In [ ]:
```
importpandasaspd
pd.read_csv('../input/traffic-signs-classification/labels.csv')
```

Importing Libraries

In [2]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.image import imread
import seaborn as sns
```

Data Collection

In [3]:
```
dir_path = '../input/gtsrb-german-traffic-sign'
```

In [4]:
```
os.listdir(dir_path)
```

Out[4]:
```
['Meta',
 'meta',
 'Meta.csv',
 'Train.csv',
 'Test.csv',
 'Test',
 'test',
 'Train',
 'train']
```

In [5]:
```
#Assigningthepathfortrainandtestimages

train_path=dir_path+'/Train'test_p
ath=dir_path+'/Test'
```

In [6]:
```
print(sorted(os.listdir(train_path)))
```

```
['0', '1', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '2', '20', '21', '22',
'23', '24', '25', '26', '27', '28', '29', '3', '30', '31', '32', '33', '34', '35', '36', '3
7', '38', '39', '4', '40', '41', '42', '5', '6', '7', '8', '9']
```

In [ ]:
```
sorted(os.listdir(test_path))
```

Visualization

Visualizing 25 random sample images from test set

In [8]:
```
import random
images_path = os.listdir(test_path)
plt.figure(figsize=(25,25))



for i in range(1,26):

    plt.subplot(5,5,i)
    random_img_path = test_path +'/'+ random.choice(images_path) rand_img =
    imread(random_img_path)
    plt.imshow(rand_img)
    plt.xlabel(rand_img.shape[1], fontsize = 20)#width of image
    plt.ylabel(rand_img.shape[0], fontsize = 20)#height of image
```

The dimensions of the images are not fixed.

Note:

Convolutional neural networks cannot perform on images that have various dimensions. We will resize these images during our model building.

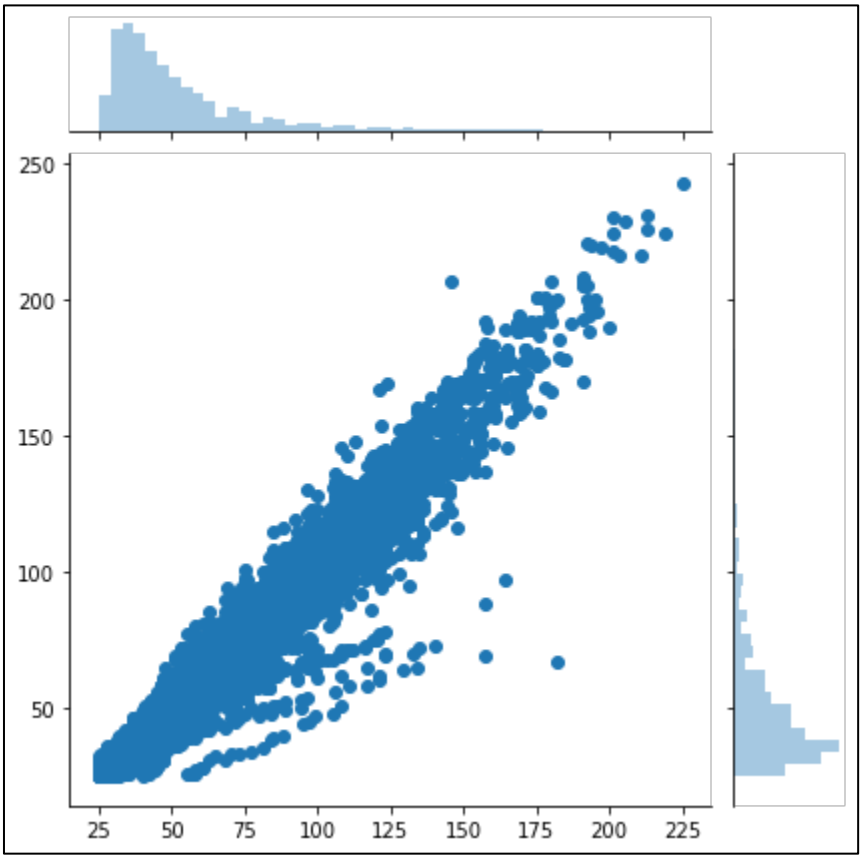
But first find the mean of the dimensions of all the images in training set.

In [9]:
```
dim1=[]di
m2=[]

forininrange(0,43):
    labels=train_path+'/{0}'.format(i)image_p
    ath=os.listdir(labels)
    forxinimage_path:
        img=imread(labels+'/'+x)dim1.a
        ppend(img.shape[0])
        dim2.append(img.shape[1])
```

Exploring the dimensions with a jointplot

In [10]:
```
sns.jointplot(dim1,dim2)plt.sh
ow()
```



In [11]:
```
np.mean(dim1)
```

Out[11]: 50.328929582493814

In [12]:
```
np.mean(dim2)
```

Out[12]: 50.83587951745773

Since the mean of both dimensions is around 50 , we will use (50x50) as the shape of images.

In [13]:
```
image_shape=(50,50)
```

Data Preprocessing

Importing the images

In [14]:
```
fromPILimportImage

images=[]
label_id=[]

foriinrange(43):
    labels=train_path+'/{0}'.format(i)image_p
    ath=os.listdir(labels)
    forxinimage_path:
        img=Image.open(labels+'/'+x)img=i
        mg.resize(image_shape)
        img=np.array(img)i
        mages.append(img)
```

Scaling the images so that the values remain between 0 and 1

In [15]:
```
#Convertingimagesintonumpyarray
images=np.array(images)
#Thepixelvalueofeachimagerangesbetween0and255
#Dividingeachimageby255willscalethevaluesbetween0and1.Thisisalsoknownasnorm
images=images/255
```

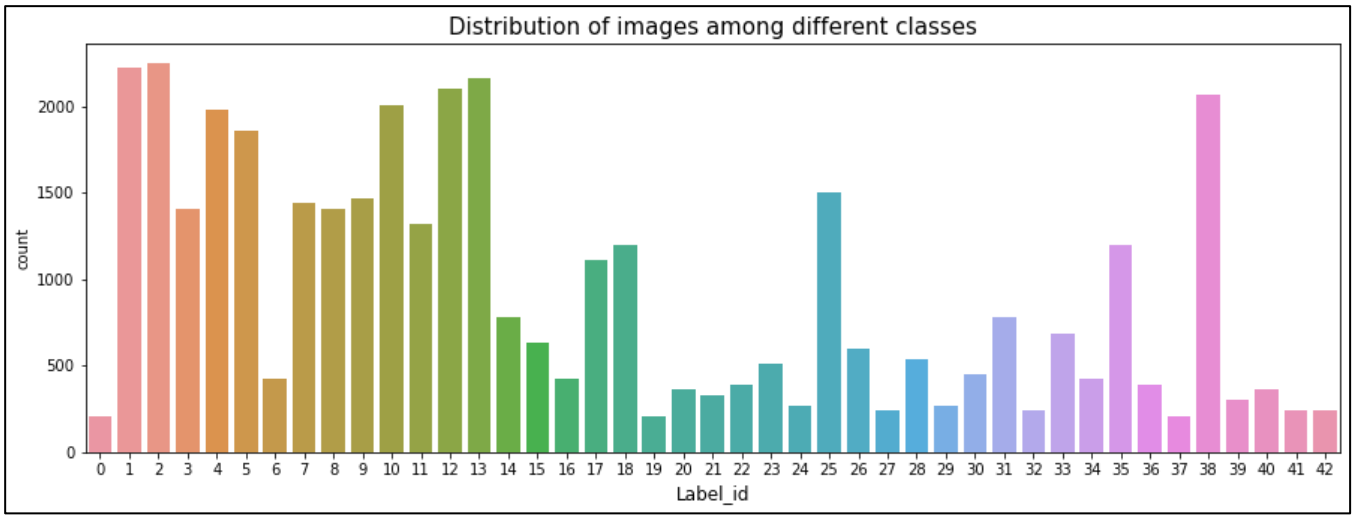
In [16]:
```
label_id=np.array(label_id)label_id.
shape
```

Out[16]: (39209,)

In [17]:
```
images.shape
```

Out[17]: (39209, 50, 50, 3)

In [18]:
```
plt.figure(figsize=(15,5))sns.count
plot(label_id)
plt.title('Distributionofimagesamongdifferentclasses',fontsize=15)plt.xlabel('Labe
l_id',fontsize=12)
plt.show()
```



In [19]:
```
#Savingthescaledimagesandlabelsforfutureuse
np.save('Training_set',images)np.s
ave('Label_Id',label_id)
```

Splitting the train data into train and validation data

In [20]:
```
importnumpyasnp
importpandasaspd
```

In [21]:
```
images=np.load('Training_set.npy')label_id=np.lo
ad('Label_Id.npy')
```

In [22]:
```
#Splittingthedata
fromsklearn.model_selectionimporttrain_test_split
x_train,x_val,y_train,y_val=train_test_split(images,label_id,test_size=0.2,random_
```

Changing target labels to categorical using one-hot encoding technique

In [23]:
```
#kerashasabuilt-infunctionforone-hotencoding.
fromtensorflow.keras.utilsimportto_categoricaly_train_cat=to_cate

gorical(y_train)

y_val_cat=to_categorical(y_val)
```

Model Building

In [24]:
```
importtensorflowastf
fromtensorflow.keras.modelsimportSequential
fromtensorflow.keras.layersimportDense,Flatten,Dropout,Conv2D,MaxPool2D
```

In [25]:
```
model=Sequential()

#1stlayer
model.add(Conv2D(filters=64,kernel_size=(3,3),input_shape=x_train.shape[1:],activatimodel.add(Max
Pool2D(pool_size=(2,2)))
model.add(Dropout(0.5))

#2ndlayer
model.add(Conv2D(filters=64,kernel_size=(3,3),activation='relu'))model.add(Ma
xPool2D(pool_size=(2,2)))
model.add(Dropout(0.5))

#3rdlayer
model.add(Conv2D(filters=64,kernel_size=(3,3),activation='relu'))model.add(Ma
xPool2D(pool_size=(2,2)))
model.add(Dropout(0.5))model.add(Flatten())

#Denselayer
model.add(Dense(128,activation='relu'))model.
add(Dropout(0.5))

#Outputlayer
model.add(Dense(43,activation='softmax'))
```

In [26]:
```
model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accur
```

In [27]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 50, 50, 64)	1792
max_pooling2d (MaxPooling2D)	(None, 25, 25, 64)	0
dropout (Dropout)	(None, 25, 25, 64)	0
conv2d_1 (Conv2D)	(None, 23, 23, 64)	36928
max_pooling2d_1 (MaxPooling2	(None, 11, 11, 64)	0
dropout_1 (Dropout)	(None, 11, 11, 64)	0
conv2d_2 (Conv2D)	(None, 9, 9, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 64)	0
dropout_2 (Dropout)	(None, 4, 4, 64)	0
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 128)	131200
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 43)	5547

Total params: 212,395
Trainable params: 212,395
Non-trainable params: 0

In [28]:
```
early_stopping=tf.keras.callbacks.EarlyStopping(monitor='val_loss',patience=2)model.fit(

    x_train,y_train,epochs=25
    ,
    batch_size=64,
    validation_data=(x_val,y_val),callbac
    ks=[early_stopping],
    verbose=2

)
```

```
Epoch 1/25
491/491 - 85s - loss: 2.6929 - accuracy: 0.2470 - val_loss: 1.7082 - val_accuracy: 0.5179 Epoch 2/25
491/491 - 83s - loss: 1.4750 - accuracy: 0.5205 - val_loss: 0.8227 - val_accuracy: 0.7754 Epoch 3/25
491/491 - 83s - loss: 0.9164 - accuracy: 0.6967 - val_loss: 0.3744 - val_accuracy: 0.9218 Epoch 4/25
491/491 - 83s - loss: 0.6258 - accuracy: 0.7922 - val_loss: 0.2281 - val_accuracy: 0.9536 Epoch 5/25
491/491 - 83s - loss: 0.4927 - accuracy: 0.8393 - val_loss: 0.1606 - val_accuracy: 0.9688 Epoch 6/25
491/491 - 83s - loss: 0.4136 - accuracy: 0.8658 - val_loss: 0.1222 - val_accuracy: 0.9733 Epoch 7/25
491/491 - 82s - loss: 0.3639 - accuracy: 0.8815 - val_loss: 0.1012 - val_accuracy: 0.9821 Epoch 8/25
491/491 - 85s - loss: 0.3263 - accuracy: 0.8935 - val_loss: 0.0797 - val_accuracy: 0.9824 Epoch 9/25
491/491 - 84s - loss: 0.2886 - accuracy: 0.9071 - val_loss: 0.0650 - val_accuracy: 0.9866
Epoch 10/25
491/491 - 84s - loss: 0.2714 - accuracy: 0.9128 - val_loss: 0.0576 - val_accuracy: 0.9876
Epoch 11/25
491/491 - 83s - loss: 0.2599 - accuracy: 0.9159 - val_loss: 0.0547 - val_accuracy: 0.9890
Epoch 12/25
491/491 - 84s - loss: 0.2444 - accuracy: 0.9226 - val_loss: 0.0559 - val_accuracy: 0.9901
Epoch 13/25
491/491 - 84s - loss: 0.2290 - accuracy: 0.9260 - val_loss: 0.0456 - val_accuracy: 0.9888
Epoch 14/25
491/491 - 83s - loss: 0.2220 - accuracy: 0.9292 - val_loss: 0.0399 - val_accuracy: 0.9923
Epoch 15/25
491/491 - 84s - loss: 0.2234 - accuracy: 0.9292 - val_loss: 0.0472 - val_accuracy: 0.9895
Epoch 16/25
491/491 - 87s - loss: 0.2015 - accuracy: 0.9369 - val_loss: 0.0342 - val_accuracy: 0.9921
Epoch 17/25
491/491 - 84s - loss: 0.1986 - accuracy: 0.9371 - val_loss: 0.0354 - val_accuracy: 0.9922
Epoch 18/25
491/491 - 84s - loss: 0.1904 - accuracy: 0.9396 - val_loss: 0.0329 - val_accuracy: 0.9934
Epoch 19/25
491/491 - 83s - loss: 0.1858 - accuracy: 0.9411 - val_loss: 0.0370 - val_accuracy: 0.9921
Epoch 20/25
491/491 - 84s - loss: 0.1785 - accuracy: 0.9429 - val_loss: 0.0300 - val_accuracy: 0.9946
Epoch 21/25
491/491 - 84s - loss: 0.1808 - accuracy: 0.9439 - val_loss: 0.0267 - val_accuracy: 0.9950
Epoch 22/25
491/491 - 83s - loss: 0.1784 - accuracy: 0.9447 - val_loss: 0.0272 - val_accuracy: 0.9946
Epoch 23/25
491/491 - 87s - loss: 0.1657 - accuracy: 0.9487 - val_loss: 0.0280 - val_accuracy: 0.9934
```

Out[28]: <tensorflow.python.keras.callbacks.History at 0x7f2a98132fd0>

Achieved highest accuracy of 99.50% on validation data

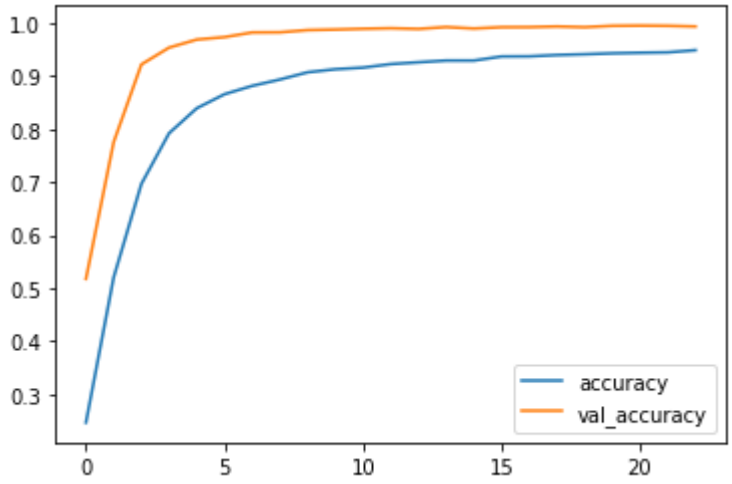
In [29]:
```
#Savingthemodel
model.save('Model.h5')
```

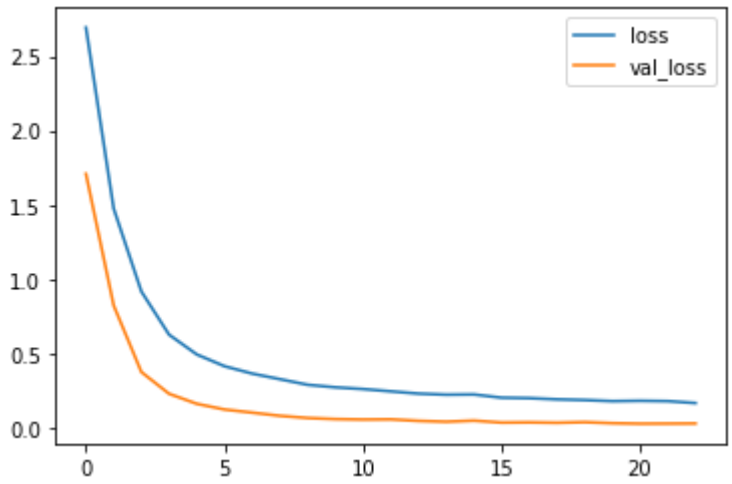
Model Evaluation

In [30]:
```
evaluation=pd.DataFrame(model.history.history)

evaluation[['accuracy','val_accuracy']].plot()evaluation[['los
s','val_loss']].plot()
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2a90091610>





Testing on test data

In [31]:
```
fromtensorflow.keras.modelsimportload_modelmodel=lo
ad_model('Model.h5')
```

Note: The test images folder in the original dataset has a blank csv file which cannot be opened with the above function. So i copied that folder and deleted that csv file and uploaded the test images again seperately. These test images are same as the test images in the original dataset

In [32]:
```
test_path='../input/test-
images/Test'test_img=sorted(os.listdir(test_path)
```

For some unknown reason , the images in kaggle kernel is not showing in the order they are in the Test folder. Upon inspection it is seen that the images are in sorted order. So using sorted() function to sort them.

In [33]:
```
#definingafunctionthatwillscaleimages
fromPILimportImage

defscaling(test_images,test_path):images=[]

    image_path=test_images

    forxinimage_path:
        img=Image.open(test_path+'/'+x)img=img.re
        size((50,50))
        img=np.array(img)ima
        ges.append(img)
```

```
    #Convertingimagesintonumpyarray
    images=np.array(images)
    #Thepixelvalueofeachimagerangesbetween0and255
    #Dividingeachimageby255willscalethevaluesbetween0and1.Thisisalsoknownas
    images=images/255

    returnimages
```

The above function can be used to scale any new traffic-sign images that can be predicted with our model. This is a general purpose function for code reusability.

In [34]:
```
test_images=scaling(test_img,test_path)
```

Test labels

In [35]:
```
test=pd.read_csv('../input/gtsrb-german-traffic-

sign/Test.csv')y_test=test['ClassId'].values

y_test
```

Out[35]:

Testing on test images

In [36]:
```
y_pred=model.predict_classes(test_images)

y_pred
```

Out[36]: array([16, 1, 38, ..., 6, 7, 10])

In [37]:
```
fromsklearn.metricsimportclassification_report

print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.98	0.99	60
1	0.99	0.98	0.99	720
2	0.99	0.99	0.99	750
3	0.99	0.93	0.96	450
4	0.98	0.99	0.99	660
5	0.94	0.97	0.96	630
6	1.00	0.93	0.96	150
7	0.98	0.98	0.98	450
8	0.97	0.98	0.98	450
9	0.95	1.00	0.97	480
10	0.99	1.00	1.00	660
11	0.97	1.00	0.98	420
12	0.96	0.94	0.95	690
13	0.99	1.00	1.00	720
14	1.00	1.00	1.00	270
15	0.90	1.00	0.95	210
16	1.00	1.00	1.00	150
17	1.00	0.92	0.96	360
18	0.99	0.95	0.97	390
19	0.92	1.00	0.96	60
20	0.72	0.99	0.83	90
21	0.94	0.64	0.76	90
22	0.96	0.97	0.97	120
23	0.99	0.96	0.98	150
24	1.00	0.97	0.98	90
25	0.96	0.96	0.96	480
26	0.95	0.90	0.93	180
27	0.80	0.47	0.59	60
28	0.97	1.00	0.99	150
29	0.74	0.99	0.85	90
30	1.00	0.77	0.87	150
31	0.96	0.94	0.95	270
32	0.85	1.00	0.92	60
33	0.97	0.99	0.98	210
34	0.98	0.99	0.99	120
35	1.00	0.99	1.00	390
36	0.96	0.99	0.98	120
37	0.98	0.98	0.98	60
38	0.95	0.99	0.97	690
39	0.98	0.99	0.98	90
40	0.98	0.99	0.98	90
41	1.00	0.58	0.74	60
42	0.96	1.00	0.98	90
accuracy			0.97	12630
macro avg	0.96	0.94	0.95	12630
weighted avg	0.97	0.97	0.97	12630

We achieved an overall accuracy of 97% on our model. This is pretty good and we can use this model for predicting some other Traffic signs as well in future.