In [1]:

```
# Importing Python libraries
import os
import pathlib
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array, array_to
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
from tensorflow.keras.models import Sequential
%matplotlib inline
```

In [2]:

```
# Load pickled data
import pickle
## Dataset Summary & Explorationotlib.gridspec as gridspec
from pandas.io.parsers import read_csv
training_file = 'C:\\Users\\Eathish\\Downloads\\traffic_signs_data\\train.p'
validation_file='C:\\Users\\Eathish\\Downloads\\traffic_signs_data\\valid.p'
testing_file = 'C:\\Users\\Eathish\\Downloads\\traffic_signs_data\\test.p'
with open(training_file, mode='rb') as f:
    train = pickle.load(f)
with open(validation_file, mode='rb') as f:
    valid = pickle.load(f)
with open(testing_file, mode='rb') as f:
    test = pickle.load(f)
sign_names = read_csv("C:\\Users\\Eathish\\Downloads\\traffic_signs_data\\signnames.csv").v
X_train, y_train = train['features'], train['labels']
X_valid, y_valid = valid['features'], valid['labels']
X_test, y_test = test['features'], test['labels']
```

Dataset Summary & Exploration

In [3]:

```
# Number of training examples
n_train = X_train.shape[0]
# Number of validation examples
n_validation = X_valid.shape[0]
# Number of testing examples.
n_test = X_test.shape[0]
# What's the shape of an traffic sign image?
image_shape = X_train[0].shape
# Unique classes/labels there are in the dataset.
classes, class_indices, class_counts  = np.unique(y_train, return_index=True, return_counts
n_classes = len(class_counts)
print("Number of training examples =", n_train)
print("Number of validation examples =", n_validation)
print("Number of testing examples =", n_test)
print("Image data shape =", image_shape)
print("Number of classes =", n_classes)
```

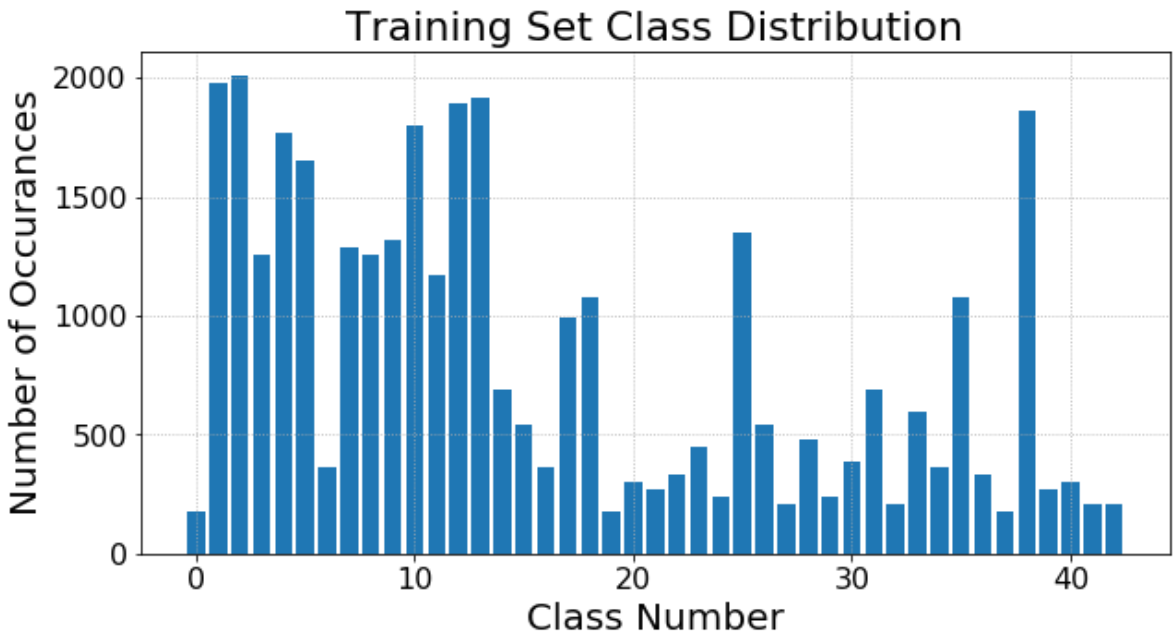
```
Number of training examples = 34799
Number of validation examples = 4410
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
```

Histogram of class distrubtions across data set splits

In [4]:

```
plt.figure(0, figsize = (10,5))
unique_train, counts_train = np.unique(y_train, return_counts = True)
plt.bar(unique_train, counts_train)

plt.title('Training Set Class Distribution', fontsize=22)
plt.xlabel('Class Number', fontsize=20)
plt.ylabel('Number of Occurances', fontsize=20)
plt.tick_params(labelsize=16)
plt.grid(linestyle=':')
```



Visualise all Images Classes

In [5]:

```
from matplotlib import pyplot
for c, c_i, c_count in zip(classes, class_indices, class_counts):
    print(c, ". Class : ", sign_names[c] )
    fig = pyplot.figure(figsize = (10, 1))
    fig.subplots_adjust(left = 0, right = 1, bottom = 0, top = 1, hspace = 0.05, wspace = 0
    for i in range(10):
        axis = fig.add_subplot(1, 10, i + 1, xticks=[], yticks=[])
        random_indices = np.random.randint(c_i, c_i+c_count, 10)
        axis.imshow(X_train[random_indices[i],:,:,:])
        #axis.text(0, 0, '{}: {}'.format(c, sign_names[c]), color='k',backgroundcolor='c',

    pyplot.show()
```

```
0 . Class :  Speed limit (20km/h)
```

```
1 . Class :  Speed limit (30km/h)
```

```
2 . Class :  Speed limit (50km/h)
```

Brief Visual Overview of the Dataset

In [7]:

```
from matplotlib import gridspec
def plot_random_each_class(n_row,n_col,X,y):

    plt.figure(figsize = (25,15))
    gs1 = gridspec.GridSpec(n_row,n_row)
    gs1.update(wspace=0.01, hspace=0.01) # set the spacing between axes.

    for c, c_i, c_count in zip(classes, class_indices, class_counts):
        # i = i + 1 # grid spec indexes from 0
        ax1 = plt.subplot(gs1[c])
        plt.axis('on')
        ax1.set_xticklabels([])
        ax1.set_yticklabels([])
        ax1.set_aspect('equal')
        #plt.subplot(4,11,i+1)
        ind_plot = np.random.randint(c_i, c_i+c_count)
        plt.imshow(X[ind_plot])
        #plt.text(2,4,str(y[ind_plot]),color='k',backgroundcolor='c', fontsize=15)
        plt.text(0, 0, '{}: {:.20}'.format(c, sign_names[c]), color='k',backgroundcolor='c'

        plt.axis('off')
    plt.show()

plot_random_each_class(7,7,X_train,y_train)
```

In [8]:

```
##Load Dataset
data_dir = 'C:\\Users\\Eathish\\Downloads\\archive\\'
train_path = 'C:\\Users\\Eathish\\Downloads\\archive\\Train'
test_path = 'C:\\Users\\Eathish\\Downloads\\archive\\'
IMG_HEIGHT = 30
IMG_WIDTH = 30
```

In [9]:

