Preparation

Install Requirement

In [1]:

```
!pip install -q tensorflow-gpu==2.0.0-beta1
!pip install -qU watermark
```

```
     |████████████████████████████████| 348.9MB 61kB/s
     |████████████████████████████████| 501kB 36.5MB/s
     |████████████████████████████████| 3.1MB 32.4MB/s
```

Custom Matplotlib Style

In [0]:

```
mpl_style = "https://gist.githubusercontent.com/m3hrdadfi/af8aca01094afb7d3e5b46de9ad8d509/raw/871ec5d721a3b438c3c896718ea4aafc91ea9744/gadfly.mplstyle"
!wget -q $mpl_style -O /root/.config/matplotlib/matplotlibrc
```

General Paramas

A random seed is a number used to initialize a pseudorandom number generator. For a seed to be used in a pseudorandom number generator, it does not need to be random

In [0]:

```
RANDOM_SEED = 141
```

Import requried packages

In [0]:

```
import tensorflow as tf

import requests
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix

import matplotlib as mpl
import matplotlib.pyplot as plt

import cv2

from sklearn.utils import shuffle

import random
import sys
import io
import re
import time
from datetime import datetime
import os
import struct
from tqdm import tqdm

from pprint import pprint


%matplotlib inline
mpl.rc_file(mpl.matplotlib_fname())
```

Persian MNIST

Data Set Information:

Attribute Information:

1. pixels
2. class:
    - 0
    - 1
    - 2
    - 3
    - 4
    - 5
    - 6
    - 7
    - 8
    - 9

In [5]:

```
!wget https://www.dropbox.com/s/op3ht07lfou9lbz/DigitDB.zip
!unzip DigitDB.zip
!ls
```

```
--2019-07-23 20:04:07--  https://www.dropbox.com/s/op3ht07lfou9lbz/DigitDB.zip
Resolving www.dropbox.com (www.dropbox.com)... 162.125.65.1, 2620:100:6021:1::a27d:4101
Connecting to www.dropbox.com (www.dropbox.com)|162.125.65.1|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/op3ht07lfou9lbz/DigitDB.zip [following]
--2019-07-23 20:04:07--  https://www.dropbox.com/s/raw/op3ht07lfou9lbz/DigitDB.zip
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc283587cd71121ccbf5863e1e0c.dl.dropboxusercontent.com/cd/0/inline/AlQTEO1pTDeyMMB
bsU5u3JV3YgWRLb-V34f9BAU6coWxg71V53agcHfmfQNMloeVp74zZirV4JolDGwbU1a3NBHosxjxmN-KbUJA0oHRq-jf8w/file
# [following]
--2019-07-23 20:04:07--  https://uc283587cd71121ccbf5863e1e0c.dl.dropboxusercontent.com/cd/0/inline/
AlQTEO1pTDeyMMBbsU5u3JV3YgWRLb-V34f9BAU6coWxg71V53agcHfmfQNMloeVp74zZirV4JolDGwbU1a3NBHosxjxmN-KbUJA
0oHRq-jf8w/file
Resolving uc283587cd71121ccbf5863e1e0c.dl.dropboxusercontent.com (uc283587cd71121ccbf5863e1e0c.dl.dr
opboxusercontent.com)... 162.125.65.6, 2620:100:6021:6::a27d:4106
Connecting to uc283587cd71121ccbf5863e1e0c.dl.dropboxusercontent.com (uc283587cd71121ccbf5863e1e0c.d
l.dropboxusercontent.com)|162.125.65.6|:443... connected.
HTTP request sent, awaiting response... 302 FOUND
Location: /cd/0/inline2/AlTNiseZqeAteUcqOiW2McGavoI-Wh1F3GGWQzt9IvLdglM99Bb4a4ID6K5ljPEIjXWEDHb4QkwW
QaImbCvjVSkfqaLeauwzA83y0YZEgKrmZT1e1srtsr3gjMeJOgA2ezrZaWkaDLs-YRpUM-GZBN1LhiX4cDKtJCxxwK2HJQHm5Heh
E46m2s3uJUJ1jqKJvUnNW1QG8hRSs1e7x6eY-Jd1Mu_S_2LK1q24VOjmLjp0rcK0TMR3TXnBDIxvbxc3cUn_tKOiaLahcPX5ASJa
WWdwQvwGGuaeuWNbc_CesPzeai51qwoaP2avrAZf4rFaryLwYQQ7wd5wfmSF5PwwxOe3/file [following]
--2019-07-23 20:04:08--  https://uc283587cd71121ccbf5863e1e0c.dl.dropboxusercontent.com/cd/0/inline
2/AlTNiseZqeAteUcqOiW2McGavoI-Wh1F3GGWQzt9IvLdglM99Bb4a4ID6K5ljPEIjXWEDHb4QkwWQaImbCvjVSkfqaLeauwzA8
3y0YZEgKrmZT1e1srtsr3gjMeJOgA2ezrZaWkaDLs-YRpUM-GZBN1LhiX4cDKtJCxxwK2HJQHm5HehE46m2s3uJUJ1jqKJvUnNW1
QG8hRSs1e7x6eY-Jd1Mu_S_2LK1q24VOjmLjp0rcK0TMR3TXnBDIxvbxc3cUn_tKOiaLahcPX5ASJaWWdwQvwGGuaeuWNbc_CesP
zeai51qwoaP2avrAZf4rFaryLwYQQ7wd5wfmSF5PwwxOe3/file
Reusing existing connection to uc283587cd71121ccbf5863e1e0c.dl.dropboxusercontent.com:443.
HTTP request sent, awaiting response... 200 OK
Length: 5290356 (5.0M) [application/zip]
Saving to: ‘DigitDB.zip’

DigitDB.zip         100%[===================>]   5.04M  --.-KB/s    in 0.1s

2019-07-23 20:04:09 (50.9 MB/s) - ‘DigitDB.zip’ saved [5290356/5290356]

Archive:  DigitDB.zip
  inflating: Train 60000.cdb
  inflating: RemainingSamples.cdb
  inflating: Test 20000.cdb
 DigitDB.zip            sample_data      'Train 60000.cdb'
 RemainingSamples.cdb  'Test 20000.cdb'
```

In [0]:

```
def resize_image(src_image, dst_image_height, dst_image_width):
    src_image_height = src_image.shape[0]
    src_image_width = src_image.shape[1]

    if src_image_height > dst_image_height or src_image_width > dst_image_width:
        height_scale = dst_image_height / src_image_height
        width_scale = dst_image_width / src_image_width
        scale = min(height_scale, width_scale)
        img = cv2.resize(src=src_image, dsize=(0, 0), fx=scale, fy=scale, interpolation=cv2.INTER_CUB
IC)
    else:
        img = src_image

    img_height = img.shape[0]
    img_width = img.shape[1]

    dst_image = np.zeros(shape=[dst_image_height, dst_image_width], dtype=np.uint8)

    y_offset = (dst_image_height - img_height) // 2
    x_offset = (dst_image_width - img_width) // 2

    dst_image[y_offset:y_offset + img_height, x_offset:x_offset + img_width] = img

    return dst_image


def read_cdb(filepath):
    with open(filepath, 'rb') as f:
        data = f.read()
        offset = 0

        # read private header
        yy = struct.unpack_from('H', data, offset)[0]
        offset += 2

        m = struct.unpack_from('B', data, offset)[0]
        offset += 1

        d = struct.unpack_from('B', data, offset)[0]
        offset += 1

        h = struct.unpack_from('B', data, offset)[0]
        offset += 1

        w = struct.unpack_from('B', data, offset)[0]
        offset += 1

        total_rec = struct.unpack_from('I', data, offset)[0]
        offset += 4

        letter_count = struct.unpack_from('128I', data, offset)
        offset += 128 * 4

        img_type = struct.unpack_from('B', data, offset)[0]  # 0: binary, 1: gray
        offset += 1

        comments = struct.unpack_from('256c', data, offset)
        offset += 256 * 1

        reserved = struct.unpack_from('245c', data, offset)
        offset += 245 * 1

        if (w > 0) and (h > 0):
            normal = True
        else:
            normal = False

        images = []
        labels = []

        for i in tqdm(range(total_rec), position=0):

            start_byte = struct.unpack_from('B', data, offset)[0]  # must be 0xff
            offset += 1

            label = struct.unpack_from('B', data, offset)[0]
            offset += 1

            if not normal:
                w = struct.unpack_from('B', data, offset)[0]
                offset += 1

                h = struct.unpack_from('B', data, offset)[0]
                offset += 1
            byte_count = struct.unpack_from('H', data, offset)[0]
            offset += 2

            image = np.zeros(shape=[h, w], dtype=np.uint8)

            if img_type == 0:
                # Binary
                for y in range(h):
                    b_white = True
                    counter = 0
                    while counter < w:
                        wb_count = struct.unpack_from('B', data, offset)[0]
                        offset += 1

                        if b_white:
                            image[y, counter:counter + wb_count] = 0  # Background
                        else:
                            image[y, counter:counter + wb_count] = 255  # ForeGround
                        b_white = not b_white  # black white black white ...
                        counter += wb_count
            else:
                # GrayScale mode
                data = struct.unpack_from('{}B'.format(w * h), data, offset)
                offset += w * h
                image = np.asarray(data, dtype=np.uint8).reshape([w, h]).T

            images.append(image)
            labels.append(label)

        return images, labels


def load_data(datapath, img_height=32, img_width=32):
    images, labels = read_cdb(datapath)
    assert len(images) == len(labels)

    x = np.zeros(shape=[len(images), img_height, img_width], dtype=np.float32)
    y = np.zeros(shape=[len(labels)], dtype=np.int)

    for i in tqdm(range(len(images)), position=0):
        image = images[i]
        image = resize_image(src_image=image, dst_image_height=img_height, dst_image_width=img_width)
        image = image / 255
        image = np.where(image >= 0.5, 1, 0)

        x[i] = image
        y[i] = labels[i]

    x, y = shuffle(x, y, random_state=0)

    return x, y
```

Load the data

In [7]:
```
trainset_dir = 'Train 60000.cdb'
testset_dir = 'Test 20000.cdb'

x_train, y_train = load_data(trainset_dir)
x_test, y_test = load_data(testset_dir)

print('Train: %s, Labels: #%s' %(x_train.shape, len(y_train)))
print('Test: %s, Labels: #%s' %(x_test.shape, len(y_test)))
```

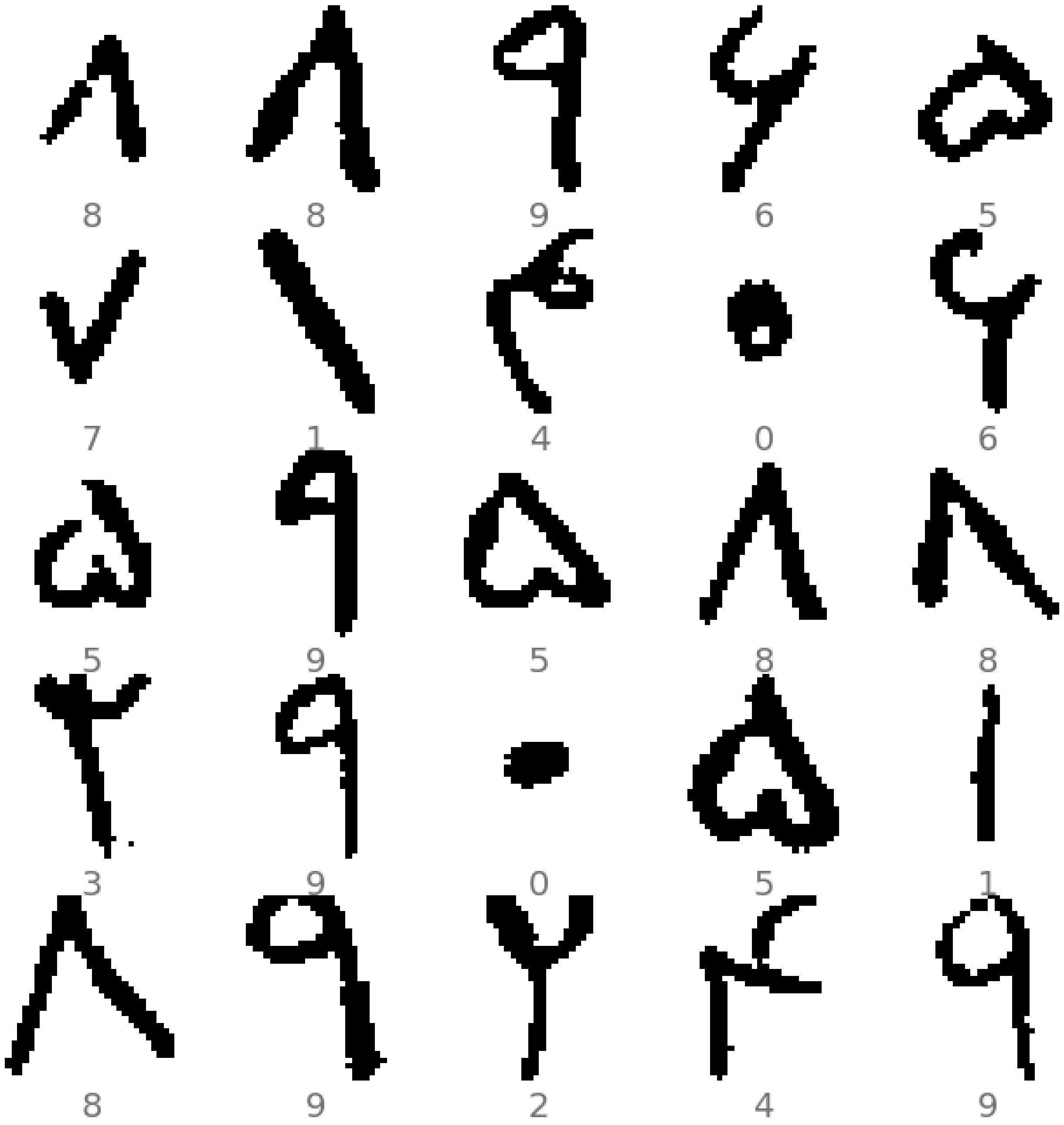
```
100%|██████████| 60000/60000 [00:07<00:00, 8047.52it/s]
100%|██████████| 60000/60000 [00:01<00:00, 43312.01it/s]
100%|██████████| 20000/20000 [00:02<00:00, 7447.51it/s]
100%|██████████| 20000/20000 [00:00<00:00, 39107.24it/s]

Train: (60000, 32, 32), Labels: #60000
Test: (20000, 32, 32), Labels: #20000
```

Visualize the data

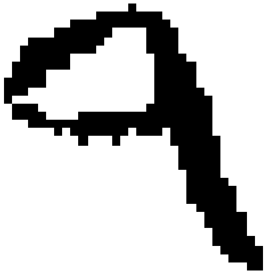
In [8]:

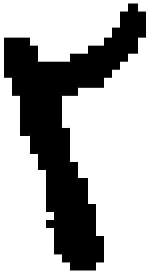
```
plt.figure(figsize=(8, 8))
for i in range(25):
    plt.subplot(5, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train[i], cmap=plt.cm.binary)
    plt.xlabel(str(y_train[i]))
plt.show()
```

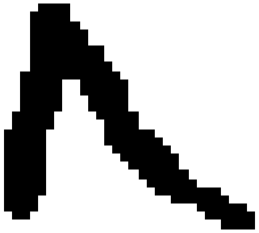

8
8
9
6
5
7
1
4
0
6
5
9
5
8
8
3
9
0
5
1
8
9
2
4
9

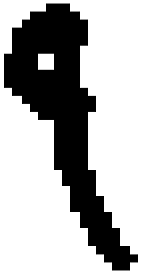
In [9]:

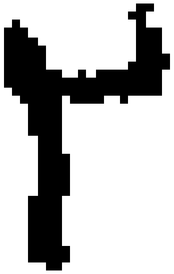
```python
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_test[i], cmap=plt.cm.binary)
    plt.xlabel(str(y_test[i]))
plt.show()
```

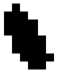

9

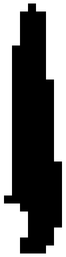

2

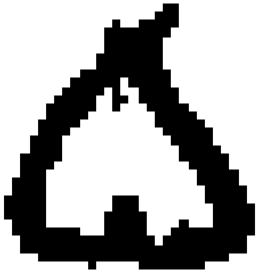

8

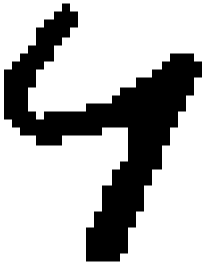

9


2

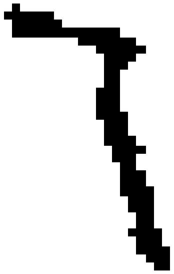

0

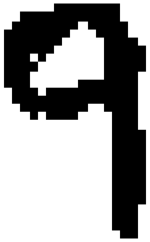

1

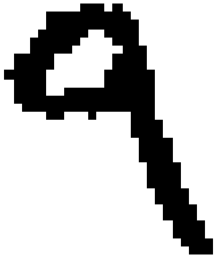

5

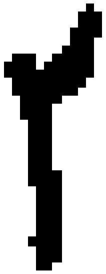

6

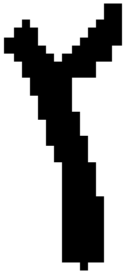
0

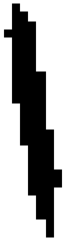

6

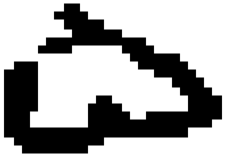

9


9

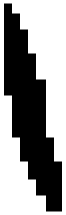

2

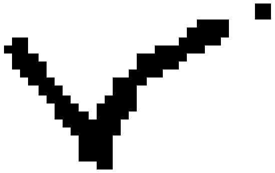

2

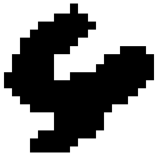

1

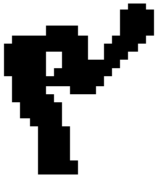

5

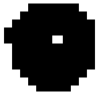
1

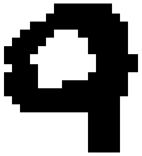

1

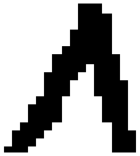

7

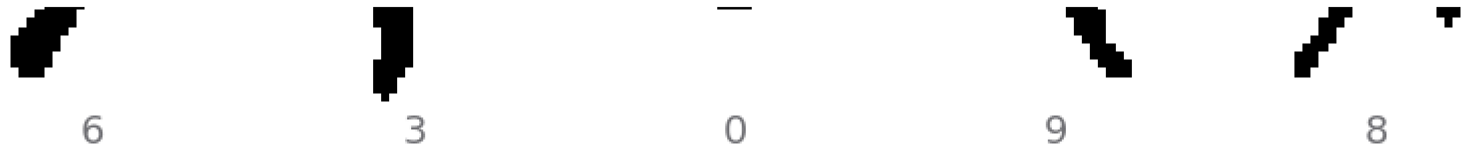












Preprocessing

```
In [0]: #
```

Configure Neural Network Models

Create non-linear model

```
In [11]: print(x_train.shape)
         print(y_train.shape)
         print(y_train[0])
```

```
(60000, 32, 32)
(60000,)
8
```

Simple Model

```
In [62]: def build_simple_model():

             model = tf.keras.Sequential([
                 tf.keras.layers.Flatten(input_shape=(32, 32)),
                 tf.keras.layers.Dense(6, activation='relu'),
                 tf.keras.layers.Dense(4, activation='relu',
                 tf.keras.layers.Dense(10, activation='softmax')
             ])

             model.compile(optimizer='adam',
                           loss='sparse_categorical_crossentropy',
                           metrics=['accuracy'])

             return model
```

```
  File "<ipython-input-62-bb10a135e7ef>", line 8
    ])
    ^
SyntaxError: invalid syntax
```

Regualization Model

In [0]:

```
def build_regularization_model():
    l1 = tf.keras.regularizers.l1(0.001)

    model = tf.keras.Sequential([
        tf.keras.layers.Flatten(input_shape=(32, 32)),
        tf.keras.layers.Dense(6, activation='relu',
                              activity_regularizer=l1),
        tf.keras.layers.Dense(4, activation='relu',
                              activity_regularizer=l1),
        tf.keras.layers.Dense(10, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

Dropout Model

In [0]:

```
def build_dropout_model():

    model = tf.keras.Sequential([
        tf.keras.layers.Flatten(input_shape=(32, 32)),
        tf.keras.layers.Dense(6, activation='relu'),
        tf.keras.layers.Dropout(rate=0.1),
        tf.keras.layers.Dense(4, activation='relu'),
        tf.keras.layers.Dropout(rate=0.1),
        tf.keras.layers.Dense(10, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

Summary of the model

In [67]:

```
model = build_simple_model()
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_2 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_5 (Dense)              (None, 6)                 6150
_________________________________________________________________
features (Dense)             (None, 4)                 28
_________________________________________________________________
dense_6 (Dense)              (None, 10)                50
=================================================================
Total params: 6,228
Trainable params: 6,228
Non-trainable params: 0
_________________________________________________________________
```

In [0]:

```
# !rm -rf logs/scaler
```

In [69]:

```
from datetime import datetime

datetime.now().strftime('%Y%m%d-%H%M%S')
```

Out[69]:

```
'20190723-204437'
```

In [0]:

```
logdir = 'logs/scaler/' + datetime.now().strftime('%Y%m%d-%H%M%S')
tb = tf.keras.callbacks.TensorBoard(logdir,
                                    histogram_freq=1,
                                    write_graph=False,
                                    write_images=False)
```

In [75]:

```
r = model.fit(x_train, y_train,
          validation_split=0.01,
          batch_size=128,
          epochs=10,
          verbose=1, callbacks=[tb])


history_dict = r.history
history_list = list(history_dict.keys())
print(history_list)
```

```
Train on 59400 samples, validate on 600 samples
Epoch 1/10
59400/59400 [==============================] - 2s 32us/sample - loss: 0.2055 - accuracy: 0.9463 - va
l_loss: 0.1576 - val_accuracy: 0.9533
Epoch 2/10
59400/59400 [==============================] - 2s 33us/sample - loss: 0.2012 - accuracy: 0.9472 - va
l_loss: 0.1508 - val_accuracy: 0.9583
Epoch 3/10
59400/59400 [==============================] - 2s 33us/sample - loss: 0.1967 - accuracy: 0.9486 - va
l_loss: 0.1595 - val_accuracy: 0.9550
Epoch 4/10
59400/59400 [==============================] - 2s 33us/sample - loss: 0.1931 - accuracy: 0.9496 - va
l_loss: 0.1527 - val_accuracy: 0.9600
Epoch 5/10
59400/59400 [==============================] - 2s 34us/sample - loss: 0.1907 - accuracy: 0.9497 - va
l_loss: 0.1547 - val_accuracy: 0.9533
Epoch 6/10
59400/59400 [==============================] - 2s 33us/sample - loss: 0.1867 - accuracy: 0.9512 - va
l_loss: 0.1485 - val_accuracy: 0.9583
Epoch 7/10
59400/59400 [==============================] - 2s 33us/sample - loss: 0.1836 - accuracy: 0.9513 - va
l_loss: 0.1518 - val_accuracy: 0.9567
Epoch 8/10
59400/59400 [==============================] - 2s 33us/sample - loss: 0.1814 - accuracy: 0.9523 - va
l_loss: 0.1502 - val_accuracy: 0.9600
Epoch 9/10
59400/59400 [==============================] - 2s 32us/sample - loss: 0.1794 - accuracy: 0.9526 - va
l_loss: 0.1495 - val_accuracy: 0.9583
Epoch 10/10
59400/59400 [==============================] - 2s 33us/sample - loss: 0.1764 - accuracy: 0.9541 - va
l_loss: 0.1443 - val_accuracy: 0.9583
['loss', 'accuracy', 'val_loss', 'val_accuracy']
```

Plotting

In [0]:

```
# !kill 1477
```

In [79]:

```
%reload_ext tensorboard

%tensorboard --logdir logs/scaler
```

Evaluation

In [0]:

```
def plot_image(i, predictions_array, true_label, img):
    predictions_array, true_label, img = predictions_array[i], true_label[i], img[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    plt.imshow(img, cmap=plt.cm.binary)

    predicted_label = np.argmax(predictions_array)
    if predicted_label == true_label:
        color = 'blue'
    else:
        color = 'red'

    plt.xlabel("{} {:2.0f}% ({})".format(str(predicted_label),
                                         100 * np.max(predictions_array),
                                         str(true_label)), color=color)


def plot_value_array(i, predictions_array, true_label):
    predictions_array, true_label = predictions_array[i], true_label[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    thisplot = plt.bar(range(10), predictions_array, color="#777777")
    plt.ylim([0, 1])
    predicted_label = np.argmax(predictions_array)

    thisplot[predicted_label].set_color('red')
    thisplot[true_label].set_color('blue')
```

