Keras_mnist_analysis_DL

August 22, 2018

0.1 Keras example: mnist analysis by DL (CV2D)

```
In [1]: %%time
        from keras.datasets import mnist
        (X_train0, y_train0), (X_test0, y_test0) = mnist.load_data()
```

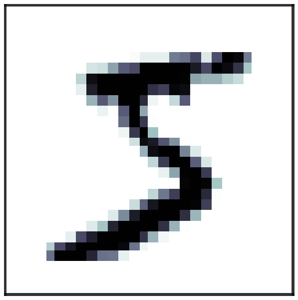
```
Using TensorFlow backend.


CPU times: user 1.01 s, sys: 584 ms, total: 1.59 s
Wall time: 874 ms
```

```
In [2]: print(X_train0.shape, X_train0.dtype)
        print(y_train0.shape, y_train0.dtype)
        print(X_test0.shape, X_test0.dtype)
        print(y_test0.shape, y_test0.dtype)
```

```
(60000, 28, 28) uint8
(60000,) uint8
(10000, 28, 28) uint8
(10000,) uint8
```

```
In [3]: import matplotlib.pyplot as plt
        import matplotlib as mpl
        %matplotlib inline
```

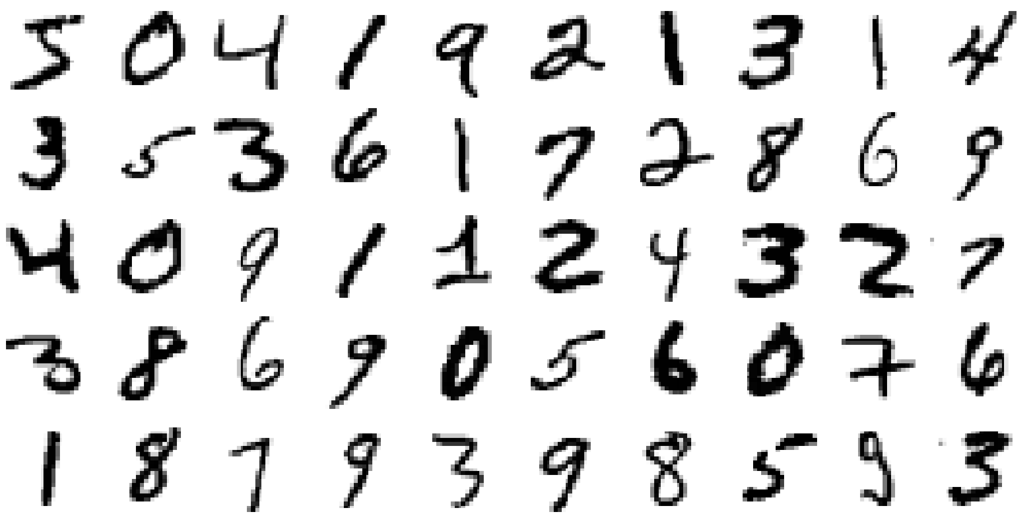
```
In [4]: plt.figure(figsize=(2, 2))
        plt.imshow(X_train0[0], cmap=mpl.cm.bone_r)
        plt.grid(False)
        plt.xticks([])
        plt.yticks([])
        plt.show()
```



0.1.1 Show images of numbers

```
In [5]: #
        import numpy as np
        # import matplotlib as mpl
        def plot_digits(instances, images_per_row=10, **options):
            size = 28
            images_per_row = min(len(instances), images_per_row)
            images = [instance.reshape(size,size) for instance in instances]
            n_rows = (len(instances) - 1) // images_per_row + 1
            row_images = []
            n_empty = n_rows * images_per_row - len(instances)
            images.append(np.zeros((size, size * n_empty)))
            for row in range(n_rows):
                rimages = images[row * images_per_row : (row + 1) * images_per_row]
                row_images.append(np.concatenate(rimages, axis=1))
            image = np.concatenate(row_images, axis=0)
            plt.imshow(image, cmap = mpl.cm.binary, **options)
            plt.axis("off")
```

```
In [6]: plt.figure(figsize=(9,9))
        example_images = np.r_[X_train0[:50]]
        plot_digits(example_images, images_per_row=10)

        plt.show()
```



0.1.2 float .

```
In [7]: X_train = X_train0.reshape(60000, 28, 28, 1).astype('float32') / 255.0
        X_test = X_test0.reshape(10000, 28, 28, 1).astype('float32') / 255.0
        print(X_train.shape, X_train.dtype)
```

```
(60000, 28, 28, 1) float32
```

0.1.3 y One-Hot-Encoding .

```
In [8]: y_train0[:5]
```

```
Out[8]: array([5, 0, 4, 1, 9], dtype=uint8)
```

```
In [9]: from keras.utils import np_utils

        Y_train = np_utils.to_categorical(y_train0, 10)
        Y_test = np_utils.to_categorical(y_test0, 10)
        Y_train[:5]
```

```
Out[9]: array([[0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
               [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
               [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
               [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
               [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]], dtype=float32)
```

0.2

0.2.1 Keras .

1. Sequential
2. add layer .
   - Dense layer
   - .
   - .
   - input_dim .
   - activation activation
3. compile .
   - loss Loss
   - optimizer
   - metrics
4. fit
   - nb_epoch epoch
   - batch_size mini batch size
   - metrics
   - Jupyter Notebook verbose=2 progress bar .

In [10]:
```
from keras.optimizers import SGD
import numpy as np
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.callbacks import ModelCheckpoint,EarlyStopping
```

In [11]:
```
# Deep Learning model
np.random.seed(0)

# Simple NN
# model = Sequential()
# model.add(Dense(15, input_dim=784, activation="sigmoid"))  # firsr layer
# model.add(Dense(10, activation="sigmoid")) # output layer
# model.compile(optimizer=SGD(lr=0.2), loss='mean_squared_error', metrics=["accuracy"

#
model = Sequential()
# model.add(Conv2D(32, kernel_size=(3, 3), input_dim=784, activation='relu'))
model.add(Conv2D(32, kernel_size=(3, 3), input_shape=(28, 28, 1), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.25))
model.add(Flatten())
```

```
        model.add(Dense(128,  activation='relu'))
        model.add(Dropout(0.5))
        model.add(Dense(10, activation='softmax'))
```

model_to_dot summary layers .

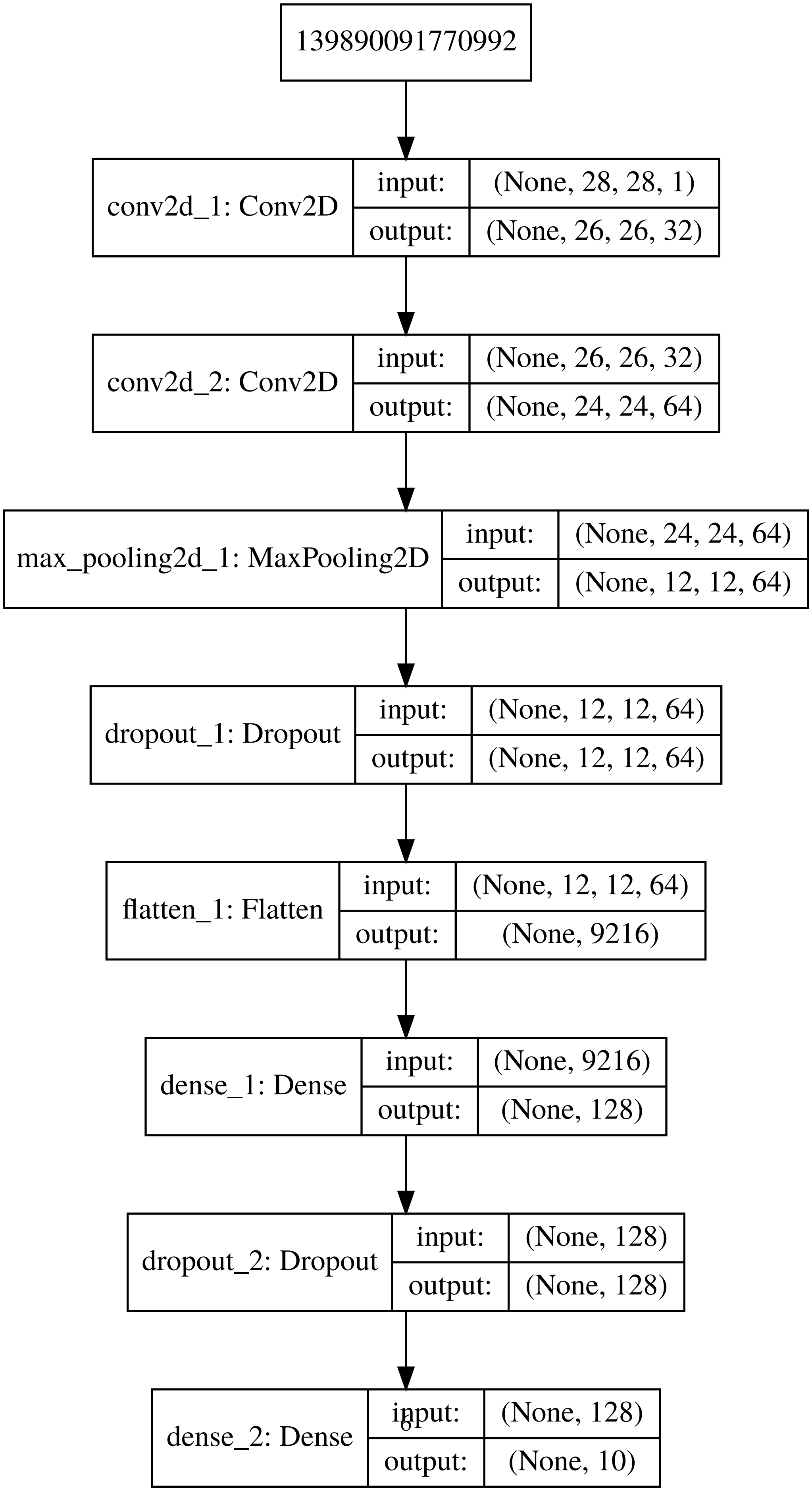
```
In [12]: # !pip install pydot
```

```
In [13]: # !pip install GraphViz
```

```
In [14]: import pydot
         import graphviz
```

```
In [15]: from IPython.display import SVG
         from keras.utils.vis_utils import model_to_dot

         SVG(model_to_dot(model, show_shapes=True).create(prog='dot', format='svg'))
```

```
Out[15]:
```


139890091770992
conv2d_1: Conv2D
input:
(None, 28, 28, 1)
output:
(None, 26, 26, 32)
conv2d_2: Conv2D
input:
(None, 26, 26, 32)
output:
(None, 24, 24, 64)
max_pooling2d_1: MaxPooling2D
input:
(None, 24, 24, 64)
output:
(None, 12, 12, 64)
dropout_1: Dropout
input:
(None, 12, 12, 64)
output:
(None, 12, 12, 64)
flatten_1: Flatten
input:
(None, 12, 12, 64)
output:
(None, 9216)
dense_1: Dense
input:
(None, 9216)
output:
(None, 128)
dropout_2: Dropout
input:
(None, 128)
output:
(None, 128)
dense_2: Dense
input:
(None, 128)
output:
(None, 10)

```
In [16]: from keras.utils import plot_model
         plot_model(model, to_file='model_DL.png')
```

```
In [17]: model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 64)        0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 9216)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 128)               1179776   
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290      
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
```

```
In [18]: l1 = model.layers[0]
         l2 = model.layers[1]
         l6 = model.layers[5]
```

```
In [19]: l1.name, type(l1), l1.output_shape, l1.activation.__name__, l1.count_params()
```

```
Out[19]: ('conv2d_1',
          keras.layers.convolutional.Conv2D,
          (None, 26, 26, 32),
          'relu',
          320)
```

```
In [20]: l2.name, type(l2), l2.output_shape, l2.activation.__name__, l2.count_params()
```

```
Out[20]: ('conv2d_2',
          keras.layers.convolutional.Conv2D,
```

```
(None, 24, 24, 64),
'relu',
18496)
```

```
In [21]: l6.name, type(l6), l6.output_shape, l6.activation.__name__, l6.count_params()
```

```
Out[21]: ('dense_1', keras.layers.core.Dense, (None, 128), 'relu', 1179776)
```

0.3 fit

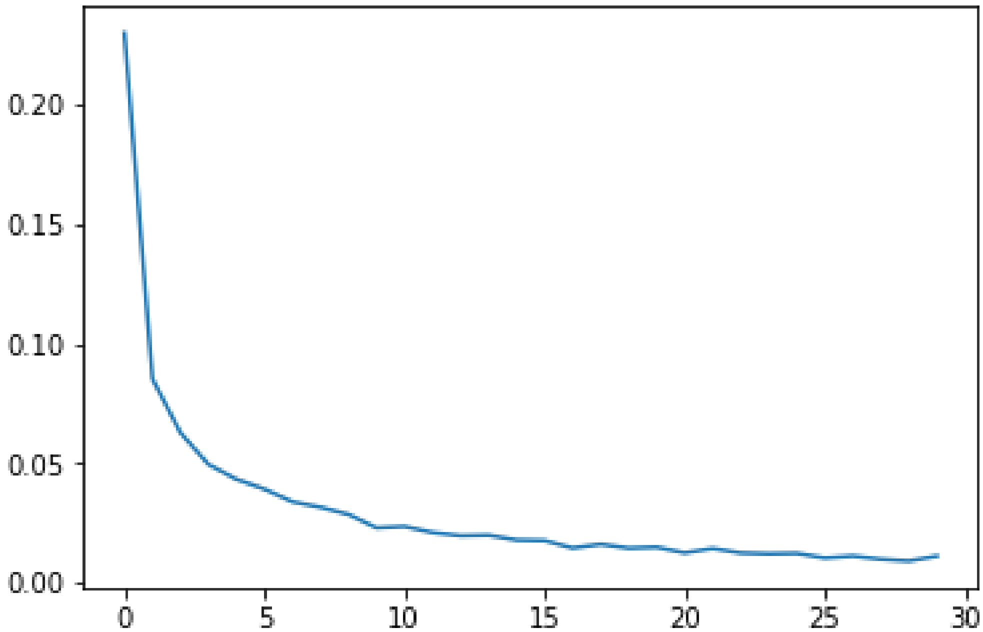
```
In [22]: model.compile(loss='categorical_crossentropy',
                       optimizer='adam',
                       metrics=['accuracy'])
```

```
In [23]: %%time
         hist = model.fit(X_train, Y_train,
                          epochs=30, batch_size=100,
                          validation_data=(X_test, Y_test),
                          verbose=1)
```

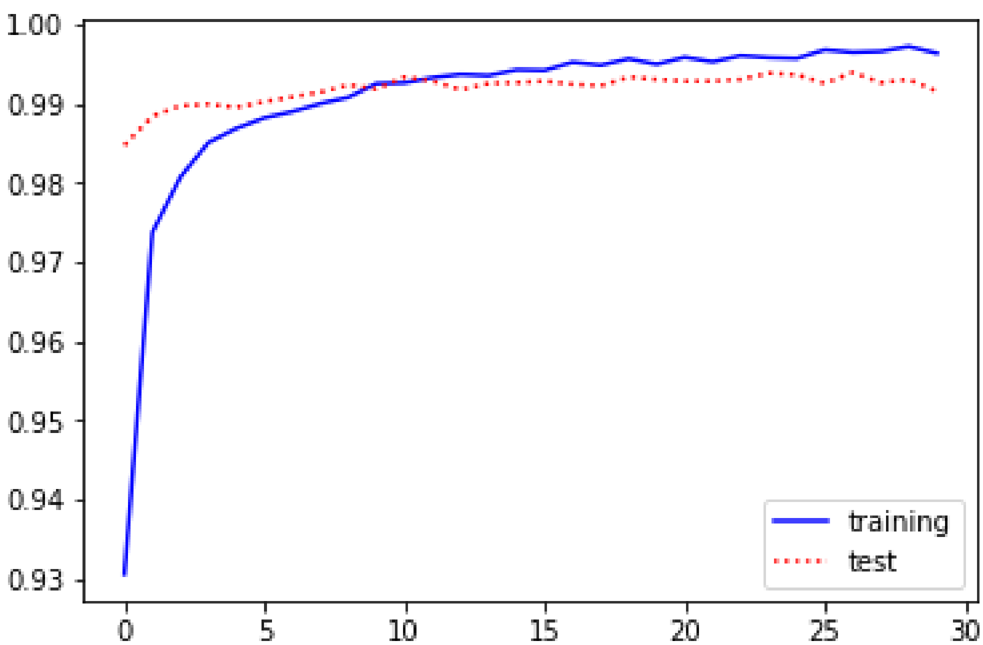
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/30
60000/60000 [==============================] - 5s 80us/step - loss: 0.2300 - acc: 0.9306 - val_
Epoch 2/30
60000/60000 [==============================] - 3s 58us/step - loss: 0.0853 - acc: 0.9737 - val_
Epoch 3/30
60000/60000 [==============================] - 3s 57us/step - loss: 0.0627 - acc: 0.9807 - val_
Epoch 4/30
60000/60000 [==============================] - 4s 60us/step - loss: 0.0494 - acc: 0.9851 - val_
Epoch 5/30
60000/60000 [==============================] - 4s 60us/step - loss: 0.0432 - acc: 0.9869 - val_
Epoch 6/30
60000/60000 [==============================] - 4s 62us/step - loss: 0.0391 - acc: 0.9882 - val_
Epoch 7/30
60000/60000 [==============================] - 4s 60us/step - loss: 0.0338 - acc: 0.9890 - val_
Epoch 8/30
60000/60000 [==============================] - 4s 60us/step - loss: 0.0315 - acc: 0.9900 - val_
Epoch 9/30
60000/60000 [==============================] - 4s 61us/step - loss: 0.0285 - acc: 0.9908 - val_
Epoch 10/30
60000/60000 [==============================] - 4s 59us/step - loss: 0.0229 - acc: 0.9925 - val_
Epoch 11/30
60000/60000 [==============================] - 3s 58us/step - loss: 0.0234 - acc: 0.9926 - val_
Epoch 12/30
60000/60000 [==============================] - 3s 57us/step - loss: 0.0209 - acc: 0.9932 - val_
Epoch 13/30
60000/60000 [==============================] - 3s 56us/step - loss: 0.0196 - acc: 0.9937 - val_
Epoch 14/30
60000/60000 [==============================] - 3s 56us/step - loss: 0.0198 - acc: 0.9935 - val_
```

```
Epoch 15/30
60000/60000 [==============================] - 4s 60us/step - loss: 0.0177 - acc: 0.9942 - val_
Epoch 16/30
60000/60000 [==============================] - 3s 56us/step - loss: 0.0175 - acc: 0.9942 - val_
Epoch 17/30
60000/60000 [==============================] - 3s 55us/step - loss: 0.0144 - acc: 0.9952 - val_
Epoch 18/30
60000/60000 [==============================] - 3s 58us/step - loss: 0.0160 - acc: 0.9948 - val_
Epoch 19/30
60000/60000 [==============================] - 3s 57us/step - loss: 0.0143 - acc: 0.9956 - val_
Epoch 20/30
60000/60000 [==============================] - 3s 52us/step - loss: 0.0147 - acc: 0.9949 - val_
Epoch 21/30
60000/60000 [==============================] - 3s 57us/step - loss: 0.0124 - acc: 0.9958 - val_
Epoch 22/30
60000/60000 [==============================] - 3s 55us/step - loss: 0.0142 - acc: 0.9952 - val_
Epoch 23/30
60000/60000 [==============================] - 3s 55us/step - loss: 0.0123 - acc: 0.9960 - val_
Epoch 24/30
60000/60000 [==============================] - 3s 54us/step - loss: 0.0120 - acc: 0.9958 - val_
Epoch 25/30
60000/60000 [==============================] - 3s 54us/step - loss: 0.0121 - acc: 0.9957 - val_
Epoch 26/30
60000/60000 [==============================] - 3s 54us/step - loss: 0.0102 - acc: 0.9968 - val_
Epoch 27/30
60000/60000 [==============================] - 3s 54us/step - loss: 0.0110 - acc: 0.9964 - val_
Epoch 28/30
60000/60000 [==============================] - 3s 53us/step - loss: 0.0097 - acc: 0.9966 - val_
Epoch 29/30
60000/60000 [==============================] - 3s 53us/step - loss: 0.0090 - acc: 0.9972 - val_
Epoch 30/30
60000/60000 [==============================] - 3s 54us/step - loss: 0.0110 - acc: 0.9963 - val_
CPU times: user 2min 11s, sys: 20 s, total: 2min 31s
Wall time: 1min 43s
```

```
In [24]: # Plot performance
         plt.plot(hist.history['loss'])
         plt.show()
```



In [25]:
```
plt.plot(hist.history['acc'], 'b-', label="training")
plt.plot(hist.history['val_acc'], 'r:', label="test")
plt.legend()
plt.show()
```



0.4

get_weights . w b .

```
In [26]: w1 = l1.get_weights()
         w1[0].shape, w1[1].shape
```

```
Out[26]: ((3, 3, 1, 32), (32,))
```

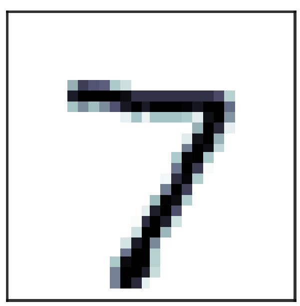
```
In [27]: w2 = l2.get_weights()
         w2[0].shape, w2[1].shape
```

```
Out[27]: ((3, 3, 32, 64), (64,))
```

0.5

predict y y predict_classes classification .

```
In [28]: plt.figure(figsize=(2, 2))
         plt.imshow(X_test0[0], cmap=mpl.cm.bone_r)
         plt.grid(False)
         plt.xticks([])
         plt.yticks([])
         plt.show()
```



```
In [29]: model.predict(X_test[:1, :])
```

```
Out[29]: array([[2.2876680e-24, 1.4718440e-16, 1.1257238e-17, 6.9971253e-15,
                 1.3605593e-22, 5.6539288e-20, 1.9341417e-31, 1.0000000e+00,
                 1.1402606e-21, 1.6830274e-14]], dtype=float32)
```

```
In [30]: model.predict_classes(X_test[:1, :], verbose=0)
```

```
Out[30]: array([7])
```

0.6 DL

save hdf5 load .

```
In [31]: model.save('my_model_dl.hdf5')
         # del model
```

```
In [32]: from keras.models import load_model

         model2 = load_model('my_model_dl.hdf5')
         model2.predict_classes(X_test[:1, :], verbose=0)
```

```
Out[32]: array([7])
```

```
In [33]: model2.predict_classes(X_test[:10, :], verbose=0)
```

```
Out[33]: array([7, 2, 1, 0, 4, 1, 4, 9, 5, 9])
```

```
In [34]: y_test0[:10]
```

```
Out[34]: array([7, 2, 1, 0, 4, 1, 4, 9, 5, 9], dtype=uint8)
```

0.6.1

```
In [35]: # Correct prediction
         model2.predict_classes(X_test[8:9, :], verbose=1)
```

```
1/1 [==============================] - 0s 963us/step
```

```
Out[35]: array([5])
```

```
In [36]: y_test0[8]
```

```
Out[36]: 5
```

```
In [37]: x_pred = model2.predict_classes(X_test, verbose=1)
```

```
10000/10000 [==============================] - 0s 28us/step
```

```
In [38]: t_count = np.sum(x_pred==y_test0) # True positive
         f_count = np.sum(x_pred!=y_test0) # False positive
         f_count==10000-t_count
```

```
Out[38]: True
```

```
In [39]: t_count,f_count
```

```
Out[39]: (9915, 85)
```

```
In [40]: accuracy = t_count/10000*100
         accuracy
```

```
Out[40]: 99.15
```

0.7 DL is great!!!