Assignment : IDfy

- @Name : Subham Sarkar
- @Github : https://github.com/SubhamIO
- @LinkedIn : https://www.linkedin.com/in/subham-sarkar-4224aa147/
- @Portfolio : https://subhamio.github.io/SubhamSarkar-PortfolioWebsite/

Title : Text Recognition from low quality License Plates using Deep Learning

Objective:

- The objective of this assignment is to build an OCR solution for the provided dataset. This specific dataset is normal and HDR readings of license plates.

Requirements:

1. Use an 80:20 train:test split on the provided dataset
2. Create a model for reading the text using any approaches or tools that you are familiar with or can learn
3. Use the available test set to check the accuracy of your model

Dataset:

1. The image dataset here contains 652 images of cropped license plates with a csv containing annotation as well.

- Link: https://medusa.fit.vutbr.cz/traffic/download/513/

Load Libraries

In [1]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%
b&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2
www.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly
ttps%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

In [2]:

```
cd drive/My\ Drive/
```

```
/content/drive/My Drive
```

In [3]:

```
import re
import cv2
import numpy as np

from matplotlib import pyplot as plt
```

```
import pandas as pd
from keras.preprocessing.image import *
from keras.layers.core import *
import tensorflow as tf
from keras.layers import *
from keras.models import *
import keras
from keras import backend as K
from keras.callbacks import *
np.random.seed(0)
from keras.utils.np_utils import to_categorical
from keras.regularizers import l2
import seaborn as sns
```

```
Using TensorFlow backend.
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

Data Acquisition

In [0]:

```
# base_dir = '/Users/subham/Desktop/2017-IWT4S-HDR_LP-dataset/'
base_dir = "./2017-IWT4S-HDR_LP-dataset/"
```

In [0]:

```
data = pd.read_csv("./2017-IWT4S-HDR_LP-dataset/" + "/trainVal.csv")
```

In [6]:

```
data.tail()
```

Out[6]:

	track_id	image_path	lp	train
647	./crop_m4/I00084.png	./crop_h4/I00084.png	2B90178	0
648	./crop_m4/I00085.png	./crop_m4/I00085.png	7B59839	0
649	./crop_m4/I00085.png	./crop_h4/I00085.png	7B59839	0
650	./crop_m4/I00086.png	./crop_m4/I00086.png	7B11123	0
651	./crop_m4/I00086.png	./crop_h4/I00086.png	7B11123	0

In [17]:

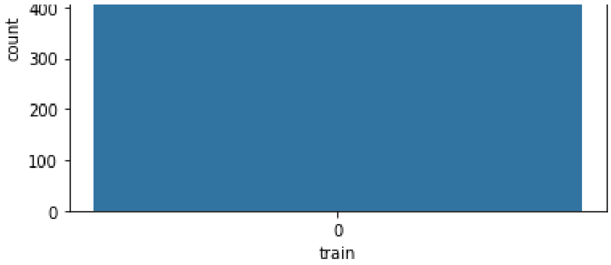
```
n = data.shape[0]
print(n)
```

```
652
```

In [18]:

```
ax = sns.countplot(x=data['train'], data=data)
print(data['train'].value_counts())
```

```
0    652
Name: train, dtype: int64
```



Observation :

- It has only test data as per README.txt provided

In [19]:

```
len(data['track_id'].unique())
```

Out[19]:

```
326
```

In [20]:

```
len(data['image_path'].unique())
```

Out[20]:

```
652
```

In [21]:

```
len(data['lp'].unique())
```

Out[21]:

```
302
```

Observation :

- Unique track_id = 326
- Unique image_path = 652
- Many : 1 relationship [track_id : image_path]
- Unique lp = 302 , this means same images are present in different folders. So, not much variation in image types also the dataset is small. Accuracy will be impacted.

Data Preprocessing

Create a dictionary of length 37 [26 alphabets and 10 numbers and space] for target map

In [0]:

```
letters = " ABCDEFGHIJKLMNPQRSTUVWXYZ0123456789"
dic = {}
for i in range(len(letters)):
    dic[i] = letters[i]
invert_dic = {}
for i in range(len(letters)):
    invert_dic[letters[i]] = i
```

In [23]:

```
print(dic)
```

```
{0: ' ', 1: 'A', 2: 'B', 3: 'C', 4: 'D', 5: 'E', 6: 'F', 7: 'G', 8: 'H', 9: 'I', 10: 'J', 11: 'K',
12: 'L', 13: 'M', 14: 'N', 15: 'P', 16: 'Q', 17: 'R', 18: 'S', 19: 'T', 20: 'U', 21: 'V', 22: 'W',
23: 'X', 24: 'Y', 25: 'Z', 26: '0', 27: '1', 28: '2', 29: '3', 30: '4', 31: '5', 32: '6', 33: '7',
34: '8', 35: '9'}
```

In [24]:

```
print(invert_dic)
```

```
{' ': 0, 'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 5, 'F': 6, 'G': 7, 'H': 8, 'I': 9, 'J': 10, 'K': 11,
'L': 12, 'M': 13, 'N': 14, 'P': 15, 'Q': 16, 'R': 17, 'S': 18, 'T': 19, 'U': 20, 'V': 21, 'W': 22,
'X': 23, 'Y': 24, 'Z': 25, '0': 26, '1': 27, '2': 28, '3': 29, '4': 30, '5': 31, '6': 32, '7': 33,
'8': 34, '9': 35}
```

In [0]:

```
X_train = []
y_train = []
X_test = []
y_test = []
```

Creating X(feature) , Y(target)

In [0]:

```
X = []
Y = []
```

In [0]:

```
for i in range(n):
    temp_y= np.zeros((8)) # Target variable size = 8 dimensional
    path = base_dir + data["image_path"][i]
    # Read the images in gray scale
    temp_x = cv2.imread(base_dir + data["image_path"][i], cv2.IMREAD_GRAYSCALE)
    # Resizing as per our need to process in our CNN architecture
    temp_x = cv2.resize(temp_x,(256,64))
    X.append(temp_x)
    # Let's loop over each ground truth and assign each character with index
    for j,k in enumerate(data["lp"][i]):
        temp_y[j] = invert_dic[k]

    Y.append(temp_y)
```

In [28]:

```
len(X), len(Y)
```

Out[28]:

```
(652, 652)
```

In [0]:

```
# import pickle

# with open('X_train', 'wb') as fp:
#     pickle.dump(X_train, fp)
```

In [0]:

```
# with open('X_test', 'wb') as fp:
#     pickle.dump(X_test, fp)
```

In [0]:

```
# with open('y_train', 'wb') as fp:
#     pickle.dump(y_train, fp)
```

In [0]:

```
# with open('y_test', 'wb') as fp:
#     pickle.dump(y_test, fp)
```

In [0]:

```
# with open ('y_test', 'rb') as fp:
#     y_test1 = pickle.load(fp)
```

Let's look at the mapping created

In [30]:

```
data['lp'][0]
```

Out[30]:

```
'9B52145'
```

{' ': 0, 'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 5, 'F': 6, 'G': 7, 'H': 8, 'I': 9, 'J': 10, 'K': 11, 'L': 12, 'M': 13, 'N': 14, 'P': 15, 'Q': 16, 'R': 17, 'S': 18, 'T': 19, 'U': 20, 'V': 21, 'W': 22, 'X': 23, 'Y': 24, 'Z': 25, '0': 26, '1': 27, '2': 28, '3': 29, '4': 30, '5': 31, '6': 32, '7': 33, '8': 34, '9': 35}

In [31]:

```
Y[0]
```

Out[31]:

```
array([35.,  2., 31., 28., 27., 30., 31.,  0.])
```

Train-Test Split (80:20)

In [32]:

```
ll = 0.8*len(X)
ll = int(ll)
X_train = X[:ll]
X_test = X[ll:]
y_train = Y[:ll]
y_test = Y[ll:]
print(len(X_train), len(y_train), len(X_test), len(y_test))
```

```
521 521 131 131
```

In [0]:

```
# reshaping the array [-1 automatically adjust the number of data points here]
X_train = np.array(X_train).reshape(-1,64,256,1)
y_train = np.array(y_train)
X_test = np.array(X_test).reshape(-1,64,256,1)
y_test = np.array(y_test)

# Normalise the data
X_train = X_train /255
X_test = X_test/255

# converting target to_categorical (encoding the target characters to 36 dimensions)
y_test = to_categorical(y_test,36)
y_train = to_categorical(y_train,36)
```

```
y_train = to_categorical(y_train,36)
```

In [34]:

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

```
(521, 64, 256, 1)
(521, 8, 36)
(131, 64, 256, 1)
(131, 8, 36)
```

In [0]:

```
print(y_train[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. 1.]
 [0. 0. 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. 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. 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. 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. 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. 1. 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. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

Data Modelling

Let's define VGG-16 architecture

- Could have used transfer learning but VGG-16 takes (224,224,3) as input dimension
- So, better define the VGG-16 architecture from scratch , so that we can specify input dimensions manually.

In [0]:

```
def VGG(shape=(64, 256, 1),n_channels=64,weight_decay=0,batch_momentum=0.99):
    bn_axis = 3
    input_ = Input(shape=shape)
    x = Conv2D(128, (3, 3), padding='same', name='block1_conv1', kernel_initializer='he_normal', ke
rnel_regularizer=l2(weight_decay))(input_)
    x = BatchNormalization(axis=bn_axis, name='bn00_x1', momentum=batch_momentum)(x)
    x = Activation('relu')(x)
    x = Conv2D(128, (3, 3), padding='same', name='block1_conv2', kernel_initializer='he_normal', ke
rnel_regularizer=l2(weight_decay))(x)
    x = BatchNormalization(axis=bn_axis, name='bn01_x2', momentum=batch_momentum)(x)
    x = Activation('relu')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='block1_pool')(x)

    # Block 2
    x = Conv2D(128, (3, 3), padding='same', name='block2_conv1', kernel_initializer='he_normal', ke
rnel_regularizer=l2(weight_decay))(x)
    x = BatchNormalization(axis=bn_axis, name='bn11_x1', momentum=batch_momentum)(x)
    x = Activation('relu')(x)
    x = Conv2D(128, (3, 3), padding='same', name='block2_conv2', kernel_initializer='he_normal', ke
rnel_regularizer=l2(weight_decay))(x)
    x = BatchNormalization(axis=bn_axis, name='bn12_x2', momentum=batch_momentum)(x)
    x = Activation('relu')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='block2_pool')(x)

    # Block 3
```

```
    x = Conv2D(256, (3, 3), padding='same', name='block3_conv1', kernel_initializer='he_normal', ke
rnel_regularizer=l2(weight_decay))(x)
    x = BatchNormalization(axis=bn_axis, name='bn21_x1', momentum=batch_momentum)(x)
    x = Activation('relu')(x)
    x = Conv2D(256, (3, 3), padding='same', name='block3_conv2', kernel_initializer='he_normal', ke
rnel_regularizer=l2(weight_decay))(x)
    x = BatchNormalization(axis=bn_axis, name='bn22_x2', momentum=batch_momentum)(x)
    x = Activation('relu')(x)
#    x = Conv2D(256, (3, 3), padding='same', name='block3_conv3',
kernel_initializer='glorot_uniform', kernel_regularizer=l2(weight_decay))(x)
#    x = BatchNormalization(axis=bn_axis, name='bn23_x3', momentum=batch_momentum)(x)
#    x = Activation('relu')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='block3_pool')(x)

    # Block 4
    x = Conv2D(512, (3, 3), padding='same', name='block4_conv1', kernel_initializer='he_normal', ke
rnel_regularizer=l2(weight_decay))(x)
    x = BatchNormalization(axis=bn_axis, name='bn31_x2', momentum=batch_momentum)(x)
    x = Activation('relu')(x)

    x = Conv2D(512, (3, 3), padding='same', name='block4_conv2', kernel_regularizer=l2(weight_decay
))(x)
    x = BatchNormalization(axis=bn_axis, name='bn32_x2', momentum=batch_momentum)(x)
    x = Activation('relu')(x)
#    x = Conv2D(512, (3, 3), padding='same', name='block4_conv3',
kernel_initializer='glorot_uniform', kernel_regularizer=l2(weight_decay))(x)
#    x = BatchNormalization(axis=bn_axis, name='bn33_x2', momentum=batch_momentum)(x)
#    x = Activation('relu')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='block4_pool')(x)

    # Block 5
    x = Conv2D(512, (3, 3), padding='same', name='block5_conv1', kernel_initializer='he_normal', ke
rnel_regularizer=l2(weight_decay))(x)
    x = BatchNormalization(axis=bn_axis, name='bn41_x2', momentum=batch_momentum)(x)
    x = Activation('relu')(x)
#    x = Conv2D(512, (3, 3), padding='same', name='block5_conv2',
kernel_initializer='glorot_uniform', kernel_regularizer=l2(weight_decay))(x)
#    x = BatchNormalization(axis=bn_axis, name='bn42_x2', momentum=batch_momentum)(x)
#    x = Activation('relu')(x)

    x = Conv2D(1024, (3, 3), padding='same', name='block5_conv3', kernel_initializer='he_normal', k
ernel_regularizer=l2(weight_decay))(x)
    x = BatchNormalization(axis=bn_axis, name='bn43_x2', momentum=batch_momentum)(x)
    x = Activation('relu')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='block5_pool')(x)

    x = Conv2D(1024, (3, 3), padding='same', name='block6_conv1', kernel_initializer='he_normal', k
ernel_regularizer=l2(weight_decay))(x)
    x = BatchNormalization(axis=bn_axis, name='bn51_x2', momentum=batch_momentum)(x)
    x = Activation('relu')(x)

    x = Conv2D(1024*2, (3, 3), padding='same', name='block6_conv12', kernel_initializer='he_normal'
, kernel_regularizer=l2(weight_decay))(x)
    x = BatchNormalization(axis=bn_axis, name='bn51_x22', momentum=batch_momentum)(x)
    x = Activation('relu')(x)
    x= Dropout(0.3, noise_shape=None, seed=None)(x)


    #block5


    X = AveragePooling2D((2, 2), strides = (2, 1), name='avg_pool1',padding ='same')(x)
    X = Reshape((8,1024*2))(X)


    X = Conv1D(512, 3, strides=1, padding='same',name = 'conv1y'  ,activation=None,  dilation_rate=
1,  use_bias=True, kernel_initializer="he_normal", kernel_regularizer=regularizers.l2(0.000))(X)
    X = BatchNormalization(axis = 2, name = 'bn01y')(X)
    X = Activation('relu')(X)
    X= Dropout(0.3, noise_shape=None, seed=None)(X)
    X = Conv1D(36, 1 , strides=1, padding='same',name = 'conv1x'  ,activation=None,  dilation_rate=
1,  use_bias=True, kernel_initializer="he_normal")(X)
    X = BatchNormalization(axis = 2, name = 'bnhe')(X)
    X = Activation('softmax')(X)
    model = Model(inputs = [input_], outputs = [X])
    return model
```

In [0]:

```
model = VGG(shape=(64, 256, 1))
```

In [0]:

```
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 64, 256, 1)        0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 64, 256, 128)      1280
_________________________________________________________________
bn00_x1 (BatchNormalization) (None, 64, 256, 128)      512
_________________________________________________________________
activation_1 (Activation)    (None, 64, 256, 128)      0
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 64, 256, 128)      147584
_________________________________________________________________
bn01_x2 (BatchNormalization) (None, 64, 256, 128)      512
_________________________________________________________________
activation_2 (Activation)    (None, 64, 256, 128)      0
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 32, 128, 128)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 32, 128, 128)      147584
_________________________________________________________________
bn11_x1 (BatchNormalization) (None, 32, 128, 128)      512
_________________________________________________________________
activation_3 (Activation)    (None, 32, 128, 128)      0
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 32, 128, 128)      147584
_________________________________________________________________
bn12_x2 (BatchNormalization) (None, 32, 128, 128)      512
_________________________________________________________________
activation_4 (Activation)    (None, 32, 128, 128)      0
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 16, 64, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 16, 64, 256)       295168
_________________________________________________________________
bn21_x1 (BatchNormalization) (None, 16, 64, 256)       1024
_________________________________________________________________
activation_5 (Activation)    (None, 16, 64, 256)       0
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 16, 64, 256)       590080
_________________________________________________________________
bn22_x2 (BatchNormalization) (None, 16, 64, 256)       1024
_________________________________________________________________
activation_6 (Activation)    (None, 16, 64, 256)       0
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 8, 32, 256)        0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 8, 32, 512)        1180160
_________________________________________________________________
bn31_x2 (BatchNormalization) (None, 8, 32, 512)        2048
_________________________________________________________________
activation_7 (Activation)    (None, 8, 32, 512)        0
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 8, 32, 512)        2359808
_________________________________________________________________
bn32_x2 (BatchNormalization) (None, 8, 32, 512)        2048
_________________________________________________________________
activation_8 (Activation)    (None, 8, 32, 512)        0
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 4, 16, 512)        0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 4, 16, 512)        2359808
_________________________________________________________________
bn41_x2 (BatchNormalization) (None, 4, 16, 512)        2048
_________________________________________________________________
```

```
activation_9 (Activation)    (None, 4, 16, 512)        0
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 4, 16, 1024)       4719616
_________________________________________________________________
bn43_x2 (BatchNormalization) (None, 4, 16, 1024)       4096
_________________________________________________________________
activation_10 (Activation)   (None, 4, 16, 1024)       0
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 2, 8, 1024)        0
_________________________________________________________________
block6_conv1 (Conv2D)        (None, 2, 8, 1024)        9438208
_________________________________________________________________
bn51_x2 (BatchNormalization) (None, 2, 8, 1024)        4096
_________________________________________________________________
activation_11 (Activation)   (None, 2, 8, 1024)        0
_________________________________________________________________
block6_conv12 (Conv2D)       (None, 2, 8, 2048)        18876416
_________________________________________________________________
bn51_x22 (BatchNormalization (None, 2, 8, 2048)        8192
_________________________________________________________________
activation_12 (Activation)   (None, 2, 8, 2048)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2, 8, 2048)        0
_________________________________________________________________
avg_pool1 (AveragePooling2D) (None, 1, 8, 2048)        0
_________________________________________________________________
reshape_1 (Reshape)          (None, 8, 2048)           0
_________________________________________________________________
conv1y (Conv1D)              (None, 8, 512)            3146240
_________________________________________________________________
bn01y (BatchNormalization)   (None, 8, 512)            2048
_________________________________________________________________
activation_13 (Activation)   (None, 8, 512)            0
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 512)            0
_________________________________________________________________
conv1x (Conv1D)              (None, 8, 36)             18468
_________________________________________________________________
bnhe (BatchNormalization)    (None, 8, 36)             144
_________________________________________________________________
activation_14 (Activation)   (None, 8, 36)             0
=================================================================
Total params: 43,456,820
Trainable params: 43,442,412
Non-trainable params: 14,408
_________________________________________________________________
```

Defining custom metrics

In [0]:

```python
def custom_loss(y_true, y_pred):
    s = K.shape(y_pred)
    y_true = K.reshape(y_true,(-1,s[-1]))
    y_pred = K.reshape(y_pred,(-1,s[-1]))
    loss = K.sum(keras.losses.categorical_crossentropy(y_true, y_pred))
    num = K.shape(y_true)[0]
    num=tf.cast(num,tf.float32)
    return K.mean(loss)/num
```

In [0]:

```python
def metric1(y_true, y_pred):

    s = K.shape(y_pred)

    # reshape such that w and h dim are multiplied together
    y_true_reshaped = K.reshape( y_true,  (-1,s[-1]) )
    y_pred_reshaped = K.reshape( y_pred,  (-1, s[-1]) )

    # correctly classified
    clf_pred =  K.argmax(y_pred_reshaped,axis = -1)
    y_true = K.argmax(y_true_reshaped,axis = -1)
    correct_pixels_per_class = K.cast( K.equal(clf_pred,y_true), dtype='float32') #if equal
```

```
    correct_pixels_per_class = K.cast( K.equal(clf_pred,y_true), dtype='float32') #if equal

    return K.sum(correct_pixels_per_class) / K.cast(K.prod(s[:-1]), dtype='float32') #accuracy
```

In [0]:

```
def metric2(y_true, y_pred):

    s = K.shape(y_pred)

    # correctly classified
    clf_pred =  K.argmax(y_pred,axis = -1)
    y_true = K.argmax(y_true,axis = -1)
    correct_pixels_per_class = K.cast(K.all( K.equal(clf_pred,y_true),axis=-1), dtype='float32') #i
f equal

    return K.sum(correct_pixels_per_class) / K.cast(K.prod(s[0]), dtype='float32') #accuracy
```

In [0]:

```
model.compile(loss = custom_loss,optimizer='adam',metrics=[metric1,metric2])
```

In [0]:

```
model.load_weights("idfy1.h5")
```

Training

In [0]:

```
datagen = ImageDataGenerator(width_shift_range=0.14,
                                 height_shift_range=0.08,
                                 fill_mode='constant',
                                 zoom_range = 0.1,
                                 rotation_range = 10,
                                 #rescale  =1./255
                                 )

mcp_save = ModelCheckpoint('idfy1.h5', save_best_only=True, monitor='val_loss', mode='min',verbose=
1)

def scheduler(epoch):
    if epoch <3 :
        return 0.001/5
    elif epoch < 10:
        return 0.001/10
    elif epoch < 15:
        return  0.00001
    elif epoch <30:
        return  0.00001/2

n = X_train.shape[0]

lr_reduce = LearningRateScheduler(scheduler,verbose = 1)

history = model.fit_generator(datagen.flow(X_train, y_train,batch_size=64),
                         epochs = 30,
                         steps_per_epoch=n//64,
                         callbacks=[lr_reduce,mcp_save],
                         validation_data=(X_test, y_test))
```

```
Epoch 1/30

Epoch 00001: LearningRateScheduler setting learning rate to 0.0002.
8/8 [==============================] - 12s 2s/step - loss: 0.5540 - metric1: 0.9953 - metric2: 0.9
705 - val_loss: 1.0427 - val_metric1: 0.9232 - val_metric2: 0.7448

Epoch 00001: val_loss improved from inf to 1.04265, saving model to idfy1.h5
Epoch 2/30

Epoch 00002: LearningRateScheduler setting learning rate to 0.0002.
```

```
8/8 [==============================] - 13s 2s/step - loss: 0.5444 - metric1: 0.9980 - metric2: 0.9
844 - val_loss: 1.0200 - val_metric1: 0.9034 - val_metric2: 0.6128

Epoch 00002: val_loss improved from 1.04265 to 1.02004, saving model to idfy1.h5
Epoch 3/30

Epoch 00003: LearningRateScheduler setting learning rate to 0.0002.
8/8 [==============================] - 12s 1s/step - loss: 0.5748 - metric1: 0.9939 - metric2: 0.9
648 - val_loss: 1.0594 - val_metric1: 0.9147 - val_metric2: 0.7031

Epoch 00003: val_loss did not improve from 1.02004
Epoch 4/30

Epoch 00004: LearningRateScheduler setting learning rate to 0.0001.
8/8 [==============================] - 12s 1s/step - loss: 0.5642 - metric1: 0.9963 - metric2: 0.9
724 - val_loss: 1.0557 - val_metric1: 0.9089 - val_metric2: 0.7083

Epoch 00004: val_loss did not improve from 1.02004
Epoch 5/30

Epoch 00005: LearningRateScheduler setting learning rate to 0.0001.
8/8 [==============================] - 11s 1s/step - loss: 0.5984 - metric1: 0.9919 - metric2: 0.9
388 - val_loss: 1.0583 - val_metric1: 0.8956 - val_metric2: 0.5816

Epoch 00005: val_loss did not improve from 1.02004
Epoch 6/30

Epoch 00006: LearningRateScheduler setting learning rate to 0.0001.
8/8 [==============================] - 12s 1s/step - loss: 0.5608 - metric1: 0.9951 - metric2: 0.9
646 - val_loss: 1.0796 - val_metric1: 0.9021 - val_metric2: 0.5868

Epoch 00006: val_loss did not improve from 1.02004
Epoch 7/30

Epoch 00007: LearningRateScheduler setting learning rate to 0.0001.
8/8 [==============================] - 13s 2s/step - loss: 0.5312 - metric1: 0.9976 - metric2: 0.9
824 - val_loss: 0.9968 - val_metric1: 0.9219 - val_metric2: 0.7396

Epoch 00007: val_loss improved from 1.02004 to 0.99679, saving model to idfy1.h5
Epoch 8/30

Epoch 00008: LearningRateScheduler setting learning rate to 0.0001.
8/8 [==============================] - 12s 2s/step - loss: 0.5390 - metric1: 0.9990 - metric2: 0.9
922 - val_loss: 1.0066 - val_metric1: 0.9060 - val_metric2: 0.6128

Epoch 00008: val_loss did not improve from 0.99679
Epoch 9/30

Epoch 00009: LearningRateScheduler setting learning rate to 0.0001.
8/8 [==============================] - 12s 1s/step - loss: 0.5338 - metric1: 0.9988 - metric2: 0.9
922 - val_loss: 1.0025 - val_metric1: 0.9093 - val_metric2: 0.6285

Epoch 00009: val_loss did not improve from 0.99679
Epoch 10/30

Epoch 00010: LearningRateScheduler setting learning rate to 0.0001.
8/8 [==============================] - 12s 1s/step - loss: 0.5339 - metric1: 0.9968 - metric2: 0.9
744 - val_loss: 1.0216 - val_metric1: 0.9212 - val_metric2: 0.7344

Epoch 00010: val_loss did not improve from 0.99679
Epoch 11/30

Epoch 00011: LearningRateScheduler setting learning rate to 1e-05.
8/8 [==============================] - 12s 1s/step - loss: 0.5480 - metric1: 0.9951 - metric2: 0.9
646 - val_loss: 0.9872 - val_metric1: 0.9212 - val_metric2: 0.7240

Epoch 00011: val_loss improved from 0.99679 to 0.98720, saving model to idfy1.h5
Epoch 12/30

Epoch 00012: LearningRateScheduler setting learning rate to 1e-05.
8/8 [==============================] - 12s 2s/step - loss: 0.5247 - metric1: 0.9990 - metric2: 0.9
941 - val_loss: 0.9816 - val_metric1: 0.8974 - val_metric2: 0.6285

Epoch 00012: val_loss improved from 0.98720 to 0.98162, saving model to idfy1.h5
Epoch 13/30

Epoch 00013: LearningRateScheduler setting learning rate to 1e-05.
```

```
8/8 [==============================] - 12s 1s/step - loss: 0.5395 - metric1: 0.9948 - metric2: 0.9
744 - val_loss: 0.9794 - val_metric1: 0.9106 - val_metric2: 0.6285

Epoch 00013: val_loss improved from 0.98162 to 0.97943, saving model to idfy1.h5
Epoch 14/30

Epoch 00014: LearningRateScheduler setting learning rate to 1e-05.
8/8 [==============================] - 12s 1s/step - loss: 0.5256 - metric1: 0.9968 - metric2: 0.9
763 - val_loss: 0.9696 - val_metric1: 0.9271 - val_metric2: 0.7552

Epoch 00014: val_loss improved from 0.97943 to 0.96959, saving model to idfy1.h5
Epoch 15/30

Epoch 00015: LearningRateScheduler setting learning rate to 1e-05.
8/8 [==============================] - 13s 2s/step - loss: 0.4945 - metric1: 0.9990 - metric2: 0.9
941 - val_loss: 0.9579 - val_metric1: 0.9138 - val_metric2: 0.6493

Epoch 00015: val_loss improved from 0.96959 to 0.95785, saving model to idfy1.h5
Epoch 16/30

Epoch 00016: LearningRateScheduler setting learning rate to 5e-06.
8/8 [==============================] - 12s 1s/step - loss: 0.5179 - metric1: 0.9993 - metric2: 0.9
941 - val_loss: 0.9514 - val_metric1: 0.9132 - val_metric2: 0.6441

Epoch 00016: val_loss improved from 0.95785 to 0.95136, saving model to idfy1.h5
Epoch 17/30

Epoch 00017: LearningRateScheduler setting learning rate to 5e-06.
8/8 [==============================] - 12s 1s/step - loss: 0.5193 - metric1: 0.9973 - metric2: 0.9
783 - val_loss: 0.9468 - val_metric1: 0.9138 - val_metric2: 0.6441

Epoch 00017: val_loss improved from 0.95136 to 0.94681, saving model to idfy1.h5
Epoch 18/30

Epoch 00018: LearningRateScheduler setting learning rate to 5e-06.
8/8 [==============================] - 12s 2s/step - loss: 0.5222 - metric1: 0.9968 - metric2: 0.9
744 - val_loss: 0.9498 - val_metric1: 0.9138 - val_metric2: 0.6493

Epoch 00018: val_loss did not improve from 0.94681
Epoch 19/30

Epoch 00019: LearningRateScheduler setting learning rate to 5e-06.
8/8 [==============================] - 12s 1s/step - loss: 0.5223 - metric1: 0.9998 - metric2: 0.9
980 - val_loss: 0.9514 - val_metric1: 0.9138 - val_metric2: 0.6493

Epoch 00019: val_loss did not improve from 0.94681
Epoch 20/30

Epoch 00020: LearningRateScheduler setting learning rate to 5e-06.
8/8 [==============================] - 12s 1s/step - loss: 0.5057 - metric1: 0.9988 - metric2: 0.9
922 - val_loss: 0.9515 - val_metric1: 0.9132 - val_metric2: 0.6493

Epoch 00020: val_loss did not improve from 0.94681
Epoch 21/30

Epoch 00021: LearningRateScheduler setting learning rate to 5e-06.
8/8 [==============================] - 12s 1s/step - loss: 0.5255 - metric1: 0.9990 - metric2: 0.9
941 - val_loss: 0.9485 - val_metric1: 0.9264 - val_metric2: 0.7604

Epoch 00021: val_loss did not improve from 0.94681
Epoch 22/30

Epoch 00022: LearningRateScheduler setting learning rate to 5e-06.
8/8 [==============================] - 12s 1s/step - loss: 0.5119 - metric1: 0.9995 - metric2: 0.9
961 - val_loss: 0.9463 - val_metric1: 0.9271 - val_metric2: 0.7604

Epoch 00022: val_loss improved from 0.94681 to 0.94630, saving model to idfy1.h5
Epoch 23/30

Epoch 00023: LearningRateScheduler setting learning rate to 5e-06.
8/8 [==============================] - 12s 2s/step - loss: 0.5257 - metric1: 0.9983 - metric2: 0.9
922 - val_loss: 0.9433 - val_metric1: 0.9264 - val_metric2: 0.7552

Epoch 00023: val_loss improved from 0.94630 to 0.94326, saving model to idfy1.h5
Epoch 24/30

Epoch 00024: LearningRateScheduler setting learning rate to 5e-06.
```

```
8/8 [==============================] - 13s 2s/step - loss: 0.4999 - metric1: 0.9988 - metric2: 0.9
902 - val_loss: 0.9423 - val_metric1: 0.9132 - val_metric2: 0.6441

Epoch 00024: val_loss improved from 0.94326 to 0.94227, saving model to idfy1.h5
Epoch 25/30

Epoch 00025: LearningRateScheduler setting learning rate to 5e-06.
8/8 [==============================] - 12s 1s/step - loss: 0.4994 - metric1: 0.9993 - metric2: 0.9
961 - val_loss: 0.9408 - val_metric1: 0.9132 - val_metric2: 0.6441

Epoch 00025: val_loss improved from 0.94227 to 0.94077, saving model to idfy1.h5
Epoch 26/30

Epoch 00026: LearningRateScheduler setting learning rate to 5e-06.
8/8 [==============================] - 12s 1s/step - loss: 0.5247 - metric1: 0.9983 - metric2: 0.9
883 - val_loss: 0.9411 - val_metric1: 0.9138 - val_metric2: 0.6441

Epoch 00026: val_loss did not improve from 0.94077
Epoch 27/30

Epoch 00027: LearningRateScheduler setting learning rate to 5e-06.
8/8 [==============================] - 12s 1s/step - loss: 0.5006 - metric1: 0.9993 - metric2: 0.9
941 - val_loss: 0.9423 - val_metric1: 0.9284 - val_metric2: 0.7552

Epoch 00027: val_loss did not improve from 0.94077
Epoch 28/30

Epoch 00028: LearningRateScheduler setting learning rate to 5e-06.
8/8 [==============================] - 12s 1s/step - loss: 0.5101 - metric1: 0.9995 - metric2: 0.9
961 - val_loss: 0.9452 - val_metric1: 0.9284 - val_metric2: 0.7552

Epoch 00028: val_loss did not improve from 0.94077
Epoch 29/30

Epoch 00029: LearningRateScheduler setting learning rate to 5e-06.
8/8 [==============================] - 12s 1s/step - loss: 0.5222 - metric1: 0.9894 - metric2: 0.9
763 - val_loss: 0.9452 - val_metric1: 0.9290 - val_metric2: 0.7604

Epoch 00029: val_loss did not improve from 0.94077
Epoch 30/30

Epoch 00030: LearningRateScheduler setting learning rate to 5e-06.
8/8 [==============================] - 13s 2s/step - loss: 0.4988 - metric1: 0.9966 - metric2: 0.9
766 - val_loss: 0.9445 - val_metric1: 0.9284 - val_metric2: 0.7604

Epoch 00030: val_loss did not improve from 0.94077
```

Utility funtion for plotting

In [0]:

```
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

In [0]:

```
nb_epoch=30
```

In [49]:

```
%matplotlib inline
score = model.evaluate(X_test, y_test, verbose=0)
```

```
score = model.evaluate(X_test, y_test, verbose=0)
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test accuracy: 0.9041666984558105
```

Test Accuracy : 90%

Let's test on test data if it works good !!

In [39]:

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import random
n = X_test.shape[0]
count = 0
pred = model.predict(X_test)
pred = np.argmax(pred,axis=-1)
true = np.argmax(y_test,axis = -1)
for i in range(n):
    if np.all(pred[i,:]==true[i]):
        count +=1
print(count/n)
```

```
0.6335877862595419
```

It gave me accurate results upto 63.4% . Thats great for this small dataset

In [40]:

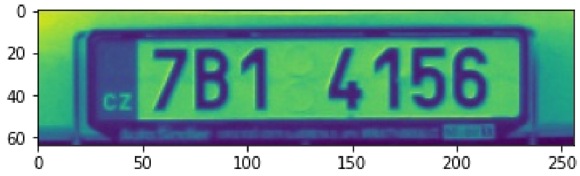
```
k = random.randint(0,100)
imgplot = plt.imshow(X_test[k,:,:,:].reshape(64,256))
arr = []
for i in range(8):
   arr.append(dic[pred[k,i]])
print(arr)
```

```
['5', 'B', '2', '9', '6', '7', '0', ' ']
```

In [42]:

```
k = random.randint(0,100)
imgplot = plt.imshow(X_test[k,:,:,:].reshape(64,256))
arr = []
for i in range(8):
   arr.append(dic[pred[k,i]])
print(arr)
```

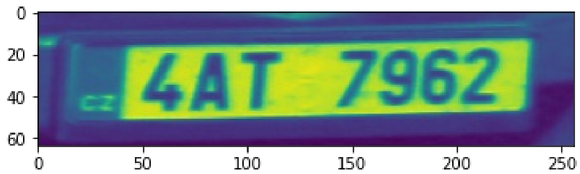
```
['7', 'B', '1', '4', '1', '5', '6', ' ']
```



In [41]:

```
k = random.randint(0,100)
imgplot = plt.imshow(X_test[k,:,:,:].reshape(64,256))
arr = []
for i in range(8):
    arr.append(dic[pred[k,i]])
print(arr)
```

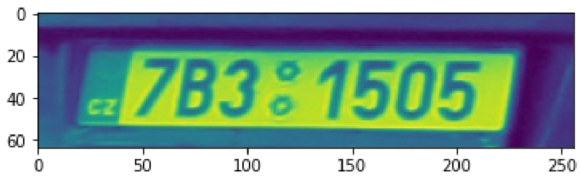
```
['4', 'A', 'X', '7', '9', '6', '2', ' ']
```



In [44]:

```
k = random.randint(0,100)
imgplot = plt.imshow(X_test[k,:,:,:].reshape(64,256))
arr = []
for i in range(8):
    arr.append(dic[pred[k,i]])
print(arr)
```

```
['7', 'B', '3', '1', '5', '0', '5', ' ']
```



In [51]:

```
k = random.randint(0,100)
imgplot = plt.imshow(X_test[k,:,:,:].reshape(64,256))
arr = []
for i in range(8):
    arr.append(dic[pred[k,i]])
print(arr)
```

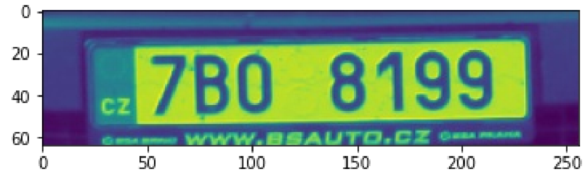
```
['9', 'B', '4', '6', '9', '8', '8', ' ']
```

In [53]:

```
k = random.randint(0,100)
```

```python
imgplot = plt.imshow(X_test[k,:,:,:].reshape(64,256))
arr = []
for i in range(8):
   arr.append(dic[pred[k,i]])
print(arr)
```

```
['7', 'B', '0', '8', '1', '9', '9', ' ']
```



Conclusion :

- Test Accuracy - 90%
- Even this first cut solution gave tremendous results.
- The dataset was very small so, it had very less training data .
- Need to modify CNN architecture and need more data for better accuracy.
- Modifications : YOLO V3 for text detection would have given better results as per different research papers.

In [0]: