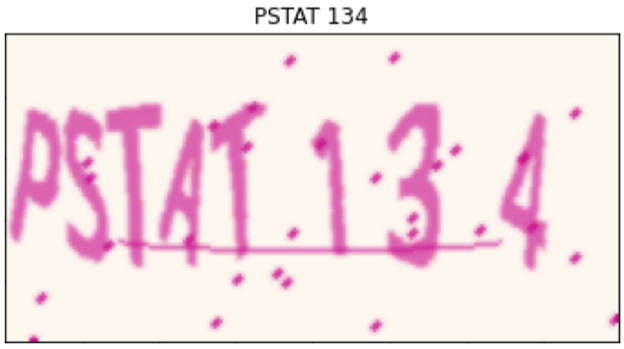
1. CAPTCHA

A CAPTCHA (a contrived acronym for "Completely Automated Public Turing test to tell Computers and Humans Apart") is a type of challenge–response test used in computing to determine whether the user is human.



The term was coined in 2003 by Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. The most common type of CAPTCHA (displayed as Version 1.0) was first invented in 1997 by two groups working in parallel. This form of CAPTCHA requires entering a sequence of letters or numbers in a distorted image. Because the test is administered by a computer, in contrast to the standard Turing test that is administered by a human, a CAPTCHA is sometimes described as a reverse Turing test.This test has received many criticisms, from people with disabilities, but also many websites use it to prevent bot spamming and raiding, and it works effectively, and its usage is widespread. Most websites use hCaptcha or reCAPTCHA. It takes the average person approximately 10 seconds to solve a typical CAPTCHA.

Source: https://en.wikipedia.org/wiki/CAPTCHA

In this notebook, we will train a convolutional neural network to identify images of CAPTCHA letters. After generating, loading, examining, and preprocessing the data, we will train the network and test its performance.

2. Generate the training and testing data

In [ ]:
```
# Import packages and set numpy random seed
import numpy as np
import tensorflow as tf
from captcha.image import ImageCaptcha
import string
import random
import matplotlib.pyplot as plt
import keras
%matplotlib inline

# define a simple captcha generator
def captcha_gen(val=None):
    captcha = ImageCaptcha(width=50, height=75)
    if not val:
        val = ''.join(random.sample((string.ascii_uppercase + string.digits),1))

    return (captcha.generate_image(val), val)
```

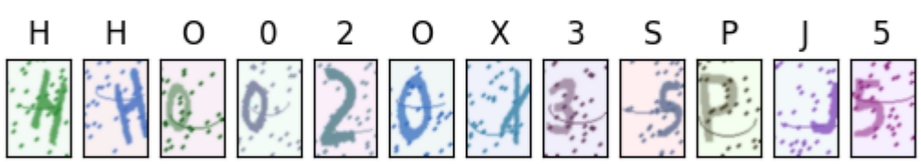
In [ ]:
```
train_size = 1000000

# X_train = np.zeros((train_size,75,50,3), dtype='int')
# y_train = np.zeros((train_size,), dtype='str')

# for i in range(train_size):
#     X_train[i],y_train[i] = captcha_gen()
#     if i%1000 == 0:
#         print(i)

# np.save('data/X_train.npy',X_train)
```

```
# np.save('data/y_train.npy',y_train)

X_train = np.load('data/X_train.npy')
y_train = np.load('data/y_train.npy')
```

In [ ]:
```
test_size = 1000

# X_test = np.zeros((test_size,75,50,3), dtype='int')
# y_test = np.zeros((test_size,), dtype='str')

# for i in range(test_size):
#     X_test[i],y_test[i] = captcha_gen()

X_test = np.load('data/X_test.npy')
y_test = np.load('data/y_test.npy')
```

3. Visualize the training data

In [ ]:
```
# show 36 examples
for i in range(36):
    plt.subplot(3,12,i+1)
    plt.imshow(np.squeeze(X_train[i]))
    plt.title(y_train[i])
    plt.xticks([])
    plt.yticks([])
```



4. Transform y categorical data with one hot encoding

In [ ]:
```
y_train_OH = np.zeros((train_size, 36))
y_test_OH = np.zeros((test_size, 36))

for i in range(train_size):
    y_train_OH[i,(string.ascii_uppercase + string.digits).find(y_train[i])] = 1

for i in range(test_size):
    y_test_OH[i,(string.ascii_uppercase + string.digits).find(y_test[i])] = 1
```

5. Build, compile, and train the Keras Model

In [ ]:
```
"""
    Model training can take huge amount of time
    Uncomment the code below can directly load the trained model
"""

model = keras.models.load_model('data/model.h5')
```

In [ ]:
```
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Flatten, Dense, Dropout
from keras.models import Sequential

model = Sequential()
# First convolutional layer accepts image input
model.add(Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu',
                 input_shape=(75, 50, 3)))
# Add a max pooling layer
model.add(MaxPooling2D(pool_size=(2, 2)))
# Add a convolutional layer
model.add(Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu'))
# Add another max pooling layer
model.add(MaxPooling2D(pool_size=(2, 2)))
# Flatten and feed to output layer
model.add(Flatten())
model.add(Dropout(0.25))
model.add(Dense(36, activation='softmax'))

# Summarize the model
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 75, 50, 32)        896

 max_pooling2d (MaxPooling2D  (None, 37, 25, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 37, 25, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 18, 12, 64)       0
 2D)

 flatten (Flatten)           (None, 13824)             0

 dropout (Dropout)           (None, 13824)             0

 dense (Dense)               (None, 36)                497700

=================================================================
Total params: 517,092
Trainable params: 517,092
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
# Compile the model
model.compile(optimizer='adadelta',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [ ]:
```
# Train the model
hist = model.fit(X_train, y_train_OH, validation_split=0.2, epochs=20, steps_per_epoch=
```

```
Epoch 1/20
5120/5120 [==============================] - 1164s 227ms/step - loss: 38.7613 - accurac
y: 0.0302 - val_loss: 5.4922 - val_accuracy: 0.0402
Epoch 2/20
5120/5120 [==============================] - 1170s 229ms/step - loss: 4.7234 - accurac
y: 0.0326 - val_loss: 3.6133 - val_accuracy: 0.0357
Epoch 3/20
5120/5120 [==============================] - 1183s 231ms/step - loss: 3.5719 - accurac
y: 0.0457 - val_loss: 3.4670 - val_accuracy: 0.0693
Epoch 4/20
5120/5120 [==============================] - 1148s 224ms/step - loss: 3.4058 - accurac
y: 0.0842 - val_loss: 3.1911 - val_accuracy: 0.1359
Epoch 5/20
5120/5120 [==============================] - 1157s 226ms/step - loss: 3.1163 - accurac
y: 0.1495 - val_loss: 2.7922 - val_accuracy: 0.2340
Epoch 6/20
5120/5120 [==============================] - 1205s 235ms/step - loss: 2.7745 - accurac
y: 0.2263 - val_loss: 2.3997 - val_accuracy: 0.3362
Epoch 7/20
5120/5120 [==============================] - 1206s 236ms/step - loss: 2.4357 - accurac
y: 0.3055 - val_loss: 2.0500 - val_accuracy: 0.4288
Epoch 8/20
5120/5120 [==============================] - 1210s 236ms/step - loss: 2.1454 - accurac
y: 0.3761 - val_loss: 1.7674 - val_accuracy: 0.5037
Epoch 9/20
5120/5120 [==============================] - 1197s 234ms/step - loss: 1.9061 - accurac
y: 0.4379 - val_loss: 1.5443 - val_accuracy: 0.5665
Epoch 10/20
5120/5120 [==============================] - 1181s 231ms/step - loss: 1.7099 - accurac
y: 0.4890 - val_loss: 1.3616 - val_accuracy: 0.6178
Epoch 11/20
5120/5120 [==============================] - 1209s 236ms/step - loss: 1.5470 - accurac
y: 0.5332 - val_loss: 1.2114 - val_accuracy: 0.6590
Epoch 12/20
5120/5120 [==============================] - 1204s 235ms/step - loss: 1.4122 - accurac
y: 0.5700 - val_loss: 1.0898 - val_accuracy: 0.6927
Epoch 13/20
5120/5120 [==============================] - 1200s 234ms/step - loss: 1.2973 - accurac
y: 0.6019 - val_loss: 0.9852 - val_accuracy: 0.7212
Epoch 14/20
5120/5120 [==============================] - 1196s 234ms/step - loss: 1.2008 - accurac
y: 0.6291 - val_loss: 0.9012 - val_accuracy: 0.7456
Epoch 15/20
5120/5120 [==============================] - 1155s 226ms/step - loss: 1.1166 - accurac
y: 0.6531 - val_loss: 0.8285 - val_accuracy: 0.7657
Epoch 16/20
5120/5120 [==============================] - 1170s 229ms/step - loss: 1.0450 - accurac
y: 0.6753 - val_loss: 0.7658 - val_accuracy: 0.7829
Epoch 17/20
5120/5120 [==============================] - 1207s 236ms/step - loss: 0.9818 - accurac
y: 0.6933 - val_loss: 0.7140 - val_accuracy: 0.7976
Epoch 18/20
5120/5120 [==============================] - 1175s 230ms/step - loss: 0.9268 - accurac
y: 0.7091 - val_loss: 0.6659 - val_accuracy: 0.8107
Epoch 19/20
5120/5120 [==============================] - 1154s 225ms/step - loss: 0.8773 - accurac
y: 0.7242 - val_loss: 0.6254 - val_accuracy: 0.8216
Epoch 20/20
4640/5120 [==========================>...] - ETA: 1:41 - loss: 0.8365 - accuracy: 0.736
4WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that y
our dataset or generator can generate at least `steps_per_epoch * epochs` batches (in t
his case, 102400 batches). You may need to use the repeat() function when building your
dataset.
5120/5120 [==============================] - 1035s 202ms/step - loss: 0.8365 - accurac
y: 0.7364 - val_loss: 0.5913 - val_accuracy: 0.8304
```

In [ ]:
```
# save the trained model
model.save('data/model.h5')
```

6. Predict and evaluate using model

In [ ]:
```
# predict test set
y_test_pred = model.predict(X_test)
```

```
32/32 [==============================] - 0s 10ms/step
```

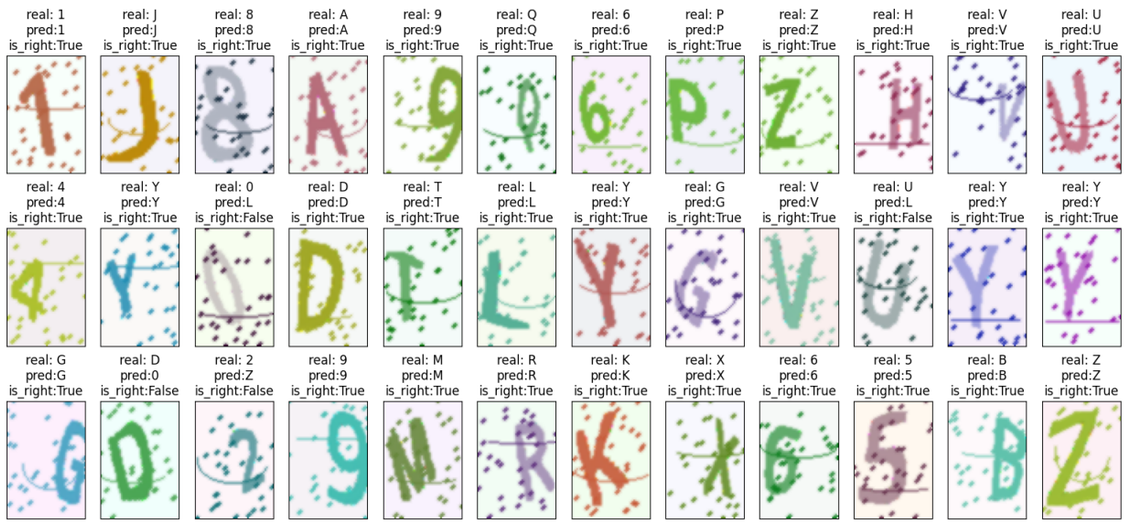
In [ ]:
```
# Obtain accuracy on test set
score = model.evaluate(x=X_test,
                       y=y_test_OH,
                       verbose=0)
print('Test accuracy:', score[1])
```

```
Test accuracy: 0.8149999976158142
```

7. Visualize the predicted result

In [ ]:
```
def one_hot_decoder(val):
    val = np.argmax(val)
    return (string.ascii_uppercase + string.digits)[val]
```

In [ ]:
```
# show 36 example of predicted plot
for i in range(36):
    plt.subplot(3,12,i+1)
    plt.imshow(np.squeeze(X_test[i]))
    plt.title('real: %s\npred:%s\nis_right:%s'%(y_test[i], one_hot_decoder(y_test_pred[
    plt.xticks([])
    plt.yticks([])
plt.subplots_adjust(right=2.5,top=1.5)
```

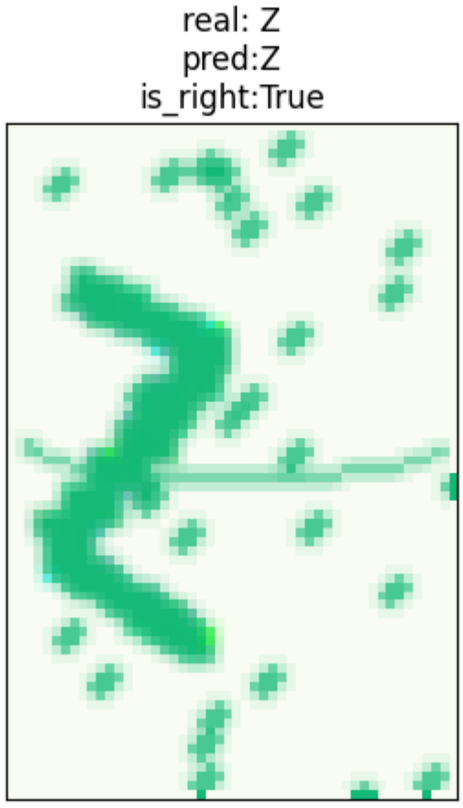


8. Playgound

In [ ]:
```
# feel free to generate another captcha and let the model to guess
random_captcha = captcha_gen()
X_random = np.zeros((1,75,50,3), dtype='int')
X_random[0] = random_captcha[0]

random_predict = model.predict(X_random)

plt.title('real: %s\npred:%s\nis_right:%s'%(random_captcha[1], one_hot_decoder(random_p
plt.imshow(np.squeeze(X_random[0]))
plt.xticks([])
plt.yticks([])
```

```
1/1 [==============================] - 0s 17ms/step
```

Out[ ]:
```
([], [])
```



In [ ]: