

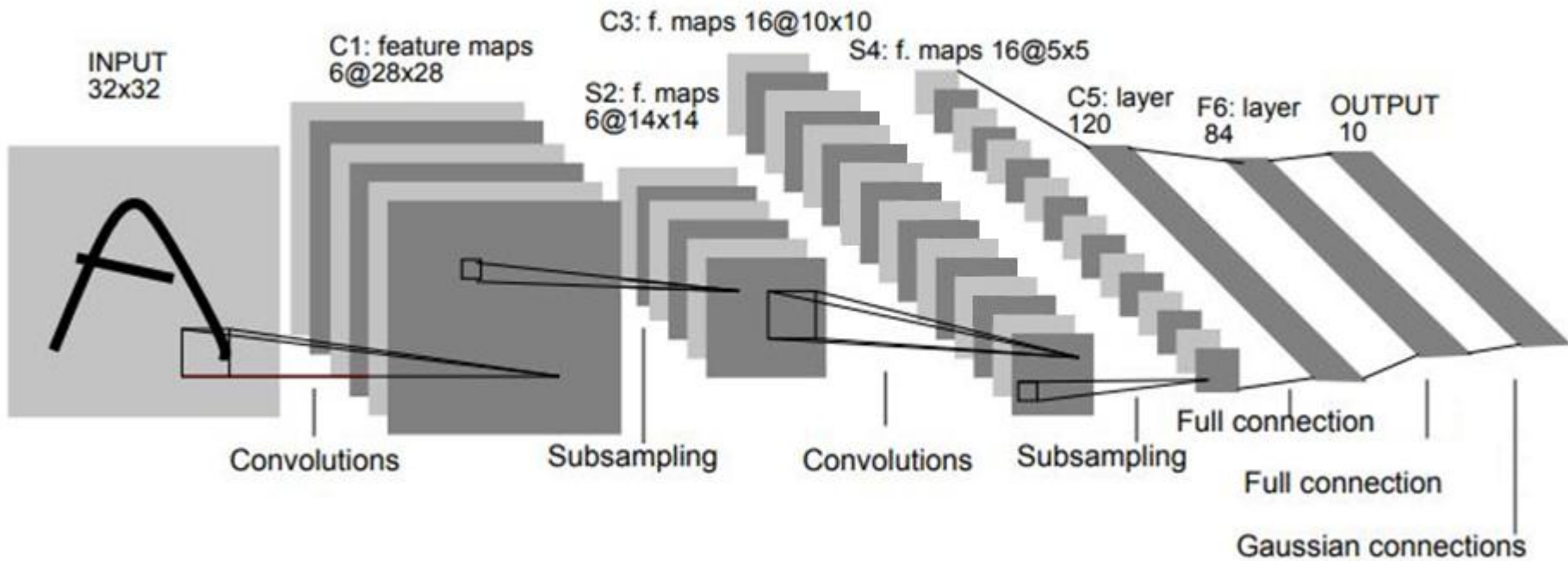
بینایی کامپیوتر فصل دوم

LENET IMPLEMENTATION IN KERAS

محمدجواد فدائی اسلام

اردیبهشت ۱۴۰۵

LENET-5



LENET-5 ARCHITECTURE SUMMARIZED TABLE

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

ABOUT KERAS

- Keras is a deep learning API written in Python, running on top of the machine learning platform **TensorFlow**.
- It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result as fast as possible is key to doing good research.*
- Keras is:
 - **Simple**
 - **Flexible**
 - **Powerful**

PACKAGE DECLARATION



 mnist2.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved



+ Code + Text



✓
0s



```
#Setup
import tensorflow as tf
import keras
#import cirq
import sympy
import numpy as np
import seaborn as sns
import collections

# visualization tools
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```



LOAD MNIST

Load the MNIST dataset distributed with Keras.