In [81]:

```
predictions = model.predict(x_test)
print(predictions.shape)
```

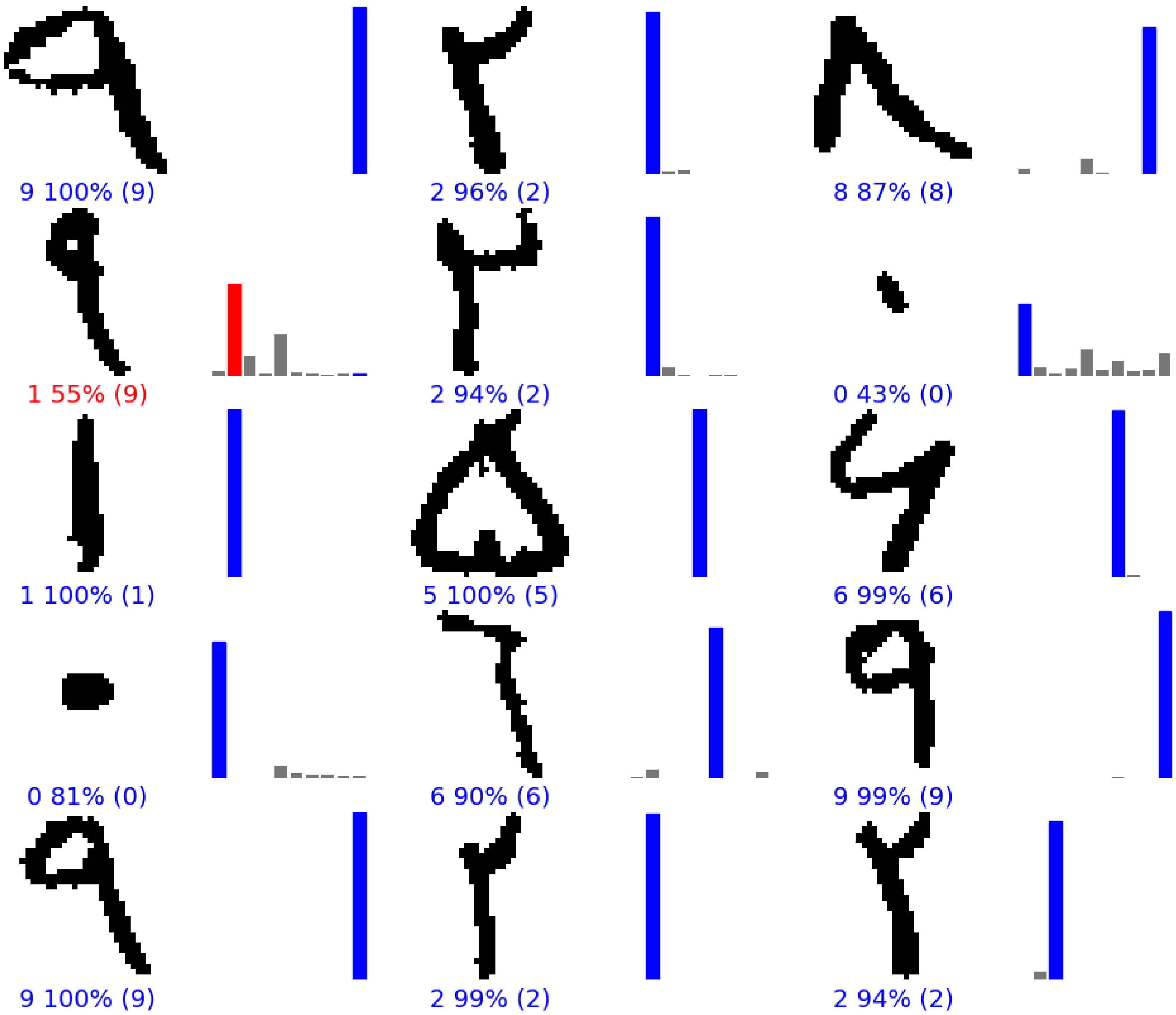
```
(20000, 10)
```

In [82]:

```
num_rows = 5
num_cols = 3
num_images = num_rows * num_cols
plt.figure(figsize=(2 * 2 * num_cols, 2 * num_rows))

for i in range(num_images):
    plt.subplot(num_rows, 2 * num_cols, 2 * i + 1)
    plot_image(i, predictions, y_test, x_test)
    plt.subplot(num_rows, 2 * num_cols, 2 * i + 2)
    plot_value_array(i, predictions, y_test)

plt.show()
```


9 100% (9)
2 96% (2)
8 87% (8)
1 55% (9)
2 94% (2)
0 43% (0)
1 100% (1)
5 100% (5)
6 99% (6)
0 81% (0)
6 90% (6)
9 99% (9)
9 100% (9)
2 99% (2)
2 94% (2)

In [0]: