In [1]:

```
import keras
import tensorflow as tf
print('TensorFlow version:', tf.__version__)
print('Keras version:', keras.__version__)
```

```
Using TensorFlow backend.


TensorFlow version: 1.11.0
Keras version: 2.2.4
```

In [2]:

```
import os
from os.path import join
import json
import random
import itertools
import re
import datetime
import cairocffi as cairo
import editdistance
import numpy as np
from scipy import ndimage
import pylab
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from keras import backend as K
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.layers import Input, Dense, Activation
from keras.layers import Reshape, Lambda
from keras.layers.merge import add, concatenate
from keras.models import Model, load_model
from keras.layers.recurrent import GRU
from keras.optimizers import SGD
from keras.utils.data_utils import get_file
from keras.preprocessing import image
import keras.callbacks
import cv2
```

In [3]:

```
sess = tf.Session()
K.set_session(sess)
```

Get alphabet

In [4]:

```python
from collections import Counter
def get_counter(dirpath, tag):
    dirname = os.path.basename(dirpath)
    ann_dirpath = join(dirpath, 'ann')
    letters = ''
    lens = []
    for filename in os.listdir(ann_dirpath):
        json_filepath = join(ann_dirpath, filename)
        ann = json.load(open(json_filepath, 'r'))
        tags = ann['tags']
        if tag in tags:
            description = ann['description']
            lens.append(len(description))
            letters += description
    print('Max plate length in "%s":' % dirname, max(Counter(lens).keys()))
    return Counter(letters)
c_val = get_counter('/data/anpr_ocr__train', 'val')
c_train = get_counter('/data/anpr_ocr__train', 'train')
letters_train = set(c_train.keys())
letters_val = set(c_val.keys())
if letters_train == letters_val:
    print('Letters in train and val do match')
else:
    raise Exception()
# print(len(letters_train), len(letters_val), len(letters_val | letters_train))
letters = sorted(list(letters_train))
print('Letters:', ' '.join(letters))
```

```
Max plate length in "anpr_ocr__train": 8
Max plate length in "anpr_ocr__train": 8
Letters in train and val do match
Letters: 0 1 2 3 4 5 6 7 8 9 A B C E H K M O P T X Y
```

Input data generator

In [5]:

```
def labels_to_text(labels):
    return ''.join(list(map(lambda x: letters[int(x)], labels)))

def text_to_labels(text):
    return list(map(lambda x: letters.index(x), text))

def is_valid_str(s):
    for ch in s:
        if not ch in letters:
            return False
    return True

class TextImageGenerator:

    def __init__(self,
                 dirpath,
                 tag,
                 img_w, img_h,
                 batch_size,
                 downsample_factor,
                 max_text_len=8):

        self.img_h = img_h
        self.img_w = img_w
        self.batch_size = batch_size
        self.max_text_len = max_text_len
        self.downsample_factor = downsample_factor

        img_dirpath = join(dirpath, 'img')
        ann_dirpath = join(dirpath, 'ann')
        self.samples = []
        for filename in os.listdir(img_dirpath):
            name, ext = os.path.splitext(filename)
            if ext in ['.png', '.jpg']:
                img_filepath = join(img_dirpath, filename)
                json_filepath = join(ann_dirpath, name + '.json')
                ann = json.load(open(json_filepath, 'r'))
                description = ann['description']
                tags = ann['tags']
                if tag not in tags:
                    continue
                if is_valid_str(description):
                    self.samples.append([img_filepath, description])

        self.n = len(self.samples)
        self.indexes = list(range(self.n))
        self.cur_index = 0

    def build_data(self):
        self.imgs = np.zeros((self.n, self.img_h, self.img_w))
        self.texts = []
        for i, (img_filepath, text) in enumerate(self.samples):
            img = cv2.imread(img_filepath)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            img = cv2.resize(img, (self.img_w, self.img_h))
            img = img.astype(np.float32)
            img /= 255
            # width and height are backwards from typical Keras convention
            # because width is the time dimension when it gets fed into the RNN
            self.imgs[i, :, :] = img
            self.texts.append(text)

    def get_output_size(self):
        return len(letters) + 1

    def next_sample(self):
        self.cur_index += 1
        if self.cur_index >= self.n:
            self.cur_index = 0
            random.shuffle(self.indexes)
        return self.imgs[self.indexes[self.cur_index]], self.texts[self.indexes[sel

    def next_batch(self):
        while True:
            # width and height are backwards from typical Keras convention
            # because width is the time dimension when it gets fed into the RNN
            if K.image_data_format() == 'channels_first':
                X_data = np.ones([self.batch_size, 1, self.img_w, self.img_h])
            else:
                X_data = np.ones([self.batch_size, self.img_w, self.img_h, 1])
            Y_data = np.ones([self.batch_size, self.max_text_len])
            input_length = np.ones((self.batch_size, 1)) * (self.img_w // self.down
            label_length = np.zeros((self.batch_size, 1))
            source_str = []

            for i in range(self.batch_size):
                img, text = self.next_sample()
                img = img.T
                if K.image_data_format() == 'channels_first':
                    img = np.expand_dims(img, 0)
                else:
                    img = np.expand_dims(img, -1)
                X_data[i] = img
                Y_data[i] = text_to_labels(text)
                source_str.append(text)
                label_length[i] = len(text)

            inputs = {
                'the_input': X_data,
                'the_labels': Y_data,
                'input_length': input_length,
                'label_length': label_length,
                #'source_str': source_str
            }
            outputs = {'ctc': np.zeros([self.batch_size])}
            yield (inputs, outputs)
```

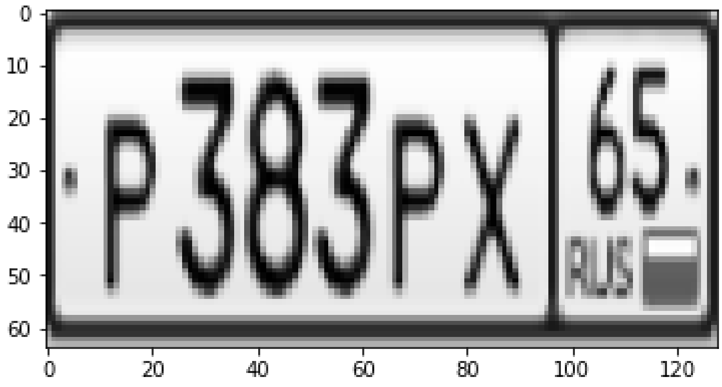
In [6]:

```
tiger = TextImageGenerator('/data/anpr_ocr__train', 'val', 128, 64, 8, 4)
tiger.build_data()
```

In [7]:

```
for inp, out in tiger.next_batch():
    print('Text generator output (data which will be fed into the neutral network):
    print('1) the_input (image)')
    if K.image_data_format() == 'channels_first':
        img = inp['the_input'][0, 0, :, :]
    else:
        img = inp['the_input'][0, :, :, 0]

    plt.imshow(img.T, cmap='gray')
    plt.show()
    print('2) the_labels (plate number): %s is encoded as %s' %
          (labels_to_text(inp['the_labels'][0]), list(map(int, inp['the_labels'][0]
    print('3) input_length (width of image that is fed to the loss function): %d ==
          (inp['input_length'][0], tiger.img_w))
    print('4) label_length (length of plate number): %d' % inp['label_length'][0])
    break
```

```
Text generator output (data which will be fed into the neutral networ
k):
1) the_input (image)
```



```
2) the_labels (plate number): P383PX65 is encoded as [18, 3, 8, 3, 18,
20, 6, 5]
3) input_length (width of image that is fed to the loss function): 30
== 128 / 4 - 2
4) label_length (length of plate number): 8
```

Loss and train functions, network architecture

In [8]:

```
def ctc_lambda_func(args):
    y_pred, labels, input_length, label_length = args
    # the 2 is critical here since the first couple outputs of the RNN
    # tend to be garbage:
    y_pred = y_pred[:, 2:, :]
    return K.ctc_batch_cost(labels, y_pred, input_length, label_length)


def train(img_w, load=False):
    # Input Parameters
    img_h = 64

    # Network parameters
    conv_filters = 16
    kernel_size = (3, 3)
    pool_size = 2
    time_dense_size = 32
    rnn_size = 512

    if K.image_data_format() == 'channels_first':
        input_shape = (1, img_w, img_h)
    else:
        input_shape = (img_w, img_h, 1)

    batch_size = 32
    downsample_factor = pool_size ** 2
    tiger_train = TextImageGenerator('/data/anpr_ocr__train', 'train', img_w, img_h
    tiger_train.build_data()
    tiger_val = TextImageGenerator('/data/anpr_ocr__train', 'val', img_w, img_h, ba
    tiger_val.build_data()

    act = 'relu'
    input_data = Input(name='the_input', shape=input_shape, dtype='float32')
    inner = Conv2D(conv_filters, kernel_size, padding='same',
                   activation=act, kernel_initializer='he_normal',
                   name='conv1')(input_data)
    inner = MaxPooling2D(pool_size=(pool_size, pool_size), name='max1')(inner)
    inner = Conv2D(conv_filters, kernel_size, padding='same',
                   activation=act, kernel_initializer='he_normal',
                   name='conv2')(inner)
    inner = MaxPooling2D(pool_size=(pool_size, pool_size), name='max2')(inner)

    conv_to_rnn_dims = (img_w // (pool_size ** 2), (img_h // (pool_size ** 2)) * co
    inner = Reshape(target_shape=conv_to_rnn_dims, name='reshape')(inner)

    # cuts down input size going into RNN:
    inner = Dense(time_dense_size, activation=act, name='dense1')(inner)

    # Two layers of bidirecitonal GRUs
    # GRU seems to work as well, if not better than LSTM:
    gru_1 = GRU(rnn_size, return_sequences=True, kernel_initializer='he_normal', na
    gru_1b = GRU(rnn_size, return_sequences=True, go_backwards=True, kernel_initial
    gru1_merged = add([gru_1, gru_1b])
    gru_2 = GRU(rnn_size, return_sequences=True, kernel_initializer='he_normal', na
    gru_2b = GRU(rnn_size, return_sequences=True, go_backwards=True, kernel_initial

    # transforms RNN output to character activations:
```

```
    inner = Dense(tiger_train.get_output_size(), kernel_initializer='he_normal',
                  name='dense2')(concatenate([gru_2, gru_2b]))
    y_pred = Activation('softmax', name='softmax')(inner)
    Model(inputs=input_data, outputs=y_pred).summary()

    labels = Input(name='the_labels', shape=[tiger_train.max_text_len], dtype='floa
    input_length = Input(name='input_length', shape=[1], dtype='int64')
    label_length = Input(name='label_length', shape=[1], dtype='int64')
    # Keras doesn't currently support loss funcs with extra parameters
    # so CTC loss is implemented in a lambda layer
    loss_out = Lambda(ctc_lambda_func, output_shape=(1,), name='ctc')([y_pred, labe

    # clipnorm seems to speeds up convergence
    sgd = SGD(lr=0.02, decay=1e-6, momentum=0.9, nesterov=True, clipnorm=5)

    if load:
        model = load_model('/data/mplate2_model.h5', compile=False)
    else:
        model = Model(inputs=[input_data, labels, input_length, label_length], outp

    # the loss calc occurs elsewhere, so use a dummy lambda func for the loss
    model.compile(loss={'ctc': lambda y_true, y_pred: y_pred}, optimizer=sgd)

    if not load:
        # captures output of softmax so we can decode the output during visualizati
        test_func = K.function([input_data], [y_pred])

        model.fit_generator(generator=tiger_train.next_batch(),
                            steps_per_epoch=tiger_train.n,
                            epochs=1,
                            validation_data=tiger_val.next_batch(),
                            validation_steps=tiger_val.n)

    return model
```

Model description and training

Next block will take about 30 minutes.

In [9]:

```
model = train(128, load=True)
```

```
______________________________________________________________________
____________________________
Layer (type)                    Output Shape         Param #     Conne
cted to
======================================================================
============================
the_input (InputLayer)          (None, 128, 64, 1)   0

______________________________________________________________________
____________________________
conv1 (Conv2D)                  (None, 128, 64, 16)  160         the_i
nput[0][0]
______________________________________________________________________
____________________________
max1 (MaxPooling2D)             (None, 64, 32, 16)   0           conv1
[0][0]
______________________________________________________________________
____________________________
conv2 (Conv2D)                  (None, 64, 32, 16)   2320        max1
[0][0]
______________________________________________________________________
____________________________
max2 (MaxPooling2D)             (None, 32, 16, 16)   0           conv2
[0][0]
______________________________________________________________________
____________________________
reshape (Reshape)               (None, 32, 256)      0           max2
[0][0]
______________________________________________________________________
____________________________
dense1 (Dense)                  (None, 32, 32)       8224        resha
pe[0][0]
______________________________________________________________________
____________________________
gru1 (GRU)                      (None, 32, 512)      837120      dense
1[0][0]
______________________________________________________________________
____________________________
gru1_b (GRU)                    (None, 32, 512)      837120      dense
1[0][0]
______________________________________________________________________
____________________________
add_1 (Add)                     (None, 32, 512)      0           gru1
[0][0]
                                                                 gru1_
b[0][0]
______________________________________________________________________
____________________________
gru2 (GRU)                      (None, 32, 512)      1574400     add_1
[0][0]
______________________________________________________________________
____________________________
gru2_b (GRU)                    (None, 32, 512)      1574400     add_1
[0][0]
```

```
_____________________________________________________________________
____________________________
concatenate_1 (Concatenate)     (None, 32, 1024)    0           gru2
[0][0]
                                                                gru2_
b[0][0]
_____________________________________________________________________
____________________________
dense2 (Dense)                  (None, 32, 23)      23575       conca
tenate_1[0][0]
_____________________________________________________________________
____________________________
softmax (Activation)            (None, 32, 23)      0           dense
2[0][0]
=====================================================================
============================
Total params: 4,857,319
Trainable params: 4,857,319
Non-trainable params: 0
_____________________________________________________________________
____________________________
```

Function to decode neural network output

In [10]:

```
# For a real OCR application, this should be beam search with a dictionary
# and language model.  For this example, best path is sufficient.

def decode_batch(out):
    ret = []
    for j in range(out.shape[0]):
        out_best = list(np.argmax(out[j, 2:], 1))
        out_best = [k for k, g in itertools.groupby(out_best)]
        outstr = ''
        for c in out_best:
            if c < len(letters):
                outstr += letters[c]
        ret.append(outstr)
    return ret
```

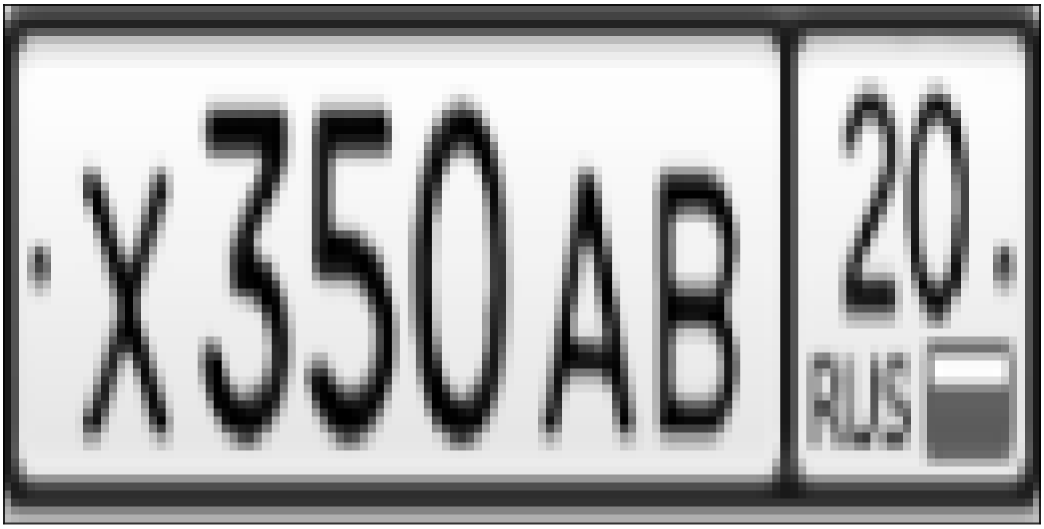
Test on validation images

In [11]:

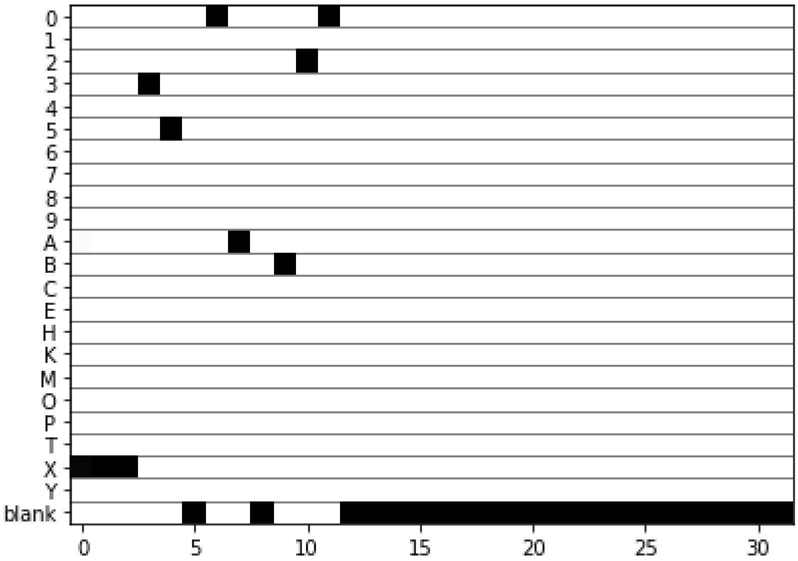
```
tiger_test = TextImageGenerator('/data/anpr_ocr__test', 'test', 128, 64, 8, 4)
tiger_test.build_data()

net_inp = model.get_layer(name='the_input').input
net_out = model.get_layer(name='softmax').output

for inp_value, _ in tiger_test.next_batch():
    bs = inp_value['the_input'].shape[0]
    X_data = inp_value['the_input']
    print ({net_inp:X_data})
    net_out_value = sess.run(net_out, feed_dict={net_inp:X_data})
    pred_texts = decode_batch(net_out_value)
    labels = inp_value['the_labels']
    texts = []
    for label in labels:
        text = ''.join(list(map(lambda x: letters[int(x)], label)))
        texts.append(text)

    for i in range(bs):
        fig = plt.figure(figsize=(10, 10))
        outer = gridspec.GridSpec(2, 1, wspace=10, hspace=0.1)
        ax1 = plt.Subplot(fig, outer[0])
        fig.add_subplot(ax1)
        ax2 = plt.Subplot(fig, outer[1])
        fig.add_subplot(ax2)
        print('Predicted: %s\nTrue: %s' % (pred_texts[i], texts[i]))
        img = X_data[i][:, :, 0].T
        ax1.set_title('Input img')
        ax1.imshow(img, cmap='gray')
        ax1.set_xticks([])
        ax1.set_yticks([])
        ax2.set_title('Activations')
        ax2.imshow(net_out_value[i].T, cmap='binary', interpolation='nearest')
        ax2.set_yticks(list(range(len(letters) + 1)))
        ax2.set_yticklabels(letters + ['blank'])
        ax2.grid(False)
        for h in np.arange(-0.5, len(letters) + 1 + 0.5, 1):
            ax2.axhline(h, linestyle='-', color='k', alpha=0.5, linewidth=1)

        #ax.axvline(x, linestyle='--', color='k')
        plt.show()
    break
```

```
{<tf.Tensor 'the_input_1:0' shape=(?, 128, 64, 1) dtype=float32>: arra
y([[[[0.79215688],
         [0.63921571],
         [0.36470589],
         ...,
         [0.57647061],
         [0.67843139],
         [0.73725492]],

        [[0.47058824],
         [0.38039216],
         [0.22352941],
         ...,
         [0.23529412],
         [0.53725493],
         [0.70588237]],

        [[0.37254903],
         [0.29019609],
         [0.14509805],
         ...,
         [0.19607843],
         [0.50980395],
         [0.68235296]],

        ...,

        [[0.37254903],
         [0.29019609],
         [0.14509805],
         ...,
         [0.19607843],
         [0.50980395],
         [0.68235296]],

        [[0.47058824],
         [0.38039216],
         [0.22352941],
         ...,
         [0.23529412],
         [0.53725493],
         [0.70588237]],

        [[0.79215688],
         [0.63921571],
         [0.36470589],
         ...,
         [0.57647061],
         [0.67843139],
         [0.73725492]]],


       [[[0.79215688],
         [0.63921571],
         [0.36470589],
         ...,
         [0.57647061],
         [0.67843139],
         [0.73725492]],

        [[0.47058824],
         [0.38039216],
         [0.22352941],
         ...,
         [0.23529412],
         [0.53725493],
         [0.70588237]],

        [[0.37254903],
         [0.29019609],
         [0.14509805],
         ...,
         [0.19607843],
         [0.50980395],
         [0.68235296]],

        ...,

        [[0.37254903],
         [0.29019609],
         [0.14509805],
         ...,
         [0.19607843],
         [0.50980395],
         [0.68235296]],

        [[0.47058824],
         [0.38039216],
         [0.22352941],
         ...,
         [0.23529412],
         [0.53725493],
         [0.70588237]],

        [[0.79215688],
         [0.63921571],
         [0.36470589],
         ...,
         [0.57647061],
         [0.67843139],
         [0.73725492]]],


       [[[0.79215688],
         [0.63921571],
         [0.36470589],
         ...,
         [0.57647061],
         [0.67843139],
         [0.73725492]],

        [[0.47058824],
         [0.38039216],
         [0.22352941],
         ...,
         [0.23529412],
         [0.53725493],
         [0.70588237]],

        [[0.37254903],
         [0.29019609],
         [0.14509805],
         ...,
         [0.19607843],
         [0.50980395],
         [0.68235296]],

        ...,

        [[0.37254903],
         [0.29019609],
         [0.14509805],
         ...,
         [0.19607843],
         [0.50980395],
         [0.68235296]],

        [[0.47058824],
         [0.38039216],
         [0.22352941],
         ...,
         [0.23529412],
         [0.53725493],
         [0.70588237]],

        [[0.79215688],
         [0.63921571],
         [0.36470589],
         ...,
         [0.57647061],
         [0.67843139],
         [0.73725492]]],


       ...,


       [[[0.79215688],
         [0.63921571],
         [0.36470589],
         ...,
         [0.57647061],
         [0.67843139],
         [0.73725492]],

        [[0.47058824],
         [0.38039216],
         [0.22352941],
         ...,
         [0.23529412],
         [0.53725493],
         [0.70588237]],

        [[0.37254903],
         [0.29019609],
         [0.14509805],
         ...,
         [0.19607843],
         [0.50980395],
         [0.68235296]],

        ...,

        [[0.37254903],
         [0.29019609],
         [0.14509805],
         ...,
         [0.19607843],
         [0.50980395],
         [0.68235296]],

        [[0.47058824],
         [0.38039216],
         [0.22352941],
         ...,
         [0.23529412],
         [0.53725493],
         [0.70588237]],

        [[0.79215688],
         [0.63921571],
         [0.36470589],
         ...,
         [0.57647061],
         [0.67843139],
         [0.73725492]]],


       [[[0.79215688],
         [0.63921571],
         [0.36470589],
         ...,
         [0.57647061],
         [0.67843139],
         [0.73725492]],

        [[0.47058824],
         [0.38039216],
         [0.22352941],
         ...,
         [0.23529412],
         [0.53725493],
         [0.70588237]],

        [[0.37254903],
         [0.29019609],
         [0.14509805],
         ...,
         [0.19607843],
         [0.50980395],
         [0.68235296]],

        ...,

        [[0.37254903],
         [0.29019609],
         [0.14509805],
         ...,
         [0.19607843],
         [0.50980395],
         [0.68235296]],

        [[0.47058824],
         [0.38039216],
         [0.22352941],
         ...,
         [0.23529412],
         [0.53725493],
         [0.70588237]],

        [[0.79215688],
         [0.63921571],
         [0.36470589],
         ...,
         [0.57647061],
         [0.67843139],
         [0.73725492]]],
       [[[0.79215688],
         [0.63921571],
         [0.36470589],
         ...,
         [0.57647061],
         [0.67843139],
         [0.73725492]],

        [[0.47058824],
         [0.38039216],
         [0.22352941],
         ...,
         [0.23529412],
         [0.53725493],
         [0.70588237]],

        [[0.37254903],
         [0.29019609],
         [0.14509805],
         ...,
         [0.19607843],
         [0.50980395],
         [0.68235296]],

        ...,

        [[0.37254903],
         [0.29019609],
         [0.14509805],
         ...,
         [0.19607843],
         [0.50980395],
         [0.68235296]],

        [[0.47058824],
         [0.38039216],
         [0.22352941],
         ...,
         [0.23529412],
         [0.53725493],
         [0.70588237]],

        [[0.79215688],
         [0.63921571],
         [0.36470589],
         ...,
         [0.57647061],
         [0.67843139],
         [0.73725492]]]])}
Predicted: X350AB20
True: X350AB20
```

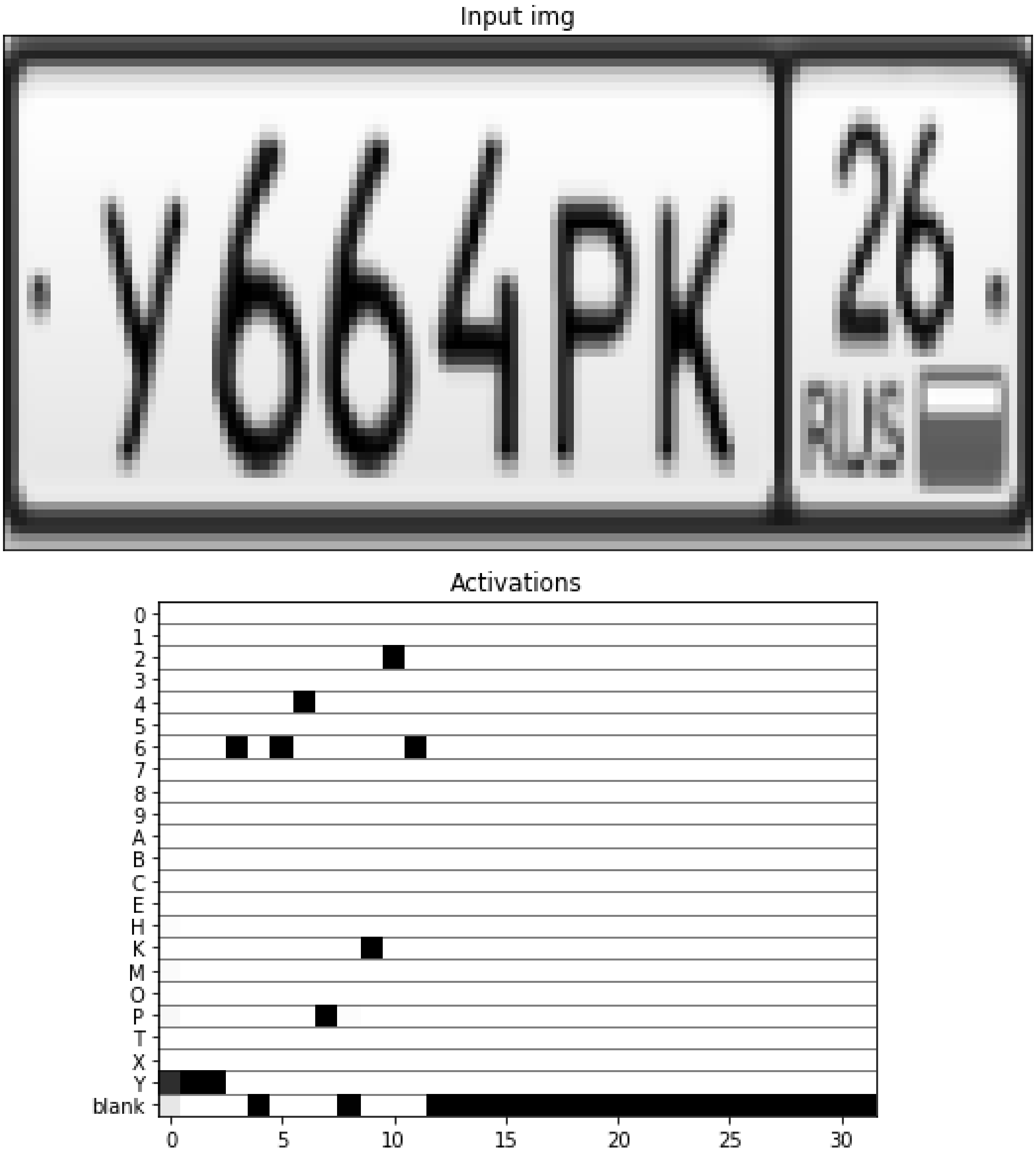
Input img



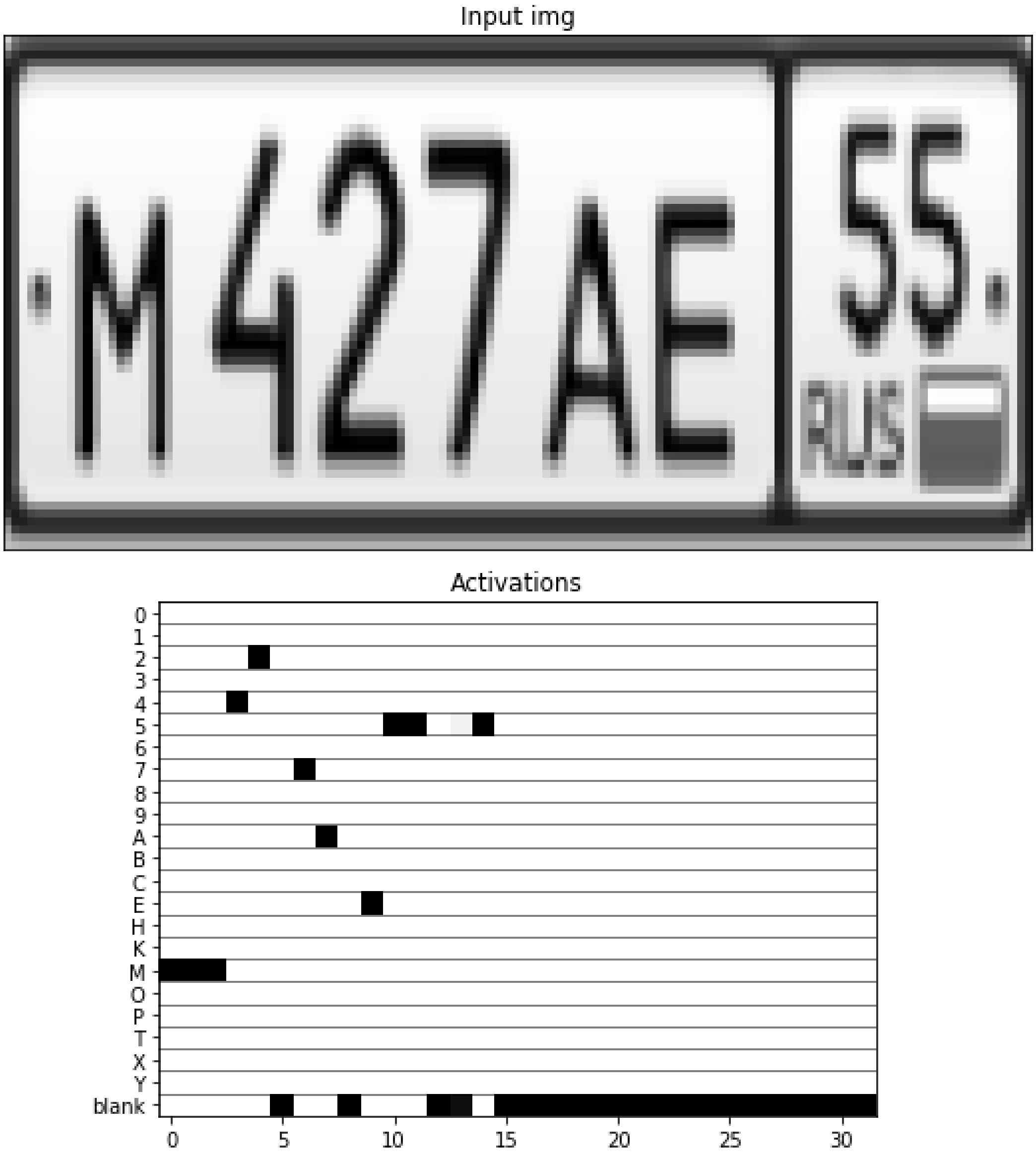
Activations



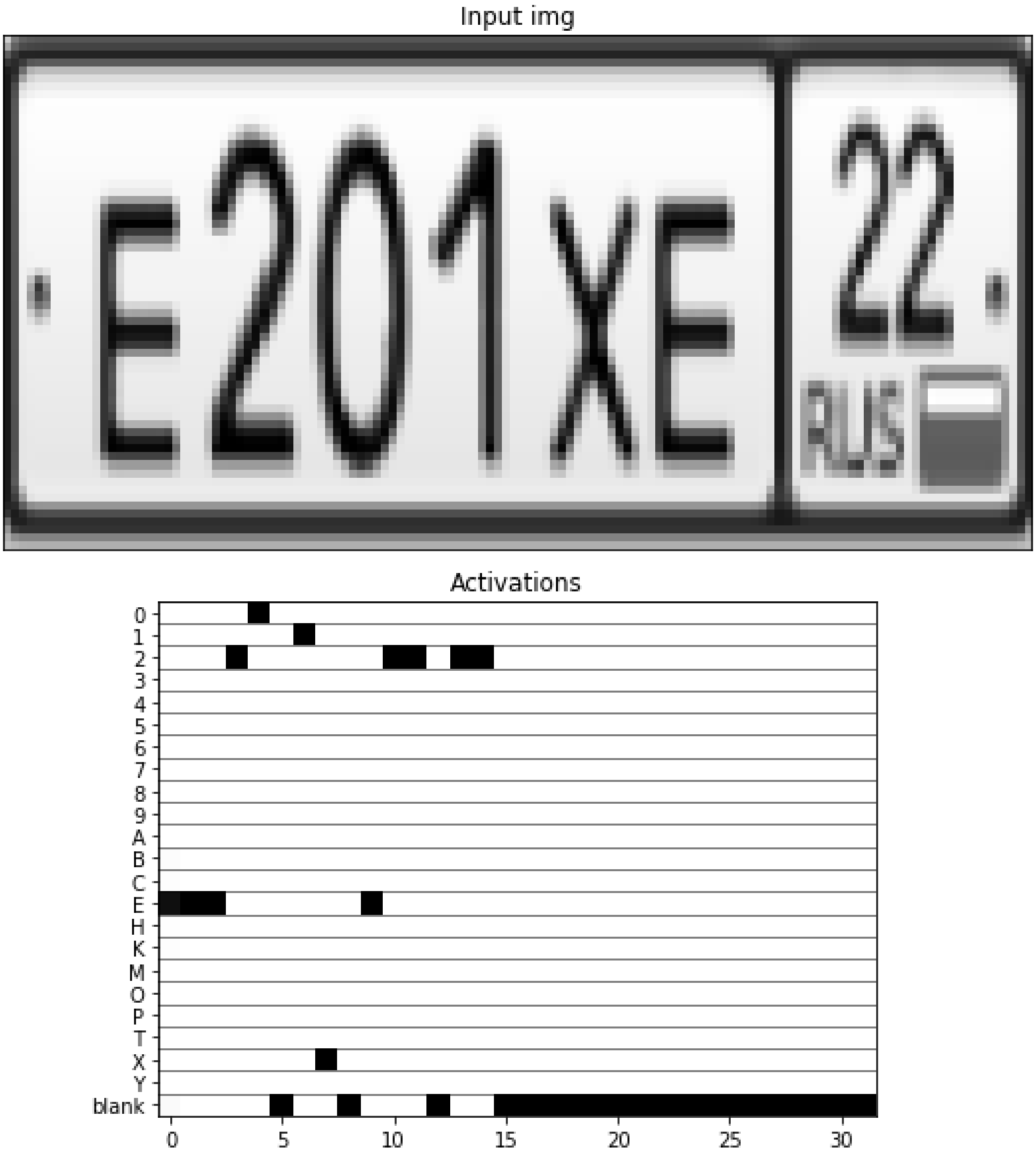
```
Predicted: Y664PK26
True: Y664PK26
```



```
Predicted: M427AE55
True: M427AE55
```

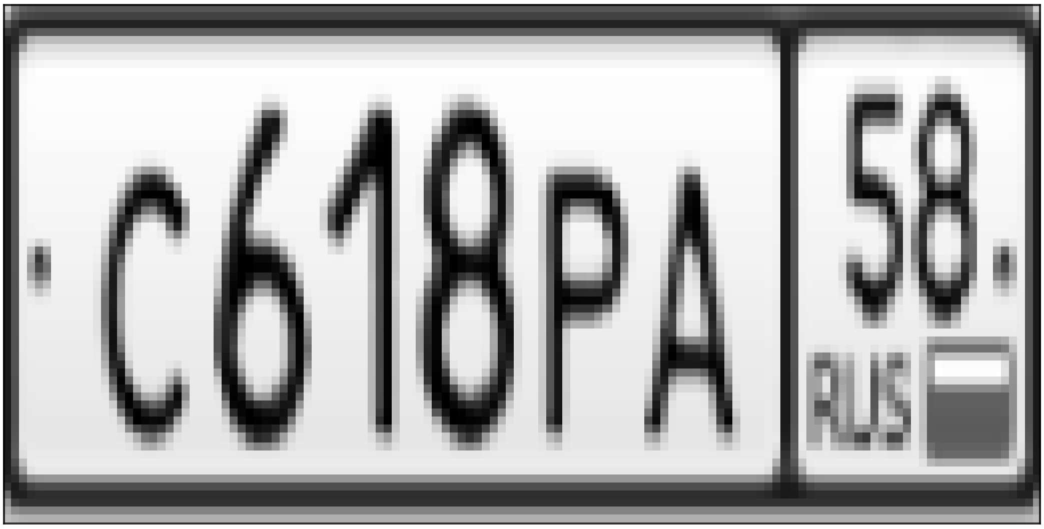


```
Predicted: E201XE22
True: E201XE22
```

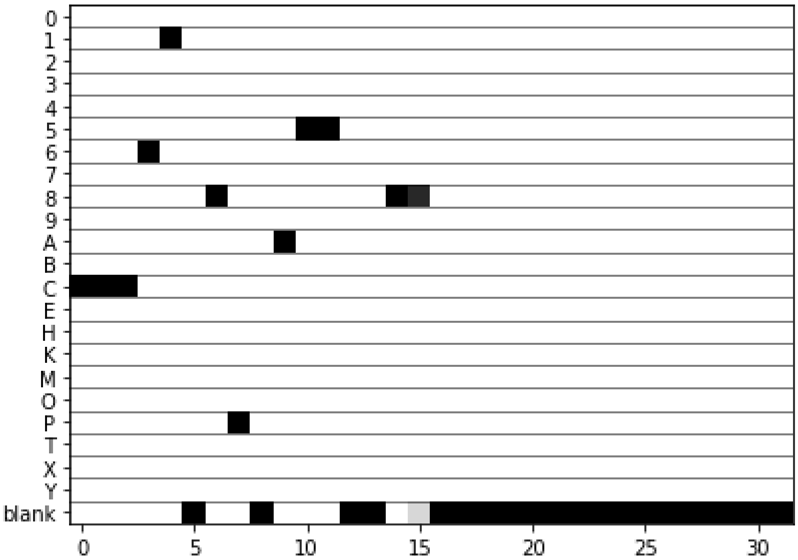


```
Predicted: C618PA58
True: C618PA58
```

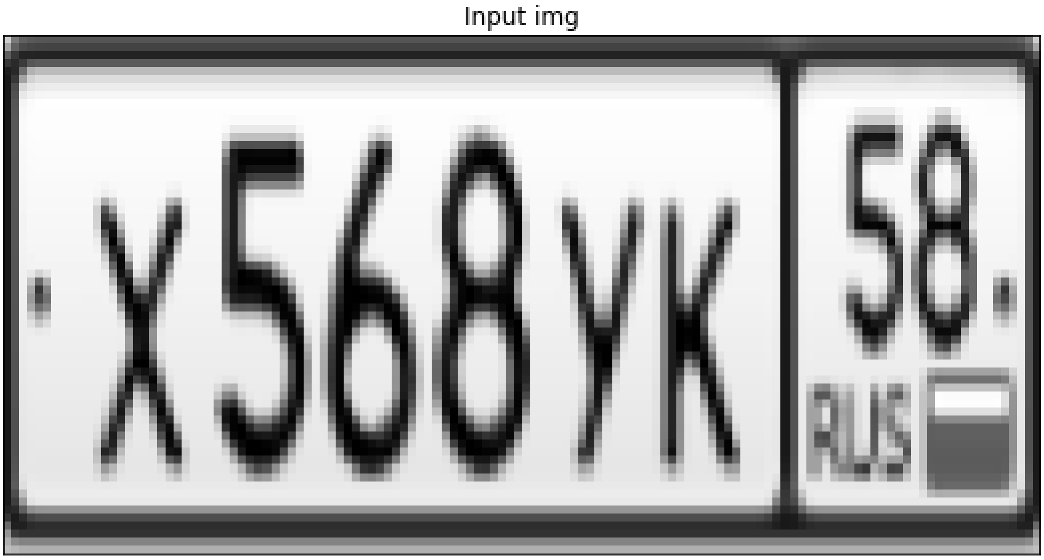
Input img



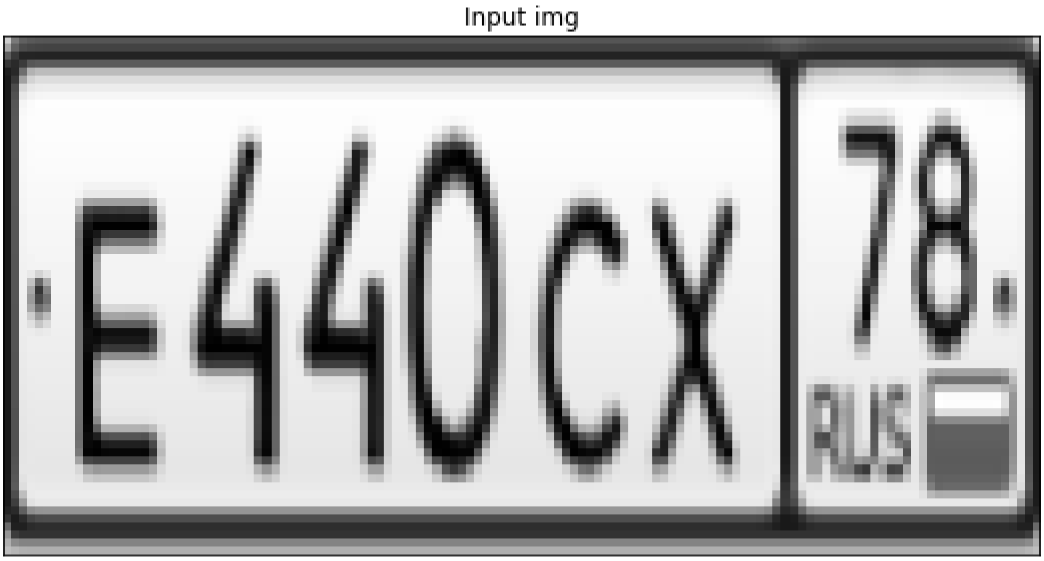
Activations

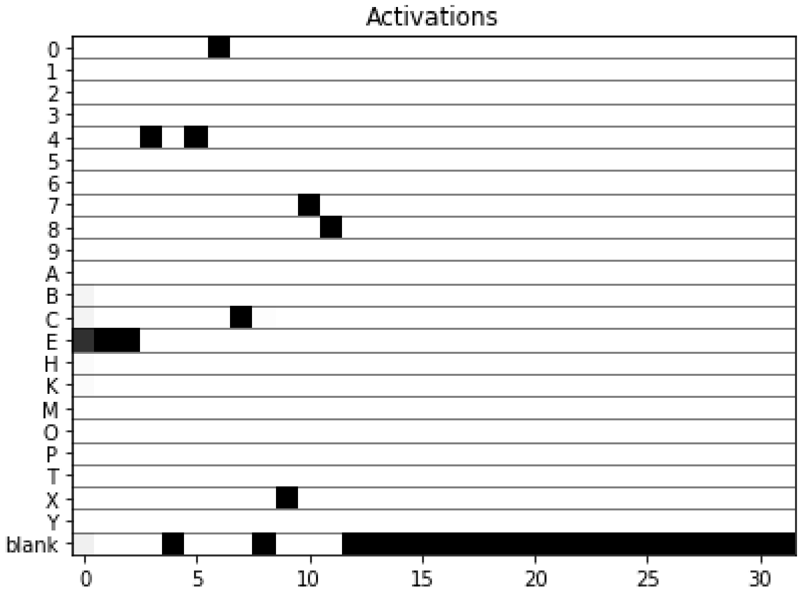


```
Predicted: X568YK58
True: X568YK58
```

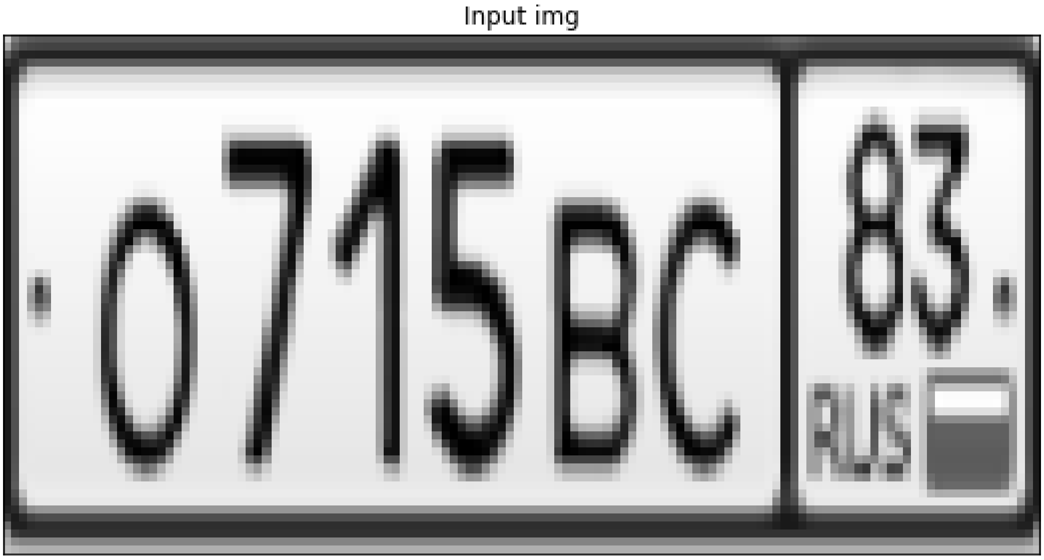


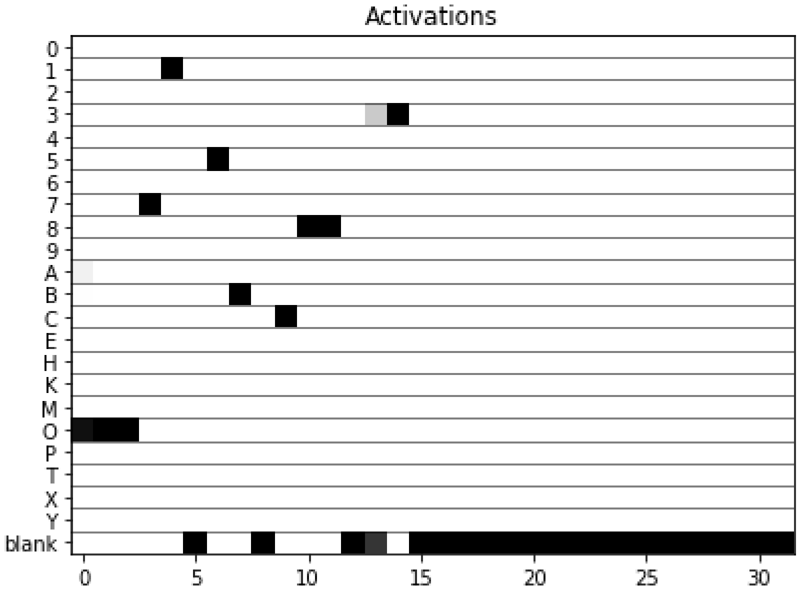
```
Predicted: E440CX78
True: E440CX78
```





```
Predicted: O715BC83
True: O715BC83
```


Input img
O715BC
83
RUS


Activations
0
1
2
3
4
5
6
7
8
9
A
B
C
E
H
K
M
O
P
T
X
Y
blank
0
5
10
15
20
25
30

In [12]:

```
def adjust_gamma(image, gamma=1.0):
    # build a lookup table mapping the pixel values [0, 255] to
    # their adjusted gamma values
    invGamma = 1.0 / gamma
    table = np.array([((i / 255.0) ** invGamma) * 255
        for i in np.arange(0, 256)]).astype("uint8")
    # apply gamma correction using the lookup table
    return cv2.LUT(image, table)
```

In [13]:

```
model.summary()
model.save("/data/mplate2_model.h5")
```

```
______________________________________________________________________
____________________________
Layer (type)                    Output Shape         Param #     Conne
cted to
======================================================================
============================
the_input (InputLayer)          (None, 128, 64, 1)   0

______________________________________________________________________
____________________________
conv1 (Conv2D)                  (None, 128, 64, 16)  160         the_i
nput[0][0]
______________________________________________________________________
____________________________
max1 (MaxPooling2D)             (None, 64, 32, 16)   0           conv1
[0][0]
______________________________________________________________________
____________________________
conv2 (Conv2D)                  (None, 64, 32, 16)   2320        max1
[0][0]
______________________________________________________________________
____________________________
max2 (MaxPooling2D)             (None, 32, 16, 16)   0           conv2
[0][0]
______________________________________________________________________
____________________________
reshape (Reshape)               (None, 32, 256)      0           max2
[0][0]
______________________________________________________________________
____________________________
dense1 (Dense)                  (None, 32, 32)       8224        resha
pe[0][0]
______________________________________________________________________
____________________________
gru1 (GRU)                      (None, 32, 512)      837120      dense
1[0][0]
______________________________________________________________________
____________________________
gru1_b (GRU)                    (None, 32, 512)      837120      dense
1[0][0]
______________________________________________________________________
____________________________
add_1 (Add)                     (None, 32, 512)      0           gru1
[0][0]
                                                                 gru1_
b[0][0]
______________________________________________________________________
____________________________
gru2 (GRU)                      (None, 32, 512)      1574400     add_1
[0][0]
______________________________________________________________________
____________________________
gru2_b (GRU)                    (None, 32, 512)      1574400     add_1
```

```
[0][0]
______________________________________________________________________
____________________________
concatenate_1 (Concatenate)     (None, 32, 1024)     0           gru2
[0][0]
                                                                 gru2_
b[0][0]
______________________________________________________________________
____________________________
dense2 (Dense)                  (None, 32, 23)       23575       conca
tenate_1[0][0]
______________________________________________________________________
____________________________
softmax (Activation)            (None, 32, 23)       0           dense
2[0][0]
______________________________________________________________________
____________________________
the_labels (InputLayer)         (None, 8)            0

______________________________________________________________________
____________________________
input_length (InputLayer)       (None, 1)            0

______________________________________________________________________
____________________________
label_length (InputLayer)       (None, 1)            0

______________________________________________________________________
____________________________
ctc (Lambda)                    (None, 1)            0           softm
ax[0][0]
                                                                 the_l
abels[0][0]
                                                                 input
_length[0][0]
                                                                 label
_length[0][0]
======================================================================
============================
Total params: 4,857,319
Trainable params: 4,857,319
Non-trainable params: 0
______________________________________________________________________
____________________________
```

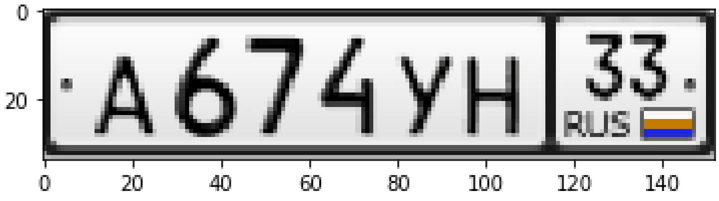
In [14]:

```
def decode_single_image(img, model):
    img_w = 128
    img_h = 64
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = cv2.resize(img, (img_w, img_h))
    #plt.imshow(img)
    img = img.astype(np.float32)
    img /= 255
    net_inp = model.get_layer(name='the_input').input
    net_out = model.get_layer(name='softmax').output
    img = img.T
    img = np.expand_dims(img, -1)
    X_data = np.ones([1, img_w, img_h, 1])
    X_data[0] = img
    net_out_value = sess.run(net_out, feed_dict={net_inp:X_data})
    return decode_batch(net_out_value)
```

In [15]:

```
# decode some png file
img = cv2.imread("/data/real_numbers/fine.png")
plt.imshow(img)
display(decode_single_image(img, model))
```

```
['A674YH33']
```

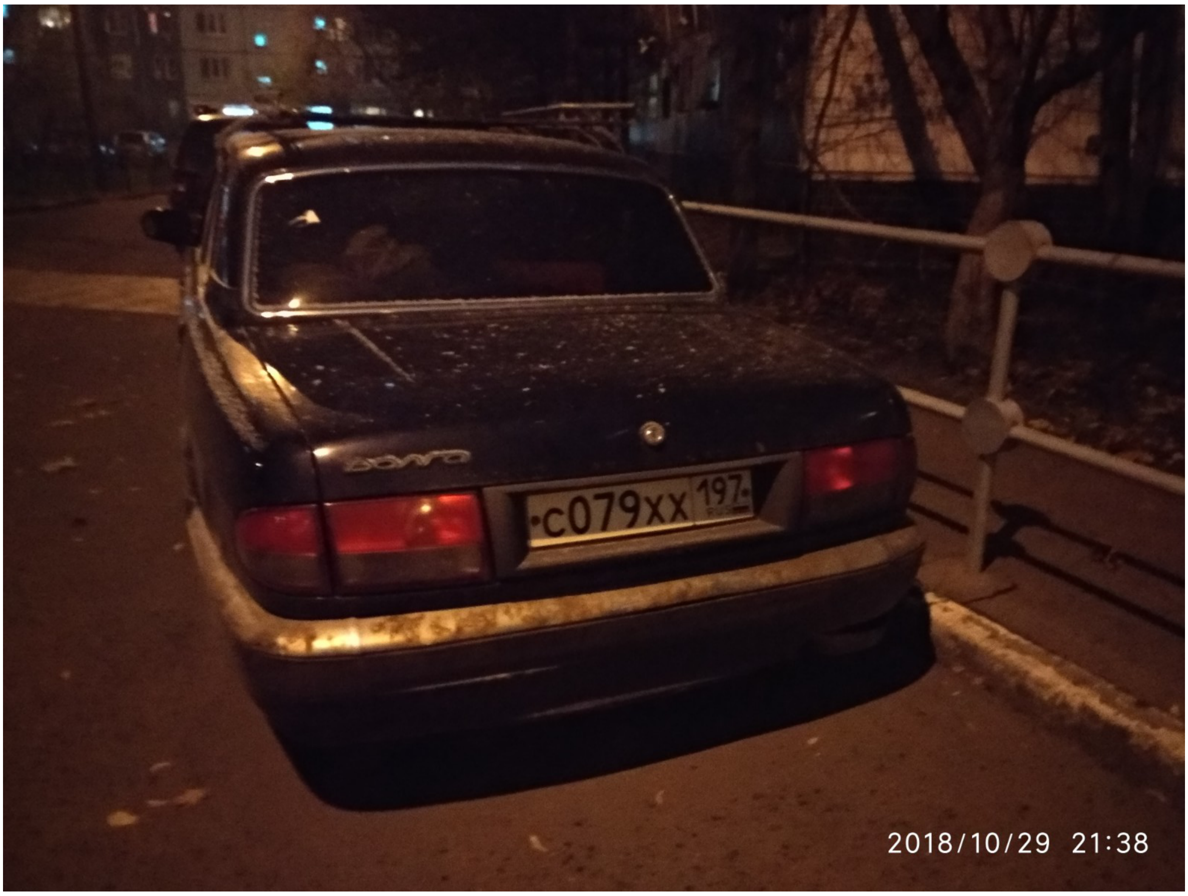


In [16]:

```python
def rotate_bound(image, angle):
    # grab the dimensions of the image and then determine the
    # center
    (h, w) = image.shape[:2]
    (cX, cY) = (w // 2, h // 2)

    # grab the rotation matrix (applying the negative of the
    # angle to rotate clockwise), then grab the sine and cosine
    # (i.e., the rotation components of the matrix)
    M = cv2.getRotationMatrix2D((cX, cY), -angle, 1.0)
    cos = np.abs(M[0, 0])
    sin = np.abs(M[0, 1])

    # compute the new bounding dimensions of the image
    nW = int((h * sin) + (w * cos))
    nH = int((h * cos) + (w * sin))

    # adjust the rotation matrix to take into account translation
    M[0, 2] += (nW / 2) - cX
    M[1, 2] += (nH / 2) - cY

    # perform the actual rotation and return the image
    return cv2.warpAffine(image, M, (nW, nH))
```

In [17]:

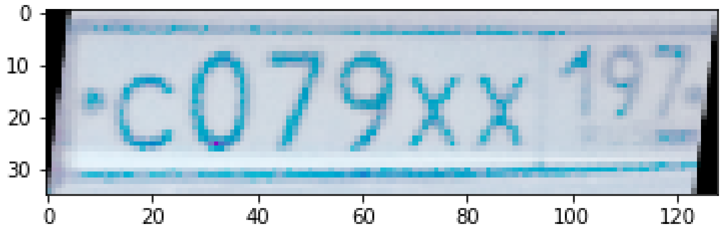
```
import PIL
img = PIL.Image.open("/data/real_numbers/1.jpg")
display(img)
```



In [18]:

```
# work on image above
img = cv2.imread("/data/real_numbers/1n.png")
img = adjust_gamma(img, 8.0)
img = rotate_bound(img, 7)
plt.imshow(img)
img = cv2.resize(img, (128, 64))
img = img[14:49, 0:128]
plt.imshow(img)
display(decode_single_image(img, model))
```

```
['C']
```

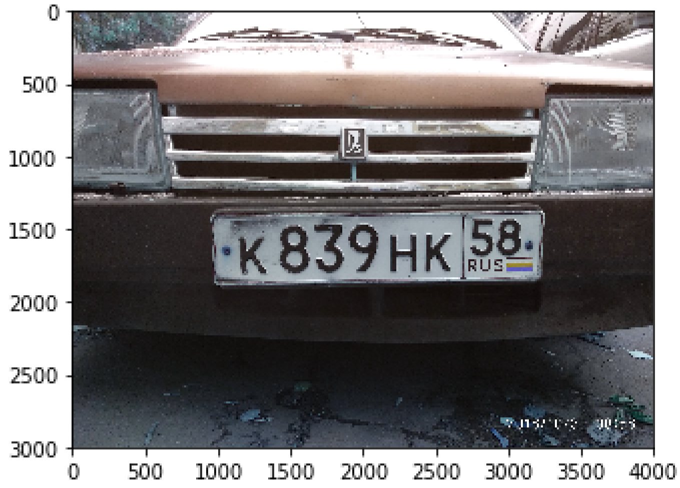


In [23]:

```
img = cv2.imread("/data/real_numbers/2.jpg")
plt.imshow(img)
```

Out[23]:

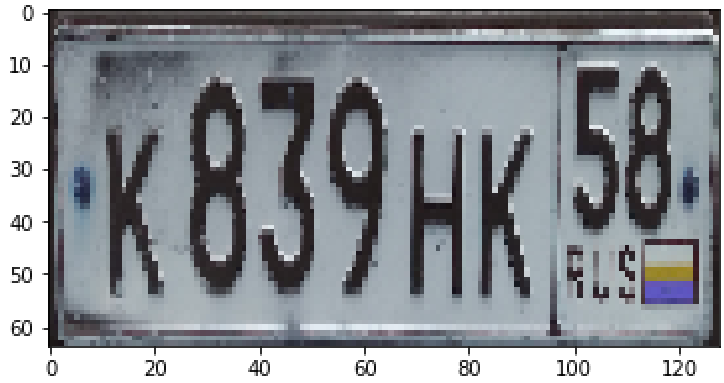
```
<matplotlib.image.AxesImage at 0x7f16907bbba8>
```



In [58]:

```
img = cv2.imread("/data/real_numbers/2.jpg")
#img = adjust_gamma(img, 2.0)
img = rotate_bound(img, 1)
#img = img[1390:1900, 900:3300]
img = img[1395:1395+128*4, 970:970+64*36]

img = cv2.resize(img, (128, 64))
plt.imshow(img)
display(decode_single_image(img, model))
```

```
['K3']
```



In [60]:

```
img = cv2.imread("/data/real_numbers/fine.png")
img = cv2.resize(img, (128, 64))
plt.imshow(img)
display(decode_single_image(img, model))
```

```
['A674YH33']
```