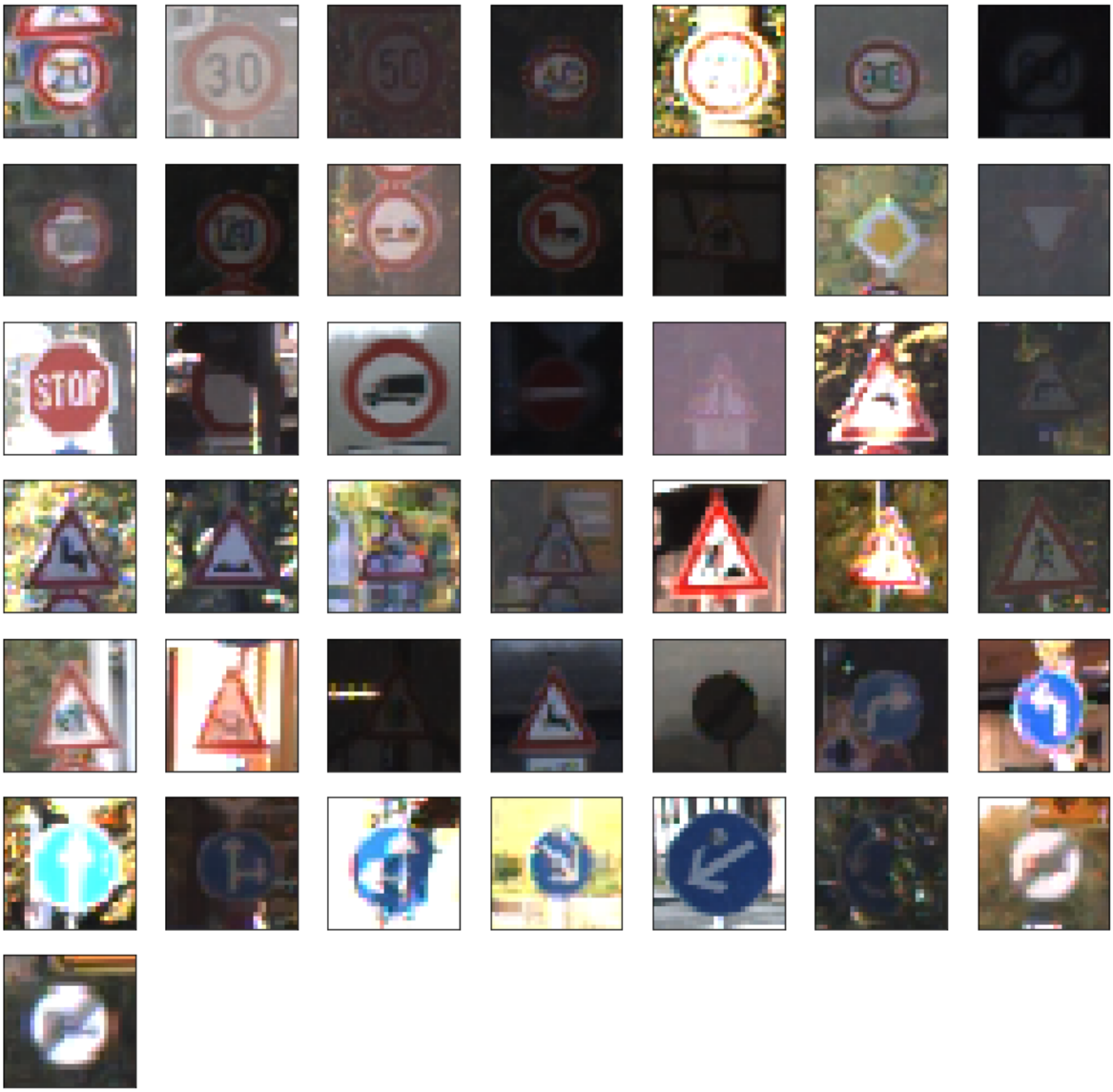
```
# Number of Classes
NUM_CATEGORIES = len(os.listdir(train_path))
NUM_CATEGORIES
```

Out[9]:

```
43
```

In [10]:

```
# Visualizing all the different Signs
img_dir = pathlib.Path(train_path)
plt.figure(figsize=(14,14))
index = 0
for i in range(NUM_CATEGORIES):
    plt.subplot(7, 7, i+1)
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    sign = list(img_dir.glob(f'{i}/*'))[0]
    img = load_img(sign, target_size=(IMG_WIDTH, IMG_HEIGHT))
    plt.imshow(img)
plt.show()
```



In [11]:

```
def load_data(data_dir):
    images = list()
    labels = list()
    for category in range(NUM_CATEGORIES):
        categories = os.path.join(data_dir, str(category))
        for img in os.listdir(categories):
            img = load_img(os.path.join(categories, img), target_size=(30, 30))
            image = img_to_array(img)
            images.append(image)
            labels.append(category)

    return images, labels
```

spliting data for training

In [12]:

```
images, labels = load_data(train_path)

# One hot encoding the labels
labels = to_categorical(labels)

# Splitting the dataset into training and test set
x_train, x_test, y_train, y_test = train_test_split(np.array(images), labels, test_size=0.4
```

Creating the Model

In [13]:

```
model = Sequential()

# First Convolutional Layer
model.add(Conv2D(filters=32, kernel_size=3, activation='relu', input_shape=(IMG_HEIGHT,IMG_
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))

# Second Convolutional Layer
model.add(Conv2D(filters=64, kernel_size=3, activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))

# Third Convolutional Layer
model.add(Conv2D(filters=64, kernel_size=3, activation='relu'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 12, 12, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928
=================================================================
Total params: 56,320
Trainable params: 56,320
Non-trainable params: 0
_________________________________________________________________
```

In [14]:

```
# Flattening the Layer and adding Dense Layer
model.add(Flatten())
model.add(Dense(units=64, activation='relu'))
model.add(Dense(NUM_CATEGORIES, activation='softmax'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 12, 12, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 64)                65600
_________________________________________________________________
dense_1 (Dense)              (None, 43)                2795
=================================================================
Total params: 124,715
Trainable params: 124,715
Non-trainable params: 0
_________________________________________________________________
```

In [15]:

```
# Compiling the model
model.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

In [16]:

```
# Fitting the model
EPOCHS = 30
history = model.fit(x_train,
                    y_train,
                    validation_data = (x_test, y_test),
                    epochs=EPOCHS,
                    steps_per_epoch=60
                   )
```

```
Epoch 1/30
60/60 [==============================] - 39s 548ms/step - loss: 5.4127 - acc
uracy: 0.0496 - val_loss: 3.6062 - val_accuracy: 0.0594
Epoch 2/30
60/60 [==============================] - 32s 538ms/step - loss: 3.2418 - acc
uracy: 0.1788 - val_loss: 2.5816 - val_accuracy: 0.3756
Epoch 3/30
60/60 [==============================] - 33s 543ms/step - loss: 2.1564 - acc
uracy: 0.4322 - val_loss: 1.4866 - val_accuracy: 0.6007
Epoch 4/30
60/60 [==============================] - 32s 535ms/step - loss: 1.2857 - acc
uracy: 0.6337 - val_loss: 0.8078 - val_accuracy: 0.7808
Epoch 5/30
60/60 [==============================] - 33s 545ms/step - loss: 0.8567 - acc
uracy: 0.7469 - val_loss: 0.5408 - val_accuracy: 0.8535
Epoch 6/30
60/60 [==============================] - 32s 535ms/step - loss: 0.6295 - acc
uracy: 0.8141 - val_loss: 0.3954 - val_accuracy: 0.9003
Epoch 7/30
60/60 [==============================] - 32s 539ms/step - loss: 0.5039 - acc
uracy: 0.8493 - val_loss: 0.3219 - val_accuracy: 0.9183
Epoch 8/30
60/60 [==============================] - 33s 550ms/step - loss: 0.4084 - acc
uracy: 0.8784 - val_loss: 0.2562 - val_accuracy: 0.9375
Epoch 9/30
60/60 [==============================] - 32s 539ms/step - loss: 0.3551 - acc
uracy: 0.8960 - val_loss: 0.2140 - val_accuracy: 0.9434
Epoch 10/30
60/60 [==============================] - 33s 545ms/step - loss: 0.3146 - acc
uracy: 0.9061 - val_loss: 0.1871 - val_accuracy: 0.9529
Epoch 11/30
60/60 [==============================] - 35s 588ms/step - loss: 0.2801 - acc
uracy: 0.9192 - val_loss: 0.1690 - val_accuracy: 0.9586
Epoch 12/30
60/60 [==============================] - 32s 535ms/step - loss: 0.2368 - acc
uracy: 0.9306 - val_loss: 0.1457 - val_accuracy: 0.9633
Epoch 13/30
60/60 [==============================] - 32s 539ms/step - loss: 0.2154 - acc
uracy: 0.9379 - val_loss: 0.1412 - val_accuracy: 0.9607
Epoch 14/30
60/60 [==============================] - 37s 621ms/step - loss: 0.2048 - acc
uracy: 0.9391 - val_loss: 0.1153 - val_accuracy: 0.9699
Epoch 15/30
60/60 [==============================] - 42s 704ms/step - loss: 0.1832 - acc
uracy: 0.9467 - val_loss: 0.0997 - val_accuracy: 0.9761
Epoch 16/30
60/60 [==============================] - 40s 661ms/step - loss: 0.1769 - acc
uracy: 0.9491 - val_loss: 0.1042 - val_accuracy: 0.9740
Epoch 17/30
60/60 [==============================] - 35s 591ms/step - loss: 0.1501 - acc
```

```
uracy: 0.9550 - val_loss: 0.1072 - val_accuracy: 0.9720
Epoch 18/30
60/60 [==============================] - 32s 537ms/step - loss: 0.1446 - acc
uracy: 0.9579 - val_loss: 0.0953 - val_accuracy: 0.9753
Epoch 19/30
60/60 [==============================] - 32s 537ms/step - loss: 0.1427 - acc
uracy: 0.9589 - val_loss: 0.0877 - val_accuracy: 0.9797
Epoch 20/30
60/60 [==============================] - 33s 557ms/step - loss: 0.1296 - acc
uracy: 0.9598 - val_loss: 0.0917 - val_accuracy: 0.9777
Epoch 21/30
60/60 [==============================] - 32s 541ms/step - loss: 0.1242 - acc
uracy: 0.9631 - val_loss: 0.0772 - val_accuracy: 0.9810
Epoch 22/30
60/60 [==============================] - 33s 546ms/step - loss: 0.1114 - acc
uracy: 0.9668 - val_loss: 0.0747 - val_accuracy: 0.9802
Epoch 23/30
60/60 [==============================] - 33s 549ms/step - loss: 0.1110 - acc
uracy: 0.9661 - val_loss: 0.0677 - val_accuracy: 0.9830
Epoch 24/30
60/60 [==============================] - 33s 544ms/step - loss: 0.1034 - acc
uracy: 0.9716 - val_loss: 0.0686 - val_accuracy: 0.9839
Epoch 25/30
60/60 [==============================] - 32s 539ms/step - loss: 0.0998 - acc
uracy: 0.9724 - val_loss: 0.0669 - val_accuracy: 0.9834
Epoch 26/30
60/60 [==============================] - 33s 548ms/step - loss: 0.0942 - acc
uracy: 0.9712 - val_loss: 0.0620 - val_accuracy: 0.9841
Epoch 27/30
60/60 [==============================] - 33s 548ms/step - loss: 0.0876 - acc
uracy: 0.9740 - val_loss: 0.0628 - val_accuracy: 0.9852
Epoch 28/30
60/60 [==============================] - 32s 542ms/step - loss: 0.0854 - acc
uracy: 0.9756 - val_loss: 0.0551 - val_accuracy: 0.9874
Epoch 29/30
60/60 [==============================] - 33s 547ms/step - loss: 0.0964 - acc
uracy: 0.9718 - val_loss: 0.0622 - val_accuracy: 0.9844
Epoch 30/30
60/60 [==============================] - 32s 537ms/step - loss: 0.0724 - acc
uracy: 0.9786 - val_loss: 0.0587 - val_accuracy: 0.9864
```

In [17]:

```
loss, accuracy = model.evaluate(x_test, y_test)

print('test set accuracy: ', accuracy * 100)
```

```
491/491 [==============================] - 6s 12ms/step - loss: 0.0587 - acc
uracy: 0.9864
test set accuracy:  98.64192605018616
```

Plotting the Accuracy and Loss values

In [18]:

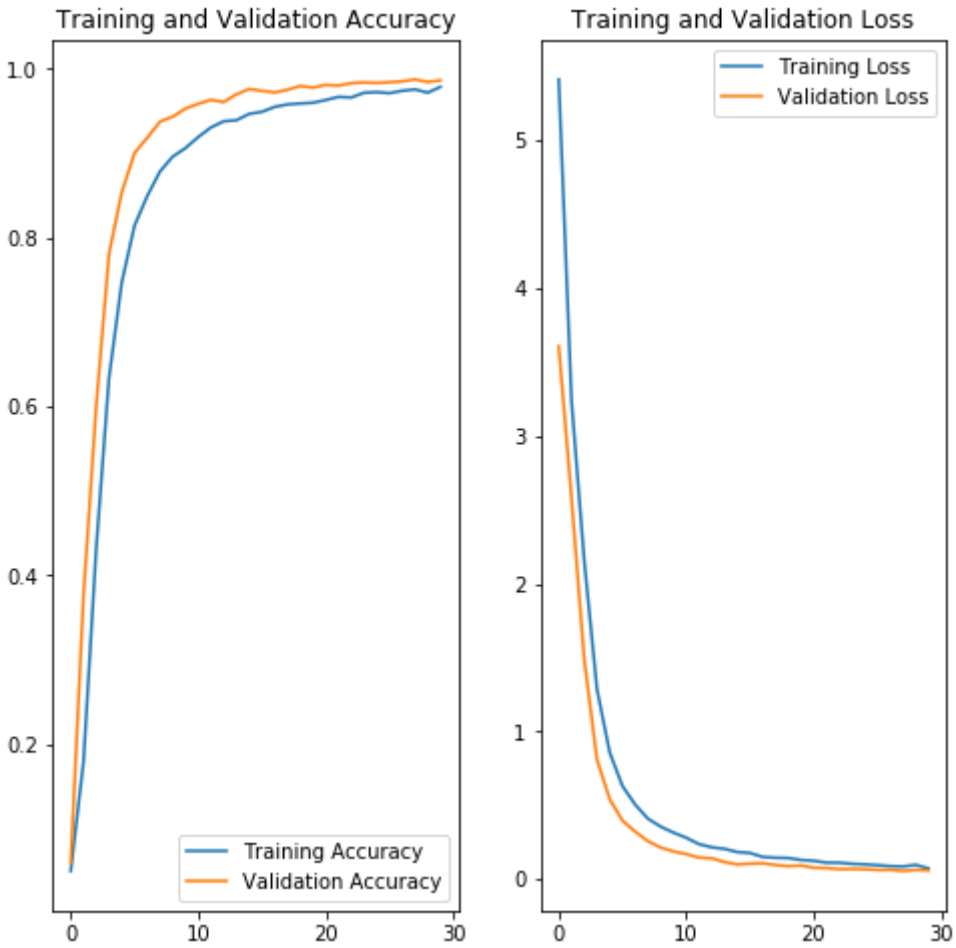
```
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

loss=history.history['loss']
val_loss=history.history['val_loss']

epochs_range = range(EPOCHS)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, accuracy, label='Training Accuracy')
plt.plot(epochs_range, val_accuracy, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



Now Prediction on test data

In [20]:

```
from sklearn.metrics import accuracy_score
import pandas as pd

Y_test = pd.read_csv(test_path + 'Test.csv')
test_labels = Y_test["ClassId"].values
test_images = Y_test["Path"].values

output = list()
for img in test_images:
    image = load_img(os.path.join(test_path, img), target_size=(30, 30))
    output.append(np.array(image))

X_test=np.array(output)
pred = model.predict_classes(X_test)

#Accuracy with the test data
print('Test Data accuracy: ',accuracy_score(test_labels, pred)*100)
```

```
C:\Users\Eathish\Anaconda3\lib\site-packages\tensorflow\python\keras\engine
\sequential.py:455: UserWarning: `model.predict_classes()` is deprecated and
will be removed after 2021-01-01. Please use instead:* `np.argmax(model.pred
ict(x), axis=-1)`,   if your model does multi-class classification   (e.g. i
f it uses a `softmax` last-layer activation).* `(model.predict(x) > 0.5).ast
ype("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 '

Test Data accuracy:  93.80047505938242
```

In [21]:

```
plt.figure(figsize = (13, 13))

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

In [16]:

```
model.save('C:\\Users\\Eathish\\Downloads\\archive\\model-1.h5')
```

In [ ]:

```

```