```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Rescale the images from [0,255] to the [0.0,1.0] range.
x_train, x_test = x_train[..., np.newaxis]/255.0, x_test[..., np.newaxis]/255.0

print("Number of original training examples:", len(x_train))
print("Number of original test examples:", len(x_test))
x_train, x_valid, y_train, y_valid = train_test_split(x_train, y_train, test_size=0.2)

y_train = keras.utils.to_categorical(y_train)
y_valid = keras.utils.to_categorical(y_valid)
y_test = keras.utils.to_categorical(y_test)
```



```
Number of original training examples: 60000
Number of original test examples: 10000
```

Show an example

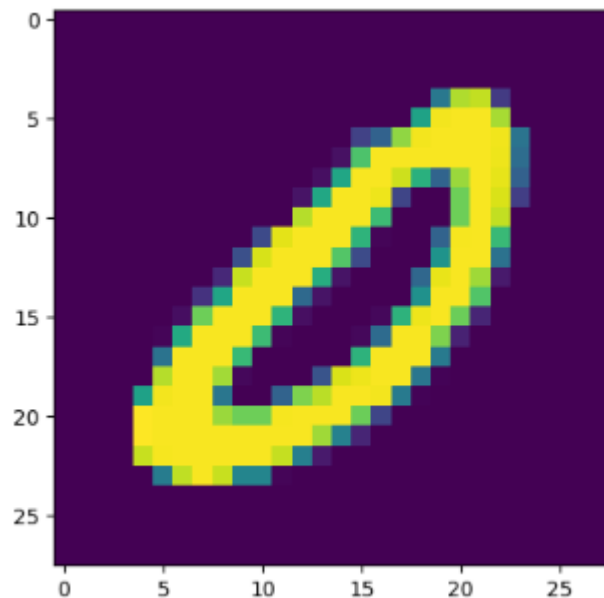
✓
0s



```
print("Size of the sample:",y_train[0])  
plt.imshow(x_train[0, :, :, 0])
```



```
Size of the sample: [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]  
<matplotlib.image.AxesImage at 0x783e9c6f3160>
```



MODEL

```
def create_classical_model():
    # A simple model based off LeNet from https://keras.io/examples/mnist\_cnn/
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Conv2D(filters = 6, kernel_size = (5, 5),
                                      activation='relu', padding="same", input_shape=(28,28,1)))
    model.add(tf.keras.layers.AveragePooling2D(2,2))
    model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'))
    model.add(tf.keras.layers.AveragePooling2D(2,2))
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(units=120, activation='relu'))
    model.add(tf.keras.layers.Dense(units=84, activation='relu'))
    model.add(tf.keras.layers.Dense(units=10, activation = 'softmax'))
    return model

model = create_classical_model()
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])
model.summary()
```

MODEL.SUMMARY()

Model: "sequential_15"

Layer (type)	Output Shape	Param #
conv2d_27 (Conv2D)	(None, 28, 28, 6)	156
average_pooling2d_24 (AveragePooling2D)	(None, 14, 14, 6)	0
conv2d_28 (Conv2D)	(None, 12, 12, 16)	880
average_pooling2d_25 (AveragePooling2D)	(None, 6, 6, 16)	0
flatten_12 (Flatten)	(None, 576)	0
dense_36 (Dense)	(None, 120)	69,240
dense_37 (Dense)	(None, 84)	10,164
dense_38 (Dense)	(None, 10)	850

Total params: 81,290 (317.54 KB)

Trainable params: 81,290 (317.54 KB)

Non-trainable params: 0 (0.00 B)

COMPILE

✓ [112] np.set_printoptions(precision=3)
44s np.set_printoptions(suppress=True)

```
model.fit(x_train,  
          y_train,  
          batch_size=128,  
          epochs=1,  
          verbose=1,  
          validation_data=(x_valid, y_valid))
```

```
cnn_results = model.evaluate(x_test, y_test)  
print(cnn_results)
```

⇒ 375/375 ————— 30s 79ms/step - accuracy: 0.9604 - loss: 0.1362 - val_accuracy: 0.9693
313/313 ————— 3s 9ms/step - accuracy: 0.9690 - loss: 0.1021
[0.08899986743927002, 0.9739999771118164]

✓ 0s ▶ print(model.predict(x_test[0:1,:,:,:]))
print(y_test[0:1:])

⇒ WARNING:tensorflow:5 out of the last 11 calls to <function TensorFlowTrainer.make_predict_function.<
1/1 ————— 0s 130ms/step
[[0. 0. 0. 0.001 0. 0. 0. 0.998 0. 0.001]]
[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]]

train_test_split

`sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)` [\[source\]](#)

Split arrays or matrices into random train and test subsets.

Quick utility that wraps input validation, `next(ShuffleSplit().split(X, y))`, and application to input data into a single call for splitting (and optionally subsampling) data into a one-liner.

Read more in the [User Guide](#).

Parameters:

***arrays** : *sequence of indexables with same length / shape[0]*

Allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes.

test_size : *float or int, default=None*

If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples. If None, the value is set to the complement of the train size. If `train_size` is also None, it will be set to 0.25.

train_size : *float or int, default=None*

If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the train split. If int, represents the absolute number of train samples. If None, the value is automatically set to the complement of the test size.

to_categorical function

[source]

```
keras.utils.to_categorical(x, num_classes=None)
```

Converts a class vector (integers) to binary class matrix.

E.g. for use with `categorical_crossentropy`.

Arguments

- **x**: Array-like with class values to be converted into a matrix (integers from 0 to `num_classes - 1`).

5 →

0
0
0
0
0
1
0
0
0
0

TF.KERAS.LAYERS.CONV2D, ARGUMENTS

- **filters**: int, the dimension of the output space (the number of filters in the convolution).
- **kernel_size**: int or tuple/list of 2 integer, specifying the size of the convolution window.
- **strides**: int or tuple/list of 2 integer, specifying the stride length of the convolution.
- **padding**: string, either "valid" or "same" (case-insensitive). "valid" means no padding.
When padding="same" and strides=1, the output has the same size as the input.
- **activation**: Activation function. If None, no activation is applied.

TF.KERAS.LAYERS.AVERAGEPOOLING2D

ARGUMENTS

- **pool_size**: int or tuple of 2 integers, factors by which to downscale (dim1, dim2). If only one integer is specified, the same window length will be used for all dimensions.
- **strides**: int or tuple of 2 integers, or None. Strides values. If None, it will default to **pool_size**. If only one int is specified, the same stride size will be used for all dimensions.
- **padding**: string, either "valid" or "same" (case-insensitive).

TF.KERAS.LAYERS.DENSE

ARGUMENTS

- **units:** Positive integer, dimensionality of the output space.
- **activation:** Activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: $a(x) = x$).

MODEL.FIT

- **x**: Input data.
- **y**: Target data. Like the input data **x**,
- **batch_size**: Integer or **None**. Number of samples per gradient update. If unspecified, **batch_size** will default to 32.
- **epochs**: Integer. Number of epochs to train the model. An epoch is an iteration over the entire **x** and **y** data provided.
- **verbose**: **"auto"**, 0, 1, or 2. Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per epoch. "auto" becomes 1 for most cases. Note that the progress bar is not particularly useful when logged to a file, so **verbose=2** is recommended when not running interactively (e.g., in a production environment). Defaults to **"auto"**.
- **validation_data**: Data on which to evaluate the loss and any model metrics at the end of each epoch. The model will not be trained on this data.

MODEL.COMPILE

- **optimizer**: String (name of optimizer) or optimizer instance. See [keras.optimizers](#).
- **loss**: Loss function. **metrics**: List of metrics to be evaluated by the model during training and testing. Typically you will use [metrics=\['accuracy'\]](#).

MODEL.PREDICT

ARGUMENTS

- **x**: Input
- **verbose**: ["auto"](#), 0, 1, or 2. Verbosity mode. 0 = silent, 1 = progress bar, 2 = single line.