

In [12]:

```
#retrieving images and labels
for i in range(classes):
    path=os.path.join(cur_path,'Train',str(i))
     #print(path)
    images=os.listdir(path)
    #print(images)
    for a in images:
        try:
            image = Image.open(path + '\\'+ a)
            image=image.resize((30,30))
            image=np.array(image)
            data.append(image)
            labels.append(i)
        except:
            print(f'error loading image{a}')
```

In [13]:

```
#change data to array
data = np.array(data)
labels = np.array(labels)
print(data)
```

```
   ...
   [ 10   9  11]
   [ 10   9  11]
   [  9   9  11]]

  [[ 11  11  13]
   [ 11  10  12]
   [  9   9  10]
   ...
   [ 10   9  11]
   [  9   9  11]
   [ 10   9  11]]

  [[ 11  11  12]
   [ 10  10  11]
   [  9   9  10]
   ...
   [ 10   9  12]
   [ 10  10  11]
   [ 10   9  11]]]]
```

In [14]:

```
print(data.shape,labels.shape)
```

```
(39209, 30, 30, 3) (39209,)
```

In [ ]:

In [15]:

```
#split data
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_
```

In [16]:

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(31367, 30, 30, 3) (7842, 30, 30, 3) (31367,) (7842,)
```

In [17]:

```
#rescale data
X_train=X_train/255
X_test=X_test/255
```

In [18]:

```
#Converting the labels into one hot encoding
y_train = to_categorical(y_train, 43)
y_test = to_categorical(y_test, 43)
```

In [19]:

```
#building model architecture
MODEL=Sequential([
    Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=X_train.shape[1
    Conv2D(filters=32, kernel_size=(3,3), activation='relu'),
    MaxPool2D(pool_size=(2, 2)),
    Dropout(rate=0.25),
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    MaxPool2D(pool_size=(2, 2)),
    Dropout(rate=0.25),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(rate=0.5),
    Dense(43, activation='softmax')

])
```

In [20]:

```
#defining learning rate,epochs and nessessary things to build the model
lr = 0.001
epochs = 30

opt = Adam(learning_rate=lr, decay=lr / (epochs * 0.5))
MODEL.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
```

In [ ]:

```

```

In [21]:

```
#fit data to train and validate it
hist = MODEL.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_test
```

```
Epoch 1/30
981/981 [==============================] - 26s 18ms/step - loss: 1.3644 -
accuracy: 0.6158 - val_loss: 0.1622 - val_accuracy: 0.9628
Epoch 2/30
981/981 [==============================] - 16s 16ms/step - loss: 0.2667 -
accuracy: 0.9187 - val_loss: 0.0826 - val_accuracy: 0.9767
Epoch 3/30
981/981 [==============================] - 16s 16ms/step - loss: 0.1564 -
accuracy: 0.9503 - val_loss: 0.0396 - val_accuracy: 0.9898
Epoch 4/30
981/981 [==============================] - 16s 16ms/step - loss: 0.1153 -
accuracy: 0.9653 - val_loss: 0.0331 - val_accuracy: 0.9913
Epoch 5/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0887 -
accuracy: 0.9726 - val_loss: 0.0276 - val_accuracy: 0.9927
Epoch 6/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0707 -
accuracy: 0.9785 - val_loss: 0.0214 - val_accuracy: 0.9946
Epoch 7/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0650 -
accuracy: 0.9790 - val_loss: 0.0208 - val_accuracy: 0.9945
Epoch 8/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0589 -
accuracy: 0.9826 - val_loss: 0.0212 - val_accuracy: 0.9954
Epoch 9/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0452 -
accuracy: 0.9861 - val_loss: 0.0177 - val_accuracy: 0.9955
Epoch 10/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0434 -
accuracy: 0.9868 - val_loss: 0.0164 - val_accuracy: 0.9966
Epoch 11/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0355 -
accuracy: 0.9883 - val_loss: 0.0210 - val_accuracy: 0.9955
Epoch 12/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0351 -
accuracy: 0.9884 - val_loss: 0.0195 - val_accuracy: 0.9962
Epoch 13/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0341 -
accuracy: 0.9897 - val_loss: 0.0165 - val_accuracy: 0.9966
Epoch 14/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0299 -
accuracy: 0.9910 - val_loss: 0.0176 - val_accuracy: 0.9959
Epoch 15/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0271 -
accuracy: 0.9918 - val_loss: 0.0161 - val_accuracy: 0.9964
Epoch 16/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0245 -
accuracy: 0.9919 - val_loss: 0.0160 - val_accuracy: 0.9966
Epoch 17/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0225 -
accuracy: 0.9934 - val_loss: 0.0183 - val_accuracy: 0.9960
Epoch 18/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0208 -
accuracy: 0.9934 - val_loss: 0.0151 - val_accuracy: 0.9967
Epoch 19/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0211 -
accuracy: 0.9932 - val_loss: 0.0163 - val_accuracy: 0.9968
Epoch 20/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0203 -
accuracy: 0.9937 - val_loss: 0.0140 - val_accuracy: 0.9971
Epoch 21/30
```

```
981/981 [==============================] - 16s 16ms/step - loss: 0.0172 -
accuracy: 0.9945 - val_loss: 0.0142 - val_accuracy: 0.9973
Epoch 22/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0172 -
accuracy: 0.9945 - val_loss: 0.0182 - val_accuracy: 0.9966
Epoch 23/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0173 -
accuracy: 0.9947 - val_loss: 0.0135 - val_accuracy: 0.9977
Epoch 24/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0157 -
accuracy: 0.9951 - val_loss: 0.0139 - val_accuracy: 0.9971
Epoch 25/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0140 -
accuracy: 0.9956 - val_loss: 0.0143 - val_accuracy: 0.9971
Epoch 26/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0159 -
accuracy: 0.9948 - val_loss: 0.0140 - val_accuracy: 0.9974
Epoch 27/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0139 -
accuracy: 0.9956 - val_loss: 0.0145 - val_accuracy: 0.9969
Epoch 28/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0124 -
accuracy: 0.9962 - val_loss: 0.0148 - val_accuracy: 0.9972
Epoch 29/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0114 -
accuracy: 0.9968 - val_loss: 0.0127 - val_accuracy: 0.9976
Epoch 30/30
981/981 [==============================] - 16s 16ms/step - loss: 0.0108 -
accuracy: 0.9964 - val_loss: 0.0164 - val_accuracy: 0.9969
```

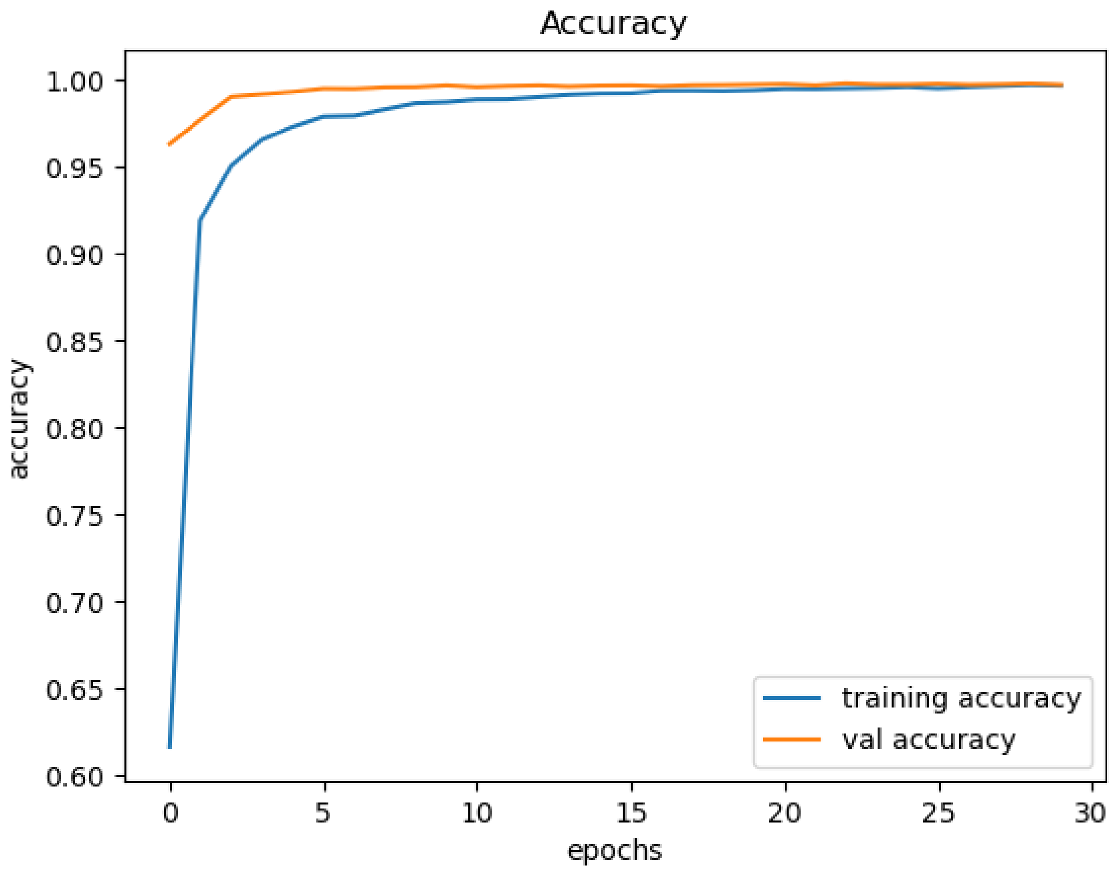
In [ ]:

```

```

In [22]:

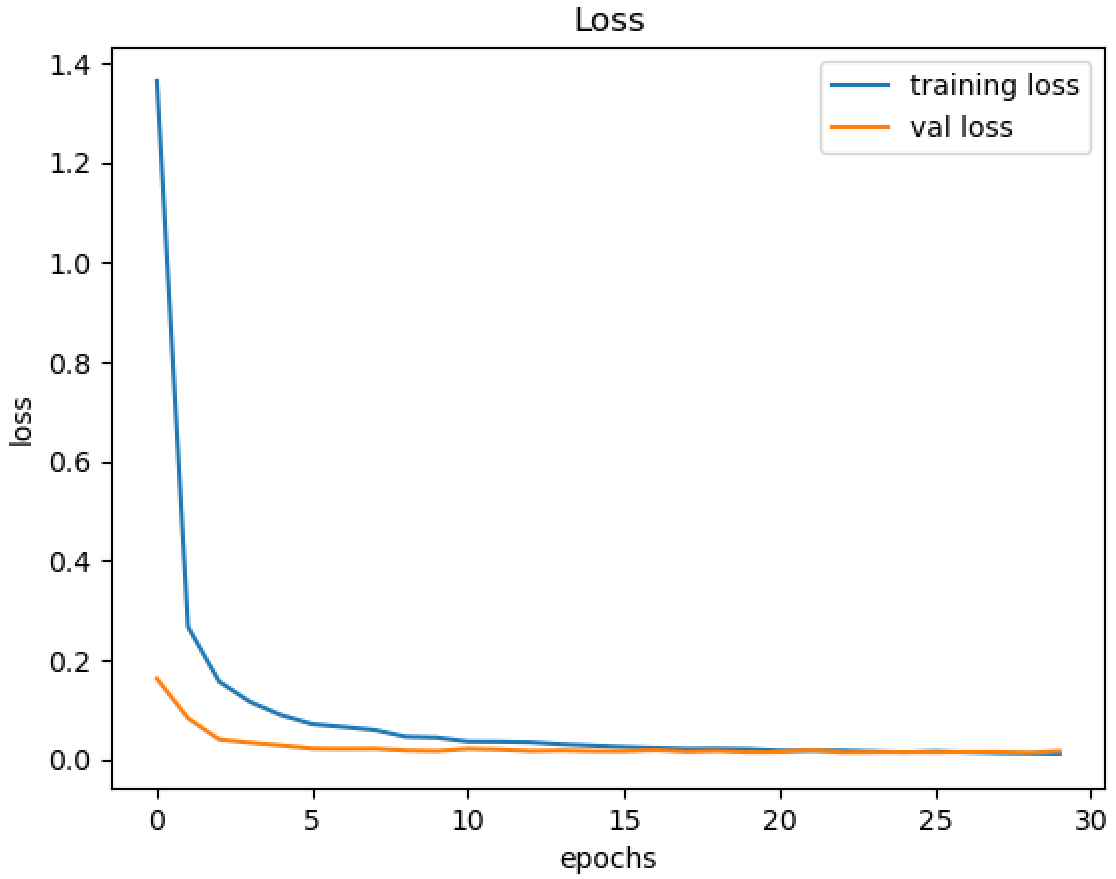
```
#ploting the performances
plt.figure(0)
plt.plot(hist.history['accuracy'], label='training accuracy')
plt.plot(hist.history['val_accuracy'], label='val accuracy')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()
```



In [ ]:

In [23]:

```
#plot the losses
plt.figure(1)
plt.plot(hist.history['loss'], label='training loss')
plt.plot(hist.history['val_loss'], label='val loss')
plt.title('Loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```



In [24]:

```
#the data has a csv file that contain the names and numbers for classes, it has a column
#column for labels
testdata = pd.read_csv('C:\\Users\\is-os\\Desktop\\archive\\Test.csv')

t_labels = testdata["ClassId"].values
imgs = testdata["Path"].values
```

In [25]:

```
#same steps as the training(open,resize,convert to array and rescale)
t_data =[]

for img in imgs:
    try:
        image = Image.open(cur_path+"\\"+img)
        image = image.resize((30,30))
        t_data.append(np.array(image))
    except:
        print(f"error in {img}")
X_test=np.array(t_data)
X_test = X_test/255
```

In [26]:

```
print(X_test)
```

```
[[[[0.45490196 0.54901961 0.68627451]
   [0.45490196 0.54117647 0.67058824]
   [0.46666667 0.54117647 0.67843137]
   ...
   [0.4        0.46666667 0.58823529]
   [0.39607843 0.47843137 0.58431373]
   [0.36470588 0.43921569 0.54509804]]

  [[0.45490196 0.55686275 0.69411765]
   [0.45490196 0.55294118 0.68627451]
   [0.45882353 0.55294118 0.68235294]
   ...
   [0.47058824 0.56078431 0.69803922]
   [0.47843137 0.56470588 0.69019608]
   [0.47843137 0.55686275 0.68235294]]

  [[0.4627451  0.55686275 0.68235294]
   [0.45490196 0.55294118 0.68627451]
   [0.44705882 0.54901961 0.6745098 ]
```

In [27]:

```
#run the prediction
pred = MODEL.predict_classes(X_test)
```

```
C:\Users\is-os\anaconda3\lib\site-packages\tensorflow\python\keras\engine
\sequential.py:455: UserWarning: `model.predict_classes()` is deprecated a
nd will be removed after 2021-01-01. Please use instead:* `np.argmax(mode
l.predict(x), axis=-1)`,   if your model does multi-class classification
(e.g. if it uses a `softmax` last-layer activation).* `(model.predict(x) >
0.5).astype("int32")`,   if your model does binary classification   (e.g.
if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

In [28]:

```
#showing metrics data
from sklearn.metrics import accuracy_score
#Accuracy with the test data
print('Test Data accuracy: ',accuracy_score(t_labels, pred)*100)
```

```
Test Data accuracy:  97.84639746634997
```

In [29]:

```
from sklearn.metrics import classification_report

print(classification_report(t_labels, pred))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	60
1	0.97	1.00	0.98	720
2	0.99	0.99	0.99	750
3	1.00	0.96	0.98	450
4	1.00	0.97	0.99	660
5	0.94	0.99	0.96	630
6	1.00	0.87	0.93	150
7	0.98	1.00	0.99	450
8	1.00	0.98	0.99	450
9	0.98	1.00	0.99	480
10	1.00	1.00	1.00	660
11	0.93	0.99	0.96	420
12	1.00	0.98	0.99	690
13	1.00	1.00	1.00	720
14	1.00	1.00	1.00	270
15	0.98	1.00	0.99	210
16	0.99	1.00	0.99	150
17	1.00	1.00	1.00	360
18	0.98	0.88	0.92	390
19	0.98	1.00	0.99	60
20	0.83	1.00	0.90	90
21	1.00	0.96	0.98	90
22	1.00	0.99	1.00	120
23	0.96	0.99	0.97	150
24	0.76	0.98	0.85	90
25	0.98	0.98	0.98	480
26	0.95	0.93	0.94	180
27	0.97	0.50	0.66	60
28	0.99	0.99	0.99	150
29	0.86	1.00	0.92	90
30	0.96	0.75	0.84	150
31	0.99	0.99	0.99	270
32	1.00	1.00	1.00	60
33	0.98	1.00	0.99	210
34	0.98	0.99	0.98	120
35	1.00	1.00	1.00	390
36	0.98	0.97	0.98	120
37	0.98	1.00	0.99	60
38	0.99	1.00	0.99	690
39	0.93	0.98	0.95	90
40	0.88	0.98	0.93	90
41	0.98	0.85	0.91	60
42	0.98	1.00	0.99	90
accuracy			0.98	12630
macro avg	0.97	0.96	0.96	12630
weighted avg	0.98	0.98	0.98	12630

In [30]:

```
#ploting the results
plt.figure(figsize = (25, 25))

index = 0
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    prediction = pred[index + i]
    actual = t_labels[index + i]
    col = 'g'
    if prediction != actual:
        col = 'r'
    plt.xlabel('Actual={} || Pred={}'.format(actual, prediction), color = col)
    plt.imshow(X_test[index + i])
plt.show()
```

In [31]:

```
# save for future uses
MODEL.save('traffic_new.h5')
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: